ระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรม

HIGH-RESOLUTION GRAPHICS FOR INDUSTRIAL PROCESS MONITORING

ภาคย พุทธรังกาศ

PHAAORN NUTASANGKAS

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2531

ระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรม

HIGH-RESOLUTION GRAPHICS FOR INDUSTRIAL PROCESS MONITORING

ภากร หุตะสังกาศ

PHAKORN HUTASANGKAS

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ กิตติ ตีรเศรษฐ

ADVISOR

Assc.Prof.KITTI TIRASESTH

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2531

บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
แบบฟอร์มการให้คะแนนการสอบวิทยานิพนธ์
สำหรับนักศึกษาระดับมหาบัณฑิต

ชื่อนักศึกษา นายภากร หุตะสังกาศ เลขประจำตัว 29.0006
ชื่อเรื่องวิทยานิพนธ์ ระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรม (High Resolution Graphics for Industrial Process Monitoring)

ชื่ออาจารย์ผู้ควบคุมการสอบ	ลายมือชื่อ	ผลการสอบ
รศ.กิตติ ตีรเศรษฐ		ดีเยี่ยม
ผศ.ดร.ฟูศักดิ์ ชีวสุวิทย์		ดีเยี่ยม
ดร.รัตติกร วรากูลศิริพันธุ์		ผ่าน
ดร.วรวัฒน์ ลิ้มโภคา		ดี

วันเดือนปี ที่สอบ 17 ตุลาคม 2531 เวลา 13.30 น. สถานที่ ห้อง A-305

(นายวิทยา โกมารชุดม)
คณบดีบัณฑิตวิทยาลัย

สารบัญ

	หน้า
บทคัดย่อ	I
ABSTRACT	II
บทที่ 1 บทนำ	1
บทที่ 2 โครงสร้างทั่วไปของระบบ	3
2.1 ส่วนควบคุมจอแสดงผลความละเอียดสูง	4
2.2 ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม	5
2.3 โปรแกรมควบคุมระบบ	6
2.4 โปรแกรมเงื่อนไขการแสดงผล	7
2.5 คอมพิวเตอร์ควบคุมระบบ	7
2.6 อุปกรณ์ร่วมอื่นๆ	8
บทที่ 3 วงจรส่วนควบคุมจอแสดงผลความละเอียดสูง	10
3.1 ส่วนเชื่อมต่อกับคอมพิวเตอร์	10
3.2 ส่วนควบคุมการแสดงผลแบบกราฟฟิค	12
3.3 หน่วยความจำแสดงผล	23
บทที่ 4 วงจรส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม	26
4.1 วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบดิจิตอล	28
4.2 วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบอนาลอก	37
4.3 วงจรเอาท์พุทสำหรับสัญญาณเตือนในแบบดิจิตอล	44
บทที่ 5 โปรแกรมควบคุมระบบ	52
5.1 ตำแหน่งแอดเดรสและฟังก์ชั่น	52
5.2 รายละเอียดเกี่ยวกับรีจิสเตอร์ STATUS	53
5.3 ชุดคำสั่งของ GDC	54
5.4 รายละเอียดเกี่ยวกับคำสั่งต่างๆที่จำเป็นและใช้ในโปรแกรมควบคุมระบบ	58

5.5 การเปลี่ยนตำแหน่ง X-Y เป็นตำแหน่งแอดเดรสในหน่วยความจำ 66
5.6 การวาดแบบเวคเตอร์ 67
5.7 การออกแบบโปรแกรมควบคุมระบบ 73

บทที่ 6 การสร้างภาพกระบวนการและการกำหนดอุปกรณ์ในกระบวนการ 86
6.1 การสร้างภาพของกระบวนการ 89
6.2 การแบ่งชั้นของภาพกระบวนการ 89
6.3 การสร้างภาพลงในคอมพิวเตอร์ 94
6.4 การสร้างชุดคำสั่งสำหรับระบบแสดงผล 102
6.5 การแยกอุปกรณ์ในไฟล์ชุดคำสั่งการวาด 105
ตัวอย่างการสร้างภาพกระบวนการและการกำหนดอุปกรณ์ในกระบวนการ 109

บทที่ 7 โปรแกรมเงื่อนไขการแสดงผล 115
7.1 การกำหนดช่วงเวลาการทำงานแต่ละงาน 115
7.2 การกำหนดสีของอุปกรณ์ในสภาวะต่างๆ 121
7.3 การกำหนดเงื่อนไขการแสดงผล 122
7.4 การประมาณค่าเวลาตอบสนองของระบบ 128
ตัวอย่างการสร้างโปรแกรมเงื่อนไขการแสดงผล 132

บทที่ 8 บทสรุป 139

กิตติกรรมประกาศ 140
เอกสารอ้างอิง 141
ภาคผนวก 1 รายละเอียดเกี่ยวกับจอแสดงผล ADI รุ่น DM-2214
ภาคผนวก 2 รายละเอียดเกี่ยวกับไอซี 82720
ภาคผนวก 3 โปรแกรมที่ใช้ในการทดลอง

หัวข้อวิทยานิพนธ์ ระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรม
นักศึกษา นายภากร หุตะสังกาศ 29.0006
อาจารย์ที่ปรึกษา รศ.กิตติ ตีรเศรษฐ
ระดับการศึกษา วิศวกรรมศาสตรมหาบัณฑิตทางวิศวกรรมไฟฟ้า
ปีการศึกษา 2531

บทคัดย่อ

ปัจจุบันนี้ความก้าวหน้าทางด้านเทคโนโลยีของคอมพิวเตอร์ได้ถูกนำมาประยุกต์ใช้ในกระบวนการทางอุตสาหกรรมเป็นอย่างมาก ไม่ว่าจะเป็นด้านการวัดหรือบันทึกค่าตัวแปร การควบคุม การแสดงผลทางด้านกราฟฟิค และอื่นๆ อย่างไรก็ดีในกรณีของกระบวนการทางอุตสาหกรรมที่มีขนาดใหญ่และซับซ้อนนั้น การใช้คอมพิวเตอร์ในการแสดงผลต่างๆ ด้วยจอแสดงผลหรือจอภาพยังมีขีดจำกัดอยู่อันเนื่องมาจากความละเอียดของจอแสดงผลเอง และข้อจำกัดของส่วนควบคุมจอแสดงผล

วิทยานิพนธ์ฉบับนี้ขอเสนอการออกแบบระบบแสดงผลที่มีความละเอียดสูงเพื่อแก้ไขข้อจำกัดดังกล่าวข้างต้น และออกแบบส่วนควบคุมเพื่อให้มีการแสดงผลพร้อมกับ 2 จอภาพได้ในเวลาเดียวกัน โดยที่ภาพจะถูกปรับให้เลื่อนขึ้นลง หรือเคลื่อนไปทางซ้ายขวาได้ จึงทำให้นำมาประยุกต์ใช้งานได้อย่างมีประสิทธิภาพดียิ่งขึ้น นอกจากนี้ผลจากการออกแบบระบบแสดงผลที่ทำงานได้ด้วยความเร็วสูง จึงทำให้สามารถนำคอมพิวเตอร์มาใช้ในการประมวลผลข้อมูลของกระบวนการ และนำผลที่ได้มาแสดงผลบนจอภาพหรือพิมพ์ออกมาทางเครื่องพิมพ์ได้ และเพื่อให้การประยุกต์ใช้ในงานกระบวนการทางอุตสาหกรรมได้ดียิ่งขึ้น ในตอนท้ายของวิทยานิพนธ์ฉบับนี้จึงได้ออกแบบและพัฒนาซอฟท์แวร์เพื่อใช้ในการกำหนดเงื่อนไขต่างๆ สำหรับนำมาใช้เป็นสัญญาณเตือนของกระบวนการทางอุตสาหกรรม โดยผู้ใช้หรือผู้ควบคุมสามารถจะกำหนดเงื่อนไขต่างๆได้โดยง่ายด้วยตัวเอง

Thesis Title HIGH-RESOLUTION GRAPHICS FOR
INDUSTRIAL PROCESS MONITORING
Name PHAKORN HUTASANGKAS
Thesis Advisor Assc. Prof. KITTI TIRASESTH
Level of Study MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
Academic Year 1988

ABSTRACT

The fast develop of computer technology has been applying to most industrial processes nowadays, such as in measurement and control; device and system, as well as the graphic display, etc.. However, in case of large and complicate industrial process, the low application of computer to display the model of the process, variables and so on, are still have some limitation due to the resolution of the CRT and the limitation of the graphic display controller itself.

This paper presents one method of the high-resolution graphic display design, in order to overcome the limitation mentioned above. Moreover, the graphic display controller is also designed to have spacial functions that the two CRTs can be used to display in the same time, and the picture displayed can be scrolled in both up - down and left - right directions. That is, it gains more efficacy for industrial processes applications. And because of the high-speed display is designed, the computer then can be used for processing of the data of the process, and display on the CRT or print out through the printer, without any affect to the displayed system. At the last of this thesis, a software for the user to designate any alarm conditions of the industrial process is also designed. Since the condition can be set by the user or the controller easily, this software then also

contribute more efficiency to the overall design and application to the industrial processes.

บทที่ 1
บทนำ

การติดตามการเปลี่ยนแปลงสภาวะของกระบวนการทางอุตสาหกรรม นับว่ามีความสำคัญอย่างมากสำหรับการควบคุมกระบวนการและการควบคุมคุณภาพของผลิตภัณฑ์ ซึ่งมีวิธีอยู่หลายวิธีแต่ละวิธีก็มีข้อดีข้อเสียแตกต่างกันออกไป วิธีหนึ่งซึ่งใช้ได้ผลดีก็คือการนำเอาคอมพิวเตอร์มาประยุกต์ใช้ในการติดตามการเปลี่ยนแปลงดังกล่าว วิธีการก็คือส่งข้อมูลสภาวะของกระบวนการผ่านตัวกลางชนิดต่างๆมาให้กับคอมพิวเตอร์ ซึ่งคอมพิวเตอร์จะนำข้อมูลเหล่านั้นมาประมวลผลแล้วนำไปแสดงผลทางจอแสดงผล โดยอาจจะใช้วิธีเปลี่ยนสี เปลี่ยนข้อความ เปลี่ยนรูป เป็นต้น ซึ่งผู้ใช้ก็จะเป็นผู้กำหนดเงื่อนไขต่างๆเหล่านี้ แต่จะกำหนดในลักษณะใดและมีขอบเขตเพียงใดก็ขึ้นอยู่กับระบบหรือโปรแกรมนั้นๆ แต่จากการศึกษาโปรแกรมเหล่านี้แล้วก็พบว่า มักจะไม่สามารถแสดงภาพกระบวนการขนาดใหญ่หรือซับซ้อนได้ ทำให้ต้องแบ่งออกเป็นหลายๆภาพหรือตัดเอารายละเอียดของภาพออกไป ซึ่งทำให้ยุ่งยากและไม่สะดวก บางโปรแกรมก็ไม่สามารถนำข้อมูลมาประมวลผลทางคณิตศาสตร์ได้ต้องแสดงผลในรูปแบบของข้อมูลดิบเลย และบางโปรแกรมก็ไม่สามารถพิมพ์ผลหรือสภาวะของกระบวนการออกทางเครื่องพิมพ์ได้ นอกจากนั้นก็ยังมีข้อกำหนดทางด้านราคาเนื่องจากส่วนใหญ่มักจะเป็นระบบขนาดใหญ่ซึ่งรวมอยู่กับเครื่องควบคุมกระบวนการ ทำให้มีราคาแพงและเป็นการใช้งานเฉพาะด้านจึงไม่สามารถนำคอมพิวเตอร์มาใช้งานอื่นได้

วิทยานิพนธ์ฉบับนี้ขอเสนอระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรม ซึ่งได้ออกแบบให้แก้ไขข้อจำกัดเหล่านั้นโดยสามารถแสดงผลได้พร้อมๆกัน 2 จอภาพ และสามารถกำหนดตำแหน่งในการแสดงผลได้โดยการเลื่อนภาพในตำแหน่งต่างๆด้วยคันโยก (JOYSTICK) ทำให้สามารถนำไปใช้กับกระบวนการขนาดใหญ่และซับซ้อนได้สะดวก การนำข้อมูลมาแสดงผลก็กระทำได้ทั้งการนำเอาข้อมูลดิบมาแสดงผลเลย หรืออาจจะนำไปประมวลผลทางคณิตศาสตร์เพื่อให้อยู่ในรูปที่ต้องการก่อนก็ได้ และสามารถแสดงการเปลี่ยนแปลงสภาวะได้ทั้งการเปลี่ยนสี การเปลี่ยนข้อความ การเปลี่ยนรูป และอื่นๆ นอกจากนั้นยังสามารถส่งสัญญาณไปควบคุมอุปกรณ์ภายนอกได้อีกด้วย ซึ่งวัตถุประสงค์ก็คือการส่งสัญญาณเตือนในกรณีต่างๆ เช่น ระดับของเหลวสูงเกินกว่าระดับที่กำหนดไว้ซึ่งอาจจะเกิดจากวาล์ว หรือ เครื่องควบคุมทำงานผิดพลาด ทั้งนี้การกำหนดเงื่อนไขต่างๆจะถูกกำหนดโดยโปรแกรม ซึ่งผู้ใช้สามารถแก้ไขหรือพัฒนาขึ้นเองได้ทำให้ระบบมีความยืดหยุ่นสูง เนื่องจากผู้ใช้สามารถพัฒนาโปรแกรมให้สอดคล้องกับความต้องการได้อย่างถูกต้อง นอก

จากนั้นส่วนควบคุมการแสดงผลยังทำงานด้วยความเร็วสูงทำให้ผู้ใช้สามารถกำหนดให้คอมพิวเตอร์พิมพ์ค่าสภาวะหรือผลต่างๆออกทางเครื่องพิมพ์ได้โดยไม่มีผลกระทบต่อความเร็วในการแสดงผลของระบบเลย และเนื่องจากระบบนี้เป็นการนำเอาคอมพิวเตอร์มาประยุกต์ใช้โดยไม่มีการใช้ฮาร์ดแวร์ (HARDWARE) ร่วมกับ ดังนั้นจึงสามารถใช้กับคอมพิวเตอร์ได้หลายชนิด ทำให้มีราคาถูกและสามารถนำคอมพิวเตอร์ไปใช้งานอื่นๆได้

ระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรมนั้นแบ่งออกเป็นส่วนใหญ่ๆได้ 6 ส่วนดังนี้

1.ส่วนควบคุมจอแสดงผลความละเอียดสูง

2.ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม

3.โปรแกรมควบคุมระบบ

4.โปรแกรมเงื่อนไขในการแสดงผล

5.คอมพิวเตอร์ควบคุมระบบ

6.อุปกรณ์ร่วมอื่นๆ เช่น เครื่องพิมพ์ คันโยก เป็นต้น

สำหรับในบทต่อไปจะเป็นการกล่าวถึงโครงสร้างโดยทั่วไปของระบบโดยย่อ ซึ่งจะอธิบายถึงการจัดระบบทั้งหมด วัตถุประสงค์ที่เลือกใช้หรือออกแบบอุปกรณ์ต่างๆ ส่วนในบทที่ 3 จะเป็นการอธิบายถึงส่วนควบคุมจอแสดงผลความละเอียดสูงและทฤษฎีที่เกี่ยวข้อง รวมถึงวงจรที่ใช้ การออกแบบ และการพัฒนาโดยละเอียด บทที่ 4 ก็จะอธิบายถึงส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม ซึ่งจะประกอบด้วยทฤษฎีที่เกี่ยวข้อง วงจรที่ใช้และการออกแบบ บทที่ 5 เป็นโครงสร้างของโปรแกรมควบคุมระบบซึ่งส่วนใหญ่จะเป็นการควบคุมส่วนแสดงผล บทที่ 6 จะเป็นการกล่าวถึงการสร้างภาพ วิธีการและข้อกำหนด ในตอนท้ายของบทก็จะเป็นตัวอย่างในการสร้างโดยละเอียด ส่วนโปรแกรมในการทำเงื่อนไขและแสดงสภาวะของกระบวนการจะอธิบายไว้ในบทที่ 7 ซึ่งในการใช้งานจริงแล้วผู้ใช้จะเป็นผู้พัฒนาขึ้นเองดังนั้นในการอธิบายก็จะใช้ตัวอย่างประกอบ และนำโปรแกรมที่สร้างไว้เป็นตัวอย่าง ส่วนในบทสุดท้ายจะเป็นการสรุปวิทยานิพนธ์ฉบับนี้ทั้งหมด รวมทั้งปัญหาที่พบ แนวทางในการพัฒนาสำหรับผู้ที่ต้องการจะศึกษาหรือวิจัยในหัวข้อเดียวกันหรือเกี่ยวข้อง

บทที่ 2
โครงสร้างโดยทั่วไปของระบบ

ระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรมนั้นแบ่งออกเป็นส่วนใหญ่ๆได้ 6 ส่วนดังนี้

1.ส่วนควบคุมจอแสดงผลความละเอียดสูง

2.ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม

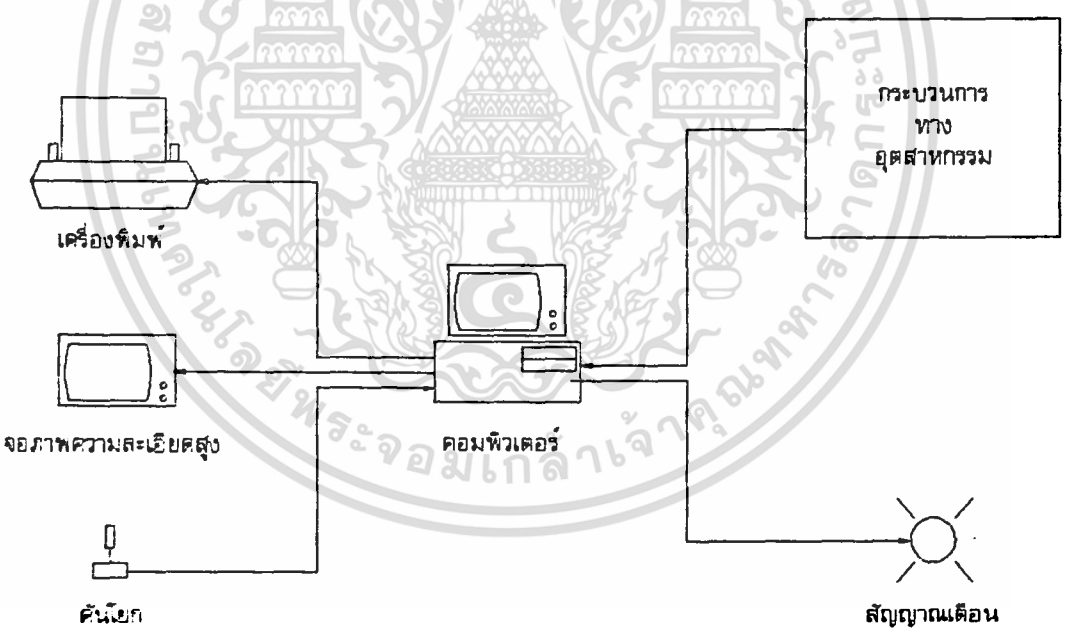
3.โปรแกรมควบคุมระบบ

4.โปรแกรมเงื่อนไขในการแสดงผล

5.คอมพิวเตอร์ควบคุมระบบ

6.อุปกรณ์ร่วมอื่นๆ เช่น เครื่องพิมพ์ คันโยก เป็นต้น

ซึ่งทั้ง 6 ส่วนสามารถเขียนเป็นผังภาพ (BLOCK DIAGRAM) รวมของระบบได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงผังภาพรวมของระบบ

2.1 ส่วนควบคุมจอแสดงผลความละเอียดสูง

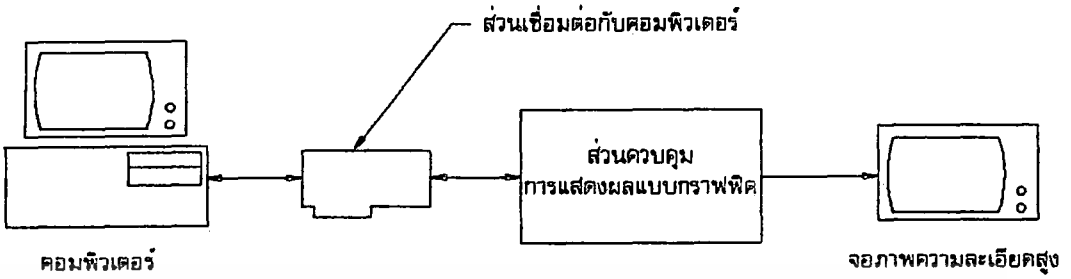
ซึ่งเป็นส่วนสำคัญที่สุดของระบบ เนื่องจากจะต้องควบคุมการแสดงผลในจอแสดงผลความละเอียดสูงที่ใช้ในการแสดงภาพของกระบวนการ จะประกอบด้วยส่วนย่อยๆอีก 3 ส่วนดังนี้

2.1.1 ส่วนเชื่อมต่อกับคอมพิวเตอร์ : ส่วนนี้จะเป็นส่วนที่เชื่อมต่อกับคอมพิวเตอร์ในการรับส่งค่าพารามิเตอร์ (PARAMETER) และคำสั่งสำหรับการวาด ซึ่งส่วนนี้ถูกออกแบบให้เป็นพอร์ทแบบขนาน (PARALLEL PORT) เพื่อความสะดวกในการนำไปประยุกต์ใช้กับคอมพิวเตอร์ชนิดอื่น ในวิทยานิพนธ์ฉบับนี้ได้ออกแบบส่วนนี้สำหรับเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์ IBM PERSONAL COMPUTER ซึ่งจะเป็นรุ่น PC XT หรือ AT ก็ได้ ส่วนการดัดแปลงไปใช้กับคอมพิวเตอร์ชนิดอื่นๆนั้นจะอธิบายไว้ในบทต่อไป

2.1.2 ส่วนควบคุมการแสดงผลแบบกราฟฟิค : ส่วนนี้เป็นส่วนควบคุมจอแสดงผลความละเอียดสูง โดยการแสดงผลจะใช้วิธีแสดงผลแบบกราฟฟิคทั้งหมด เนื่องจากการแสดงผลด้วยภาพของกระบวนการจะทำให้ผู้ใช้ปฏิบัติงานได้สะดวกและมีประสิทธิภาพมากกว่าการใช้ข้อความหรือตัวเลข การออกแบบได้เลือกใช้ไอซีควบคุมการแสดงผลแบบกราฟฟิค (GRAPHICS DISPLAY CONTROLLER) เบอร์ 82720 ของ Intel หรือ uPD7220 ของ NEC และมีหน่วยความจำสำหรับการแสดงผลได้สูงสุด 1 เมกกะไบท์ (MBytes)

2.1.3 จอแสดงผลความละเอียดสูง : จะเป็นจอแสดงผลแบบสีหรือโมโนโครม (MONOCHORME) ก็ได้ แต่ในการใช้งานจริงควรจะเป็นจอแสดงผลแบบสี เนื่องจากการเปลี่ยนแปลงสภาวะของอุปกรณ์บางชนิด เช่น วาล์ว หรือ ปั๊ม จะแสดงการเปลี่ยนสภาวะด้วยการเปลี่ยนสี ดังนั้นถ้าหากใช้จอแสดงผลแบบโมโนโครมอาจจะสังเกตุเห็นได้ไม่ชัดเจน การใช้จอแสดงผลแบบสีจะทำให้สามารถติดตามการเปลี่ยนแปลงของสภาวะได้ง่าย และจอแสดงผลที่ใช้จะต้องมีความละเอียดสูงเป็นสำคัญ ส่วนความละเอียดจะต้องมีขนาดเท่าใดนั้นสามารถกำหนดได้จากโปรแกรม เนื่องจากผู้ใช้สามารถกำหนดได้จากโปรแกรมควบคุมระบบว่าจอแสดงผลจะมีความละเอียดเท่าใดซึ่งจะต้องตรวจสอบและกำหนดค่าพารามิเตอร์เหล่านี้อย่างระมัดระวัง เนื่องจากการกำหนดค่าพารามิเตอร์เหล่านี้ผิดพลาดอาจจะทำให้จอแสดงผลเกิดความเสียหายได้ ซึ่งจะอธิบายการกำหนดค่าพารามิเตอร์เหล่านี้ และการเลือกใช้จอแสดงผลโดยละเอียดในบทต่อไป

ทั้ง 3 ส่วนนี้สามารถเขียนเป็นผังภาพได้ดังรูปที่ 2.2

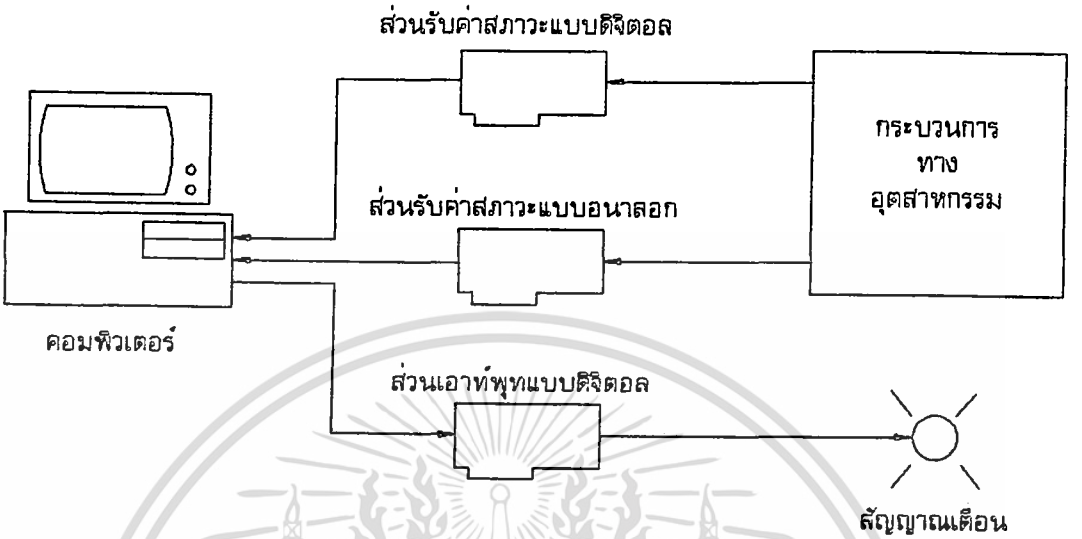


รูปที่ 2.2 แสดงผังภาพของส่วนควบคุมจอแสดงผลความละเอียดสูง

2.2 ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม

ส่วนนี้เป็นส่วนที่รับค่าสภาวะจากกระบวนการทางอุตสาหกรรม ซึ่งในท้องตลาดก็มีจำหน่ายอยู่หลายชนิดขึ้นอยู่กับชนิดของกระบวนการ ประเภทการใช้งาน และรูปแบบการเชื่อมต่อกับกระบวนการ ซึ่งจะมีทั้งแบบดิจิตอล (DIGITAL) และแบบอนาล็อก (ANALOG) ในวิทยานิพนธ์ฉบับนี้ได้ออกแบบไว้ 2 ชนิดคือ แบบดิจิตอลซึ่งค่าแรงดันที่รับเข้ามาจะต้องอยู่ในช่วง 3 ถึง 25 V. สำหรับสภาวะ "เปิด" (ON) และอยู่ในช่วง 0 ถึง 3 V. สำหรับสภาวะ "ปิด" (OFF) และการรับค่าสภาวะแบบอนาล็อกนั้นกำหนดให้รับค่าแบบลูปกระแส (CURRENT LOOP) ซึ่งมีค่าอยู่ระหว่าง 4 ถึง 20 mA. เหตุผลที่เลือกใช้การรับค่าแบบลูปกระแสก็คือ ในการใช้งานจริงนั้นคอมพิวเตอร์และส่วนแสดงผลจะอยู่ห่างจากกระบวนการมาก ดังนั้นการส่งค่าสภาวะจากกระบวนการด้วยลูปกระแสจะเหมาะสมกว่าการส่งด้วยแรงดัน และในการใช้ลูปกระแสก็สามารถเปลี่ยนเป็นแรงดันได้ถ้าต้องการโดยการใช้ความต้านทานเพียงตัวเดียว และนอกจากส่วนนี้จะมีการรับค่าสภาวะจากกระบวนการแล้ว ก็ยังได้ออกแบบให้มีส่วนเอาท์พุท (OUTPUT) เป็นแบบดิจิตอลด้วย เพื่อจุดประสงค์ในการควบคุมอุปกรณ์บางชนิด เช่นสัญญาณเตือนในกรณีที่กระบวนการเกิดความผิดพลาดไปจากเงื่อนไขที่กำหนด เช่น ระดับของของเหลวในถังเพิ่มสูงขึ้นจนเกินจุดสูงสุดซึ่งอาจจะเกิดจากความผิดพลาดของเครื่องควบคุมกระบวนการ หรืออุณหภูมิในจุดต่างๆสูงเกิดกว่าขีดกำหนดซึ่งอาจจะเกิดอันตรายได้ ดังนั้นผู้ควบคุมจะได้ทราบถึงสัญญาณเตือนเหล่านั้น นอกเหนือไปจากสภาวะต่างๆที่ปรากฏบนจอแสดงผลซึ่งบางครั้งอาจจะไม่ได้สังเกตุอยู่ตลอดเวลา

การต่อรวมเข้ากับระบบและการต่อเอาท์พุทสัญญาณเตือนจะมีลักษณะดังในรูปที่ 2.3



รูปที่ 2.3 แสดงผังภาพของส่วนเชื่อมต่อกับ
กระบวนการทางอุตสาหกรรมและการต่อเอาท์พุทสัญญาณเตือน

2.3 โปรแกรมควบคุมระบบ

เป็นโปรแกรมที่ใช้กำหนดพารามิเตอร์ต่างๆของไอซีควบคุมการแสดงผลแบบกราฟฟิค ซึ่งก็คือกำหนดพารามิเตอร์ในการแสดงผลนั่นเอง และดังที่ได้กล่าวมาแล้วในหัวข้อ 2.1.3 ของบทนี้ว่าการกำหนดค่าพารามิเตอร์จะต้องกระทำให้ถูกต้อง เพราะว่าการผิดพลาดในขั้นตอนนี้อาจจะทำให้เกิดความเสียหายต่อจอแสดงผลได้ นอกจากนั้นโปรแกรมนี้ยังมีการเตรียมคำสั่งเกี่ยวกับการวาด และคำสั่งอื่นๆไว้ด้วย

2.4 โปรแกรมเงื่อนไขในการแสดงผล

ในส่วนนี้ได้ออกแบบไว้ให้ผู้ใช้พัฒนาขึ้นเองเกือบทั้งหมด โดยมีการเตรียมคำสั่งที่จำเป็นไว้ให้เพื่อความสะดวกและรวดเร็วไม่ต้องพัฒนาขึ้นเอง วัตถุประสงค์ที่ได้ออกแบบไว้เช่นนี้ก็เนื่องมาจาก กระบวนการทางอุตสาหกรรมนั้นมีแตกต่างกันมากมาย นอกจากนั้นความต้องการในการแสดงผลของผู้ใช้แต่ละรายก็แตกต่างกันออกไป ดังนั้นการออกแบบให้ผู้ใช้ทุกรายสามารถใช้งานได้สะดวกจึงเป็นเรื่องยาก การที่ผู้ใช้ได้พัฒนาโปรแกรมในส่วนนี้ขึ้นเองถึงแม้ว่าจะต้องเสียเวลาในช่วงเริ่มต้นบ้าง แต่ก็ทำให้ระบบสามารถใช้งานได้ตามความประสงค์ของผู้ใช้ โปรแกรมเงื่อนไขในการแสดงผลนั้นจะแบ่งออกเป็นหลายส่วนด้วยกัน เช่นการกำหนดสีของอุปกรณ์ต่างๆให้เปลี่ยนเป็นสีใดเมื่อมีการทำงาน กำหนดช่วงเวลาการพิมพ์ทางเครื่องพิมพ์ เงื่อนไขการส่งสัญญานเตือน หรือการนำเอาข้อมูลดิบที่ได้รับมาจากกระบวนการมาประมวลผลก่อนก็อยู่ในส่วนนี้ ซึ่งผู้ใช้ก็จะต้องเป็นผู้กำหนดเอง เช่นนำค่าอุณหภูมิที่อ่านได้จากจุดที่ 1 ถึง 4 มาเฉลี่ยกันก่อนแล้วจึงนำไปแสดงผลเป็นต้น

2.5 คอมพิวเตอร์ควบคุมระบบ

ในวิทยานิพนธ์ฉบับนี้คอมพิวเตอร์ที่ใช้ควบคุมระบบนั้นได้ออกแบบให้ใช้กับเครื่องไมโครคอมพิวเตอร์ IBM PERSONAL COMPUTER รุ่น PC XT หรือ AT ก็ได้ และจะต้องมีหน่วยความจำชนิดอ่านและเขียนได้ (RAM : RANDOM ACCESS MEMORY) อย่างน้อย 256 กิโลไบท์ (KBytes) และเครื่องขับจานแม่เหล็ก (DISK DRIVE) อย่างน้อย 1 ตัว ส่วนจอแสดงผลจะเป็นชนิดใดก็ได้ขึ้นอยู่กับความต้องการของผู้ใช้ และถ้ามีการประมวลผลทางคณิตศาสตร์ที่ซับซ้อนมากๆก็ควรจะมีหน่วยประมวลผลทางคณิตศาสตร์ด้วยเพื่อให้คอมพิวเตอร์ทำงานได้เร็วขึ้น เหตุผลที่เลือกใช้คอมพิวเตอร์ชนิดนี้เนื่องจากมีราคาถูกและหาได้ง่าย นอกจากนั้นยังมีประสิทธิภาพค่อนข้างสูง และความเร็วในการทำงานของระบบจะขึ้นอยู่กับความเร็วในการประมวลผลของคอมพิวเตอร์ที่ควบคุมระบบด้วย ดังนั้นการเลือกใช้คอมพิวเตอร์ชนิดใดเป็นตัวควบคุมระบบก็จะต้องพิจารณาเงื่อนไขเหล่านี้ด้วย โดยที่ส่วนเชื่อมต่อกับคอมพิวเตอร์ (หัวข้อที่ 1.1) นั้นสามารถดัดแปลงไปใช้กับคอมพิวเตอร์ชนิดอื่นได้ง่าย เนื่องจากออกแบบให้ใช้พอร์ทแบบขนานจึงทำให้ดัดแปลงได้ง่าย

2.6 อุปกรณ์ร่วมอื่นๆ

อุปกรณ์ร่วมอื่นๆนั้นขึ้นอยู่กับความต้องการของผู้ใช้เอง เช่นคันโยกนั้นถ้าหากไม่ต้องการก็สามารถใช้แป้นพิมพ์ควบคุมทิศทางได้เช่นกัน หรือเครื่องพิมพ์ก็ขึ้นอยู่กับว่าผู้ใช้ต้องการพิมพ์ผลหรือไม่ หรืออาจจะมีอุปกรณ์อื่นๆนอกเหนือจากนี้ก็ได้ เช่น เมาส์ (MOUSE) หรือดิจิไตเซอร์ (DIGITIZER) ซึ่งใช้ช่วยในการสร้างภาพกระบวนการก็จะทำให้ทำงานได้สะดวกและรวดเร็วขึ้น

การเลือกใช้อุปกรณ์ร่วมอื่นๆกับระบบนี้นั้นสิ่งสำคัญสิ่งหนึ่งที่จะต้องคำนึงถึงคือ การอ้างตำแหน่งแอดเดรส (ADDRESS) ของหน่วยอินพุทเอาท์พุท (INPUT/OUTPUT : I/O) ซึ่งอาจจะมีการซ้อนทับกัน

ในระบบที่ทำการทดลองประกอบด้วย

1.ส่วนควบคุมจอแสดงผลความละเอียดสูง : ประกอบด้วย

- ส่วนเชื่อมต่อกับคอมพิวเตอร์ : ออกแบบสำหรับคอมพิวเตอร์ IBM PC/XT/AT
- ส่วนควบคุมการแสดงผลแบบกราฟฟิค : มีหน่วยความจำสำหรับการแสดงผล (VIDEO RAM) ขนาด 64 X 3 (3ชุด RGB) กิโลไบท์ (KBytes) ขนาดความยาวของข้อมูล 16 บิท (BIT)
- จอแสดงผลความละเอียดสูง : ใช้จอแสดงผลความละเอียดสูงแบบสีของ ADI รุ่น DM-2214 ขนาด 14" ความละเอียดสูงสุด 770 X 350 จุด

2.ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม : ประกอบด้วย

- ส่วนอินพุทแบบดิจิตอล จำนวน 16 อินพุท แรงดัน 0 ถึง 25 V.
- ส่วนอินพุทแบบอนาลอก จำนวน 8 อินพุท กระแส 4 ถึง 20 mA.
- ส่วนเอาท์พุทแบบดิจิตอล จำนวน 16 เอาท์พุท เป็นหน้าสัมผัสรีเลย์ทนแรงดันได้สูงสุด 150 V_{DC} กระแสสูงสุด 1 A. ขนาดโหลด (LOAD) สูงสุด 10 W.

3.โปรแกรมควบคุมระบบ : ใช้ภาษาปาสคาล (PASCAL) ด้วยตัวแปลภาษาเทอร์โบปาสคาล เวอร์ชั่น 4.0 (TURBO PASCAL Version 4.0)

4.โปรแกรมเงื่อนไขในการแสดงผล : ใช้ภาษาปาสคาล (PASCAL) ด้วยตัวแปลภาษาเทอร์โบปาสคาล เวอร์ชั่น 4.0

5.คอมพิวเตอร์ควบคุมระบบ : ไมโครคอมพิวเตอร์ IBM PC/AT ขนาดหน่วยความจำ 1 เมกกะไบท์ เครื่องขับจานแม่เหล็กขนาด 1.2 เมกกะไบท์ 2 ตัว และฮาร์ดดิสค์ (HARDDISK) ขนาด 40 เมกกะไบท์ พอร์ทแบบขนานเซ็นทรอนิคส์ (CENTRONIC) 2 พอร์ท พอร์ทอนุกรม RS-232C และพอร์ทสำหรับคันโยกอย่างละ 1 พอร์ท การ์ดแสดงผล EGA (ENHANCED GRAPHICS ADAPTER) 128 กิโลไบท์ จอแสดงผลของ ADI รุ่น DM-2214

6.อุปกรณ์ร่วมอื่นๆ : ประกอบด้วย

- เครื่องพิมพ์ชนิดดอทแมททริกซ์ (DOT MATRIX PRINTER) ของ NEC รุ่น PINWRITER P5XL
- พล็อตเตอร์ (PLOTTER) ของ ROLAND รุ่น DXY-980A
- คันโยก

รูปที่ 2.4 เป็นภาพถ่ายจากระบบที่ใช้ในการทดลองในวิทยานิพนธ์นี้ โดยส่วนควบคุมจอแสดงผลความละเอียดสูงอยู่ทางด้านซ้ายของภาพ เป็นกล่องโลหะสีดำ สำหรับส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรมนั้นได้ติดตั้งไว้ในคอมพิวเตอร์ควบคุมระบบซึ่งอยู่ทางตอนกลางของภาพ ทางด้านขวามือของภาพจะเป็นเครื่องพิมพ์ชนิดดอทแมททริกซ์ และพล็อตเตอร์ ส่วนคันโยกจะวางอยู่ทางข้างแป้นพิมพ์ของคอมพิวเตอร์ควบคุมระบบ

รูปที่ 2.4 แสดงระบบที่ใช้ในการทดลอง

บทที่ 3
ส่วนควบคุมจอแสดงผลความละเอียดสูง

ส่วนควบคุมจอแสดงผลความละเอียดสูง เป็นส่วนที่มีความสำคัญอย่างมากในระบบ เนื่องจากจะเป็นการนำเอาข้อมูลสภาวะต่างๆที่ได้รับมาจากกระบวนการทางอุตสาหกรรม และข้อมูลที่ได้จากการประมวลผลของคอมพิวเตอร์นำมาแสดงผลในรูปของภาพ หรือผังภาพของกระบวนการนั้นๆ ซึ่งจะทำให้ผู้ใช้หรือผู้ควบคุมสามารถตรวจสอบการทำงานของกระบวนการได้ตลอดเวลา

ดังที่ได้กล่าวมาแล้วในบทที่ 2 ว่าส่วนควบคุมจอแสดงผลความละเอียดสูงแบ่งออกเป็นส่วนใหญ่ๆได้ 2 ส่วนคือ ส่วนเชื่อมต่อกับคอมพิวเตอร์และส่วนควบคุมการแสดงผลแบบกราฟฟิค ในบทนี้จะขออธิบายแต่ละส่วนโดยละเอียด ดังนี้

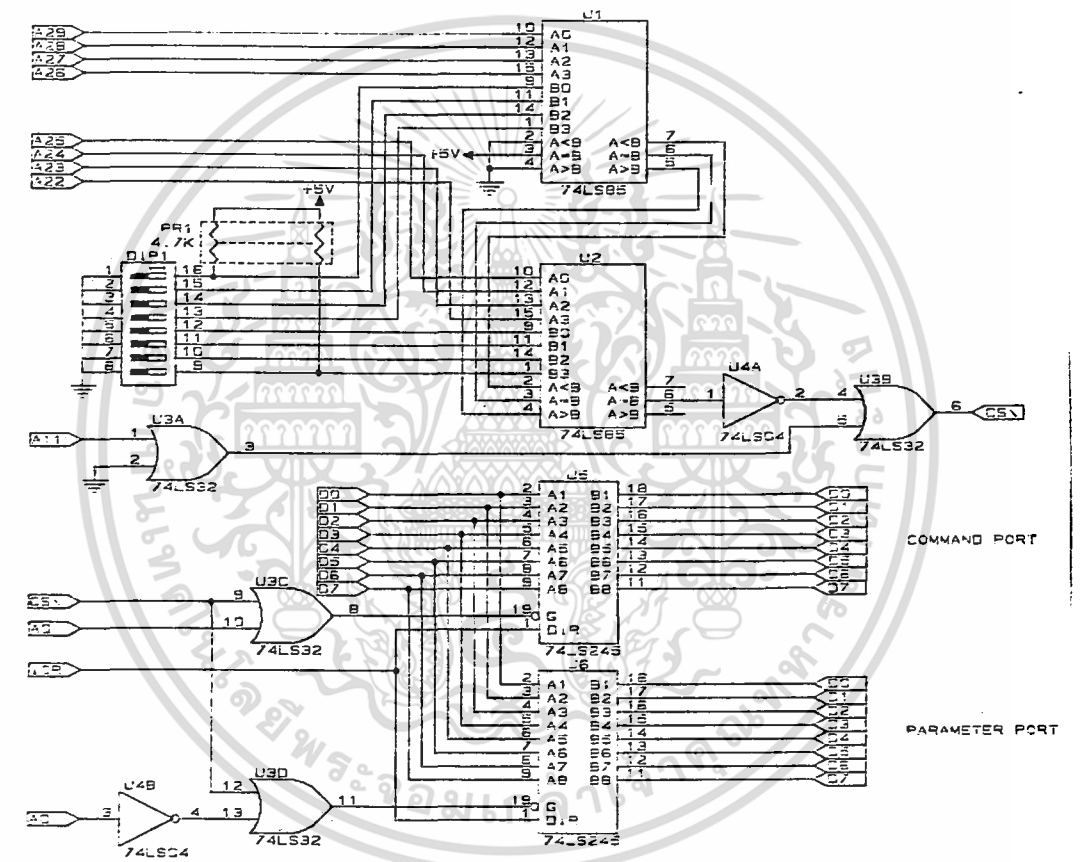
3.1 ส่วนเชื่อมต่อกับคอมพิวเตอร์

ส่วนนี้เป็นวงจรที่ใช้เชื่อมต่อระหว่างคอมพิวเตอร์และส่วนควบคุมการแสดงผลแบบกราฟฟิค การออกแบบได้ออกแบบไว้สำหรับ IBM PC/XT ซึ่งจะมีลักษณะเป็นคาร์ด (CARD) เสียบลงในสล็อท (SLOT) ของ IBM PC/XT และจะมีคอนเนคเตอร์ (CONNECTER) สำหรับต่อสายนำสัญญาณไปยังส่วนควบคุมการแสดงผลแบบกราฟฟิค

หลักการทำงานของวงจร

จากรูปที่ 3.1 เป็นวงจรที่ออกแบบและทดลองเป็นส่วนที่ใช้ในการรับส่งข้อมูลระหว่างคอมพิวเตอร์และส่วนควบคุมการแสดงผลแบบกราฟฟิค ลักษณะของวงจรจะเป็นพอร์ทแบบขนานอยู่ที่แอดเดรส 0300H และ 0301H ซึ่งเป็นทั้งอินพุทพอร์ทและเอาท์พุทพอร์ท เนื่องจากการส่งชุดคำสั่งและพารามิเตอร์ต่างๆก็จะใช้เอาท์พุทพอร์ท และการตรวจสอบบิทต่างๆหรือการอ่านค่าพารามิเตอร์หรือรีจิสเตอร์บางชนิดมาตรวจสอบก็จะต้องใช้อินพุทพอร์ท การออกแบบให้เป็นพอร์ทแบบขนานนี้วัตถุประสงค์ก็เพื่อที่จะทำให้การดัดแปลงไปใช้กับคอมพิวเตอร์ชนิดอื่นเป็นไปได้ง่าย เนื่องจากคอมพิวเตอร์แต่ละชนิดจะมีโครงสร้างทางฮาร์ดแวร์แตกต่างกันออกไป ดังนั้นถ้าออกแบบให้ส่วนควบคุมการแสดงผลติดต่อกับคอมพิวเตอร์โดยตรงแล้ว เมื่อนำวงจรนี้ไปใช้กับคอมพิวเตอร์ชนิดอื่นก็จะต้องออกแบบวงจรใหม่ ซึ่งการออกแบบวงจรในส่วนที่เกี่ยวข้องกับส่วนควบคุมการแสดงผลจะมีความยุ่งยากมาก เพราะว่าส่วนควบคุมการแสดงผลมีการใช้บัสและสัญญาณแตกต่างไปจากคอมพิวเตอร์มาก ดังนั้นการ

ออกแบบให้มีส่วน เชื่อมต่อกับคอมพิว เตอร์ เช่นนี้ก็จะทำให้ไม่ต้องออกแบบวงจรของส่วนควบคุมการแสดงผลใหม่ เพียงแต่ออกแบบส่วน เชื่อมต่อกับคอมพิว เตอร์ใหม่ เท่านั้นซึ่งวงจรดังกล่าวก็ไม่มีความซับซ้อนมากนักสามารถออกแบบวงจรได้ใน เวลาอันสั้น นอกจากนั้นยังสามารถ เปลี่ยนตำแหน่งแอด เดรสที่ใช้งานอยู่ได้ตามต้องการ



รูปที่ 3.1 วงจรส่วน เชื่อมต่อกับคอมพิว เตอร์

วงจรนี้ส่วนหนึ่งเป็นวงจรที่ใช้กำหนดแอดเดรสของส่วนเชื่อมต่อกับคอมพิวเตอร์ และอาจจะมีอุปกรณ์อื่นๆที่ใช้แอดเดรสในตำแหน่ง 0300H อยู่แล้ว ดังนั้นวงจรนี้จะต้องปรับตั้งได้เพื่อให้แอดเดรสแตกต่างกัน ซึ่งในการออกแบบก็ได้ออกแบบให้ปรับตั้งได้ตั้งแต่ 0300H - 03FFH โดยการปรับตั้งที่ดิพสวิทช์ DIP1 ดังนี้

DIP1							
1	2	3	4	5	6	7	8
A2	A3	A4	A5	A6	A7	A8	A9

การปรับตั้งดิพสวิทช์นี้จะมีสภาวะกลับกับสัญญาณที่ต้องการ เช่นถ้าตั้งไว้ที่ "ON" จะมีลอจิก (LOGIC) เป็น "0" เช่นในการทดลองได้ใช้แอดเดรส 0300H - 0303H ก็จะต้องให้สวิทช์ 7,8 อยู่ในสภาวะ "OFF" และ 1,2,3,4,5,6 อยู่ในสภาวะ "ON" และข้อสำคัญสวิทช์ตัวที่ 8 จะต้องอยู่ในสภาวะ "OFF" เสมอ เนื่องจากใน IBM PC นั้นแอดเดรสที่ 0000H - 01FAH ถูกใช้ไปแล้ว

3.2 ส่วนควบคุมการแสดงผลแบบกราฟฟิค

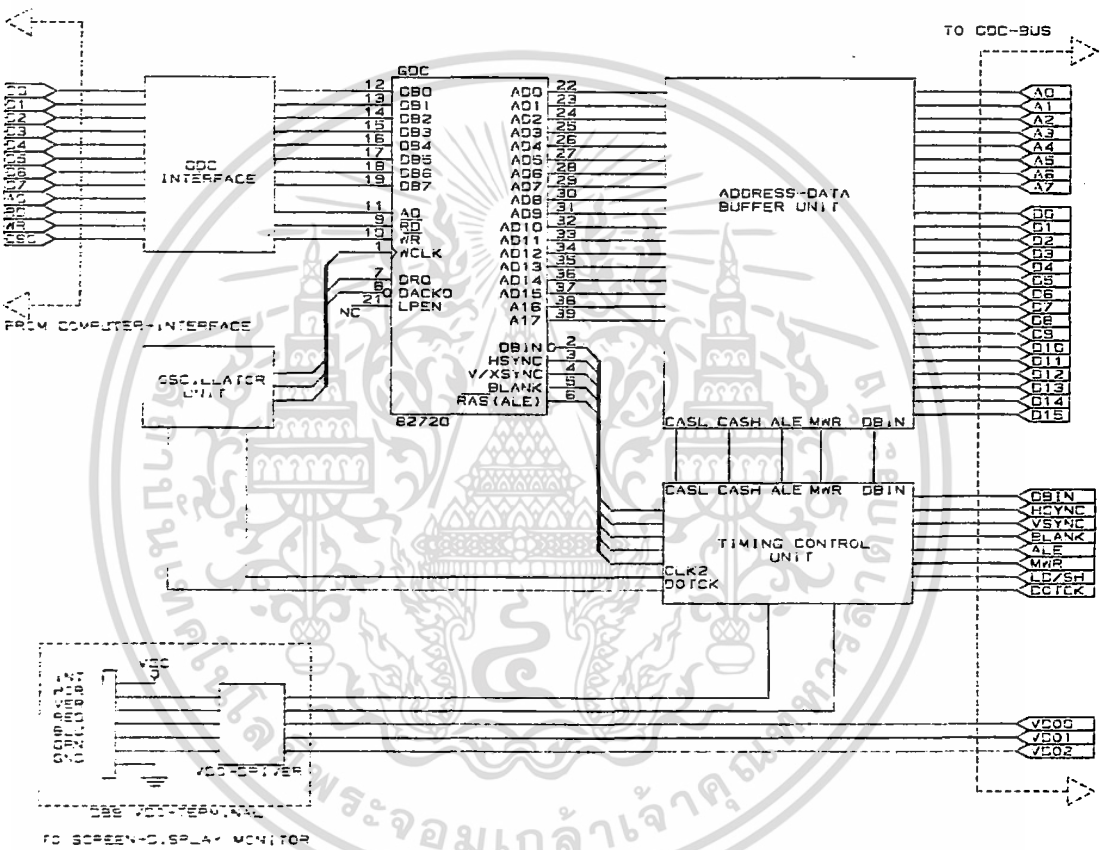
ส่วนนี้เป็นส่วนสำคัญที่จะควบคุมการแสดงผลทั้งหมด อุปกรณ์ที่สำคัญในส่วนนี้คือไอซีควบคุมการแสดงผลแบบกราฟฟิค (GRAPHICS DISPLAY CONTROLLER (GDC)) เบอร์ 82720 หรือ 7220 ซึ่งมีโครงสร้างดังในภาคผนวก 2 การทำงานของวงจรแบ่งออกได้เป็นส่วนใหญ่ได้ 2 ส่วนคือ

- GRAPHICS DISPLAY CONTROLLER CIRCUIT
- VIDEO MEMORY CIRCUIT

ในส่วนแรกเป็นวงจรที่ทำหน้าที่ควบคุมการแสดงผลทั้งหมดซึ่งควบคุมโดยมี GDC เป็นหน่วยประมวลผลกลาง นอกจากนี้ก็จะประกอบด้วยวงจรในส่วนต่างๆดังนี้

- GDC INTERFACE CIRCUIT
- ADDRESS/DATA BUFFER UNIT
- TIMING CONTROL UNIT
- OSCILLATOR UNIT

การเชื่อมต่อระหว่างวงจรต่างๆ เหล่านี้ ดังแสดงในรูปที่ 3.2



รูปที่ 3.2 ผังภาพของส่วนควบคุมการแสดงผลแบบกราฟฟิค

3.2.1 IC 82720 (GDC) เป็นหน่วยประมวลผลเกี่ยวกับการแสดงผลทั้งหมด ทำหน้าที่สร้างสัญญาณควบคุมส่วนต่างๆ เช่นจอแสดงผลความละเอียดสูง การอ่านและเขียนหน่วยความจำแสดงผล (VIDEO MEMORY) รวมทั้งควบคุมการรับส่งข้อมูลกับคอมพิวเตอร์ ด้วย ไอซี 82720 ที่มีจำหน่ายในท้องตลาดจะมีขนาด 40 ขาชนิด DIP (Dual In line Package) ดังมีรายละเอียดขาต่างๆทั้ง 40 ขา ดังรูปที่ 3.3

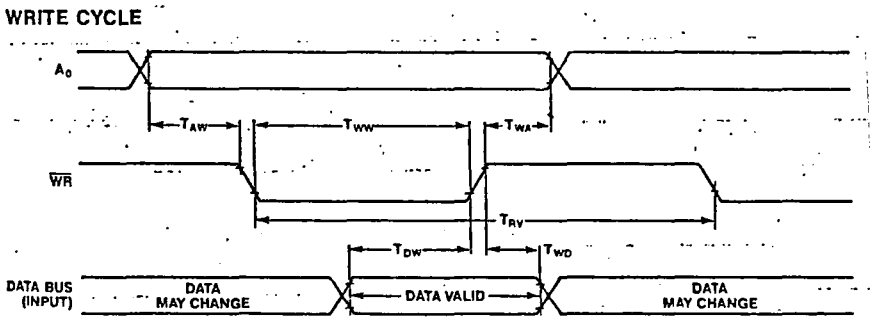
PIN	SYMBOL	DIRECTION	FUNCTION
1	2xWCLK	IN	CLOCK INPUT
2	$\overline{DBIN}$	OUT	DISPLAY MEMORY READ INPUT FLAG
3	HSYNC	OUT	HORIZONTAL VIDEO SYNC OUTPUT
4	V/EXT SYNC	IN/OUT	VERTICAL VIDEO SYNC OUTPUT OR EXTERNAL VSYNC INPUT
5	BLANK	OUT	CRT BLANKING OUTPUT
6	ALE($\overline{RAS}$)	OUT	ADDRESS LATCH ENABLE OUTPUT
7	DRQ	OUT	DMA REQUEST OUTPUT
8	$\overline{DACK}$	IN	DMA ACKNOWLEDGE INPUT
9	$\overline{RD}$	IN	READ STROBE INPUT FOR MICROPROCESSOR INTERFACE
10	$\overline{WR}$	IN	WRITE STROBE INPUT FOR MICROPROCESSOR INTERFACE
11	A0	IN	ADDRESS SELECT INPUT FOR MICROPROCESSOR INTERFACE
12-19	DB0-7	IN/OUT	BIDIREXTION DATA BUS TO HOST MICROPROCESSOR
20	GND	–	GROUND
21	LPEN	IN	LIGHT PEN DETECT INPUT
22-34	AD0-12	IN/OUT	ADDRESS AND DATA LINES TO DISPLAY MEMORY
35-37	AD13-15	IN/OUT	UTILIZATION VARIES WITH MODE OF OPERATION
38	A16	OUT	UTILIZATION VARIES WITH MODE OF OPERATION
39	A17	OUT	UTILIZATION VARIES WITH MODE OF OPERATION
40	VCC	–	+5V

รูปที่ 3.3 หน้าที่ของขาต่างๆของ GDC

และมีสัญญาณควบคุมการทำงานต่างๆดังนี้

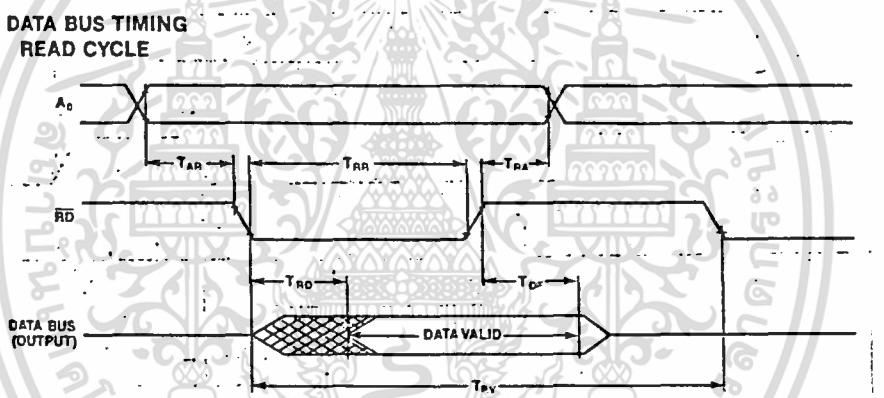
- สัญญาณติดต่อกับคอมพิวเตอร์ GDC มีการรับส่งข้อมูลกับคอมพิวเตอร์โดยข้อมูลที่ติดต่อกันนี้แบ่งออกเป็นคำสั่งและพารามิเตอร์ ข้อมูลเหล่านี้มีสัญญาณแอดเดรส A0 เป็นตัวกำหนด ถ้า A0 เป็น 0 ข้อมูลที่รับส่งก็จะเป็นคำสั่ง ถ้า A0 เป็น 1 ข้อมูลที่รับส่งก็คือพารามิเตอร์ ซึ่งสัญญาณก็จะแบ่งออกเป็นการรับและการส่งโดยมีการทำงานดังนี้

เมื่อคอมพิวเตอร์ต้องการส่งข้อมูลให้กับ GDC จะมีสัญญาณ WR มีสัญญาณควบคุมดังในรูปที่ 3.4



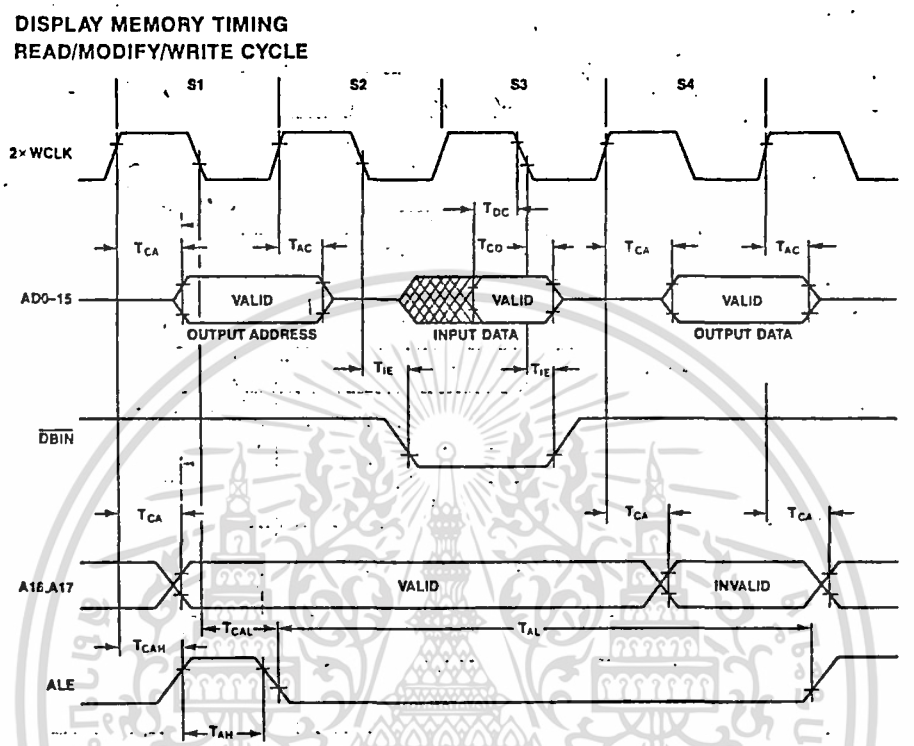
รูปที่ 3.4 สัญญาณ WR

และเมื่อคอมพิวเตอร์ต้องการรับข้อมูลจาก GDC ก็จะมีสัญญาณ RD เป็นสัญญาณควบคุมดังในรูปที่ 3.3



รูปที่ 3.5 สัญญาณ RD

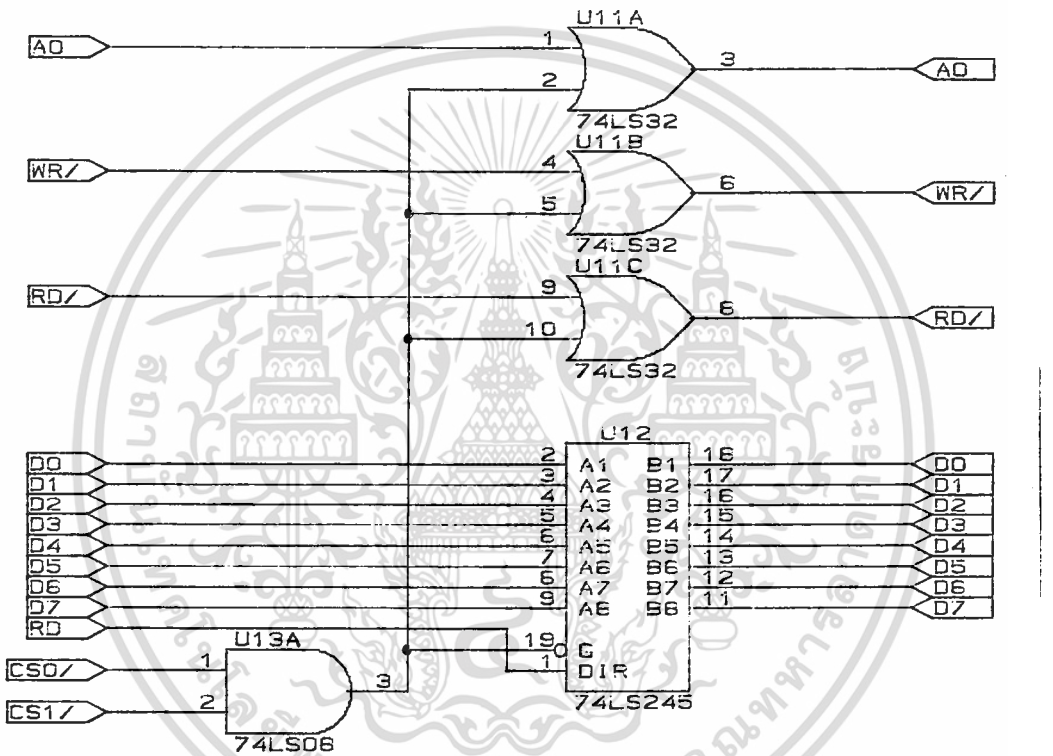
- สัญญาณอ่านข้อมูลจากหน่วยความจำแสดงผล มีสัญญาณตามรูปที่ 3.5 ซึ่งประกอบด้วยแอดเดรสเพื่อบอกตำแหน่งที่จะอ่านข้อมูล และในแอดเดรสบัส (ADDRESS BUS) นี้จะมีการส่งข้อมูลออกมาด้วยแต่ในช่วงเวลาที่แตกต่างกัน โดยมีสัญญาณ ALE มีสัญญาณที่ใช้ในการแยกข้อมูลและแอดเดรสออกจากกัน ในช่วงแรก GDC จะส่งแอดเดรสให้กับหน่วยความจำแสดงผลเพื่อกำหนดตำแหน่งและให้ ALE เป็น 1 เพื่อบอกว่าสัญญาณในขณะนั้นเป็นสัญญาณของแอดเดรส เมื่อสัญญาณนาฬิกาผ่านไป 1 ลูก ALE ก็จะกลับเป็น 0 และสัญญาณแอดเดรสก็จะถูกตัดออกจากบัสและให้ DBIN เป็น 0 ในช่วงเวลาต่อมาเพื่อให้หน่วยความจำแสดงผลส่งข้อมูลออกมาให้ GDC โดยข้อมูลดังกล่าวก็จะถูกส่งออกมาในแอดเดรสบัสเช่นเดียวกัน



รูปที่ 3.6 สัญญาณการติดต่อกับหน่วยความจำแสดงผล

– สัญญาณเขียนข้อมูลหน่วยความจำแสดงผล จากรูปที่ 3.6 จะเห็นได้ว่าหลังจากที่มีการอ่านข้อมูลจากหน่วยความจำแสดงผลแล้วก็จะมีอีกช่วงเวลาการทำงานอีกช่วงหนึ่งต่อจากช่วงแรกนั่นคือการเขียนข้อมูลลงในหน่วยความจำแสดงผล ดังนั้นการเขียนข้อมูลในหน่วยความจำแสดงผลจะต้องมีการอ่านข้อมูลก่อนเสมอ ในช่วงการเขียนข้อมูลนี้จะมีสัญญาณ WR เป็นตัวควบคุมอีกสัญญาณหนึ่ง

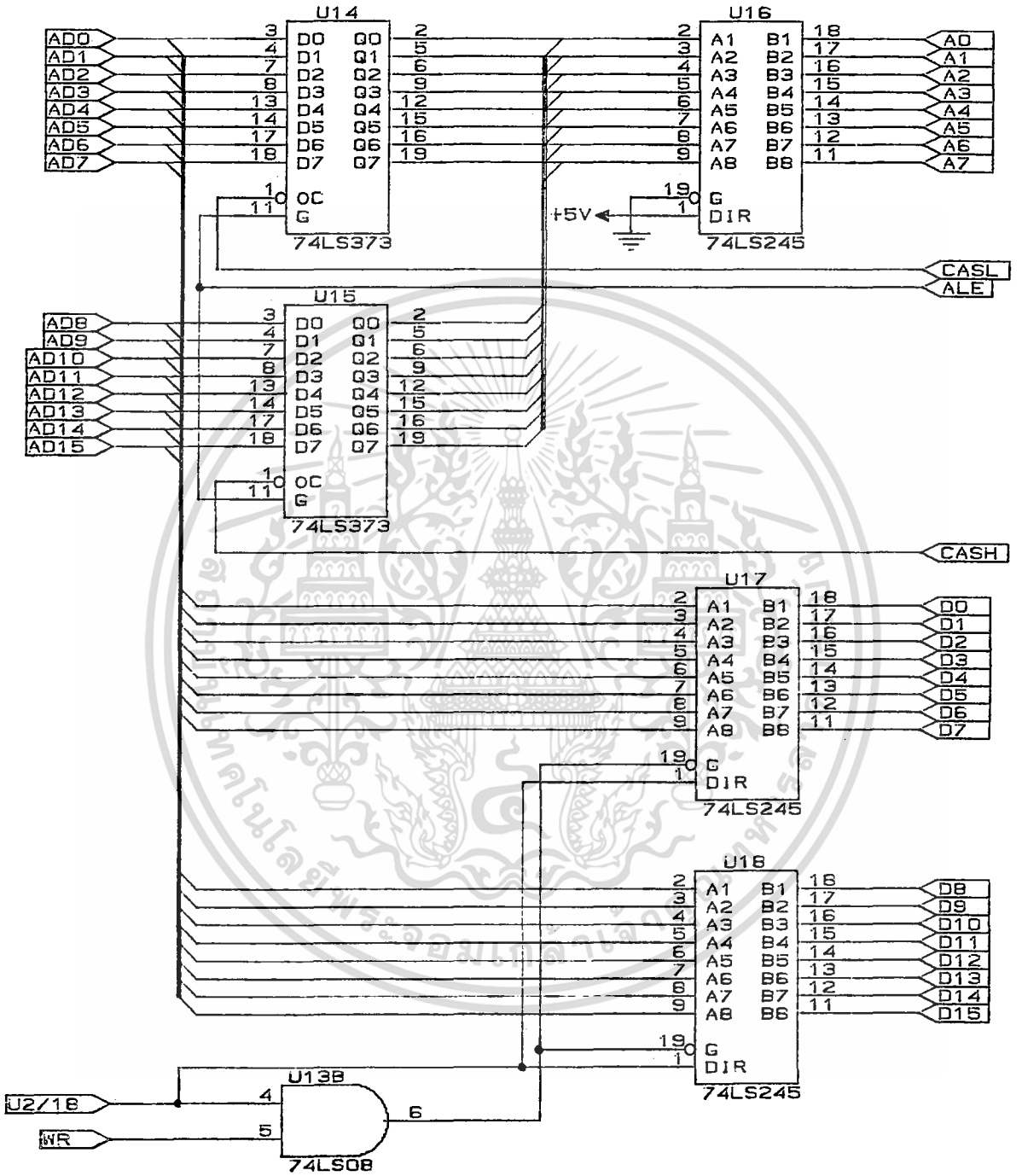
3.2.2 GDC INTERFACE CIRCUIT เป็นวงจรบัฟเฟอร์ (BUFFER) สัญญาณต่างๆ ที่ได้รับจากส่วนเชื่อมต่อกับคอมพิวเตอร์ และทำหน้าที่เป็นวงจรขับ (DRIVER) สัญญาณต่างๆที่ส่งไปยังส่วนเชื่อมต่อกับคอมพิวเตอร์ เนื่องจากการเชื่อมต่อระหว่างส่วนควบคุมการแสดงผลกับส่วนเชื่อมต่อกับคอมพิวเตอร์ต้องใช้สายเคเบิ้ล (CABLE) ที่มีความยาว และเป็นการสื่อสารข้อมูลแบบขนานที่ระดับ TTL ทำให้มีสัญญาณรบกวนและระดับแรงดันที่อาจจะต่ำกว่าปกติได้



รูปที่ 3.7 GDC INTERFACE CIRCUIT

จากรูปที่ 3.7 การรับส่งข้อมูลสามารถกระทำได้โดยการให้สัญญาณ CS0 หรือ CS1 สัญญาณใดสัญญาณหนึ่งเป็น 0 สัญญาณอื่นๆจึงจะสามารถผ่านไปยัง GDC หรือส่งจาก GDC ไปยังคอมพิวเตอร์ได้ สัญญาณ CS0 และ CS1 ก็จะเกิดจากการที่คอมพิวเตอร์ทำการติดต่อกับพอร์ท 0300H และ 0301H ตามลำดับ

3.2.3 ADDRESS/DATA BUFFER CIRCUIT ทำหน้าที่มัลติเพล็กซ์ (MULTIPLEX) สัญญาณแอดเดรสและข้อมูลให้กับหน่วยความจำแสดงผล ดังในรูปที่ 3.8



รูปที่ 3.8 ADDRESS/DATA BUFFER CIRCUIT

วงจรส่วนนี้ทำหน้าที่แยกสัญญาณแอดเดรสและข้อมูลออกจากกัน โดยสัญญาณแอดเดรสจะถูกแยกด้วยสัญญาณ ALE เมื่อสัญญาณนี้เป็น 1 U14 และ U15 ก็จะทำหน้าที่แลทช์ (LATCH) แอดเดรสเอาไว้เพื่อบอกตำแหน่งของหน่วยความจำ แต่การส่งสัญญาณแอดเดรสแบ่งออกเป็น 2 ช่วงคือแอดเดรสไบท์สูงและแอดเดรสไบท์ต่ำ การจัดการการส่งแอดเดรสในลักษณะนี้ก็กระทำด้วย U16 โดยมีสัญญาณ CASL และ CASH เป็นตัวกำหนดการส่งสัญญาณจาก U14 และ U15 ตามลำดับ

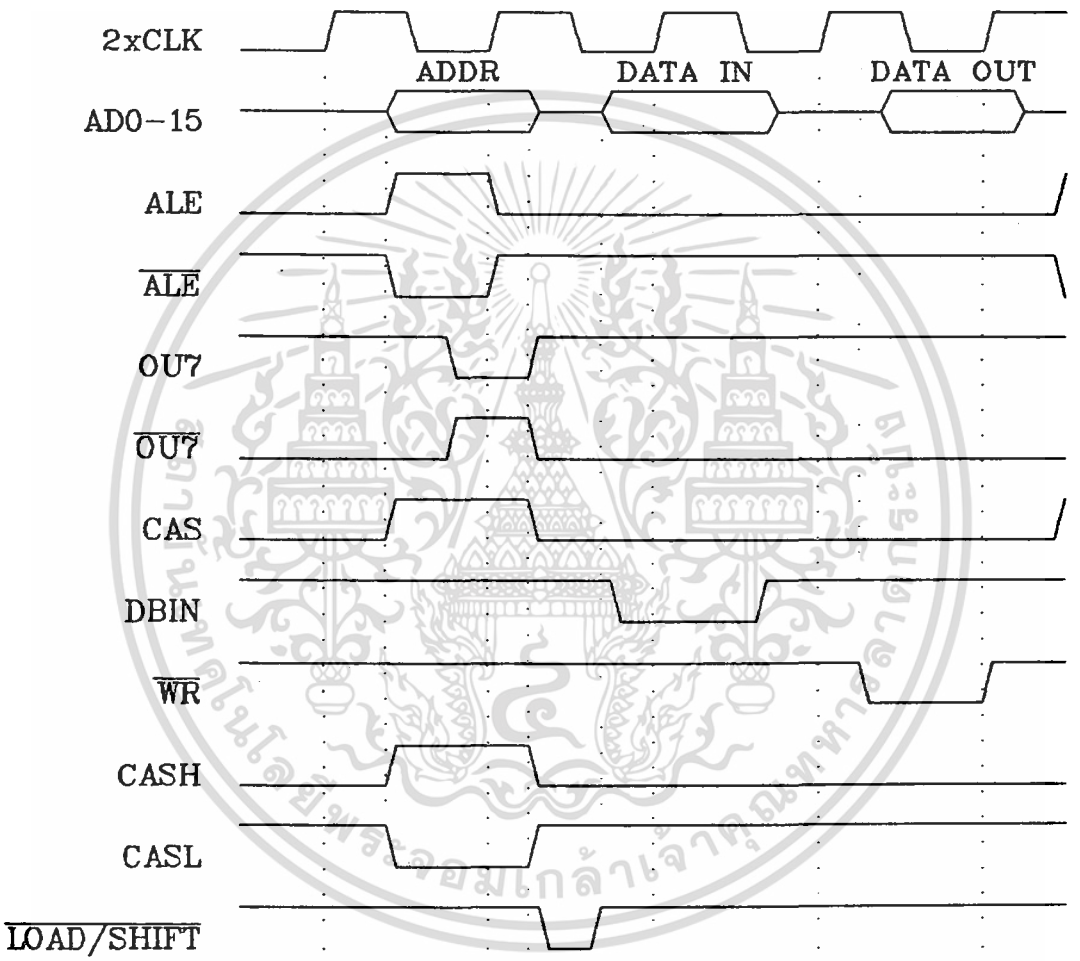
ส่วนสัญญาณข้อมูลนั้นจะถูกควบคุมการรับและส่งด้วย U17 และ U18 โดยทิศทางการไหลของข้อมูลขึ้นอยู่กับสัญญาณ WR เมื่อต้องการอ่านข้อมูล U17 และ U18 จะเปิดให้ข้อมูลจากหน่วยความจำแสดงผลผ่านไปยัง GDC และในกรณีที่ต้องการเขียนข้อมูลก็เช่นเดียวกัน แต่การไหลของข้อมูลก็เป็นไปในทางกลับกัน

3.2.4 TIMING CONTROL UNIT แบ่งการทำงานออกเป็น 2 ส่วนคือ

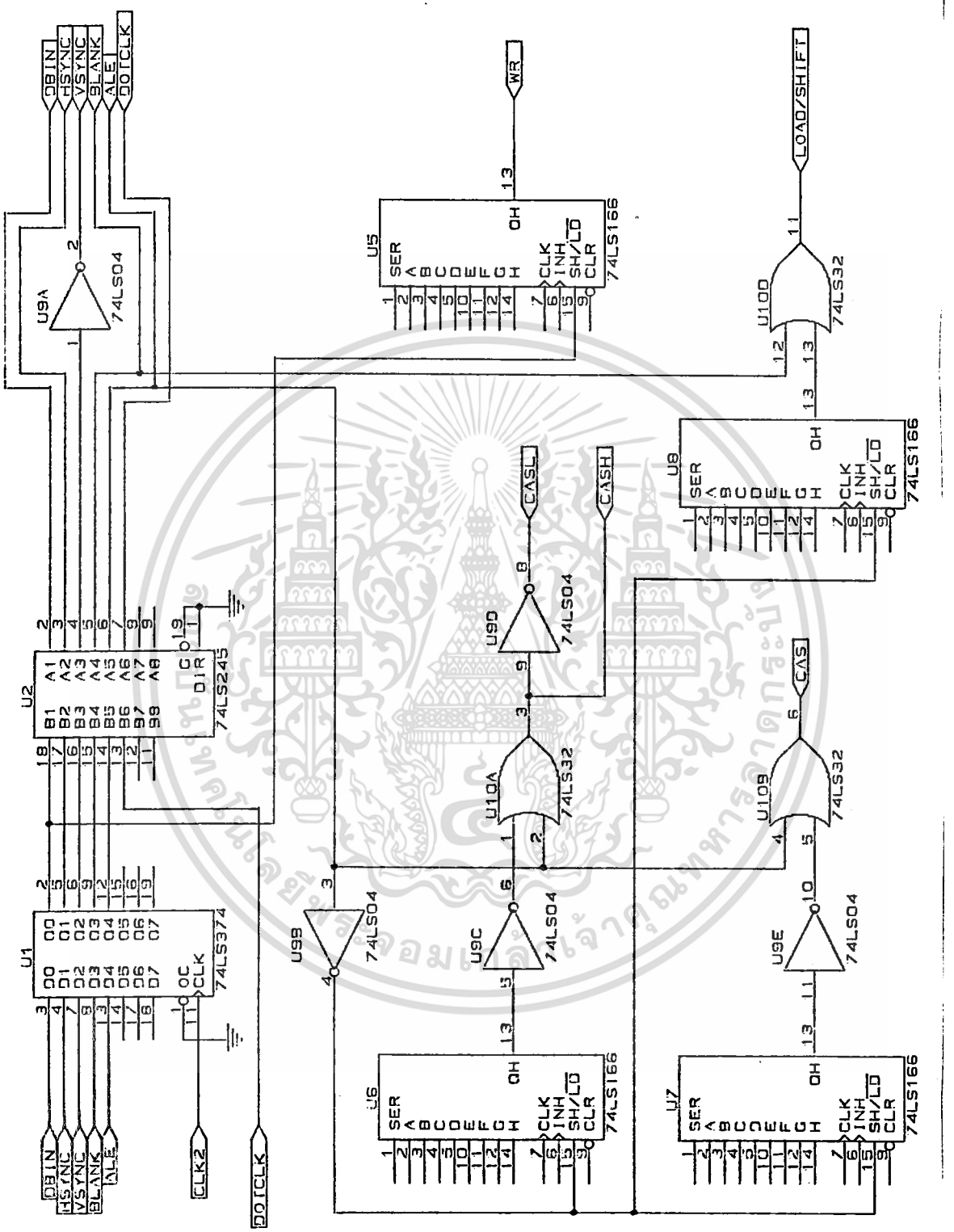
- วงจรขับ
- วงจรสร้างสัญญาณควบคุม

วงจรขับจะทำหน้าที่ขยายสัญญาณเพื่อส่งไปยังจุดต่างๆในบัส และส่งสัญญาณให้กับจอแสดงผลโดยสัญญาณจะผ่าน U1 เพื่อทำการจัดช่วงสัญญาณโดยการแลทช์ด้วยสัญญาณควบคุม CLK2 และส่งให้กับ U2 เพื่อนำไปใช้ในส่วนอื่นต่อไป

วงจรสร้างสัญญาณควบคุมทำหน้าที่สร้างสัญญาณให้กับวงจรในส่วนต่างๆโดยมีสัญญาณในช่วงต่างๆดังนี้



รูปที่ 3.9 TIMING ของสัญญาณควบคุม



รูปที่ 3.10 TIMING CONTROL UNIT

สัญญาณ CAS เป็นสัญญาณที่มีลักษณะเดียวกับ RAS ซึ่งใช้ในการควบคุมหน่วยความจำแสดงผลซึ่งเป็นหน่วยความจำชนิดไดนามิค แต่ CAS มีช่วงเวลาในขอบขาลงของสัญญาณช้ากว่า RAS ประมาณ 250 nSec การหน่วงเวลา (DELAY) สัญญาณในลักษณะนี้กระทำได้โดยใช้ชิฟท์รีจิสเตอร์ (SHIFT REGISTER) U7 ที่ตำแหน่งนี้จะมีสัญญาณดังในรูปที่ 3.9 ขา OU7 และนำสัญญาณนี้ผ่านอินเวอร์เตอร์ (INVERTER) และนำไปทำลอจิก OR กับสัญญาณ ALE จะได้สัญญาณ CAS ดังในรูปที่ 3.9

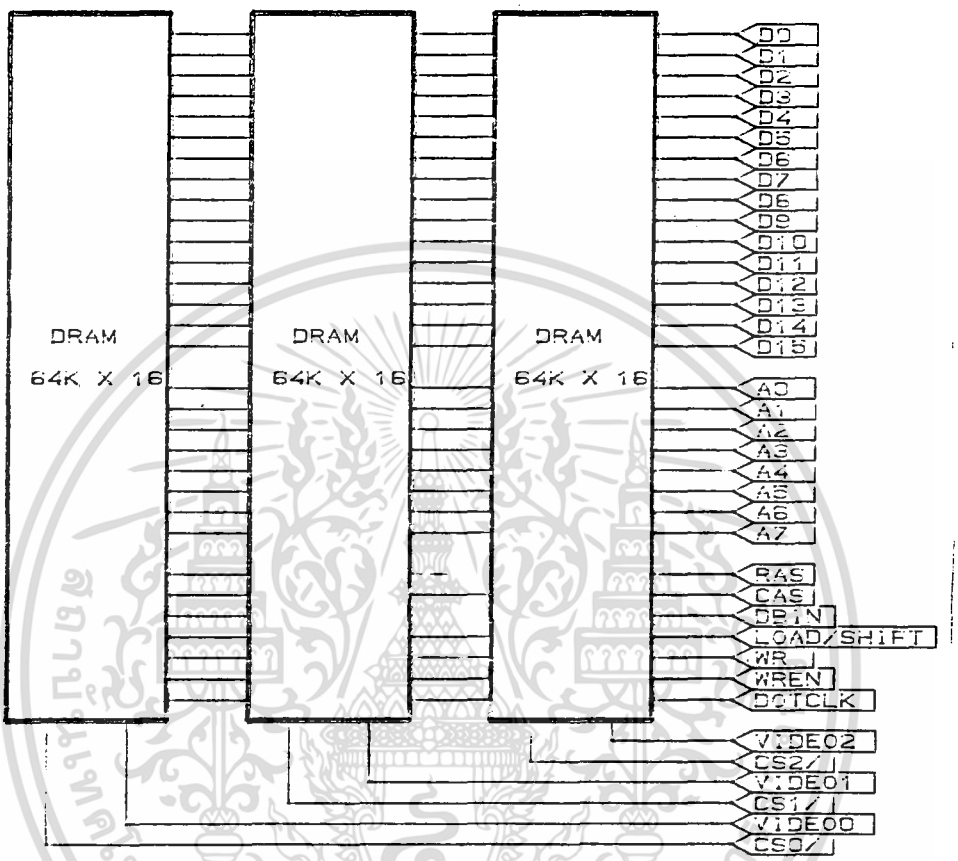
สัญญาณ CASH และ CASL ทำหน้าที่ควบคุมวงจรมัลติเพลกซ์ดังที่ได้อธิบายมาแล้ว มีการทำงานเช่นเดียวกับสัญญาณ CAS แต่สัญญาณที่ U7 นั้นมีช่วงเวลาสั้นกว่าสัญญาณที่ U6 ดังรูป (สัญญาณ OU5) และนำสัญญาณนี้ผ่านอินเวอร์เตอร์จะได้สัญญาณ CASL ดังรูป (สัญญาณ CASL)

สัญญาณ WR จากรูปที่ 3.9 และข้อมูลที่ต้องการเขียนลงในหน่วยความจำแสดงผลจะถูกส่งออกมาจาก GDC ในช่วงของสัญญาณนาฬิกาลูกสุดท้ายของการทำงานในช่วงการอ่านและเขียนหน่วยความจำแสดงผล ดังนั้นสัญญาณนี้จึงต้องถูกหน่วงเวลาให้มีช่วงเวลาการทำงานตรงกับช่วงเวลาของสัญญาณ OUTDATA โดยนำสัญญาณ DBIN ส่งให้กับ U5 และใช้สัญญาณ CLK4 เป็นสัญญาณควบคุมจะได้สัญญาณ WR ตามรูปที่ 3.9

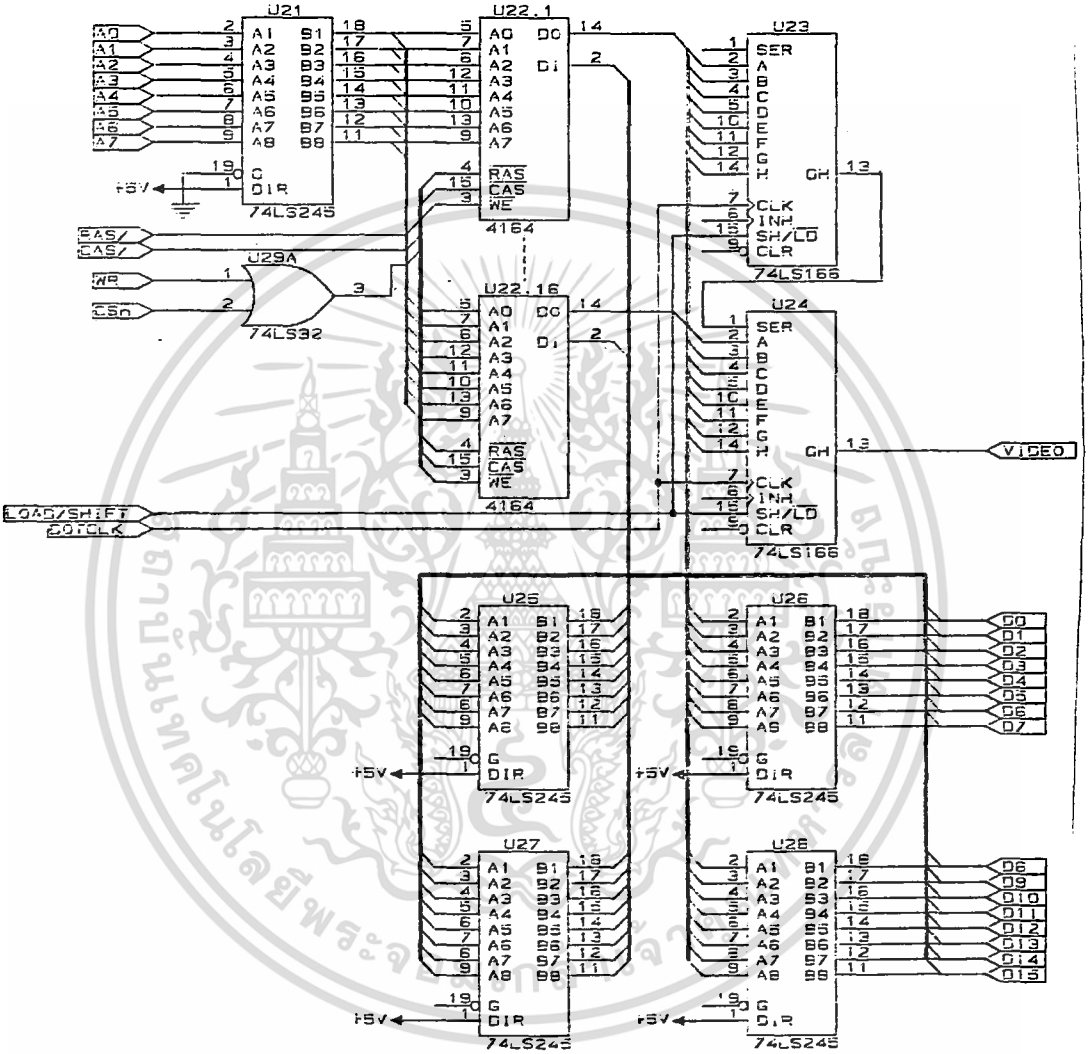
สัญญาณ LOAD/SHIFT ทำหน้าที่นำข้อมูลหน่วยความจำส่งให้กับวงจรชิฟท์รีจิสเตอร์เพื่อส่งข้อมูลภาพเป็นแบบอนุกรม โดยสัญญาณนี้มีช่วงเวลาการทำงานดังรูป (สัญญาณ LOAD/SHIFT) เป็นสัญญาณที่ถูกหน่วงเวลาหลังจากการแอดเดรสถูกส่งออกมาในบัสเพียงเล็กน้อย การหน่วงเวลาในช่วงนี้กระทำได้โดยใช้สัญญาณจาก U7 ส่งให้กับ U8 และควบคุมการทำงานด้วยสัญญาณ DOTCLK ที่ให้กับขา CLK ของ U8 และนำมาทำลอจิก AND กับสัญญาณ BLANK เพื่อให้ไม่มีการส่งสัญญาณภาพในระหว่างที่ยังไม่เกิดภาพ

ก็จะสามารถผ่านเข้าออกระหว่างหน่วยความจำแสดงผลกับ GDC ได้

ส่วนการรีเฟรช (REFRESH) หน่วยความจำชนิดไดนามิคนั้นกระทำโดย GDC ซึ่งจะส่งสัญญาณ RAS และแอดเดรสเพื่อทำการรีเฟรชอยู่ตลอดเวลา



รูปที่ 3.12 ผังภาพของหน่วยความจำแสดงผล



รูปที่ 3.13 วงจรหน่วยความจำแสดงผล

บทที่ 4
ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม

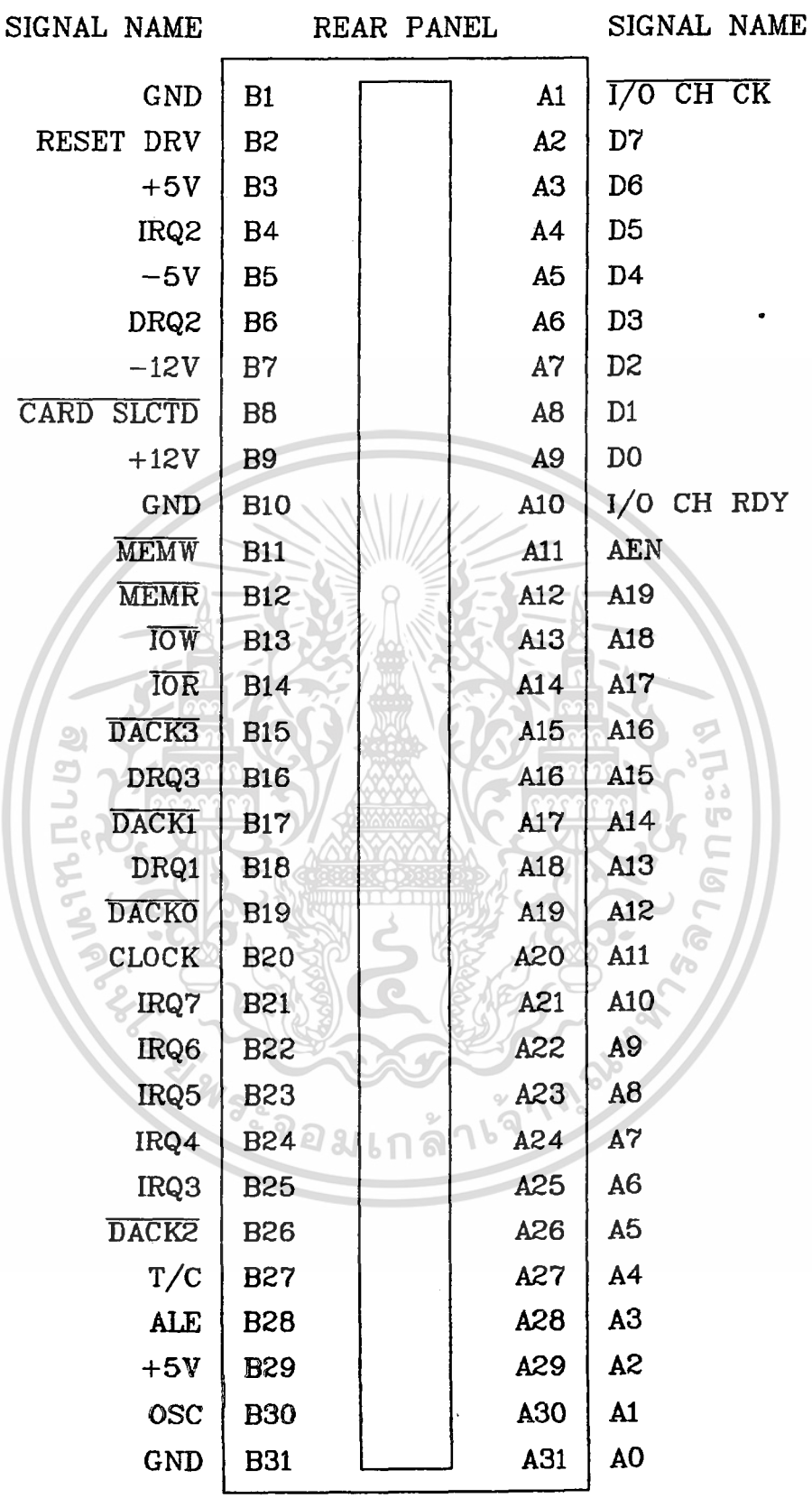
การรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมนั้นเป็นส่วนสำคัญส่วนหนึ่งของระบบ ซึ่งหมายความว่าส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรมก็มีความสำคัญอย่างมากด้วย เนื่องจากข้อมูลสภาวะของกระบวนการที่คอมพิวเตอร์ได้รับจะต้องถูกต้อง เชื่อถือได้ มีความแม่นยำสูง และสามารถเชื่อมต่อกับกระบวนการทางอุตสาหกรรมทั่วไปได้ง่าย นอกจากนั้นจะต้องมีการปรับปรุง (UPDATE) ข้อมูลให้ทันสมัยอยู่เสมอ ดังนั้นการออกแบบในส่วนนี้จะต้องทำการศึกษาการเชื่อมต่อกับกระบวนการทางอุตสาหกรรม ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรมแบ่งออกเป็น 2 ชนิด คือแบบดิจิตอลและแบบอนาลอก

นอกจากการรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมแล้ว ในวิทยานิพนธ์ฉบับนี้ได้ออกแบบส่วนเอาท์พุทของคอมพิวเตอร์ที่ใช้ในการควบคุมระบบเพื่อการสร้างสัญญาณเตือนในรูปแบบต่างๆซึ่งเป็นแบบดิจิตอล ดังนั้นในบทนี้จะแยกออกเป็นส่วนใหญ่ๆได้ดังนี้คือ

1. วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบดิจิตอล
2. วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบอนาลอก
3. วงจรเอาท์พุทสำหรับสัญญาณเตือนในแบบดิจิตอล

วงจรต่างๆที่ออกแบบนี้ได้ออกแบบไว้สำหรับไมโครคอมพิวเตอร์ IBM PC/XT/AT ซึ่งแพร่หลายอยู่ในท้องตลาดขณะนี้ และได้ออกแบบไว้เป็นลักษณะของคาร์ด (CARD) เพื่อให้เสียบลงในสล็อท (SLOT) ของไมโครคอมพิวเตอร์ดังกล่าวได้ทันที นอกจากนั้นยังได้ออกแบบให้แอดเดรส (ADDRESS) ของคาร์ดเหล่านั้นผู้ใช้สามารถกำหนดได้ด้วยการปรับตั้งดิพสวิทช์ (DIP SWITCH) ที่ติดอยู่บนคาร์ด ดังนั้นจึงสามารถขยายจำนวนช่องรับค่าสภาวะเพิ่มขึ้นอีกได้โดยใช้วงจรแบบเดียวกันแต่ปรับตั้งแอดเดรสให้แตกต่างกัน

หมายเหตุ ชื่อขาสัญญาณของวงจรในรูปต่างๆที่จะกล่าวต่อไปขาสัญญาณใดที่ขึ้นต้นด้วย AXX และ BXX จะเป็นชื่อขาสัญญาณในสล็อทของไมโครคอมพิวเตอร์ IBM PC/XT เช่น A12,B15 ก็จะเป็นตำแหน่งที่ 12 ด้าน A และตำแหน่งที่ 15 ด้าน B ตามลำดับ

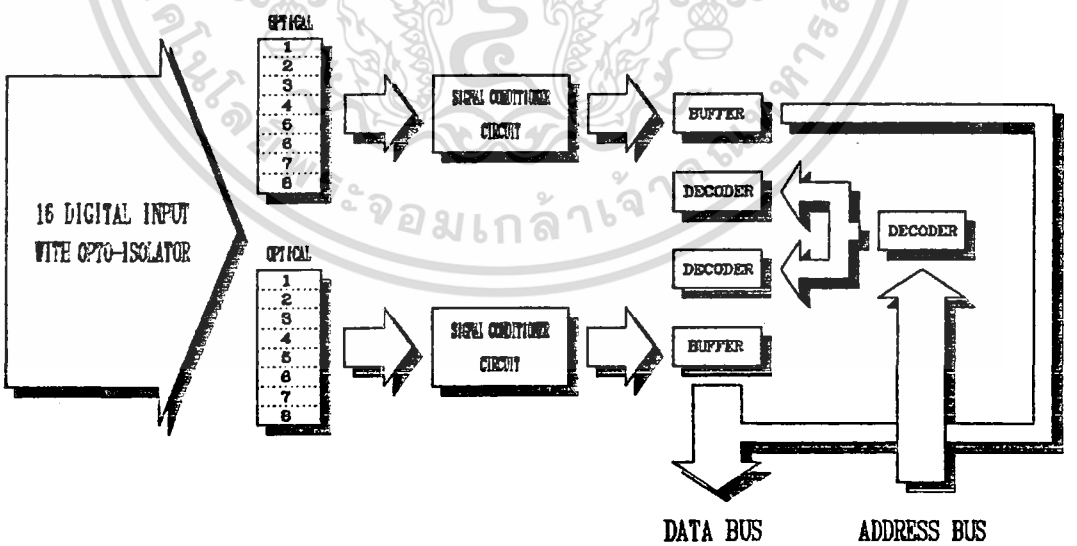


รูปที่ 4.1 แสดงการจัดตำแหน่งขาสล็อทของ IBM PC/XT

4.1 วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบดิจิตอล

วงจรรับค่าสภาวะในแบบดิจิตอลที่ได้ออกแบบไว้มีผังภาพดังรูปที่ 4.2 และมีรายละเอียดดังนี้

- ส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรมและส่วนเชื่อมต่อกับคอมพิวเตอร์ถูกแยกออกจากกันอย่างเด็ดขาดทางไฟฟ้า แต่จะเชื่อมต่อกันด้วยแสง (OPTICAL ISOLATED) เพื่อความปลอดภัยของผู้ใช้และคอมพิวเตอร์
- รับค่าสภาวะจากกระบวนการได้ 16 จุดต่อคาร์ด โดยแบ่งออกเป็น 2 ส่วนคือจุดที่ 0 - 7 และ 8 - 15 เพื่อความสะดวกในการควบคุม
- มีไดโอดเปล่งแสง (LED) สำหรับแสดงผลสภาวะที่รับมาจากกระบวนการ
- อุปกรณ์เชื่อมต่อด้วยแสง (OPTO-ISOLATOR) สามารถทนแรงดันได้สูงสุด 2500V.
- แรงดันอินพุท 0 - 25V. โดย 0 - 3V.มีค่าสภาวะเป็น "0" และ 3 - 25V. มีค่าสภาวะเป็น "1"
- กระแสอินพุทสูงสุด 30mA. แบบต่อเนื่อง
- ตำแหน่งแอดเดรสเลือกได้ตั้งแต่ 0280H - 72F7H
- การตอบสนองความถี่สูงสุด 10KHz
- ความต้องการกำลังงาน +5V_{DC} 150mA.
- คอนเนคเตอร์ (CONNECTOR) แบบ 37 ขา D-TYPE

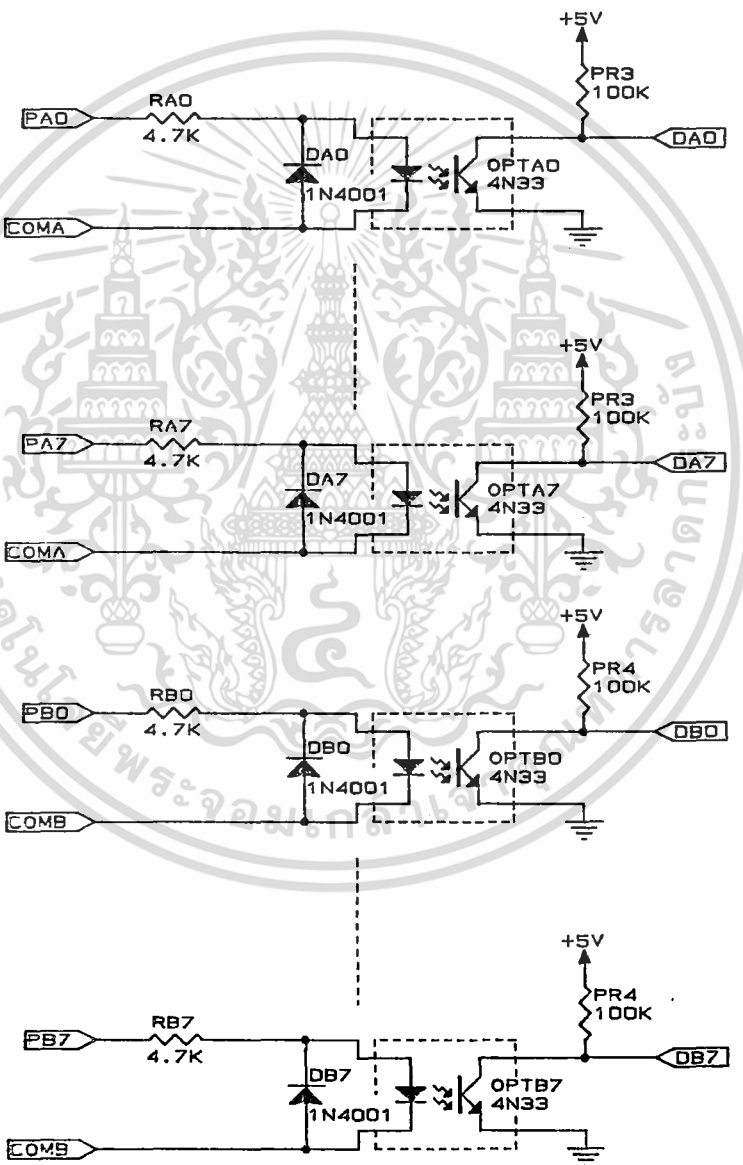


รูปที่ 4.2 ผังภาพของ
วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบดิจิตอล

หลักการทำงานของวงจร

1. วงจรเชื่อมต่อกระบวนการทางอุตสาหกรรม

วงจรในส่วนนี้ทำหน้าที่รับค่าสภาวะจากกระบวนการทางอุตสาหกรรมซึ่งสามารถรับค่าแรงดันสูงสุดได้ถึง 25V. โดยกำหนดให้ 0-3V. เป็นสภาวะ "ปิด" 3-25V. เป็นสภาวะ "เปิด" และวงจรนี้จะทำหน้าที่ส่งผ่านค่าสภาวะไปยังวงจรอื่นๆ โดยลดค่าแรงดันลงจาก 0-25V. เป็น 0-5V. เพื่อให้เหมาะสมกับอุปกรณ์ดิจิตอลอิเลคทรอนิคส์ที่เป็น TTL วงจรในส่วนนี้ดังแสดงในรูปที่ 4.3



รูปที่ 4.3 วงจรเชื่อมต่อกระบวนการทางอุตสาหกรรม

จากรูปที่ 4.3 สัญญาณ PA0 - PA7 และ PB0 - PB7 เป็นสัญญาณอินพุทที่รับมาจากกระบวนการทางอุตสาหกรรม ส่วนสัญญาณ DA0 - DA7 และ DB0 - DB7 เป็นสัญญาณทางดิจิตอลที่มีระดับของแรงดันตามมาตรฐาน TTL

กระแสอินพุทจากกระบวนการจะถูกจำกัดด้วยค่าความต้านทานของ R (RA0 - RA7 และ RB0 - RB7) ซึ่งกระแสนี้ยอมให้ไหลได้สูงสุด 30 mA. และต่ำสุด 3 mA. ในการออกแบบจะกำหนดให้กระแสนี้มีค่าประมาณ 5 mA. จาก

$$R = \frac{V_{in} - V_f}{I_f}$$

$$P = I_f^2 \times R$$

เมื่อ V_{in} = ระดับแรงดันอินพุทสูงสุด

V_f = แรงดันตกคร่อมไดโอดเปล่งแสง

R = ค่าความต้านทานที่ใช้ในการจำกัดกระแส

P = ค่ากำลังงานสูญเสียที่ R

ดังนั้น

$$R = \frac{25 - 0.7}{5 \times 10^{-3}}$$

$= 4860$ ohm. ใช้ค่า 4.7 Kohm.

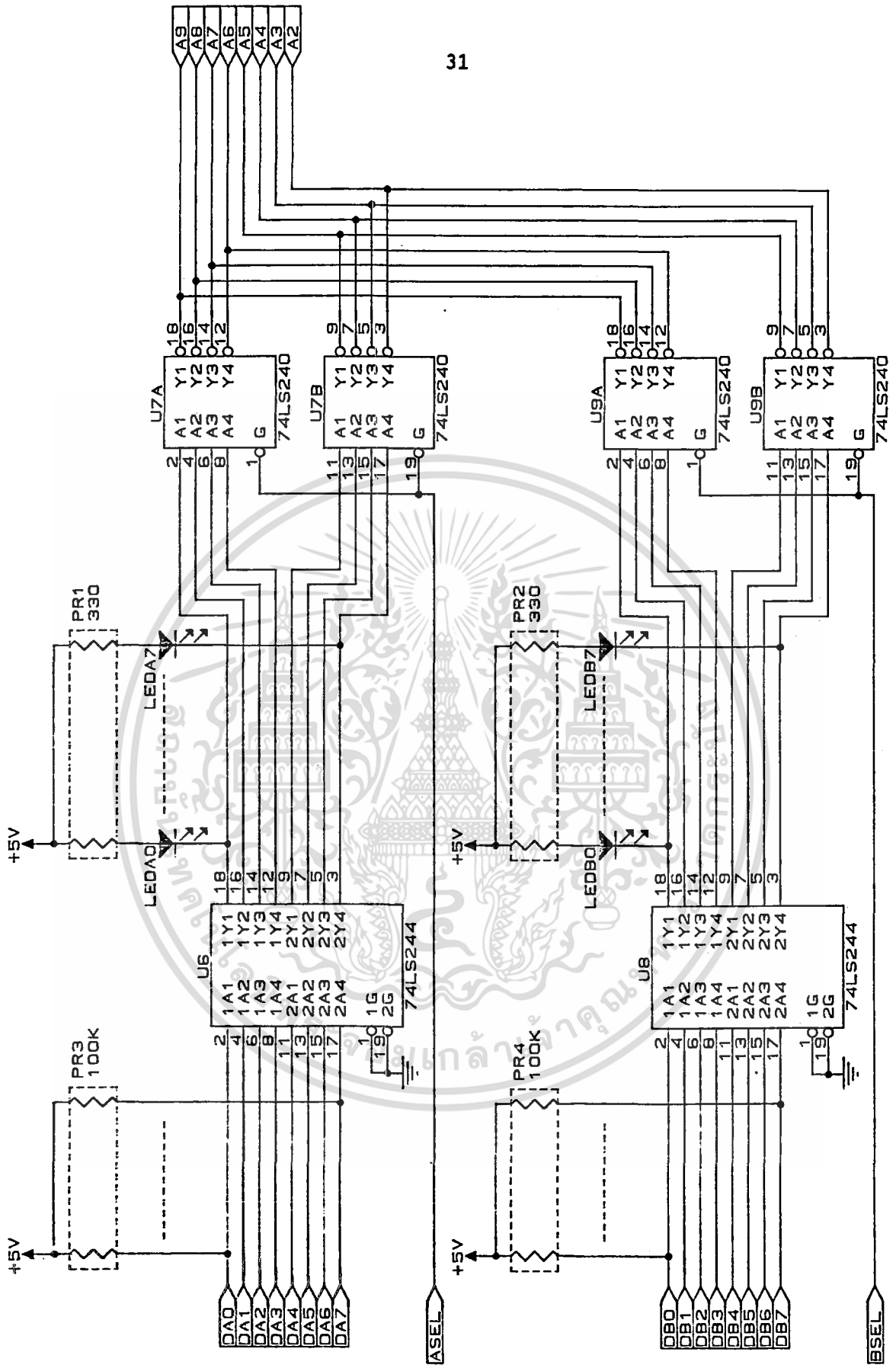
$P = (5 \times 10^{-3})^2 \times 4.7 \times 10^3$

$= 0.1175$ W. ใช้ค่า 0.5 W.

ส่วนไดโอด (DIODE) DA0 - DA7 และ DB0 - DB7 นั้นมีไว้เพื่อป้องกันแรงดันกลับขั้วซึ่งอาจจะทำให้วงจรเสียหายได้

2. วงจรบัฟเฟอร์ (BUFFER) และวงจรขับไดโอดเปล่งแสง

วงจรบัฟเฟอร์ทำหน้าที่เป็นตัวกั้นระหว่างข้อมูลค่าสภาวะที่รับมาจากกระบวนการซึ่งเปลี่ยนเป็นระดับ TTL แล้ว กับบัสข้อมูล (DATA BUS) ของคอมพิวเตอร์ ส่วนวงจรขับไดโอดเปล่งแสงนั้นใช้สำหรับตรวจสอบค่าสภาวะของคาร์ด



รูปที่ 4.4 วงจรบัฟเฟอร์และวงจรขับไดโอดเปล่งแสง

สัญญาณ DA0 - DA7 และ DB0 - DB7 เป็นสัญญาณที่ได้รับมาจากวงจรเชื่อมต่อกระบวนการ จากนั้น U6 และ U8 ทำหน้าที่ขับไดโอดเปล่งแสง และสัญญาณเดียวกันนี้จะถูกส่งไปยัง U7 และ U9 ซึ่งทำหน้าที่บัฟเฟอร์ที่ถูกควบคุมการเปิดเกท (GATE) ด้วยสัญญาณ ASEL และ BSEL ซึ่งสัญญาณทั้งสองนี้จะถูกส่งมาจากวงจรถอดรหัสแอดเดรส (ADDRESS DECODER) ส่วนขา A2 - A9 เป็นชื่อตามบัส (BUS) ของ IBM PC/XT ซึ่งเป็นบัสข้อมูล (DATA BUS)

3.วงจรถอดรหัสแอดเดรส (ADDRESS DECODER)

วงจรนี้เป็นวงจรที่ใช้กำหนดแอดเดรสของคาร์ดนั้นๆ และเนื่องจากอาจจะมีคาร์ดชนิดเดียวกันนี้มากกว่า 1 คาร์ดในคอมพิวเตอร์เครื่องเดียวกัน ดังนั้นวงจรนี้จะต้องปรับตั้งได้ เพื่อให้แอดเดรสแตกต่างกัน ซึ่งในการออกแบบก็ได้ออกแบบให้ปรับตั้งได้ตั้งแต่ 0280H - 72F7H โดยการปรับตั้งที่ดิพสวิทช์ DIP1 - DIP4 ดังนี้

- DIP1 สำหรับตั้งหลักที่ 1 ตั้งได้ตั้งแต่ 0 - 7 (0XXX - 7XXX)

DIP1							
1	2	3	4	5	6	7	8
0XXX	1XXX	2XXX	3XXX	4XXX	5XXX	6XXX	7XXX

- DIP2 สำหรับตั้งหลักที่ 3 ตั้งได้ตั้งแต่ 7 - F ส่วนหลักที่ 2 ถูกกำหนดให้เป็น "2" เสมอ (X28X - X2FX)

DIP2							
1	2	3	4	5	6	7	8
X28X	X29X	X2AX	X2BX	X2CX	X2DX	X2EX	X2FX

- DIP3 และ DIP4 สำหรับตั้งหลักที่ 4 โดย DIP3 สำหรับจุดอินพุทที่ 0 - 7 และ DIP4 สำหรับ 8 - 15 (X2X0 - X2X7)

DIP3 & DIP4							
1	2	3	4	5	6	7	8
X2X0	X2X1	X2X2	X2X3	X2X4	X2X5	X2X6	X2X7

สมมติว่าต้องการตั้งให้คาร์ดอยู่ที่แอดเดรส 0280H และ 0283H โดยจุดที่ 0 - 7 อยู่ที่ 0280H และ 8 - 15 อยู่ที่ 0283H ก็จะต้องปรับตั้งดิพสวิทช์ดังนี้

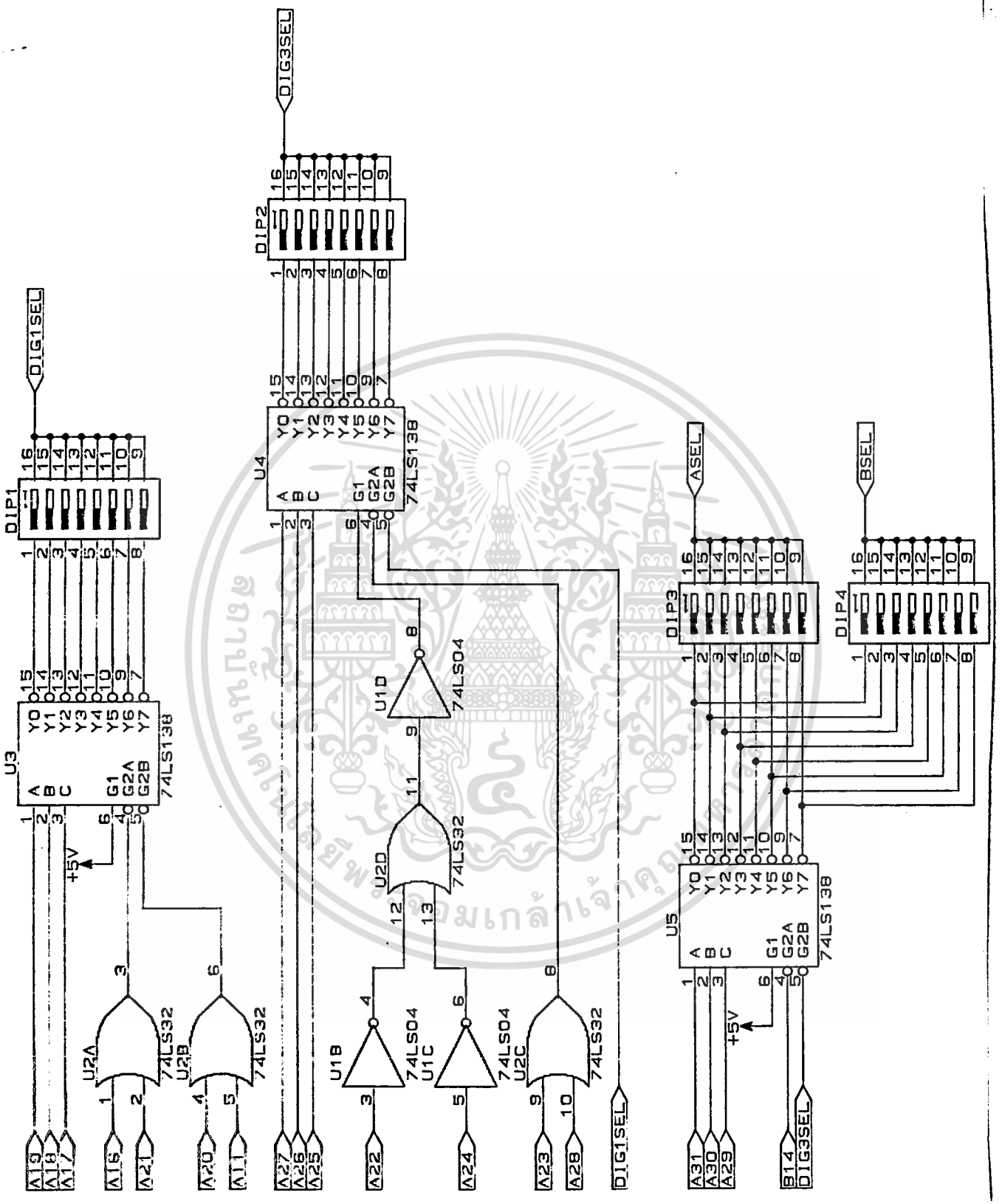
DIP1 ให้สวิทช์ตัวที่ 1 อยู่ในสภาวะ "เปิด"

DIP2 ให้สวิทช์ตัวที่ 1 อยู่ในสภาวะ "เปิด"

DIP3 ให้สวิทช์ตัวที่ 4 อยู่ในสภาวะ "เปิด"

DIP4 ให้สวิทช์ตัวที่ 1 อยู่ในสภาวะ "เปิด"

สวิทช์ตัวอื่นนอกจากที่กำหนดไว้ให้อยู่ในสภาวะ "ปิด"



รูปที่ 4.5 วงจรถอดรหัสแอดเดรส

วงจรในส่วนบนของรูปที่ 4.5 จะเป็นการถอดรหัสในหลักที่ 1 ซึ่งเป็นการกำหนดแอดเดรสสำหรับ 0XXX - 7XXX สัญญาณอินพุทคือ A19,A18,17,A16,A21,A20,A11 ซึ่งเป็นแอดเดรสบัส (ADDRESS BUS) ส่วนเอาท์พุทคือสัญญาณ DIG1SEL

วงจรในส่วนกลางจะเป็นการถอดรหัสในหลักที่ 3 (หลักที่ 2 ถูกกำหนดให้เป็น "2" เสมอ) นั่นคือแอดเดรสตำแหน่ง X28X - X2FX สัญญาณอินพุทคือ A27,A26,A25,A22,A24,A23,A28 และสัญญาณ DIG1SEL ส่วนสัญญาณเอาท์พุทคือสัญญาณ DIG3SEL

วงจรในส่วนล่างสุดก็เป็นการถอดรหัสในหลักที่ 4 คือ X2X0 - X2X7 และจะมีดิพสวิทช์ 2 ชุดคือ DIP3 และ DIP4 สำหรับอินพุทที่ 0 - 7 และ 8 - 16 ตามลำดับ การตั้งดิพสวิทช์ทั้งสองตัวนี้จะต้องไม่ตั้งไว้ที่ตำแหน่งเดียวกันซึ่งจะทำให้การทำงานผิดพลาดได้

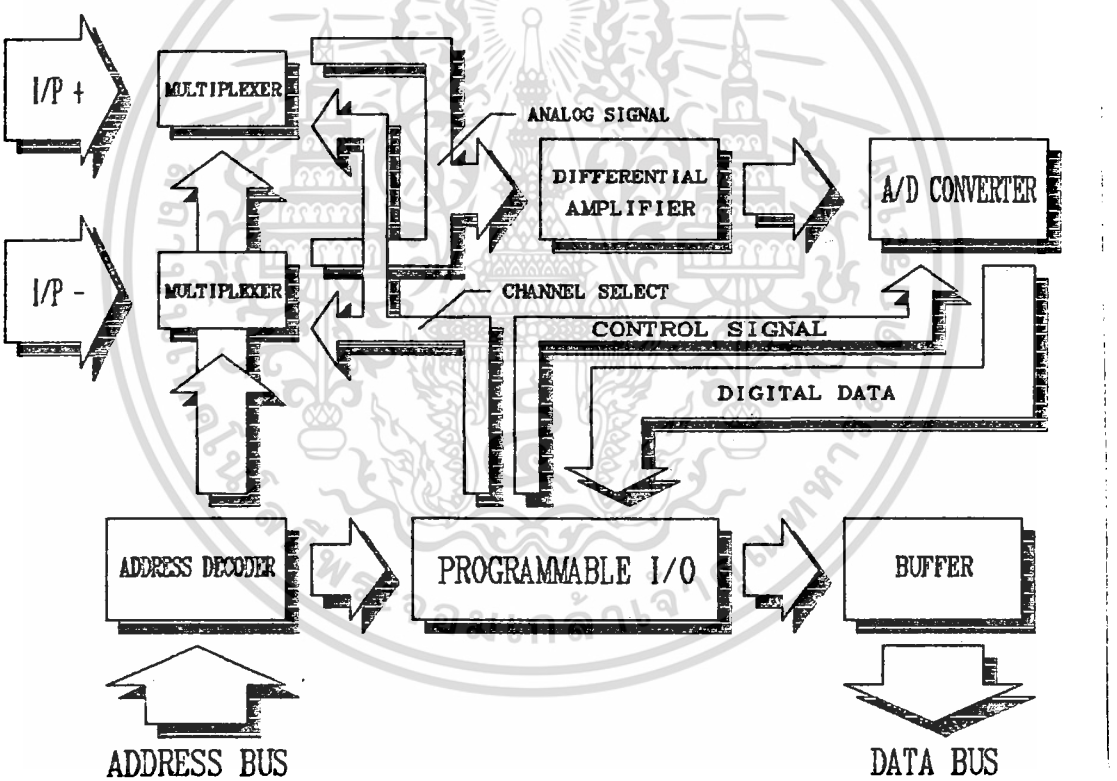
คอนเนคเตอร์สำหรับคาร์ดนี้เป็นแบบ 37 ขา D-TYPE ซึ่งมีการต่อดังนี้

PIN No.	DESCRIPTION	PIN No.	DESCRIPTION
1	PA0 [จุดที่0]	20	PB0 [จุดที่8]
2	ไม่ได้ต่อ	21	ไม่ได้ต่อ
3	PA2 [จุดที่1]	22	PB1 [จุดที่9]
4	ไม่ได้ต่อ	23	ไม่ได้ต่อ
5	PA4 [จุดที่2]	24	PB2 [จุดที่10]
6	ไม่ได้ต่อ	25	ไม่ได้ต่อ
7	PA6 [จุดที่3]	26	PB3 [จุดที่11]
8	ไม่ได้ต่อ	27	ไม่ได้ต่อ
9	PA7 [จุดที่4]	28	PB4 [จุดที่12]
10	ไม่ได้ต่อ	29	ไม่ได้ต่อ
11	PA10[จุดที่5]	30	PB5 [จุดที่13]
12	ไม่ได้ต่อ	31	ไม่ได้ต่อ
13	PA12[จุดที่6]	32	PB6 [จุดที่14]
14	ไม่ได้ต่อ	33	ไม่ได้ต่อ
15	PA14[จุดที่7]	34	PB7 [จุดที่15]
16	ไม่ได้ต่อ	35	ไม่ได้ต่อ
17	COMA	36	COMB
18	ไม่ได้ต่อ	37	GND
19	+5V.		

4.2 วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบอนาลอก

วงจรรับค่าสภาวะในแบบอนาลอกที่ได้ออกแบบไว้มีผังภาพดังในรูปที่ 4.6 และมีรายละเอียดดังนี้

- ความละเอียดในการเปลี่ยนสัญญาณอนาลอกเป็นดิจิตอลขนาด 8 บิท
- สัญญาณอนาลอกอินพุท 4 - 20 mA.
- สามารถรับสัญญาณได้ 8 ช่องสัญญาณ โดยแยกออกจากกันอิสระ เพื่อป้องกันสัญญาณรบกวนจากการใช้กราวนด์ร่วม
- ตำแหน่งแอดเดรสเลือกได้ตั้งแต่ 300H - 3FFH
- ความต้องการกำลังงาน +/- 5V 200mA. และ +/- 12V 200mA.
- คอนเนคเตอร์ 25 ขา D-TYPE

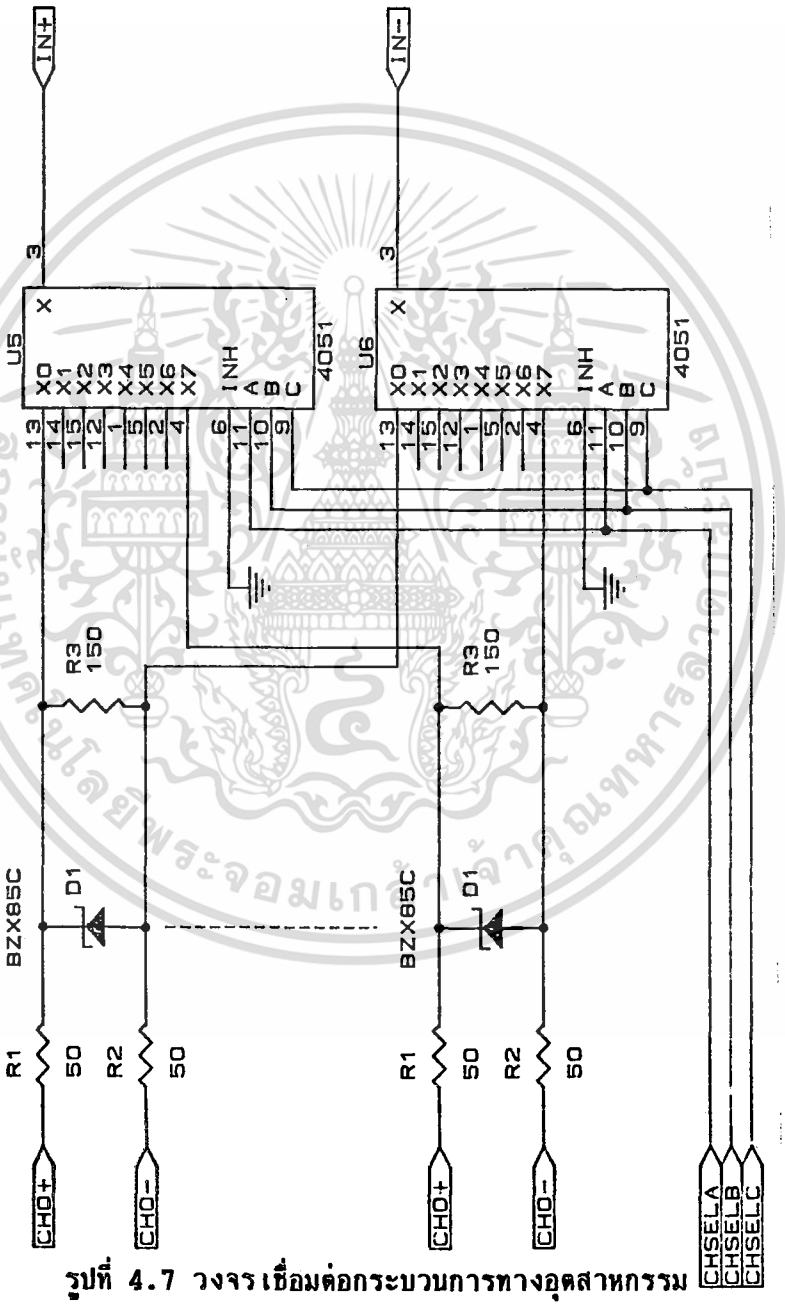


รูปที่ 4.6 ผังภาพของ
วงจรรับค่าสภาวะจากกระบวนการทางอุตสาหกรรมในแบบอนาลอก

หลักการทำงานของวงจร

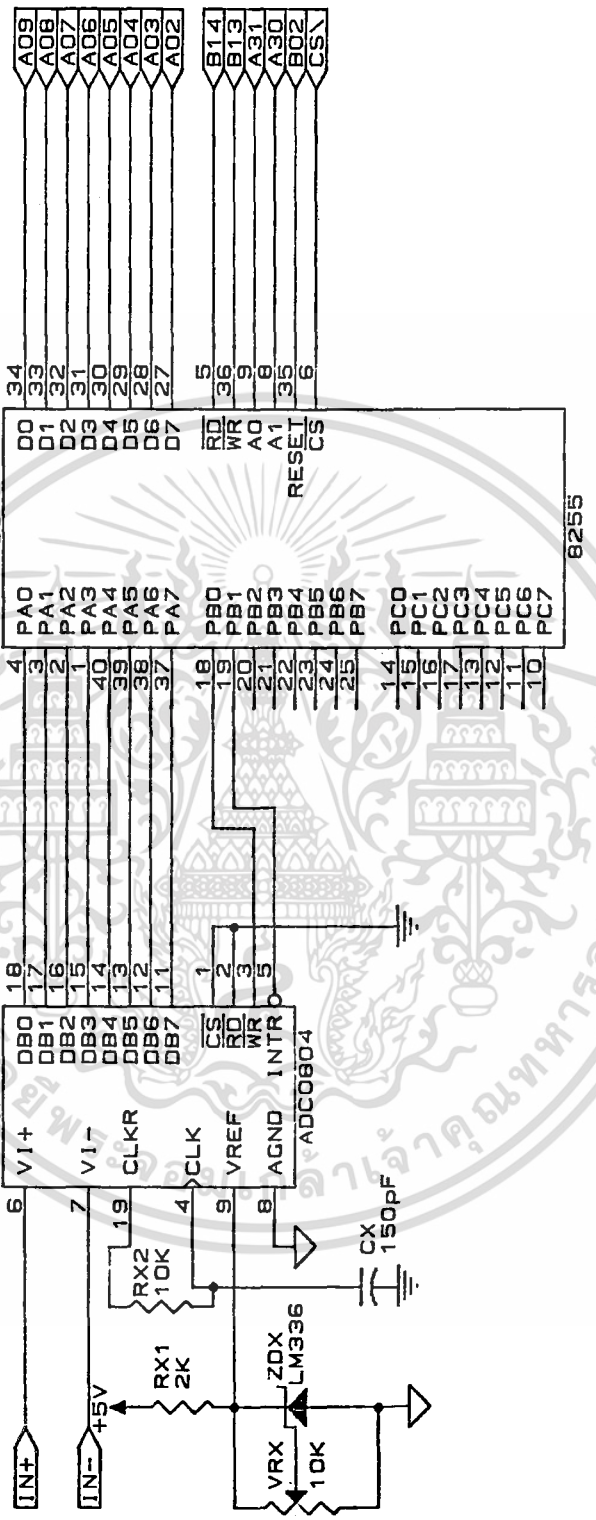
1. วงจรเชื่อมต่อกระบวนการทางอุตสาหกรรมและเลือกช่องสัญญาณอินพุท

ค่าสภาวะที่รับมาจากกระบวนการทางอุตสาหกรรม เป็นแบบลูปกระแส 4 - 20mA. ซึ่งจะถูกเปลี่ยนให้เป็นแรงดัน 1 - 5V. ด้วยความต้านทาน (RESISTOR) ขนาด 250 โอห์ม โดยมีซีเนอร์ไดโอดเป็นตัวป้องกันกระแสเกิน จากนั้นจะผ่านวงจรอนาลอกสวิทช์ (ANALOG SWITCH) ซึ่งจะทำหน้าที่เลือกช่องสัญญาณที่ต้องการ ดังวงจรต่อไปนี้



รูปที่ 4.7 วงจรเชื่อมต่อกระบวนการทางอุตสาหกรรม และเลือกช่องสัญญาณอินพุท

2. อุปกรณ์เปลี่ยนสัญญาณอนาลอกเป็นดิจิตอลจะทำการเปลี่ยนสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอลขนาด 8 บิท ใช้ไอซีเบอร์ ADC0804 โดยมีไอซี LM336 เป็นตัวสร้างแรงดันอ้างอิง แรงดันอ้างอิงสามารถปรับละเอียดได้จาก VRX โดยที่กราวนด์ (GROUND) ของสัญญาณอนาลอกและส่วนของวงจรดิจิตอลจะแยกออกจากกันตามที่ได้แสดงไว้ในวงจร เพื่อการป้องกันสัญญาณรบกวนจากวงจรดิจิตอล เนื่องจากวงจรดิจิตอลทำงานในลักษณะการสวิทช์ดังนั้นจึงเกิดสัญญาณรบกวนขึ้นในระบบกราวนด์ ถ้าหากใช้กราวนด์ร่วมกันแล้วจะทำให้ไอซีเปลี่ยนสัญญาณอนาลอกเป็นดิจิตอลทำงานผิดพลาดไปอย่างมาก สัญญาณอนาลอกอินพุทสำหรับไอซีดังกล่าวจะได้รับมาจากอนาลอกสวิทช์ของวงจรเลือกช่องสัญญาณ (รูปที่ 4.7) จากนั้นสัญญาณอนาลอกอินพุทจะถูกเปลี่ยนให้เป็นข้อมูลทางดิจิตอลที่มีความสัมพันธ์กับสัญญาณอนาลอกดังกล่าว ข้อมูลทางดิจิตอลนี้จะถูกส่งไปยังคอมพิวเตอร์ควบคุมระบบ เพื่อนำไปประมวลผลต่อไป การเชื่อมต่อระหว่างส่วนนี้กับคอมพิวเตอร์คอมพิวเตอร์ควบคุมระบบจะผ่านไอซี 8255 ซึ่งเป็นพอร์ทชนิดโปรแกรมได้ (PROGRAMMABLE I/O) ดังนั้นข้อมูลทางดิจิตอลจะต้องถูกอ่านผ่านทางพอร์ทนี้ไม่สามารถอ่านโดยตรงจากไอซี ADC0804 ได้ ทั้งที่ไอซี ADC 0804 นี้สามารถออกแบบให้ติดต่อกับบัสของคอมพิวเตอร์ควบคุมระบบได้ แต่เนื่องจากไอซีเบอร์นี้มีการอินเตอร์รัพท์ (INTERRUPT) เมื่อทำการเปลี่ยนข้อมูลเสร็จจึงได้ออกแบบไว้ให้มีไอซี 8255 เป็นตัวจัดการอีกชั้นหนึ่ง เหตุผลที่ได้ออกแบบไว้ในลักษณะนี้ก็เนื่องจากว่าวัตถุประสงค์ของวิทยานิพนธ์นี้ต้องการให้ผู้ใช้สามารถพัฒนาซอฟท์แวร์และฮาร์ดแวร์ต่างๆได้โดยง่าย การใช้อินเตอร์รัพท์กับคอมพิวเตอร์ควบคุมระบบจะเป็นการยุ่งยากสำหรับผู้ใช้ในการพัฒนาซอฟท์แวร์และฮาร์ดแวร์ การอ่านข้อมูลและควบคุมข้อมูลผ่านทางพอร์ทจะทำให้ซอฟท์แวร์ไม่ยุ่งยาก และผู้ใช้สามารถออกแบบหรือพัฒนาฮาร์ดแวร์ได้โดยง่ายด้วย



รูปที่ 4.8 อุปกรณ์เปลี่ยนสัญญาณอนาลอกเป็นดิจิตอล

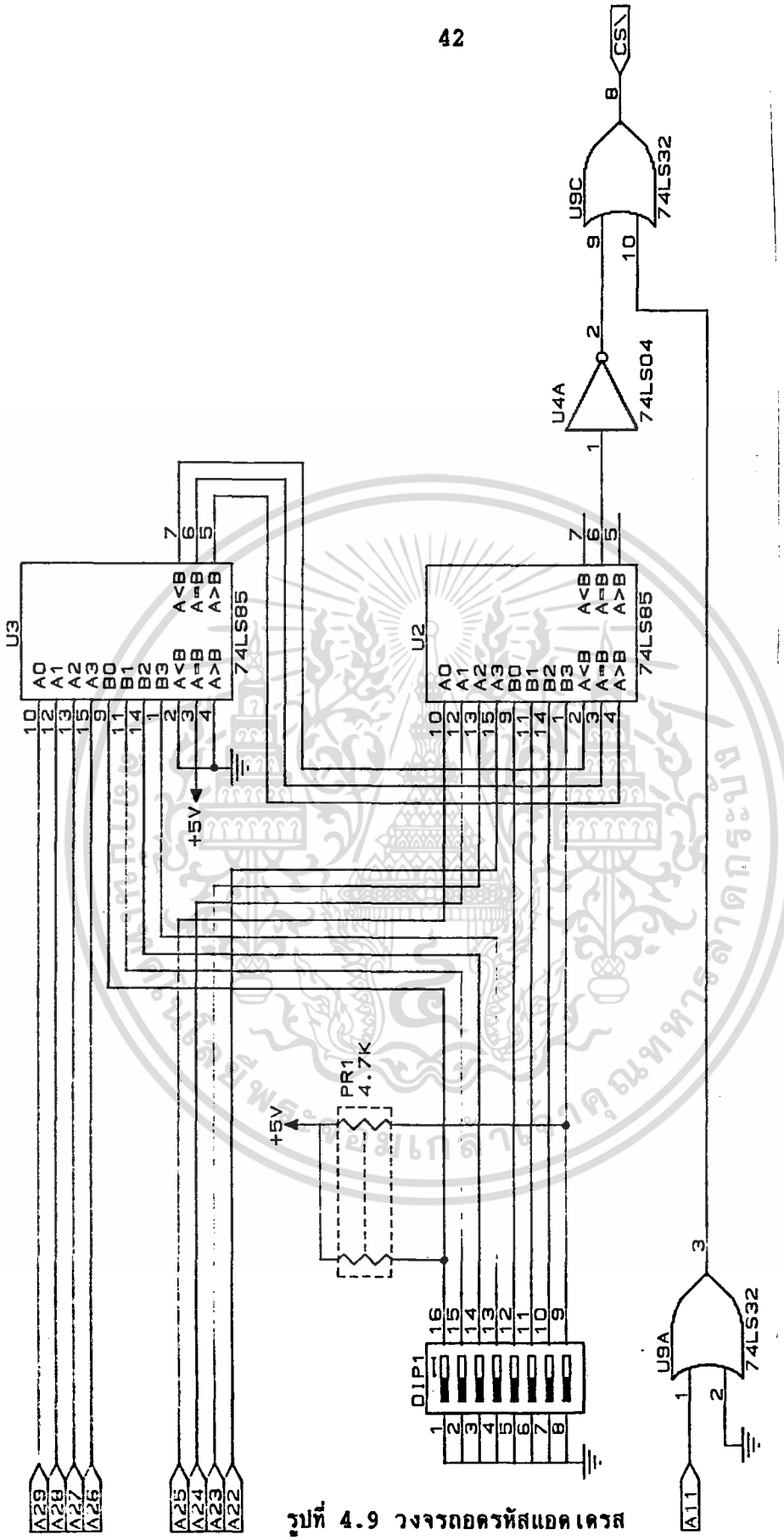
3.วงจรถอดรหัสแอดเดรส (ADDRESS DECODER)

วงจรนี้เป็นวงจรที่ใช้กำหนดแอดเดรสของคาร์ดนั้นๆ และเนื่องจากอาจจะมีคาร์ดชนิดเดียวกันนี้มากกว่า 1 คาร์ดในคอมพิวเตอร์เครื่องเดียวกัน ดังนั้นวงจรนี้จะต้องปรับตั้งได้เพื่อให้แอดเดรสแตกต่างกัน ซึ่งในการออกแบบก็ได้ออกแบบให้ปรับตั้งได้ตั้งแต่ 0300H - 03FFH โดยการปรับตั้งที่ดิพสวิทช์ DIP1 ดังนี้

DIP1							
1	2	3	4	5	6	7	8
A2	A3	A4	A5	A6	A7	A8	A9

การปรับตั้งดิพสวิทช์นี้จะมีสภาวะกลับกับสัญญาณที่ต้องการ เช่นถ้าตั้งไว้ที่ "ON" จะมีลอจิก (LOGIC) เป็น "0" ดังตัวอย่างเช่น

แอดเดรส 0380H - 0383H จะต้องให้สวิทช์ 6,7,8 อยู่ในสภาวะ "OFF" และ 1,2,3,4,5 อยู่ในสภาวะ "ON" และข้อสำคัญสวิทช์ตัวที่ 8 จะต้องอยู่ในสภาวะ "OFF" เสมอ เนื่องจากใน IBM PC นั้นแอดเดรสที่ 0000H - 01FAH ถูกใช้ไปแล้ว



รูปที่ 4.9 วงจรถอดรหัสแอดเดรส

คอนเนคเตอร์สำหรับคาร์ดนี้เป็นแบบ 25 ขา D-TYPE ซึ่งมีการต่อดังนี้

PIN No.	DESCRIPTION	PIN No.	DESCRIPTION
1	CH0+	9	CH4+
2	CH0-	10	CH4-
3	CH1+	11	CH5+
4	CH1-	12	CH5-
5	CH2+	13	CH6+
6	CH2-	14	CH6-
7	CH3+	15	CH7+
8	CH3-	16	CH7-

ขา 17 - 25 ซึ่งไม่ได้แสดงไว้ในตารางนั้นไม่ได้มีการต่อใช้งาน

4.3 วงจรเอาท์พุทสำหรับสัญญาณเตือนในแบบดิจิตอล

วงจรเอาท์พุทสำหรับสัญญาณเตือนได้ออกแบบไว้สำหรับอุปกรณ์เตือนหรือแจ้งภัยมีผังภาพดังในรูปที่ 4.10 และมีรายละเอียดดังนี้

- ส่วนเอาท์พุทถูกแยกออกจากกันอย่างเด็ดขาดทางไฟฟ้าด้วยหน้าสัมผัสของรีเลย์แบบ REED RELAY DRY CONTACT ทนแรงดันได้สูงสุด 150V_{DC} กระแส 1 A. และกำลังงานขนาด 10 วัตต์

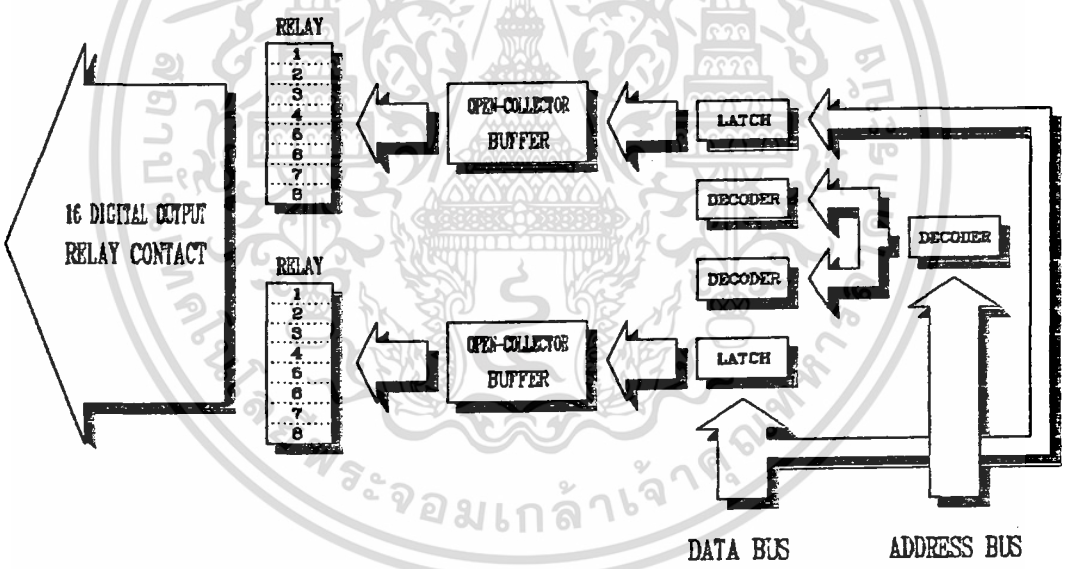
- มีเอาท์พุท 16 จุดต่อคาร์ด โดยแบ่งออกเป็น 2 ส่วนคือจุดที่ 0 - 7 และ 8 - 15 เพื่อความสะดวกในการควบคุม

- ตำแหน่งแอดเดรสเลือกได้ตั้งแต่ 0280H - 72F7H

- มีไดโอดเปล่งแสง (LED) สำหรับแสดงผลสภาวะที่ส่งออกไป

- ความต้องการกำลังงาน +5V_{DC} 700mA.

- คอนเนคเตอร์เป็นแบบ 37 ขา D-TYPE



รูปที่ 4.10 ผังภาพของ
วงจรเอาท์พุทสำหรับสัญญาณเตือนในแบบดิจิตอล

หลักการทำงานของวงจร

1. วงจรถอดรหัสแอดเดรส (ADDRESS DECODER)

วงจรนี้เป็นวงจรที่ใช้กำหนดแอดเดรสของคาร์ดนั้นๆ และเนื่องจากอาจจะมีคาร์ดชนิดเดียวกันนี้มากกว่า 1 คาร์ดในคอมพิวเตอร์เครื่องเดียวกัน ดังนั้นวงจรนี้จะต้องปรับตั้งได้ เพื่อให้แอดเดรสแตกต่างกัน ซึ่งในการออกแบบก็ได้ออกแบบให้ปรับตั้งได้ตั้งแต่ 0280H - 72F7H โดยการปรับตั้งที่ดิพสวิทช์ DIP1 - DIP4 ดังนี้

- DIP1 สำหรับตั้งหลักที่ 1 ตั้งได้ตั้งแต่ 0 - 7 (0XXX - 7XXX)

DIP1							
1	2	3	4	5	6	7	8
0XXX	1XXX	2XXX	3XXX	4XXX	5XXX	6XXX	7XXX

- DIP2 สำหรับตั้งหลักที่ 3 ตั้งได้ตั้งแต่ 7 - F ส่วนหลักที่ 2 ถูกกำหนดให้เป็น "2" เสมอ (X28X - X2FX)

DIP2							
1	2	3	4	5	6	7	8
X28X	X29X	X2AX	X2BX	X2CX	X2DX	X2EX	X2FX

- DIP3 และ DIP4 สำหรับตั้งหลักที่ 4 โดย DIP3 สำหรับจุดอินพุทที่ 0 - 7 และ DIP4 สำหรับ 8 - 15 (X2X0 - X2X7)

DIP3 & DIP4							
1	2	3	4	5	6	7	8
X2X0	X2X1	X2X2	X2X3	X2X4	X2X5	X2X6	X2X7

สมมติว่าต้องการตั้งให้คาร์ดอยู่ที่แอดเดรส 0280H และ 0283H โดยจุดที่ 0 - 7 อยู่ที่ 0280H และ 8 - 15 อยู่ที่ 0283H ก็จะต้องปรับตั้งดิพสวิทช์ดังนี้

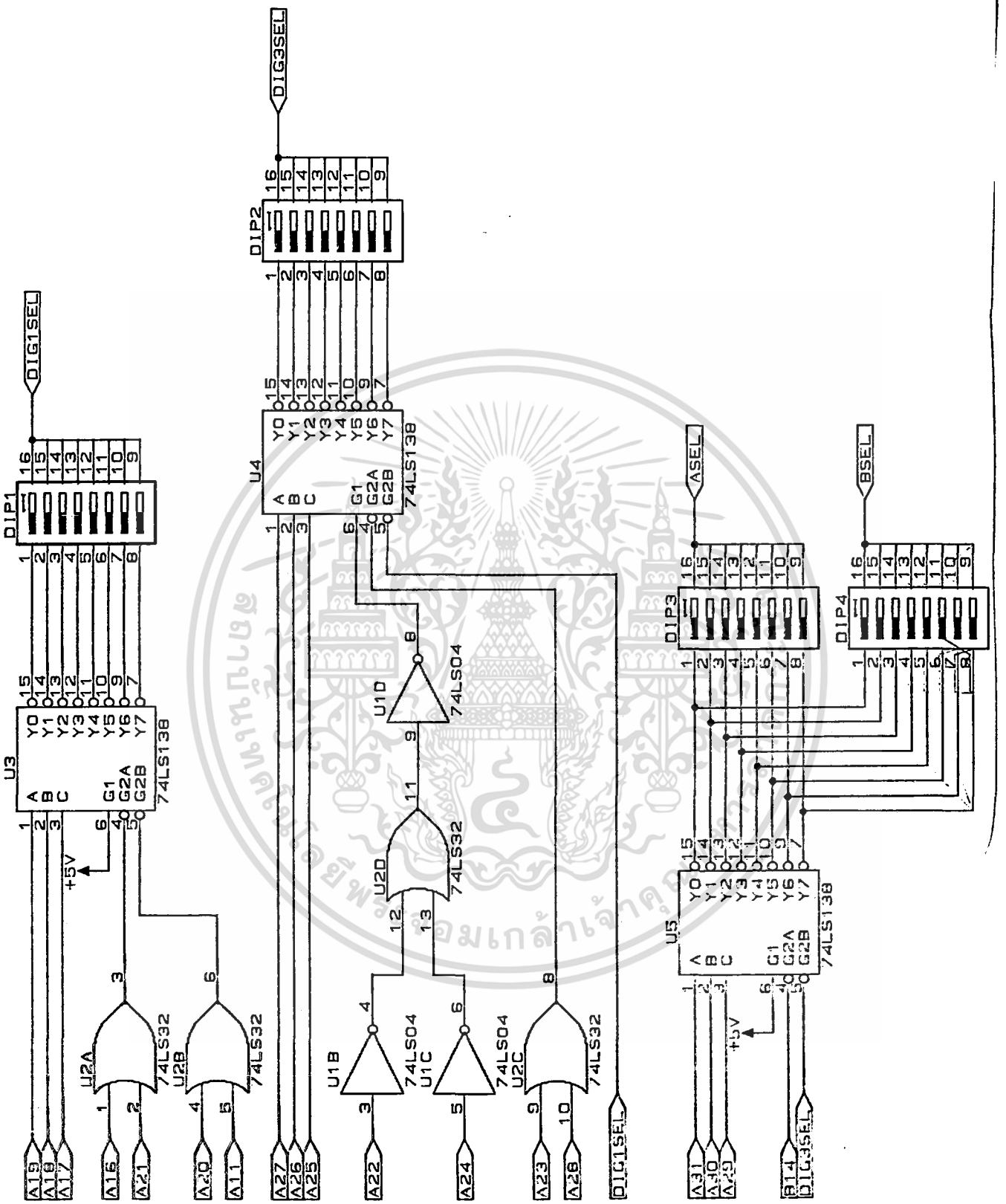
DIP1 ให้สวิทช์ตัวที่ 1 อยู่ในสภาวะ "เปิด"

DIP2 ให้สวิทช์ตัวที่ 1 อยู่ในสภาวะ "เปิด"

DIP3 ให้สวิทช์ตัวที่ 4 อยู่ในสภาวะ "เปิด"

DIP4 ให้สวิทช์ตัวที่ 1 อยู่ในสภาวะ "เปิด"

สวิทช์ตัวอื่นนอกจากที่กำหนดไว้ให้อยู่ในสภาวะ "ปิด"



รูปที่ 4.11 วงจรถอดรหัสแอดเดรส

วงจรในส่วนบนของรูปที่ 4.11 จะเป็นการถอดรหัสในหลักที่ 1 ซึ่งเป็นการกำหนดแอดเดรสสำหรับ 0XXX - 7XXX สัญญาณอินพุทคือ A19,A18,17,A16,A21,A20,A11 ซึ่งเป็นแอดเดรสบัส (ADDRESS BUS) ส่วนเอาท์พุทคือสัญญาณ DIG1SEL

วงจรในส่วนกลางจะเป็นการถอดรหัสในหลักที่ 3 (หลักที่ 2 ถูกกำหนดให้เป็น "2" เสมอ) นั่นคือแอดเดรสตำแหน่ง X28X - X2FX สัญญาณอินพุทคือ A27,A26,A25,A22,A24,A23,A28 และสัญญาณ DIG1SEL ส่วนสัญญาณเอาท์พุทคือสัญญาณ DIG3SEL

วงจรในส่วนล่างสุดก็เป็นการถอดรหัสในหลักที่ 4 คือ X2X0 - X2X7 และจะมีดิพสวิทช์ 2 ชุดคือ DIP3 และ DIP4 สำหรับอินพุทที่ 0 - 7 และ 8 - 16 ตามลำดับ การตั้งดิพสวิทช์ทั้งสองตัวนี้จะต้องไม่ตั้งไว้ที่ตำแหน่งเดียวกันซึ่งจะทำให้การทำงานผิดพลาดได้

คอนเนคเตอร์สำหรับคาร์ดนี้เป็นแบบ 37 ขา D-TYPE ซึ่งมีการต่อดังนี้

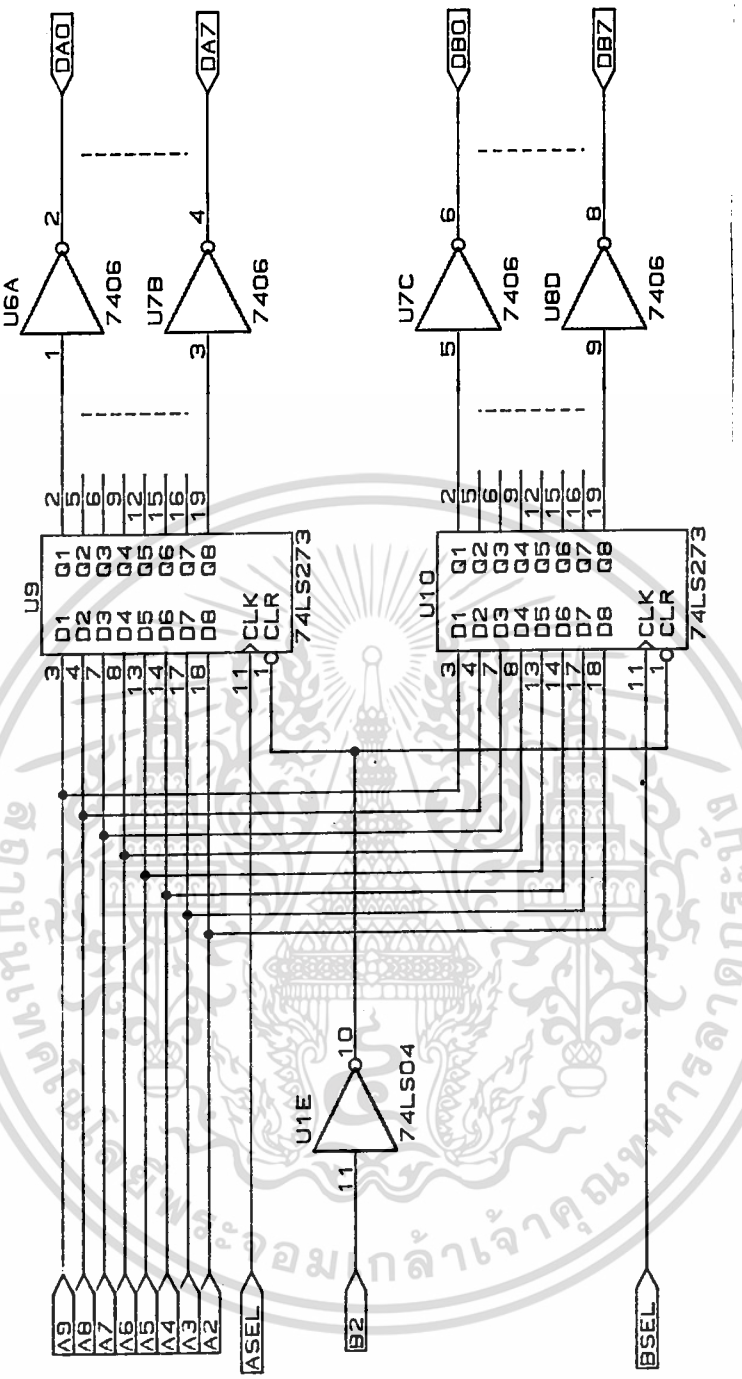
PIN No.	DESCRIPTION	PIN No.	DESCRIPTION
1	ไม่ได้ต่อ	20	PB0B
2	PA0A	21	PB1A
3	PA0B	22	PB1B
4	PA1A	23	PB2A
5	PA1B	24	PB2B
6	PA2A	25	PB3A
7	PA2B	26	PB3B
8	PA3A	27	PB4A
9	PA3B	28	PB4B
10	PA4A	29	PB5A
11	PA4B	30	PB5B
12	PA5A	31	PB6A
13	PA5B	32	PB7B
14	PA6A	33	PB8A
15	PA6B	34	PB8B
16	PA7A	35	ไม่ได้ต่อ

17	PA7B	36	+5V
18	ไม่ได้ต่อ	37	GND
19	PB0A		

2.วงจรแลทช์ (LATCH) และวงจรขับรีเลย์

วงจรแลทช์จะทำหน้าที่เป็นตัวเก็บข้อมูลที่ส่งมาจากคอมพิวเตอร์แล้วค้างค่านั้นไว้จนกว่าจะมีข้อมูลอื่นส่งมาเปลี่ยนแปลง

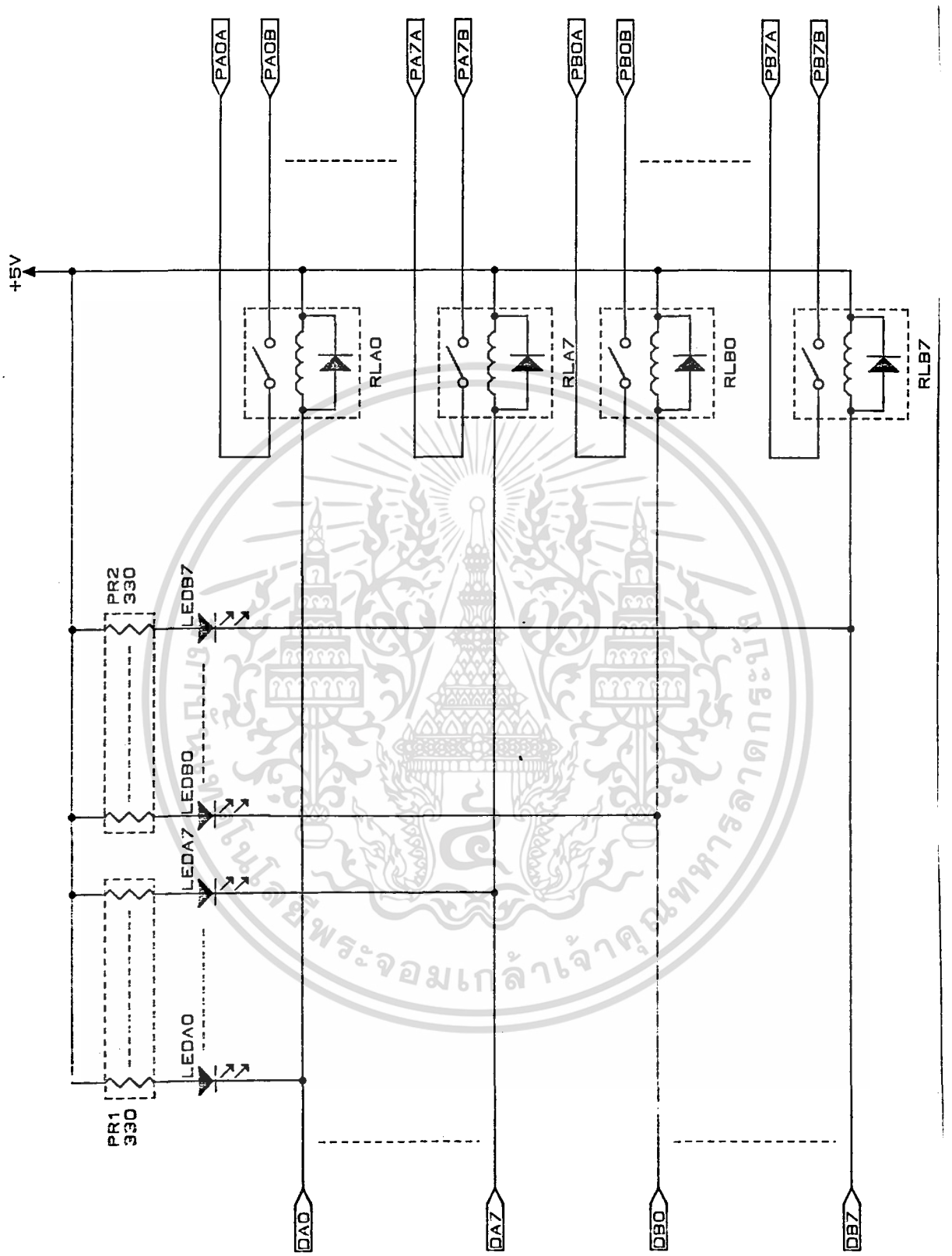
สัญญาณจากบัสข้อมูลจะถูกส่งมายัง U8 และ U10 โดยมีสัญญาณ ASEL และ BSEL เป็นสัญญาณที่สั่งให้เก็บข้อมูลและค้างค่าดังกล่าวไว้ ข้อมูลดังกล่าวจะถูกส่งไปยัง U6 - U8 ซึ่งเป็นวงจรขับรีเลย์และไดโอดเปล่งแสง



รูปที่ 4.12 วงจรแลทช์และวงจรขับรีเลย์

3.วงจรรีเลย์และไดโอดเปล่งแสง

วงจรนี้แสดงการต่อรีเลย์และไดโอดเปล่งแสงรวมทั้งการต่อขาสัญญาณเอาท์พุท ดังในรูปที่ 4.13



รูปที่ 4.13 วงจรรีเลย์และไดโอดเปล่งแสง

บทที่ 5
โปรแกรมควบคุมระบบ

โปรแกรมควบคุมระบบ เป็นโปรแกรมที่พัฒนาขึ้น เพื่อใช้ในการควบคุมการทำงานของส่วนแสดงผลกราฟฟิคความละเอียดสูง ส่วนสำคัญที่สุดของส่วนแสดงผลนี้คือไอซี 82720 ซึ่งเป็น GRAPHICS DISPLAY CONTROLLER (GDC) นั้น มีสมรรถนะในการแสดงผลทางด้านกราฟฟิคสูง การสั่งงาน GDC จะใช้คำสั่ง (COMMAND) แล้วตามด้วยพารามิเตอร์ต่างๆตามที่กำหนด การส่งคำสั่งและพารามิเตอร์เหล่านี้ได้ออกแบบให้ส่งผ่านทางพอร์ทแบบขนาน ดังที่ได้อธิบายรายละเอียดทางฮาร์ดแวร์ไว้ในบทที่ 3

5.1 ตำแหน่งแอดเดรสและฟังก์ชั่น

การควบคุม GDC จะกำหนดค่าผ่านทางรีจิสเตอร์ STATUS และรีจิสเตอร์ FIFO (FIRST-IN FIRST-OUT) แอดเดรส A0 จะเป็นตัวกำหนดว่าค่าที่ส่งมายัง FIFO นั้นจะเป็นคำสั่งหรือเป็นพารามิเตอร์ ดังนี้

A0	I/O COMMAND	I/O ADDRESS	FUNCTION
0	READ	XXXX+0	READ STATUS REGISTER
0	WRITE	XXXX+0	WRITE PARAMETER INTO FIFO
1	READ	XXXX+1	READ FIFO
1	WRITE	XXXX+1	WRITE COMMAND INTO FIFO

XXXX คือแอดเดรสเริ่มต้นของ GDC ในวิทยานิพนธ์ฉบับนี้ใช้ที่ 0300H

5.2 รายละเอียดเกี่ยวกับรีจิสเตอร์ STATUS

รีจิสเตอร์ตัวนี้จะถูกคอมพิวเตอร์ที่ควบคุมระบบอ่านค่าอยู่เสมอ เนื่องจากการโปรแกรมชุดคำสั่งจะต้องใช้รีจิสเตอร์นี้เป็นตัวบอกสภาวะต่างๆ บิทต่างๆของรีจิสเตอร์นี้มีดังนี้

บิทที่	รายละเอียด
0	DATA READY
1	FIFO FULL
2	FIFO EMPTY
3	DRAWING IN PROGRESS
4	DMA EXECUTE
5	VERTICAL SYNC ACTIVE
6	HORIZONTAL BLANK ACTIVE
7	LIGHT PEN DETECT

บิทที่ 0 DATA READY : ถ้าบิทนี้เป็น 1 หมายถึงมีข้อมูลอยู่ใน FIFO แล้ว ถ้าเป็น 0 หมายถึงข้อมูลกำลังอยู่ในระหว่างการส่งจาก FIFO ไปยังรีจิสเตอร์ที่ใช้ในการเชื่อมต่อกับไมโครโปรเซสเซอร์ (MICROPROCESSOR INTERFACE DATA REGISTER) ซึ่งอยู่ในตัว GDC ดังนั้นจะต้องทำการตรวจสอบบิทนี้ทุกครั้งก่อนทำการอ่านข้อมูล

บิทที่ 1 FIFO FULL : ถ้าบิทนี้เป็น 1 หมายถึง FIFO เต็มแล้ว ซึ่งจะเกิดขึ้นเมื่อคอมพิวเตอร์ควบคุมระบบส่งข้อมูลมายัง FIFO ขณะที่ GDC ยังทำคำสั่งเดิมไม่เสร็จ หรือ GDC กำลังอ่านข้อมูลใน FIFO อยู่ เมื่อ FLAG นี้เป็น 0 ก็แสดงว่ามีที่ว่างใน FIFO อย่างน้อย 1 ไบท์ ดังนั้นจะต้องทำการตรวจสอบบิทนี้ทุกครั้งก่อนทำการเขียนข้อมูลลงไปใน FIFO

บิทที่ 2 FIFO EMPTY : ถ้าบิทนี้เป็น 1 แสดงว่า FIFO ว่างทั้งหมด และ GDC ได้กระทำคำสั่งที่ผ่านมาเสร็จเรียบร้อยแล้ว

บิทที่ 3 DRAWING IN PROGRESS : ขณะที่ GDC กำลังวาดอยู่บิทนี้จะเป็น 1

บิทที่ 4 DMA EXECUTE : บิทนี้จะเป็น 1 ขณะที่กำลังทำ DMA

บิทที่ 5 VERTICAL SYNC : บิทนี้จะเป็น 1 ขณะที่กำลังทำกระบวนการ VERTICAL RETRACE SYNC

บิทที่ 6 HORIZONTAL BLANKING ACTIVE : เมื่อบิทนี้เป็น 1 แสดงว่ากำลังทำ HORIZONTAL RETRACE BLANKING

บิทที่ 7 LIGHT PEN DETECT : ถ้าบิทนี้เป็น 1 แสดงว่ารีจิสเตอร์แอดเดรสของปากกาแสง (LIGHT PEN ADDRESS (LAD)) มีค่าตำแหน่งของปากกาแสงเก็บอยู่ บิทนี้จะเป็น 0 เมื่อค่าใน LAD ถูกอ่านไปยัง FIFO ด้วยคำสั่ง LIGHT PEN READ

5.3 ชุดคำสั่งของ GDC

คำสั่งของ GDC มีทั้งสิ้น 20 คำสั่ง แบ่งออกได้เป็น 4 กลุ่ม ดังนี้

กลุ่มที่ 1 คำสั่งควบคุมวิดีโอ (VIDEO CONTROL COMMAND)

1.RESET ใช้ในการรีเซ็ท (RESET) GDC

0	0	0	0	0	0	0	0

2.SYNC ใช้ในการกำหนดรูปแบบในการแสดงผล

0	0	0	0	1	1	1	DE

3.VSYNC ใช้ในการเลือก MASTER หรือ SLAVE สำหรับ VIDEO SYNCHRONIZATION MODE

0	1	1	0	1	1	1	M

4.CCHAR ใช้ในการกำหนดเคอร์เซอร์ (CURSOR) และความสูงของตัวอักษร

0	1	0	0	1	0	1	1

กลุ่มที่ 2 คำสั่งควบคุมจอแสดงผล (DISPLAY CONTROL COMMAND)

1.START ใช้ในการสั่งให้เริ่มสแกน (SCAN) จอแสดงผล

0	1	1	0	1	0	1	1

2.BCTRL ใช้ในการควบคุม BLANKING และ UNBLANKING ของจอแสดงผล

0	0	0	0	1	1	0	DE

3.ZOOM ใช้ในการกำหนดอัตราการขยายในการวาดและแสดงผล

0	1	0	0	0	1	1	0

4.CURS ใช้ในการกำหนดตำแหน่งของเคอร์เซอร์ในหน่วยความจำแสดงผล (DISPLAY MEMORY)

0	1	0	0	1	0	0	1

5.PRAM ใช้ในการกำหนดแอดเดรสเริ่มต้นและความยาวของพื้นที่แสดงผลและกำหนดค่า 8 ไบท์ของตัวอักษรกราฟฟิค (GRAPHICS CHARACTER)

0	1	1	1	SA			

6.PITCH ใช้ในการกำหนดความกว้างของหน่วยความจำแสดงผล

0	1	0	0	0	1	1	1

กลุ่มที่ 3 คำสั่งควบคุมการวาด (DRAWING CONTROL COMMAND)

1.WDAT ใช้ในการใส่ข้อมูลลงในหน่วยความจำแสดงผล

0	0	1	TYPE	0	MOD

2.MASK ใช้ในการกำหนดค่ารีจิสเตอร์ MASK

0	1	0	0	1	0	1	0

3.FIGS ใช้ในการกำหนดพารามิเตอร์สำหรับตัวประมวลผลการวาด (DRAWING PROCESSOR)

0	1	0	0	1	1	0	0

4.FIGD ใช้ในการสั่งให้มีการวาดตามพารามิเตอร์ที่กำหนดโดย FIGS

0	1	1	0	1	1	0	0

5.GCHRD ใช้ในการวาดตัวอักษรกราฟฟิคลงในหน่วยความจำแสดงผล

0	1	1	0	1	0	0	0

กลุ่มที่ 4 คำสั่งอ่านข้อมูล (DATA READ COMMAND)

1.RDAT ใช้ในการอ่านข้อมูลจากหน่วยความจำแสดงผล

1	0	1	TYPE	0	MOD

2.CURD ใช้ในการอ่านตำแหน่งของ เคอร์เซอร์

1	1	1	0	0	0	0	0

3.LPRD ใช้ในการอ่านแอดเดรสของปากกาแสง (LIGHT PEN)

1	1	0	0	0	0	0	0

กลุ่มที่ 5 คำสั่งควบคุมการ DMA (DIRECT MEMORY ACCESS)

1.DMAR ใช้ในการขออ่านข้อมูลแบบ DMA

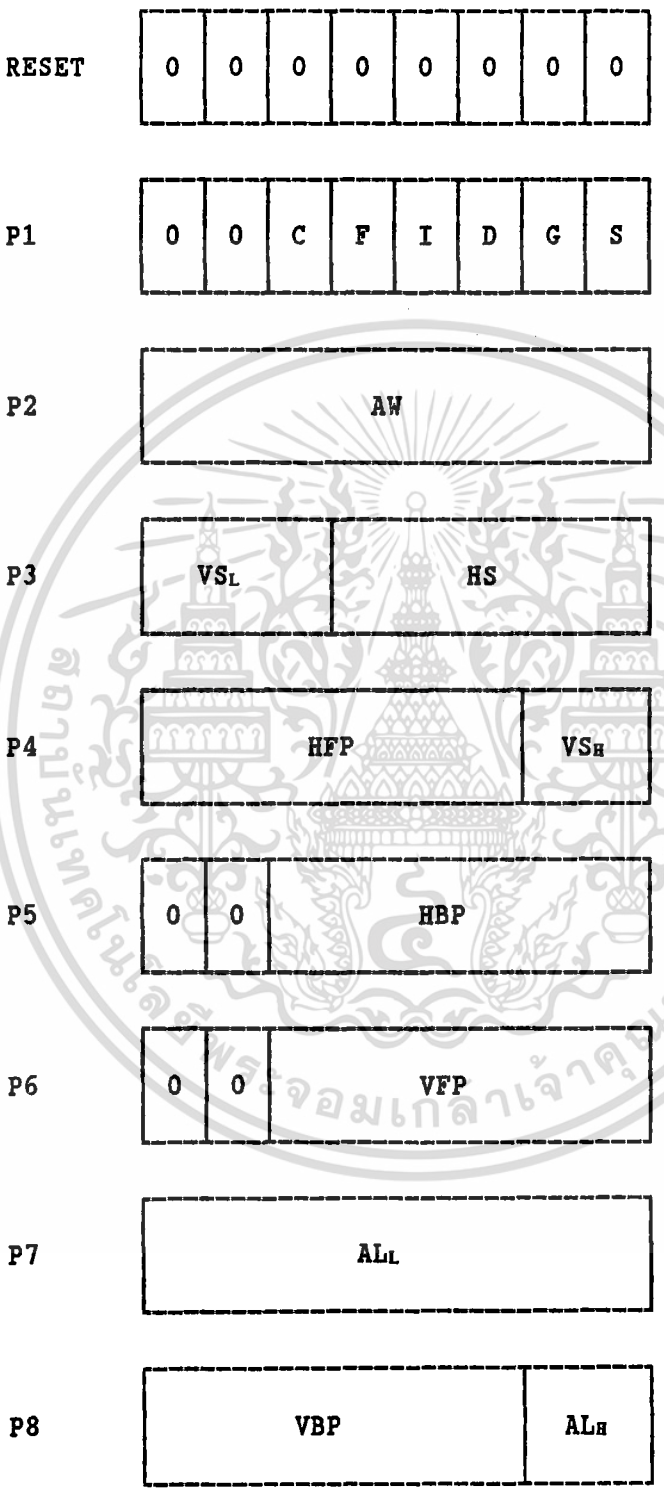
1	0	1	TYPE	1	MOD

2.DMAW ใช้ในการขอ เขียนข้อมูลแบบ DMA

0	0	1	TYPE	1	MOD

5.4 รายละเอียดเกี่ยวกับคำสั่งต่างๆที่จำเป็นและใช้ในโปรแกรมควบคุมระบบ

5.4.1 คำสั่ง RESET



AW = ACTIVE DISPLAY WORDS PER LINE MINUS 2

VS_L = VERT.SYNC WIDTH, LOW ORDER BITS

HS = HORZ.SYNC WIDTH MINUS 1

HFP = HORZ.FRONT PORCH WIDTH MINUS 1

VS_H = VERT.SYNC WIDTH, HIGH ORDER BITS

HBP = HORZ.BACK PORCH MINUS 1

VFP = VERT.FRONT PORCH WIDTH

AL_L = ACTIVE DISPLAY LINES PER VIDEO FIELD, LOW BITS

VBP = VERTICAL BACK PORCH WIDTH

AL_H = ACTIVE DISPLAY LINES PER VIDEO FIELD, HIGH BITS

และพารามิเตอร์ใน P1

C	G	DISPLAY MODE
0	1	GRAPHICS MODE
I	S	VIDEO FRAMING
0	0	NON-INTERLACED
1	1	INTERLACED

NON-INTERLACES S = I = 0

INTERLACED S = I = 1

D = DYNAMIC RAM REFRESH CYCLES ENABLE (0 = DISABLE, 1 = ENABLE)

F	DRAWING TIME WINDOW
0	DRAWING DURING ACTIVE DISPLAY TIME & RETRACE BLINKING
1	DRAWING ONLY DURING RETRACE BLINKING

5.4.2 คำสั่ง VSYNC

VSYNC	0	1	1	0	1	1	1	M

M = MASTER MODE, เมื่อ M = 1 GDC จะกำเนิดสัญญาณพัลซ์ VERTICAL SYNC. ถ้า M = 0 GDC จะรับสัญญาณดังกล่าวจากภายนอก

5.4.3 คำสั่ง CCHAR

CCHAR	0	1	0	0	1	0	1	1
P1	0	0	0	0	0	0	0	0
P2	1	1	0	0	0	0	0	0
P3	0	0	0	0	0	0	0	0

5.4.4 คำสั่ง START

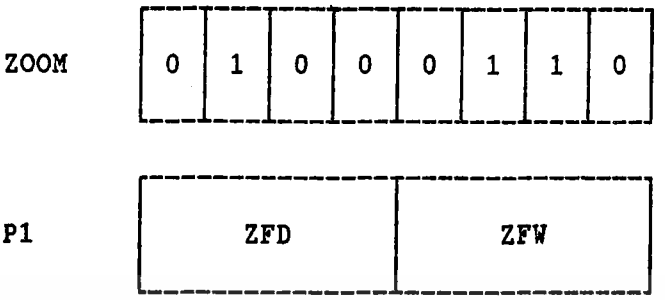
START	0	1	1	0	1	0	1	1

5.4.5 คำสั่ง BCTRL

BCTRL	0	0	0	0	1	0	0	E

E = DISPLAY ENABLE = 1 และ DISABLE = 0

5.4.6 คำสั่ง ZOOM



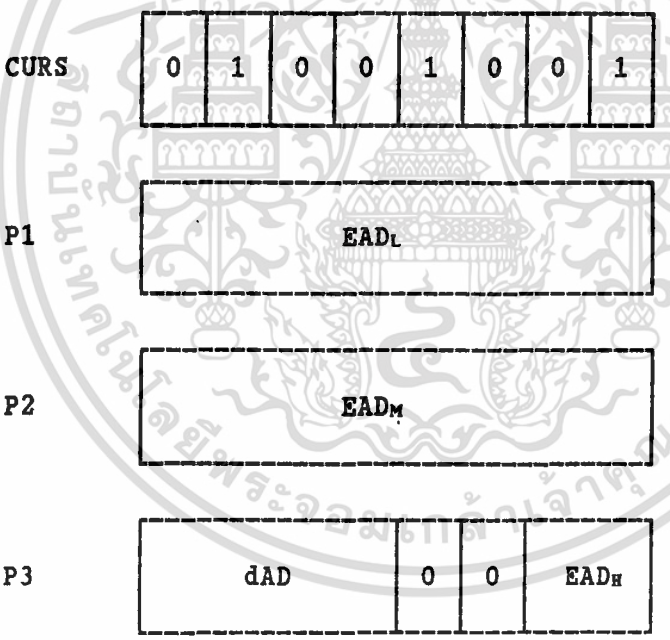
ZFD = อัตราการขยายของการแสดงผล

ZFW = อัตราการขยายของตัวอักษรและ AREA FILLING

อัตราการขยาย 0000_2 = 1 เท่า (ไม่มีการขยาย)

1111_2 = 16 เท่า

5.4.7 คำสั่ง CURS



EAD = HIGH ORDER EXECUTION WORD

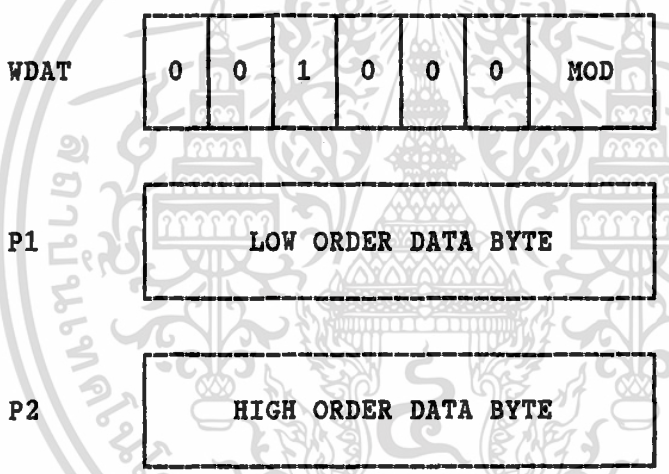
dAD = DOT ADDRESS

5.4.8 คำสั่ง WDAT

คำสั่ง WDAT (WRITE DATA) เป็นคำสั่งที่นำข้อมูลเข้าไปเก็บไว้ในหน่วยความจำแสดงผลได้โดยตรง ตำแหน่งแอดเดรสที่ต้องการจะเก็บไว้ในรีจิสเตอร์ EAD ในคำสั่งที่ใช้กำหนดตำแหน่งเริ่มต้นของเคอร์เซอร์ EAD ซึ่งมีขนาด 18 บิทนั้นจะเปลี่ยนค่าเป็นค่าสุดท้ายโดยอัตโนมัติเมื่อการวาดสิ้นสุดลง ทิศทางในการเขียนข้อมูลหรือเก็บข้อมูลนี้จะถูกกำหนดโดยพารามิเตอร์ DIR ซึ่งเป็นพารามิเตอร์ตัวแรกในคำสั่ง FIGS

ในคำสั่ง WDAT เองก็ได้แบ่งเป็น 3 แบบแต่ละแบบมี OPCODE ไม่เหมือนกัน แบบแรกเมื่อข้อมูลมีขนาด 2 ไบท์ แบบที่สองเมื่อข้อมูลมีขนาด 1 ไบท์และเป็นไบท์ต่ำ (LOW ORDER DATA BYTE) แบบสุดท้ายเมื่อข้อมูลมีขนาด 1 ไบท์เช่นกันแต่เป็นไบท์สูง (HIGH ORDER DATA BYTE)

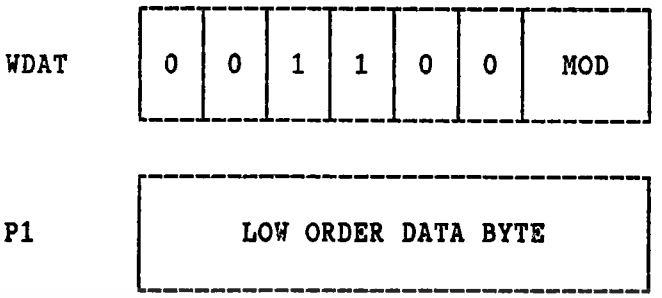
แบบที่ 1 เมื่อข้อมูลมีขนาด 2 ไบท์



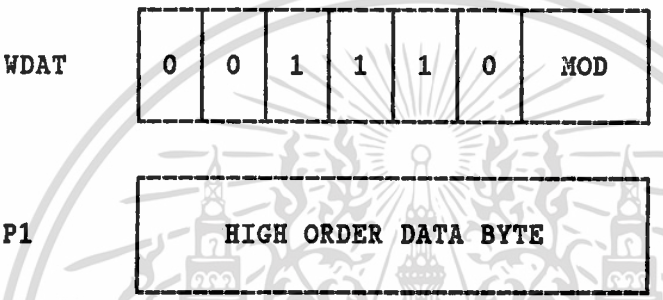
MOD = RMW MEMORY CYCLE

MOD		FUNCTION
0	0	REPLACE WITH PATTERN
0	1	COMPLEMENT
1	0	RESET TO ZERO
1	1	SET TO ONE

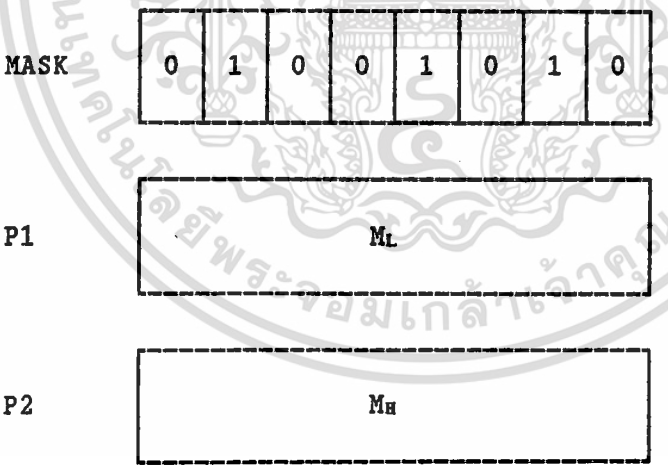
แบบที่ 2 เมื่อข้อมูลมีขนาด 1 ไบท์และเป็นไบท์ต่ำ



แบบที่ 3 เมื่อข้อมูลมีขนาด 1 ไบท์และเป็นไบท์สูง



5.4.9 คำสั่ง MASK



M_L = LOW ORDER BYTE

M_H = HIGH ORDER BYTE

5.4.10 คำสั่ง FIGS

FIGS เป็นคำสั่งที่ใช้กำหนดพารามิเตอร์สำหรับตัวประมวลผลการวาด มีการกำหนดพารามิเตอร์ต่างๆดังนี้

FIGS	0	1	0	0	1	1	0	0

P1	SL	R	A	GC	L	DIR

SL	R	A	GC	L	OPERATION
0	0	0	0	0	CHARACTER DISPLAY MODE DRAWING,INDIVIDUAL DOT DRAWING, DMA, WDAT, AND RDAT
0	0	0	0	1	STRAIGHT LINE DRAWING
0	0	0	1	0	GRAPHICS CHARACTER DRAWING AND AREA FILLING WITH GRAPHICS CHARACTER PATTERN
0	0	1	0	0	ARC AND CIRCLE DRAWING
0	1	0	0	0	RECTANGLE DRAWING
1	0	0	0	0	SLANTED GRAPHICS CHARACTER DRAWING AND SLANTED AREA FILLING

SL : SLANTED GRAPHICS CHARACTER

R : RECTANGLE

A : ARC\CIRCLE

GC : GRAPHICS CHARACTER

L : LINE (VECTOR)

DIR : DRAWING DIRECTION

หลังจากใช้คำสั่ง FIGS แล้วจะต้องใช้คำสั่งใดคำสั่งหนึ่งต่อไปนี้ไปทำการเริ่มกระบวนการ READ-MODIFY-WRITE (RWM)

FIGD,GCHRD,WDAT,RDAT

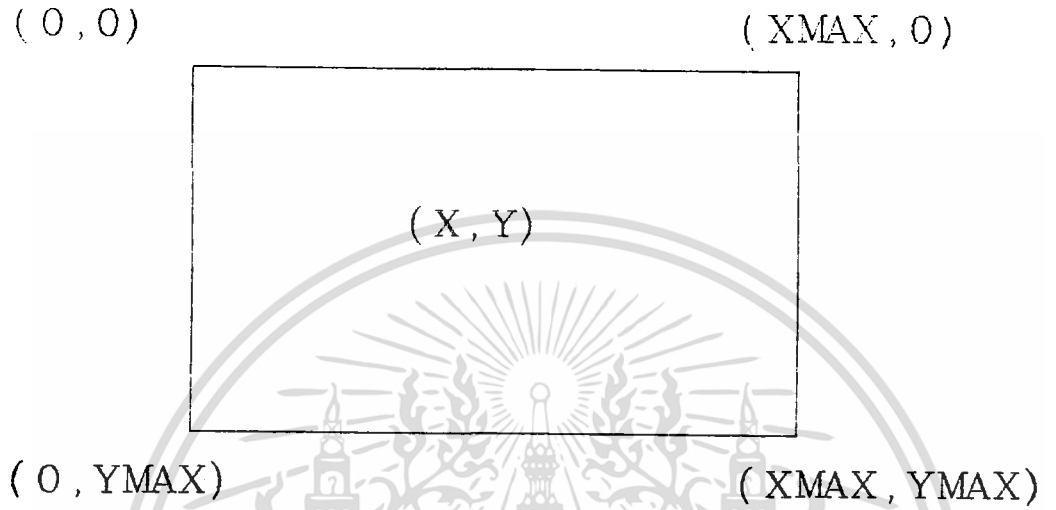
โดยคำสั่งต่างๆ เหล่านี้มีวิธีใช้และรูปแบบในการใช้แตกต่างกันดังนี้

คำสั่ง	วิธีการ	SL	R	A	GC	L
FIGD	LINE OR VECTOR DRAWING	0	0	0	0	1
	ARC AND CIRCLE DRAWING	0	0	1	0	0
	RECTANGLE DRAWING	0	1	0	0	0
	SINGLE DOT DRAWING	0	0	0	0	0
GCHRD	AREA FILLING	0	0	0	1	0
	SLANTED AREA FILLING	1	0	0	1	0
WDAT	SINGLE WORD	0	0	0	0	0
	SUCCESIVE WORD WRITES	0	0	0	0	0
	CHARACTER MODE READING	0	0	0	0	0
DMAW	DMA WRITE SEQUENCE	0	0	0	0	0
DMAR	DMA READ SEQUENCE	0	0	0	0	0

จากตารางคำสั่ง WDAT นั้นค่าพารามิเตอร์เหมือนกันทั้งหมด เนื่องจาก WDAT ใช้พารามิเตอร์อื่นประกอบในการเลือกทำคำสั่ง

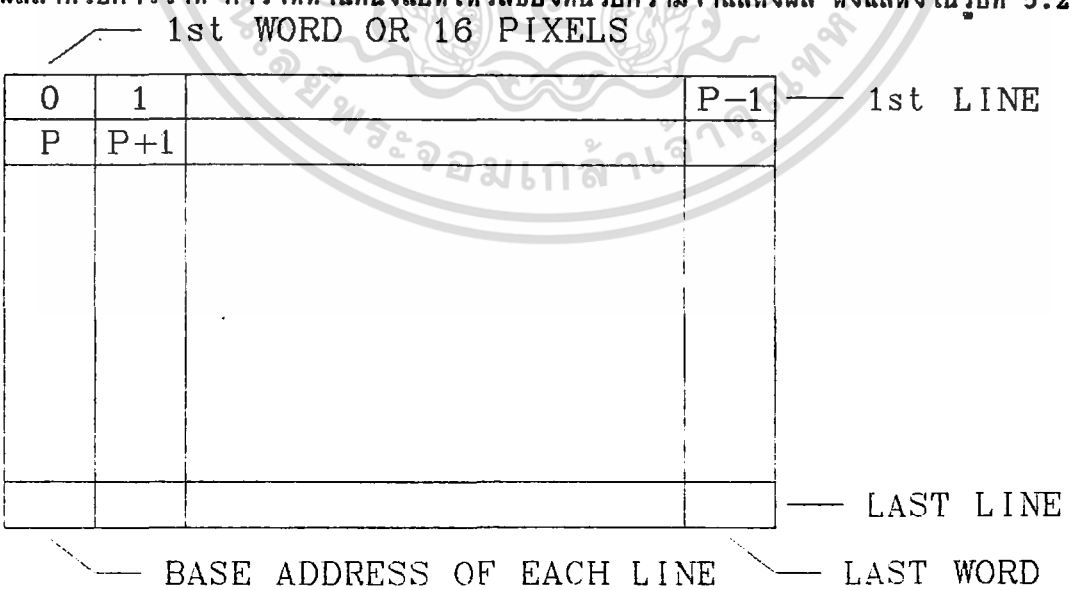
5.5 การเปลี่ยนตำแหน่ง X-Y เป็นตำแหน่งแอดเดรสในหน่วยความจำ

เนื่องจากการกำหนดตำแหน่งการวาดนั้นกำหนดด้วยระบบแกน X และแกน Y ซึ่งเริ่มต้นที่ (0,0) และแกน X มีการเพิ่มค่าจากซ้ายไปขวา และแกน Y มีการเพิ่มค่าจากบนลงล่างดังในรูปที่ 5.1



รูปที่ 5.1 แสดงการกำหนดตำแหน่ง X-Y ในการวาด

แต่การจัดแอดเดรสของหน่วยความจำแสดงผลนั้นจะแตกต่างกัน ดังนั้นจึงต้องมีการนำค่า X-Y ที่ต้องการมาทำการคำนวณเพื่อเปลี่ยนให้เป็นตำแหน่งแอดเดรสของหน่วยความจำแสดงผล เนื่องจาก GDC ต้องการพารามิเตอร์ที่เป็นแอดเดรสของหน่วยความจำแสดงผลสำหรับการวาด การจัดตำแหน่งแอดเดรสของหน่วยความจำแสดงผล ดังแสดงในรูปที่ 5.2



รูปที่ 5.2 แสดงการจัดแอดเดรสของหน่วยความจำแสดงผล

จากรูปที่ 5.2 จะเห็นได้ว่าการจัดแอดเดรสของหน่วยความจำแสดงผลนั้นจะเรียงตามแกน X นั่นคือเพิ่มค่าจากซ้ายไปขวา และจากบนลงล่างเหมือนกับการจัดตำแหน่งในระบบ X-Y แต่หน่วยความจำแสดงผล 1 แอดเดรสนั้นประกอบด้วยจุด 16 จุด ดังนั้น

เมื่อ P = จำนวน 16 BIT WORDS ใน 1 เส้น

ดังนั้น $P = (X_{MAX} + 1) / 16$

ในที่นี้ X_{MAX} มีค่าเท่ากับ 1023 ดังนั้น $P = 64_{10}$ หรือ 40H

จะได้ LINE BASE ADDRESS (LBA) = P * Y

ดังนั้นแอดเดรสของหน่วยความจำแสดงผลคือ

EAD = LBA + TRUNC(X/16)

และตำแหน่งของบิทคือ dAD = X MOD 16

จากสมการต่างๆที่กล่าวมานี้นำมาเขียนเป็นอัลกอริทึมได้ดังนี้

```
procedure AddressConversion;
begin
  LBA := 64 * Y;
  EAD := LBA + TRUNC(X/16);
  dAD := X MOD 16;
end;
```

5.6 การวาดแบบเวคเตอร์ (VECTOR DRAWING)

การวาดแบบเวคเตอร์หรือการวาดเส้นตรงนี้ต้องการพารามิเตอร์ 9 ไบท์ตามหลังคำสั่ง FIGS ไบท์แรกจะเป็น "L" bit set และการกำหนดทิศทางในการวาด ส่วนอีก 8 ไบท์จะเป็นการกำหนดค่ารีจิสเตอร์ของ DIGITAL DIFFERENTIAL ANALYZER (DDA : เป็นฮาร์ดแวร์ที่อยู่ในตัว GDC ทำหน้าที่คำนวณทางคณิตศาสตร์จะทำงานไปพร้อมกับการอ่านข้อมูลด้วยฮาร์ดแวร์ซึ่งใน GDC เรียกว่า READ MODIFY WRITE (RWM)) ซึ่งประกอบด้วย DC,D,D2 และ D1 ค่าเหล่านี้จะเป็นจุดเริ่มต้นและจุดสิ้นสุดของเวคเตอร์ การหาค่าต่างๆเหล่านี้กระทำได้โดย

ขั้นที่ 1 คำนวณหาระยะห่างเป็นจำนวนจุดทางด้านแกน X และแกน Y ด้วยสมการ

$$dX := X_2 - X_1$$

$$dY := Y_2 - Y_1$$

เมื่อ X_2 คือจุดสุดท้ายของเวคเตอร์ทางด้านแกน X

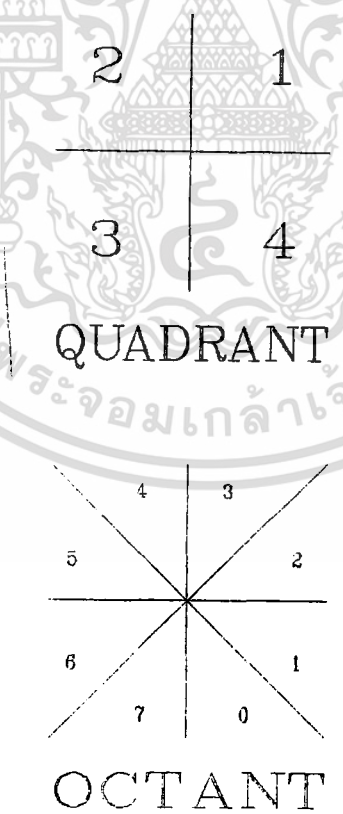
X_1 คือจุดเริ่มต้นของเวคเตอร์ทางด้านแกน Y

Y_2 คือจุดสุดท้ายของเวคเตอร์ทางด้านแกน Y

Y_1 คือจุดเริ่มต้นของเวคเตอร์ทางด้านแกน Y

ค่าของ dX และ dY สามารถเป็นได้ทั้งบวกและลบ ค่าทั้งสองคือระยะห่างเป็นจำนวนจุดของทั้งสองแกนแต่ไม่ใช่จำนวนจุดที่จะวาด ตัว GDC จะทำการวาดจุดแรกลงไปในหน่วยความจำในตำแหน่งของเคอร์เซอร์ขณะที่ DDA ทำการคำนวณแอดเดรสของจุดที่ 2 เมื่อ GDC ทำการวาดจุดที่ 2 DDA ก็จะทำการคำนวณจุดต่อไป เป็นเช่นนี้ไปเรื่อยๆจนกระทั่งครบทุกจุดในเวคเตอร์หรือในเส้นนั้น โดยจำนวนจุดที่จะวาดทั้งหมดจะอยู่ในรีจิสเตอร์ DC และเมื่อ GDC ทำการวาดถึงจุดสุดท้ายค่าในรีจิสเตอร์ DC จะมีค่าเป็นศูนย์และเคอร์เซอร์จะเคลื่อนที่ไปอยู่ที่จุดสุดท้าย

ขั้นที่ 2 เป็นการหาค่า QUADRANT และค่า OCTANT จากค่า dX และ dY ที่ได้จะถูกนำไปประมวลผลเพื่อหา QUADRANT และ OCTANT ซึ่งการกำหนด QUADRANT และ OCTANT ของ GDC มีการกำหนดดังในรูปที่ 5.3



รูปที่ 5.3 การกำหนด QUADRANT และ OCTANT

การหา QUADRANT จะหาได้จากการเปรียบเทียบเครื่องหมายของ dX และ dY ตามตารางต่อไปนี้

เครื่องหมายของ dX	เครื่องหมายของ dY	QUADRANT
+	-	1
-	-	2
-	+	3
+	+	4

ส่วนการหา OCTANT จะนำค่า QUADRANT ที่หาได้มาทำการประมวลผลร่วมกับการเปรียบเทียบค่าสัมบูรณ์ (ABSOLUTE) ของ dX และ dY ($|dX|, |dY|$) ตามตารางดังต่อไปนี้

$|dX| > |dY|$

QUADRANT	OCTANT
1	2
2	5
3	6
4	1

$|dX| < |dY|$

QUADRANT	OCTANT
1	3
2	4
3	7
4	0

ถ้า $|dX| = |dY|$ แสดงว่าเวคเตอร์หรือเส้นที่จะวาดอยู่ในแนวเส้นทะแยงมุมพอดี ดังนั้นค่าที่ได้จะคาบอยู่ระหว่าง 2 OCTANT ให้ใช้ค่าเลขคู่เป็นค่า OCTANT

ขั้นที่ 3 เป็นการกำหนดแกนการวาดอิสระ (INDEPENDENT DRAWING AXIS) และคำนวณค่าพารามิเตอร์ในการวาด แกนที่เป็นแกนวาดอิสระจะเป็นแกนที่มีความยาวมากกว่าอีกแกนหนึ่ง เช่น แกน X จะเป็นแกนวาดอิสระเมื่อ $|dX| > |dY|$ จะเป็นแกน Y เมื่อ $|dX| <= |dY|$ ดังตารางต่อไปนี้

OCTANT	INDEPENDENT AXIS	DEPENDENT AXIS
0	Y	X
1	X	Y
2	X	Y
3	Y	X
4	Y	X
5	X	Y
6	X	Y
7	Y	X

ค่าพารามิเตอร์อื่นๆในการวาดของ DDA จะหาได้จากการคำนวณจากค่าของ $|dX|$ และ $|dY|$ ซึ่งประกอบด้วย DC,D,D2,D1 ตามสมการดังนี้

$$DC = |dI|$$

$$D = (2 * |dD|) - |dI|$$

$$D2 = 2 * (|dD| - |dI|)$$

$$D1 = 2 * |dD|$$

เมื่อ dI เป็นระยะห่างของแกนการวาดอิสระ

dD เป็นระยะห่างของแกนการวาดไม่อิสระ

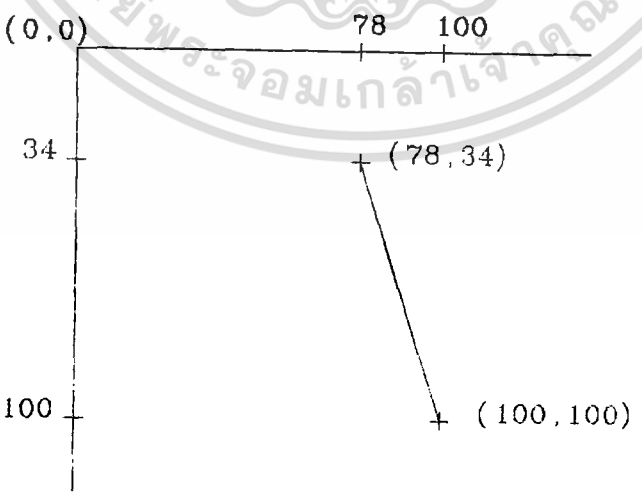
ดังนั้นการคำนวณหาค่าพารามิเตอร์สำหรับการวาดแบบเวคเตอร์หรือการวาดเส้นจะมีอัลกอริทึม (ALGORITHM) ดังนี้

```
procedure VectorDrawingParameterCalculation;
begin { VectorDrawingParameterCalculation }
  dX := X2-X1;
  dY := Y2-Y1;
  if (dX >= 0) and (dY < 0)  then Quadrant := 1 else
  if (dX < 0)  and (dY < 0)  then Quadrant := 2 else
  if (dX < 0)  and (dY >= 0) then Quadrant := 3 else
  if (dX >= 0) and (dY >= 0) then Quadrant := 4;
  case Quadrant of
    1 : begin
          if abs(dX) > abs(dY) then Octant := 2 else
          if abs(dX) < abs(dY) then Octant := 3 else
          if abs(dX) = abs(dY) then Octant := 2;
        end;
    2 : begin
          if abs(dX) > abs(dY) then Octant := 5 else
          if abs(dX) < abs(dY) then Octant := 4 else
          if abs(dX) = abs(dY) then Octant := 4;
        end;
    3 : begin
          if abs(dX) > abs(dY) then Octant := 6 else
          if abs(dX) < abs(dY) then Octant := 7 else
          if abs(dX) = abs(dY) then Octant := 6;
        end;
    4 : begin
          if abs(dX) > abs(dY) then Octant := 1 else
          if abs(dX) < abs(dY) then Octant := 0 else
          if abs(dX) = abs(dY) then Octant := 0;
        end;
  end;
```

```
  if Octant in [0,3,4,7] then
  begin
    dI := dY;
    dD := dX;
  end
  else { Octant in [1,2,5,6] }
  begin
    dI := dX;
    dD := dY;
  end;
  DC := abs(dI);
  D  := (2 * abs(dD)) - abs(dI);
  D2 := 2 * (abs(dD) - abs(dI));
  D1 := 2 * abs(dD);
end { VectorDrawingParameterCalculation };
```

ตัวอย่างการวาดแบบเวคเตอร์

ตัวอย่างนี้จะทำให้เข้าใจการคำนวณหาพารามิเตอร์ต่างๆสำหรับการวาดแบบเวคเตอร์ได้ดีขึ้นจะขอยกตัวอย่างการวาดแบบเวคเตอร์และแสดงวิธีการคำนวณ โดยกำหนดให้จุดเริ่มต้นอยู่ที่ (100,100) และจุดสุดท้ายอยู่ที่ (78,34) ดังในรูปที่ 5.4



รูปที่ 5.4 ตัวอย่างการวาดแบบเวคเตอร์

ขั้นแรกเป็นการคำนวณหา dX และ dY จะได้

$$dX = X_2 - X_1 = 78 - 100 = -22$$

$$dY = Y_2 - Y_1 = 34 - 100 = -66$$

ขั้นที่ 2 เป็นการหาค่า QUADRANT และค่า OCTANT โดยสังเกตุจากเครื่องหมายของ dX และ dY จะได้ QUADRANT = 2 และค่าสัมบูรณ์ของ dX น้อยกว่า dY ที่ QUADRANT = 0 จะได้ค่า OCTANT = 4

ขั้นที่ 3 เป็นการหาแกนการวาดอิสระ จากค่า OCTANT = 4 จะได้แกน Y เป็นแกนอิสระและแกน X เป็นแกนไม่อิสระ ดังนั้น

$$dI = dY = -66$$

$$dD = dX = -22$$

ดังนั้นจะได้ค่าพารามิเตอร์ต่างๆดังนี้

$$DC = |dI| = 66$$

$$D = (2 * |dD|) - |dI|$$

$$= (2 * 22) - 66 = -22$$

$$D2 = 2 * (|dD| - |dI|)$$

$$= 2 * (22 - 66) = -88$$

$$D1 = 2 * |dD| = 2 * 22 = 44$$

5.7 การออกแบบโปรแกรมควบคุมระบบ

จากชุดคำสั่งต่างๆดังที่ได้กล่าวมาแล้วจะนำมาเขียนเป็นโปรแกรมที่ใช้ในการควบคุมระบบที่ได้ออกแบบไว้ ซึ่งพารามิเตอร์ต่างๆที่จะโปรแกรมก็ขึ้นอยู่กับฮาร์ดแวร์ที่ออกแบบ

ก่อนอื่นจะต้องทำการ INITIAL GDC ซึ่งจะต้องใช้ชุดคำสั่งหลายชุดคำสั่งโดยมีอัลกอริทึมดังนี้

```
procedure InitialGDC;
begin {InitialGDC}
  RESETCommand;
  VSYNCCommand;
  PITCHCommand;
  PRAMCommand;
  CCHARCommand;
  ZOOMCommand;
  STARTCommand;
end {InitialGDC};
```

จากอัลกอริทึมข้างบนจะอธิบายแยกออกเป็นส่วนๆดังนี้

1.คำสั่ง RESET

- RESET OPCODE = 00
- P1 = MODE BITS

  P1 = 1FH (0 0 0 1 1 1 1 1)

  (0 0 C F I D G S)

  C = 0, G = 1 GRAPHIC MODE

  S = I = 1 INTERLACED

  D = 1 ENABLE DYNAMIC RAM REFRESHING

  F = 1 DRAWING DURING RETRACE BLINKING

- P2 = ACTIVE WORDS PER LINE

  P2 = 2BH = 43 + 2 = 45 WORDS PER LINE

  = 45 x 16 = 720 DOTS PER LINE

- P3 = VSYNC AND HSYNC WIDTHS

  P3 = 65H (0 1 1 0 0 1 0 1)

  VS_L = (0 1 1)

  HS = 5 = 5 + 1 = 6

- P4 = HRPORCH AND VSYNC WIDTHS

  P4 = 0FH (0 0 0 0 1 1 1 1)

  HFP = 3

  VS_H = (1 1)

  VS = VS_H + VS_L = 1 1 0 1 1 = 27

- P5 = HBPORCH WIDTH

  P5 = 03H = 3 + 1 = 4

- P6 = VFPORCH WIDTH

  P6 = 0AH = 10 + 1 = 11

- P7 = ACTIVE LINES PER VIDEO FIELD

  P7 = AL_L = 00H = (0 0 0 0 0 0 0 0)

- P8 = VBPORCH WIDTH AND ACTIVE LINE COUNT

  P8 = 19H (0 0 0 1 1 0 0 1)

  VBP = (0 0 0 1 1 0) = 06H

  AL_H = (0 1)

  AL = (0 1 0 0 0 0 0 0 0 0)

  = 256 = 512 LINES PER VIDEO FIELD

2.คำสั่ง VSYNC

- VSYNC OPCODE + MASTER\SLAVE BIT

- VSYNC OPCODE + MASTER MODE = 6EH

3.คำสั่ง PITCH

- PITCH OPCODE = 47H

- P1 = DISPLAY MEMORY WIDTH

  P1 = 40H = 64 x 16 = 1024 DOTS PER LINE

4.คำสั่ง PRAM

- PRAM OPCODE + STARTING ADDRESS OF 0

  PRAM OPCODE + 00 = 70H

- P1 = DISPLAY STARTING WORD ADDRESS, LOW BYTE

  = 00

- P2 = DISPLAY WINDOW STARTING WORD ADDRESS, HIGH BYTE
  = 00
- P3 = WINDOW LENGTH LOW BITS + STARTING TOP BITS
  = 00
- P4 = MODE BITS + WINDOW LENGTH TOP BITS
  = 10H

5.คำสั่ง CCHAR

- CCHAR OPCODE = 4BH
- P1 = SWEEP LINE PER CHARACTER - 1 = 00
- P2 = C0H
- P3 = 00

6.คำสั่ง ZOOM

- ZOOM OPCODE = 46H
- P1 = DISPLAY & WRITEING ZOOM MAGNIFICATION FACTOR
  = 00

7.คำสั่ง START

- START OPCODE = 6BH

ดังนั้นในวิทยานิพนธ์นี้จะ INITIAL GDC โดยมีชุดคำสั่งและพารามิเตอร์ต่างๆดังนี้

COMMAND	PARAMETER	OPCODE	ADDRESS
RESET		00	301
	P1	1F	300
	P2	2B	300
	P3	65	300
	P4	0F	300
	P5	03	300
	P6	0A	300
	P7	00	300
	P8	19	300

VSYNC		6E	301
PITCH		47	301
	P1	40	300
PRAM		70	301
	P1	00	300
	P2	00	300
	P3	00	300
	P4	10	300
CCHAR		4B	301
	P1	00	300
	P2	C0	300
	P3	00	300
ZOOM		46	301
	P1	00	300
START		6B	301

จากตารางดังกล่าวสามารถนำมาเขียนเป็นอัลกอริทึมได้ดังนี้

```
procedure RESETCommand;
begin
  Port[$301] := RESET.OpCode;
  for i := 0 to 7 do Port[$300] := RESET.P[i];
end;
procedure VSYNCCommand;
begin
  Port[$301] := VSYNC.OpCode;
end;
procedure PITCHCommand;
begin
  Port[$301] := PITCH.OpCode;
  Port[$300] := PITCH.P1;
end;
procedure PRAMCommand;
begin
  Port[$301] := PRAM.OpCode;
  for i := 0 to 3 do Port[$300] := PRAM.P[i];
end;
procedure CCHARCommand;
begin
  Port[$301] := CCHAR.OpCode;
  for i := 0 to 2 do Port[$300] := CCHAR.P[i];
end;
procedure ZOOMCommand (P1 : byte);
begin
  Port[$301] :=  ZOOM.OpCode;
  Port[$300] :=  P1;
end;
```

```
procedure STARTCommand;
begin
  Port[$301] := START.OpCode;
end;
```

นอกจากคำสั่งที่ใช้ในการ INITIAL แล้วคำสั่งอื่นๆที่จำเป็นได้แก่ คำสั่งที่ใช้ในการวาด คำสั่งตรวจสอบ FIFO เป็นต้น การตรวจสอบ FIFO จำเป็นจะต้องทำทุกครั้งก่อนที่จะส่งคำสั่งใหม่เข้าไปเพื่อความแน่ใจว่า GDC ได้อ่านคำสั่งเก่าเข้าไปประมวลผลเรียบร้อยแล้ว มิฉะนั้นแล้วจะทำให้การทำงานของ GDC ผิดพลาดได้ การตรวจสอบ FIFO กระทำได้โดยการตรวจสอบบิทที่ 2 (FIFO EMPTY) ถ้าบิทนี้เป็น 1 แสดงว่า FIFO ว่าง สามารถส่งคำสั่งใหม่ไปให้ GDC ได้ ดังอัลกอริทึม

```
procedure CheckFIFO;
begin
  repeat until FIFOEmpty;
end;
```

นอกจากนั้นก็ยังมีการทำ BLANKING โดยใช้คำสั่ง BCTRL และการเขียนหรือใส่ข้อมูลลงในหน่วยความจำแสดงผลด้วยคำสั่ง WDAT ดังอัลกอริทึม

```
procedure BCTRLCommand;
begin
  Port[$301] :=  BCTRL.OpCode;
end;
procedure WDATCommand;
begin
  Port[$301] := WDAT.OpCode; ประกอบด้วย OpCode + Type + Mode
  Port[$300] := Lo(Data); LOW BYTE ของข้อมูล
  Port[$300] := Hi(Data); HIGH BYTE ของข้อมูล
end;
```

การวาดเส้นจะต้องประกอบด้วยคำสั่งการเลื่อนเคอร์เซอร์มายังตำแหน่งที่ต้องการด้วยคำสั่ง CURS แล้วใช้คำสั่ง FIGS ซึ่งใช้ในการกำหนดพารามิเตอร์ในการวาด และสุดท้ายจึงใช้คำสั่ง FIGD ในการวาดตามพารามิเตอร์ดังกล่าว ซึ่งคำสั่งต่างๆที่กล่าวมานี้มีอัลกอริทึมดังนี้

```
procedure CURSCommand (daD : byte; Address : word);
begin
  Port[$301] := CURS.OpCode;
  Port[$300] := Lo(Address);
  Port[$300] := Hi(Address);
  Port[$300] := daD;
end;
procedure FIGSCommand (P1 : byte; DC,D,D2,D1,DM:integer);
begin
  Port[$301] := FIGS.OpCode;
  Port[$300] := P1; {VECTOR DRAW SETTING}
  Port[$300] := Lo(DC);  {P2}
  Port[$300] := Hi(DC);  {P3}
  Port[$300] := Lo(D) ;  {P4}
  Port[$300] := Hi(D) ;  {P5}
  Port[$300] := Lo(D2);  {P6}
  Port[$300] := Hi(D2);  {P7}
  Port[$300] := Lo(D1);  {P8}
  Port[$300] := Hi(D1);  {P9}
  Port[$300] := Lo(DM);  {P10}
  Port[$300] := Hi(DM);  {P11}
end;
procedure FIGDCommand;
begin
  Port[$301] := FIGD.OpCode;
end;
```

ดังนั้นในการวาดเส้นจะมีอัลกอริทึมดังนี้

```
procedure VectorDrawing (X1,Y1,X2,Y2 : word);
begin {VectorDrawing}
  Vector-Drawing Parameter Calculation;
  AddressConversion;
  CURSCommand;
  FIGSCommand;
  FIGDCommand;
end {VectorDrawing};
```

นอกจากคำสั่งต่างๆที่ได้กล่าวมาแล้วก็ได้พัฒนาคำสั่งอื่นๆซึ่งเป็นการประยุกต์ใช้คำสั่งต่างๆ เหล่านี้โดยนำมาประกอบกันเป็นชุดคำสั่งใหม่ เช่น การเลือกสี การวาดเส้นโดยมีการกำหนดสีได้และการเลื่อนภาพเป็นต้น การเลือกสีและการวาดเส้นโดยสามารถกำหนดสีได้นั้นใช้ชุดคำสั่งของการวาดแบบเวคเตอร์แล้วทำการวาดลงไปในหน่วยความจำของสีต่างๆ โดยใช้หลักการผสมสีของแสดงทำให้สามารถสร้างสีได้ถึง 8 สี จากหน่วยความจำที่แบ่งออกเป็น 3 สี คือ แดง เขียว และน้ำเงิน (RGB) ดังนั้นจะได้สีต่างๆดังนี้

R	G	B	สี
0	0	0	BLACK
0	0	1	BLUE
0	1	0	GREEN
0	1	1	CYAN
1	0	0	RED
1	0	1	MAGENTA
1	1	0	YELLOW
1	1	1	WHITE

เมื่อทำการวาดเส้นนั้นการวาดจะต้องลบสีอื่นๆที่ไม่ต้องการออกด้วย เช่นต้องการวาดเส้นสีแดงก็จะต้องลบเส้นสีเขียวและเส้นสีน้ำเงินในตำแหน่งเดียวกันออกไปด้วย มิฉะนั้นถ้าตำแหน่งที่ต้องการวาดมีเส้นอื่นวาดอยู่แล้วหรือมีการตัดผ่านเส้นจะทำให้สีเกิดการผสมกันขึ้นและกลายเป็นสีอื่นไป และ GDC มีแต่คำสั่งสำหรับการวาดเส้นไม่มีคำสั่งสำหรับลบเส้น แต่ GDC ก็สามารถกำหนดชนิดของเส้นได้ ดังนั้นการวาดจึงแบ่งออกเป็น 2 ชนิดคือ วาดเส้นเต็ม และวาดเส้นเพื่อลบโดยการใส่ค่า 00 ลงในหน่วยความจำแสดงผล ดังอัลกอริทึมต่อไปนี้

```
procedure Line;
begin { Line }
  SetLineType(SolidLine);
  case Color of
    Black : begin { --- }
          SetLineType(HiddenLine);
          R-VectorDrawing;
          G-VectorDrawing;
          B-VectorDrawing;
        end;
    Blue : begin { B-- }
          B-VectorDrawing;
          SetLineType(HiddenLine);
          R-VectorDrawing;
          G-VectorDrawing;
        end;
    Green : begin { -G- }
          G-VectorDrawing;
          SetLineType(HiddenLine);
          R-VectorDrawing;
          B-VectorDrawing;
        end;
    Cyan : begin { BG- }
```

```
            B-VectorDrawing;
            G-VectorDrawing;
            SetLineType(HiddenLine);
            R-VectorDrawing;
          end;
      Red : begin { --R }
            R-VectorDrawing;
            SetLineType(HiddenLine);
            G-VectorDrawing;
            B-VectorDrawing;
          end;
      Magenta : begin { B-R }
            R-VectorDrawing;
            B-VectorDrawing;
            SetLineType(HiddenLine);
            G-VectorDrawing;
          end;
      Yellow : begin { -GR }
            R-VectorDrawing;
            G-VectorDrawing;
            SetLineType(HiddenLine);
            B-VectorDrawing;
          end;
      White : begin { BGR }
            R-VectorDrawing;
            G-VectorDrawing;
            B-VectorDrawing;
          end;
    end;
  end { DrawLine };
```

ส่วนคำสั่ง SetLineType นั้นเป็นการใช้คำสั่ง PRAM นั่นเอง ซึ่งคำสั่งนี้ใช้ในการกำหนด PATTERN ของการวาด เมื่อให้ PATTERN เป็น FFFFH ก็จะได้เส้นทึบ (SOLID LINE) ถ้ากำหนดให้ PATTERN เป็น 0000 ก็จะได้เส้นสีดำ (HIDDEN LINE) เพื่อเป็นการลบเส้นเก่าออก ดังอัลกอริทึม

```
procedure SetLinetype;
begin
  Port[$301] := PRAM.OpCode;
  case LineStyle of
    SolidLine : begin
      Port[$300] := $FF;
      Port[$300] := $FF;
    end;
    HiddenLine : begin
      Port[$300] := $00;
      Port[$300] := $00;
    end;
end {SetLineType};
```

การเลื่อนภาพขึ้นลงและซ้ายขวาก็ใช้คำสั่ง PRAM เช่นเดียวกัน โดยใช้วิธีเปลี่ยนแอดเดรสเริ่มต้นของภาพ การเปลี่ยนแอดเดรสเริ่มต้นของภาพนั้นได้ออกแบบไว้ 2 วิธีคือ การแป้นพิมพ์ลูกศร (ARRAOW KEY) และการใช้คันโยก (JOYSTICK) ทั้ง 2 วิธีนี้จะไปทำการเพิ่มหรือลดค่าแอดเดรสเริ่มต้นเพื่อให้ภาพเลื่อนไปมาได้ตามต้องการ โดยการเลื่อนภาพขึ้นหรือลงนั้นจะต้องทำการเพิ่มหรือลดค่าแอดเดรสครั้งละเท่ากับ LBA หรือจำนวน WORD ใน 1 เส้น (เท่ากับ 1 จุดภาพในแกน Y) ส่วนการเลื่อนไปทางซ้ายและขวาจะมีค่าเท่ากับ 1 บิท (เท่ากับ 1 จุดภาพในแกน X) ส่วนการเลื่อนในแนวเฉียงซ้ายขวาและขึ้นลงก็จะต้องทำการเพิ่มหรือลดค่าในทั้ง 2 แกน ดังอัลกอริทึมการเลื่อนภาพต่อไปนี้

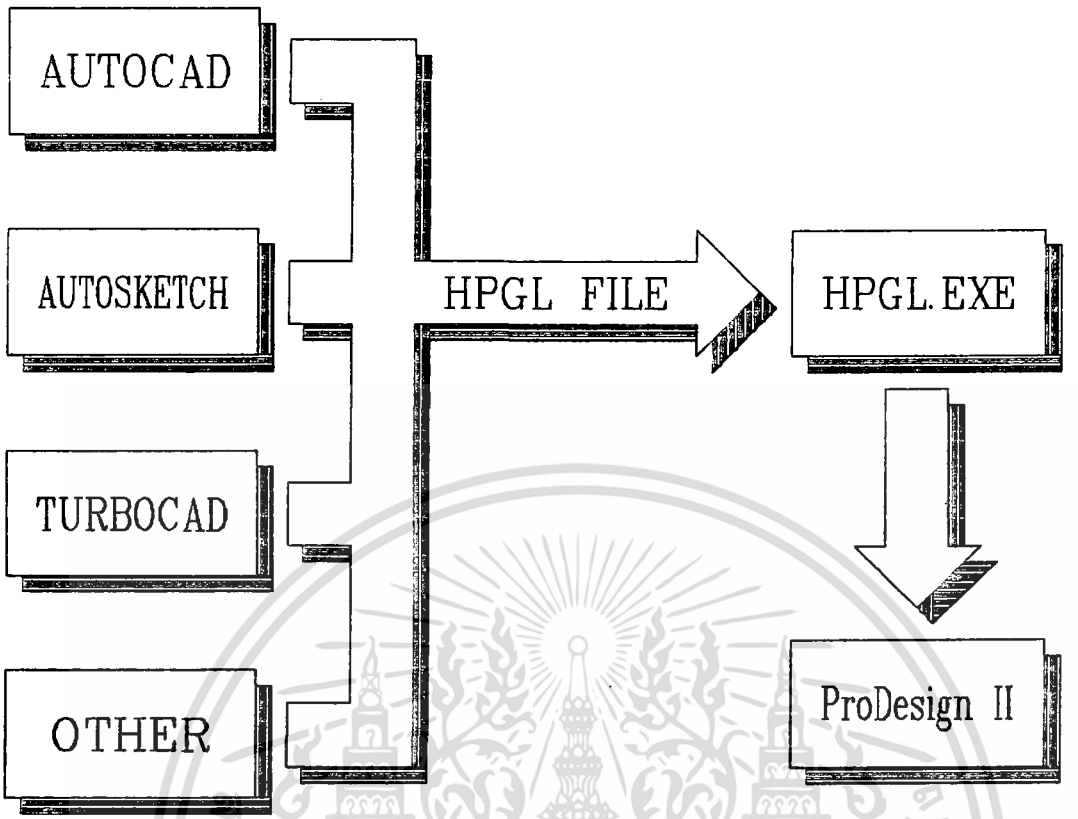
```
procedure Scrolling;
begin
  case Direction of
    Up : Address := Address + NumberOfWordInLine;
    Down  : Address := Address - ;
    Right : Address := Address - 1;
    Left  : Address := Address + 1;
    LeftUp  : begin
                Address := Address + 1;
                Address := Address + NumberOfWordInLine;
              end;
    LeftDown: begin
                Address := Address + 1;
                Address := Address - NumberOfWordInLine;
              end;
    RightUp  : begin
                 Address := Address - 1;
                 Address := Address + NumberOfWordInLine;
               end;
    RightDown: begin
                 Address := Address - 1;
                 Address := Address - NumberOfWordInLine;
               end;
  end;
  Port[$301] := PRAM.OpCode;
  Port[$300] := Lo(Address);
  Port[$300] := Hi(Address);
  Port[$300] := P3;
  Port[$300] := P4;
end;
```

บทที่ 6
การสร้างภาพกระบวนการและการกำหนดอุปกรณ์ในกระบวนการ

การสร้างภาพหรือผังภาพของกระบวนการเพื่อนำมาแสดงผลด้วยระบบนี้นั้น ในการออกแบบได้กำหนดให้ผู้ใช้เป็นผู้สร้างขึ้นเองโดยใช้โปรแกรมสำเร็จรูปต่างๆที่มีอยู่ในท้องตลาด เช่น AutoCAD,ProDesign II,AutoSketch และอื่นๆ วัตถุประสงค์ที่ได้ออกแบบระบบไว้ลักษณะนี้ก็เนื่องจากว่า ในปัจจุบันเทคโนโลยีของคอมพิวเตอร์ช่วยในการออกแบบและวาด (COMPUTER-AIDED DESIGN AND DRAFTING) ได้พัฒนาไปมากประกอบกับโปรแกรมสำเร็จรูปสำหรับงานดังกล่าวก็มีแพร่หลาย และโปรแกรมต่างๆเหล่านี้ใช้งานได้ง่ายและมีประสิทธิภาพสูง ดังนั้นในการออกแบบจึงกำหนดให้ใช้โปรแกรมสำเร็จรูปเหล่านี้ในการสร้างภาพหรือผังภาพของกระบวนการ แต่การนำโปรแกรมสำเร็จรูปเหล่านี้มาใช้ก็มีข้อกำหนดอยู่บ้างคือ

- ต้องสามารถกำหนดชั้น (LAYER) ในการวาดได้
- ต้องมีเอาท์พุทเป็นภาษา HPGL (Hewlett Packard Graphics Language) และต้องสามารถเก็บไฟล์ HPGL ดังกล่าวลงในแผ่นจานแม่เหล็กได้

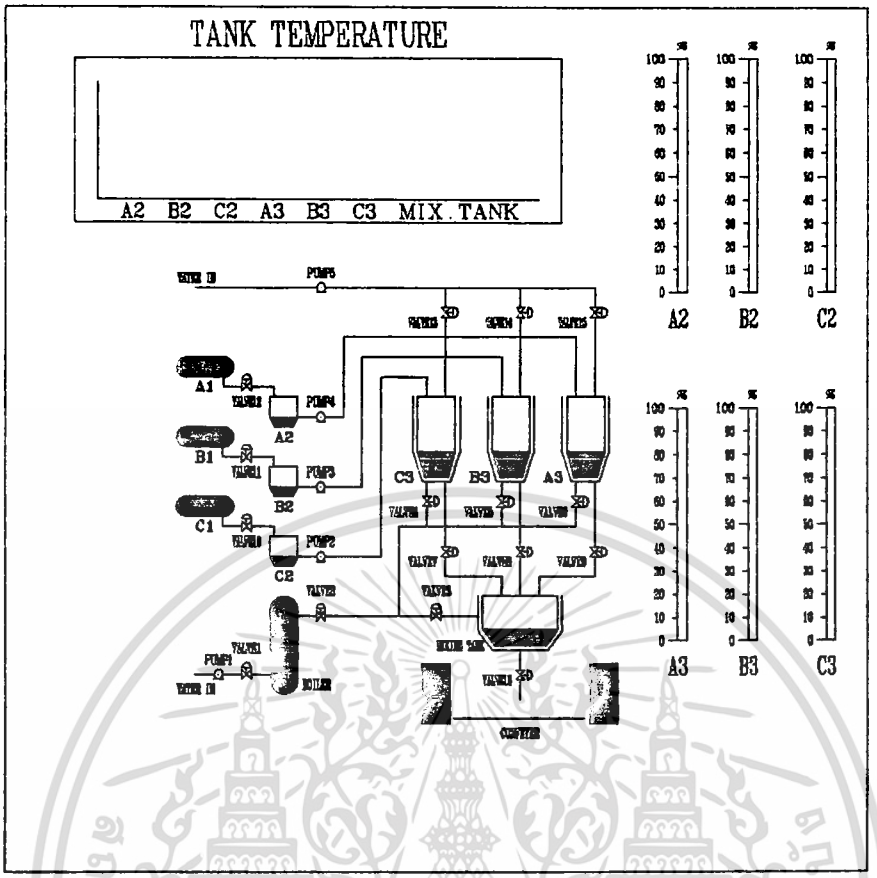
การที่มีข้อกำหนดเหล่านี้ก็เนื่องมาจาก การออกแบบโปรแกรมควบคุมระบบได้กำหนดให้การวาดของส่วนแสดงผลกราฟฟิคความละเอียดสูงเป็นการวาดชนิดวาดเส้นตรงเพียงอย่างเดียวเนื่องจากการวาดเส้นตรงเป็นคำสั่งที่ทำงานได้เร็วที่สุด เส้นโค้งและวงกลมต่างๆก็ใช้เส้นตรงเล็กๆประกอบกับ ซึ่งโปรแกรมสำเร็จรูปที่สามารถกำหนดการวาดให้เป็นลักษณะนี้ได้ คือโปรแกรม ProDesign II ดังนั้นถ้าผู้ใช้ใช้โปรแกรมสำเร็จรูปอื่นในการวาดก็จะต้องใช้ ProDesign II มาเปลี่ยนชุดคำสั่งในการวาดให้เป็นไปตามที่กำหนดเสียก่อน ดังในรูปที่ 6.1



รูปที่ 6.1 แสดงผังภาพของการเชื่อมต่อไฟล์กับ ProDesign II

จากรูปที่ 6.1 จะเห็นได้ว่าการสร้างภาพของกระบวนการด้วยโปรแกรมสำเร็จรูปอื่นๆจะต้องใช้ ProDesign II ในการเปลี่ยนชุดคำสั่งจาก HPGL เป็นชุดคำสั่งที่โปรแกรมควบคุมการแสดงผลกำหนด หรือผู้ใช้อาจจะใช้ ProDesign II ในการสร้างภาพของกระบวนการเลยก็ได้

ขั้นตอนและข้อกำหนดในการสร้างภาพของกระบวนการนั้นมีอยู่หลายข้อ แต่ในการใช้งานจริงแล้วผู้ใช้อาจจะเปลี่ยนแปลงข้อกำหนดต่างๆได้ เนื่องจากระบบมีความยืดหยุ่นในลักษณะนี้สูง แต่ในขั้นตอนหลักจะมีอยู่ 5 ขั้นตอน ส่วนข้อกำหนดต่างๆก็จะได้อธิบายรวมไว้หัวข้อที่เกี่ยวข้องด้วย โดยสมมติให้กระบวนการที่ต้องการมีลักษณะดังรูปที่ 6.2



รูปที่ 6.2 แสดงภาพตัวอย่างของกระบวนการ

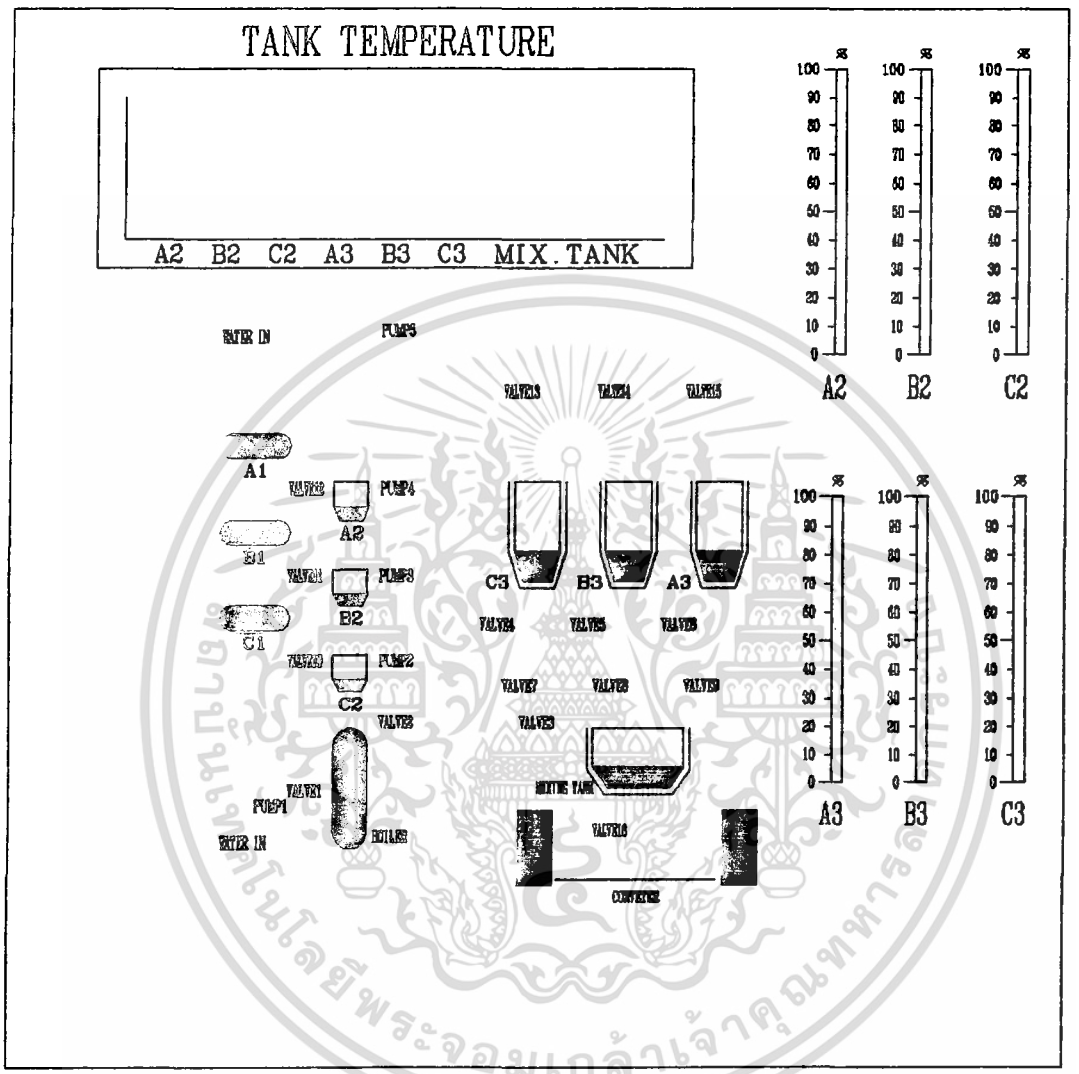
สมมติให้กระบวนการตัวอย่างเป็นการผสมสารเคมีซึ่งเป็นของเหลว 3 ชนิด (สาร-เคมี A,B,C) เข้าด้วยกันแล้วนำไปบรรจุกระป๋อง โดยที่วัตถุดิบของสารเคมี A จะเก็บอยู่ที่ถัง A1 (สารเคมี B ที่ B1 และ C ที่ C1) แล้วนำมาอุ่นให้อุณหภูมิสูงขึ้นที่ถัง A2 (B2,C2) โดยมีวาล์ว12 (11,10) เป็นตัวควบคุม จากนั้นจะต้องนำไปผสมกับน้ำและอุ่นให้อุณหภูมิสูงขึ้นอีกในขณะที่ผสมกับน้ำที่ถัง A3 (B3,C3) หลังจากนั้นก็จะส่งไปยังถังผสม (MIXING TANK) ซึ่งจะอุณหภูมิจะถูกควบคุมให้คงที่ จากนั้นก็เป็นการบรรจุกระป๋องโดยการเปิดวาล์ว16 เมื่อกระป๋องบรรจุเลื่อนเข้ามาถึงตำแหน่งบรรจุ ดังนั้นการสร้างภาพก็จะต้องกำหนดว่าอุปกรณ์ใดจะสร้างอย่างไรตามขั้นตอนต่อไปนี้

ขั้นตอนและข้อกำหนดในการสร้างภาพของกระบวนการ

6.1 การสร้างภาพของกระบวนการ : ในขั้นตอนนี้ผู้ใช้อาจจะยังไม่ต้องสร้างลงในคอมพิวเตอร์ก็ได้ อาจจะใช้วิธีวาดลงในกระดาษเสียก่อน เนื่องจากขั้นตอนนี้เป็นการสร้างภาพของกระบวนการทั้งหมดขึ้นมาเพื่อใช้อ้างอิงในการวาดภายหลังเท่านั้น และนอกเหนือจากภาพของกระบวนการที่จะปรากฏบนจอแสดงผลแล้ว ก็สามารถกำหนดหรือสร้างภาพอื่นๆที่ต้องการแสดงผลลงไปด้วยก็ได้ เช่นกราฟแสดงอุณหภูมิหรือระดับเป็นต้น ส่วนการแสดงข้อความนั้นก็สามารถสร้างลงไปได้ด้วยเช่นกัน แต่สำหรับข้อความที่มีการเปลี่ยนแปลงจะไม่เหมาะสมในการแสดงผลบนจอแสดงผลความละเอียดสูง เนื่องจากส่วนควบคุมการแสดงผลความละเอียดสูงได้ออกแบบสำหรับการแสดงผลแบบกราฟฟิคดังนั้นการแสดงข้อความจะกระทำได้ช้าซึ่งจะทำให้เวลาการตอบสนองรวมของระบบช้าลงด้วย การแสดงข้อความจึงควรจะแสดงบนจอแสดงผลของคอมพิวเตอร์จะแสดงผลได้รวดเร็วกว่า

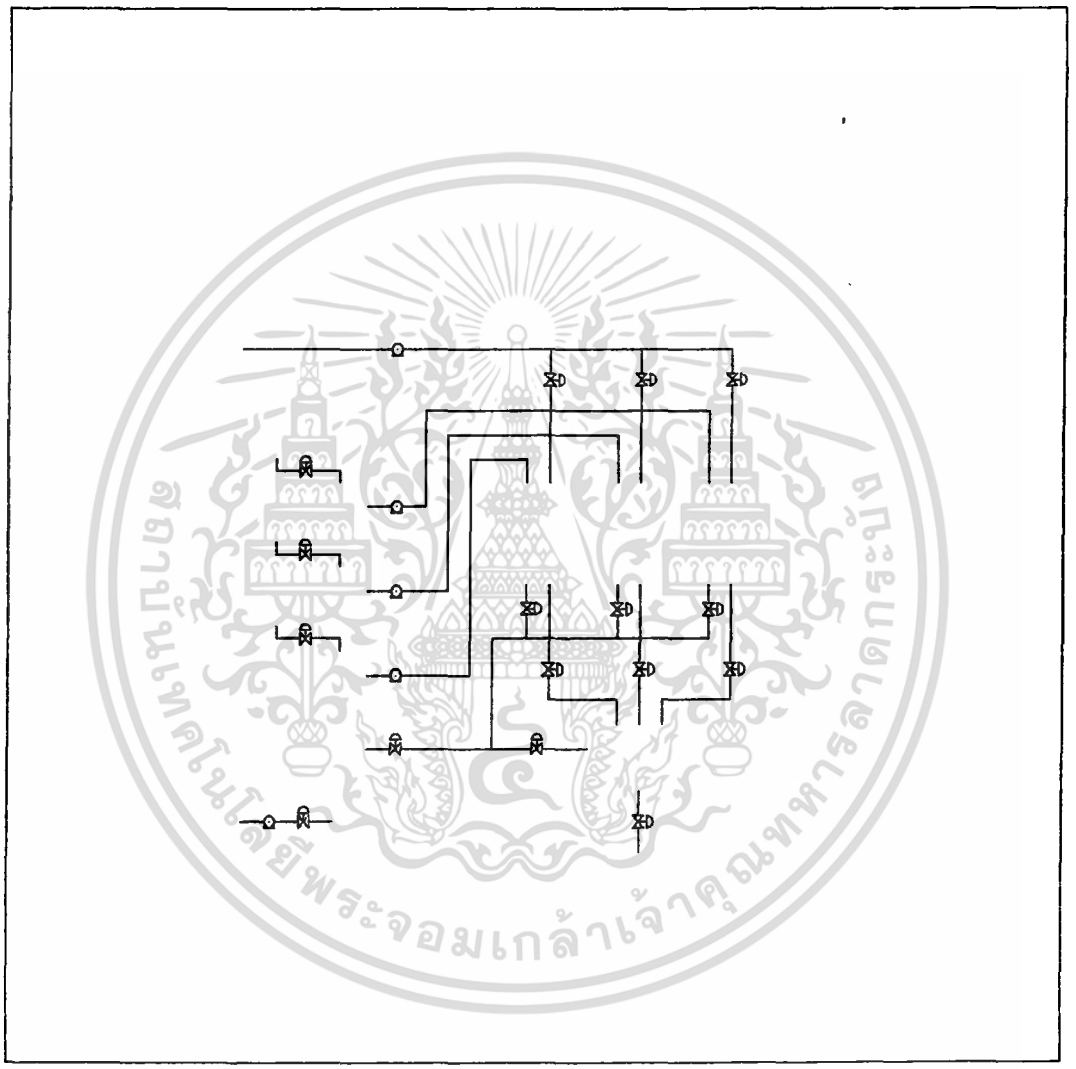
6.2 การแบ่งชั้นของภาพกระบวนการ : การแบ่งชั้นของภาพออกเป็นหลายๆชั้นจะทำให้สะดวกในการตรวจสอบและแก้ไขภาพของกระบวนการโดยเฉพาะกระบวนการที่มีความซับซ้อนสูงๆ หลักการของการแบ่งชั้นก็จะแบ่งตามกลุ่มของอุปกรณ์เป็นหลัก เช่นในตัวอย่างจากรูปที่ 6.2 นั้นก็จะแบ่งอุปกรณ์ออกเป็นกลุ่มได้ดังนี้

กลุ่มที่ 1 อุปกรณ์ที่ไม่มีการเปลี่ยนแปลงสภาวะ เช่น ข้อความหรือตัวอักษรที่ใช้เป็นชื่อของอุปกรณ์ต่างๆ และถัง (TANK) ซึ่งการเปลี่ยนแปลงจะเป็นระดับของของเหลวในถังเท่านั้น ดังแสดงในรูปที่ 6.3



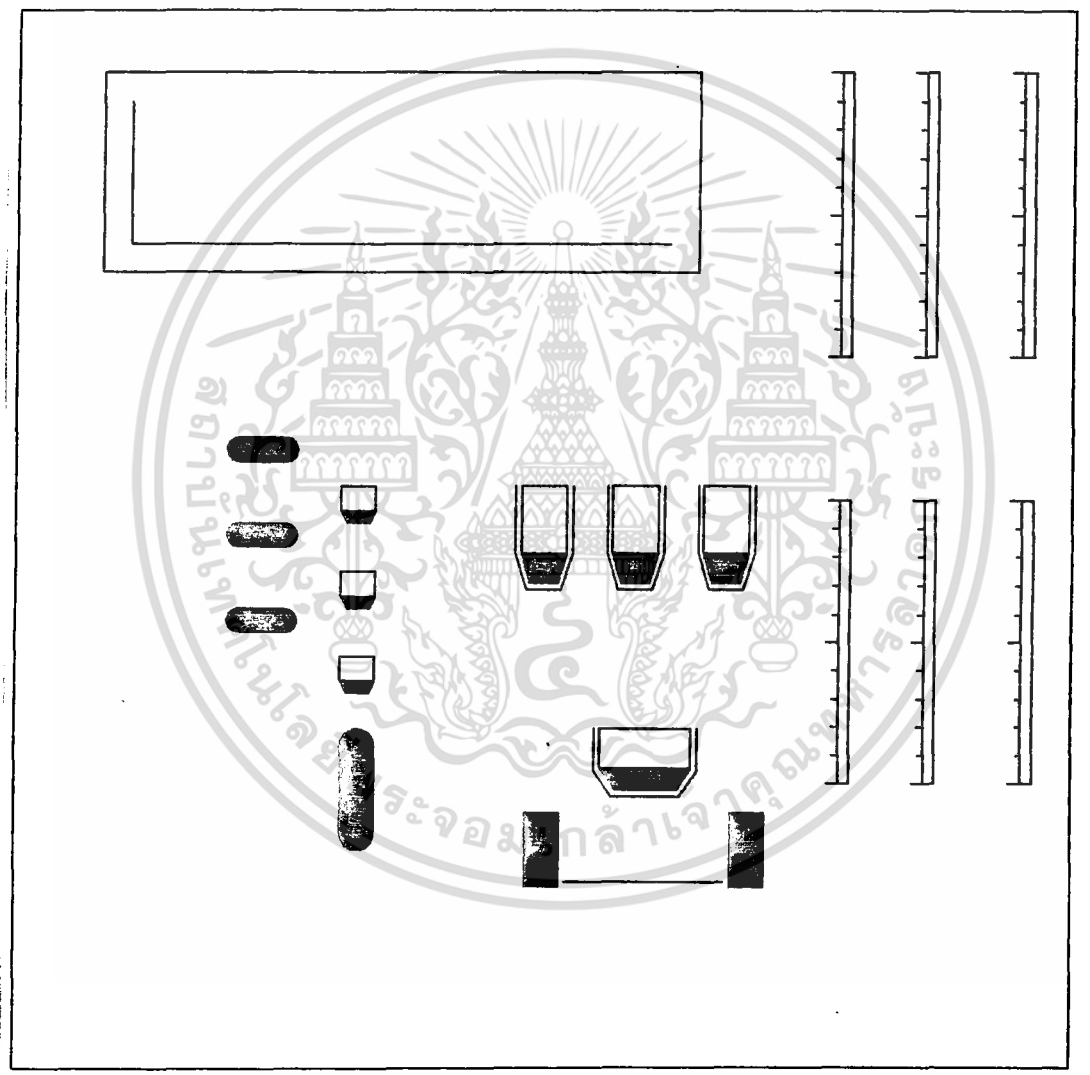
รูปที่ 6.3 แสดงอุปกรณ์ที่ไม่มีการเปลี่ยนแปลงสภาวะ

กลุ่มที่ 2 อุปกรณ์ที่แสดงการเปลี่ยนแปลงสภาวะโดยการเปลี่ยนสี เช่น ท่อ (PIPE) วาล์ว (VALVE) ปั๊ม (PUMP) ซึ่งอุปกรณ์พวกนี้จะมีสภาวะเป็นแบบดิจิตอลคือ ปิดและเปิดเท่านั้น ดังนั้นผู้ใช้จะเป็นผู้กำหนดว่าขณะที่อยู่ในสภาวะปิดจะเป็นสีใด และเมื่ออยู่ในสภาวะเปิดจะเป็นสีใด ซึ่งอุปกรณ์แต่ละตัวก็สามารถกำหนดสีได้เฉพาะตัว เช่น วาล์วตัวที่ 1 ขณะที่ปิดมีสีขาวและเมื่อเปิดจะเปลี่ยนเป็นสีเขียว เป็นต้น ดังแสดงในรูปที่ 6.4



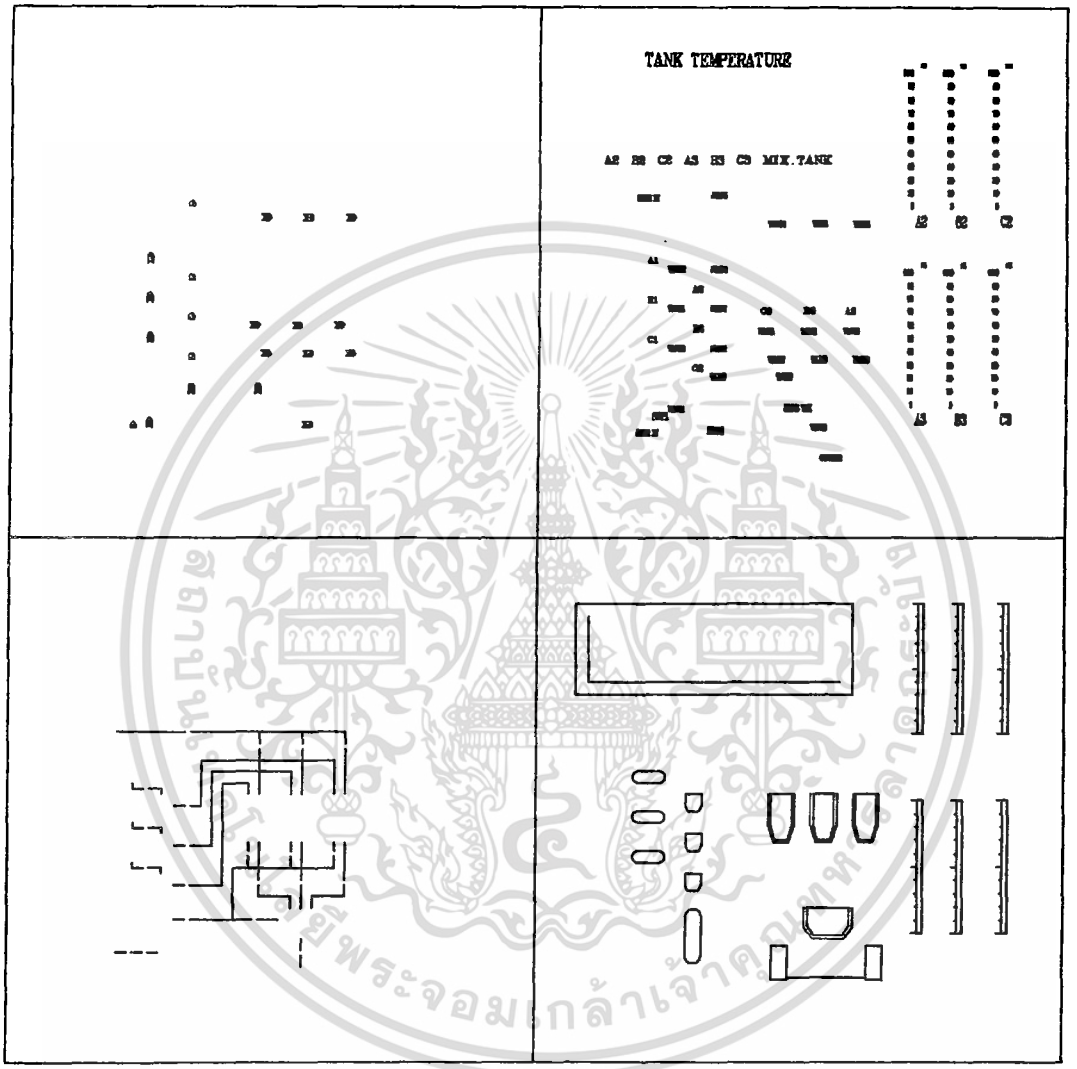
รูปที่ 6.4 แสดงอุปกรณ์ที่แสดงการเปลี่ยนแปลงสภาวะโดยการเปลี่ยนสี

กลุ่มที่ 3 อุปกรณ์ที่แสดงการเปลี่ยนแปลงสภาวะโดยการเปลี่ยนระดับ : เช่น ระดับของเหลวในถัง อุณหภูมิ ความดัน เป็นต้น ซึ่งผู้ใช้อาจจะสร้างให้เป็นภาพของถังและระดับของเหลวในถัง หรือใช้กราฟก็ได้ อุปกรณ์กลุ่มนี้ส่วนใหญ่แล้วจะเป็นการรับค่าสภาวะแบบอนาลอกมาจากกระบวนการ ซึ่งระดับของของเหลวจะเป็นค่าร้อยละ (PERCENTAGE) ของระดับสูงสุดของถัง ถ้าค่าสภาวะที่รับมาแบบดิจิตอลแล้วก็สามารถเปลี่ยนระดับได้เช่นเดียวกันคือกำหนดการเปลี่ยนระดับให้เปลี่ยนเป็นขั้นๆ การแยกอุปกรณ์ในกลุ่มนี้ตามตัวอย่างจะเป็นดังในรูปที่ 6.5



รูปที่ 6.5 แสดงอุปกรณ์ที่แสดงการเปลี่ยนแปลงสภาวะโดยการเปลี่ยนระดับ

จากตัวอย่างในรูปที่ 6.2 จะแบ่งเป็นกลุ่ม โดยกลุ่มทั้ง 3 กลุ่มที่กล่าวมาข้างต้นจะสามารถแบ่งกลุ่มของอุปกรณ์เป็นกลุ่มย่อยๆได้ เพื่อให้สะดวกในการตรวจสอบและแก้ไข เช่น อุปกรณ์ในกลุ่มที่ 2 อาจจะแยกเป็นอุปกรณ์แต่ละชนิด ดังนั้นการแบ่งชั้นจะแบ่งได้ดังในรูปที่ 6.6



รูปที่ 6.6 แสดงการแบ่งอุปกรณ์ออกเป็นชั้น

6.3 **การสร้างภาพลงในคอมพิวเตอร์ :** ขั้นตอนนี้จะเป็นการนำแบบที่ได้ร่างไว้ทั้งหมดนำมาสร้างลงในคอมพิวเตอร์โดยแบ่งออกเป็นชั้นตามที่ได้กำหนดไว้ ส่วนวิธีการสร้างนั้นจะขึ้นอยู่กับโปรแกรมที่นำมาใช้ซึ่งแต่ละโปรแกรมจะมีคำสั่งและการใช้งานแตกต่างกันออกไป แต่ในการสร้างจำเป็นจะต้องสร้างตามเงื่อนไขที่กำหนด คือการแบ่งแยกอุปกรณ์แต่ละตัวออกจากกัน เนื่องจากภาพที่สร้างแล้วอุปกรณ์แต่ละตัวโดยเฉพาะอุปกรณ์ที่แสดงการเปลี่ยนแปลงสภาวะโดยการเปลี่ยนสีจะถูกเก็บเป็นชุดคำสั่งในการวาด ซึ่งประกอบด้วยตำแหน่งของจุดต่างๆโดยอุปกรณ์แต่ละตัวอาจจะมีความยาวของคำสั่งไม่เท่ากัน ส่วนอุปกรณ์ที่แสดงการเปลี่ยนแปลงสภาวะโดยการเปลี่ยนระดับนั้นจะมีความยาวของชุดคำสั่งเท่ากันซึ่งจะมีวิธีแตกต่างกันออกไปโดยจะกล่าวถึงในภายหลัง ดังนั้นการที่จะแยกอุปกรณ์ชนิดแรกแต่ละตัวออกจากกันก็จะต้องมีคำสั่งหรือตำแหน่งกำหนดไว้ในการวาดอุปกรณ์แต่ละตัว ในวิทยานิพนธ์ฉบับนี้ใช้วิธีกำหนดจุดก่อนที่จะทำการวาดอุปกรณ์ทุกครั้ง เนื่องจากได้กำหนดให้คำสั่งในการวาดเป็นคำสั่งวาดเส้นตรงทั้งหมดดังนั้นการวาดจุดลงในภาพจะทำให้เกิดชุดคำสั่งที่มีลักษณะต่อไปนี้

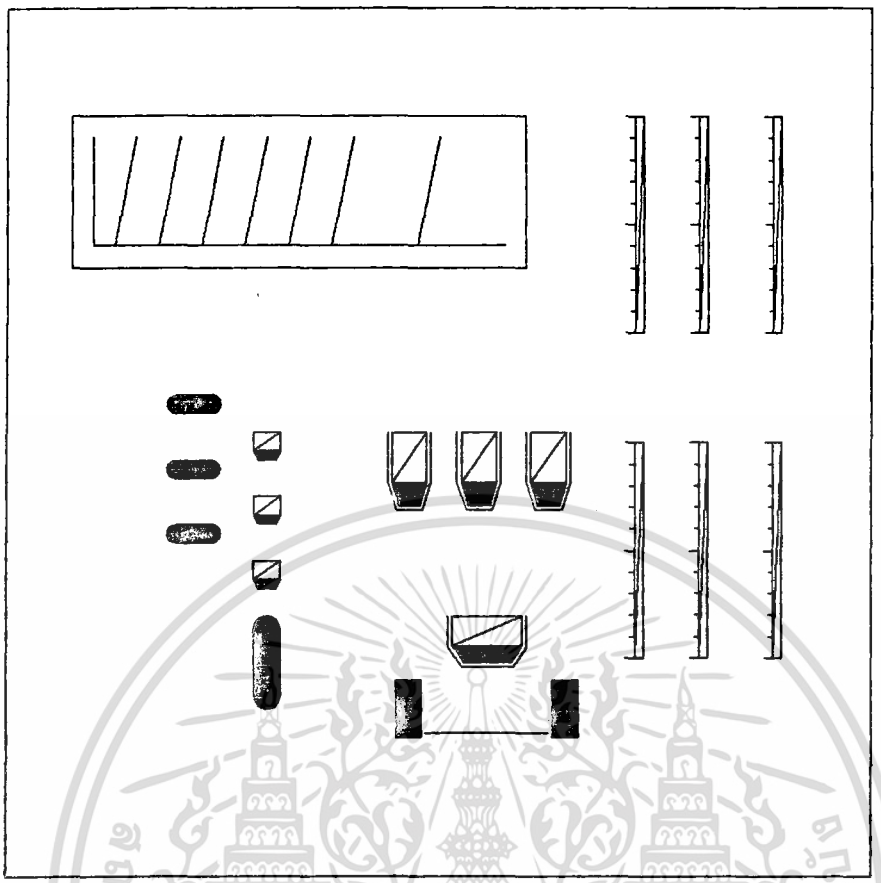
```
MOVE X,Y
LINE X,Y
```

ซึ่งค่าตำแหน่ง (COORDINATE) ของทั้งสองคำสั่งที่อยู่ติดกันจะมีค่าเดียวกัน ดังนั้นจะใช้ชุดคำสั่งดังกล่าวนี้เป็นจุดตรวจสอบสำหรับการแบ่งแยกอุปกรณ์ ดังตัวอย่างเช่นการวาดท่อตามตัวอย่างจากรูปที่ 6.6 เมื่อถูกแปลเป็นชุดคำสั่งในการวาดแล้วจะได้ดังนี้

```
MOVE 223 230   <-------------- เป็นการสร้างจุดแสดงว่าเป็นจุดเริ่มของอุปกรณ์
SETC 2 <---- คำสั่งนี้ไม่มีความจำเป็นจะถูกตัดออกในภายหลัง
LINE 223 230   <-------
LINE 245 230
MOVE 255 230   <---------- จุดเริ่มต้นของอุปกรณ์ตัวต่อมา
LINE 255 230   <------
LINE 277 230
MOVE 287 230   <----------
LINE 287 230   <-----
LINE 309 230
MOVE 341 299   <-----------
```

```
LINE 341 299    <----┘
LINE 363 299
     .
     .
LINE 513 643
LINE 513 551
MOVE 599 643    <---------
LINE 599 643    <----┘
LINE 599 551
MOVE 685 643    <---------
LINE 685 643    <----┘
LINE 685 551
MOVE 1 1        <---------  เป็นจุดสิ้นสุดของชุดคำสั่ง
SETC 0          <----┘
```

จากตัวอย่างที่แสดงไว้ข้างต้นจะ เห็นได้ว่าวิธีการค่อนข้างยุ่งยากและสับสนในการสร้างซึ่งผู้ใช้อาจจะกำหนดวิธีการขึ้น เองก็ได้แต่สำหรับในวิทยานิพนธ์ฉบับนี้ได้ใช้วิธีนี้ เป็นหลัก และการแบ่งแยกอุปกรณ์จากชุดคำสั่งในการวาดนั้นก็อาจจะใช้วิธีแยก เองโดยใช้ข้อกำหนดดังกล่าวข้างต้น หรืออาจจะใช้โปรแกรม CNVRTDEV.EXE ซึ่งได้พัฒนาขึ้นสำหรับการแบ่งแยกอุปกรณ์ที่ใช้วิธีแยกด้วยจุด ส่วนอุปกรณ์ที่แสดงการ เปลี่ยนแปลงสภาวะด้วยการเปลี่ยนระดับนั้นค่าที่กำหนดจะมี เพียงจุดสูงสุดและต่ำสุด เท่านั้นดังนั้นในการกำหนดระดับการเปลี่ยนแปลงของถังหนึ่งถังก็จะ เป็น เพียง เส้นทะแยงมุม เพียงหนึ่ง เส้น เท่านั้น ดังรูปที่ 6.7



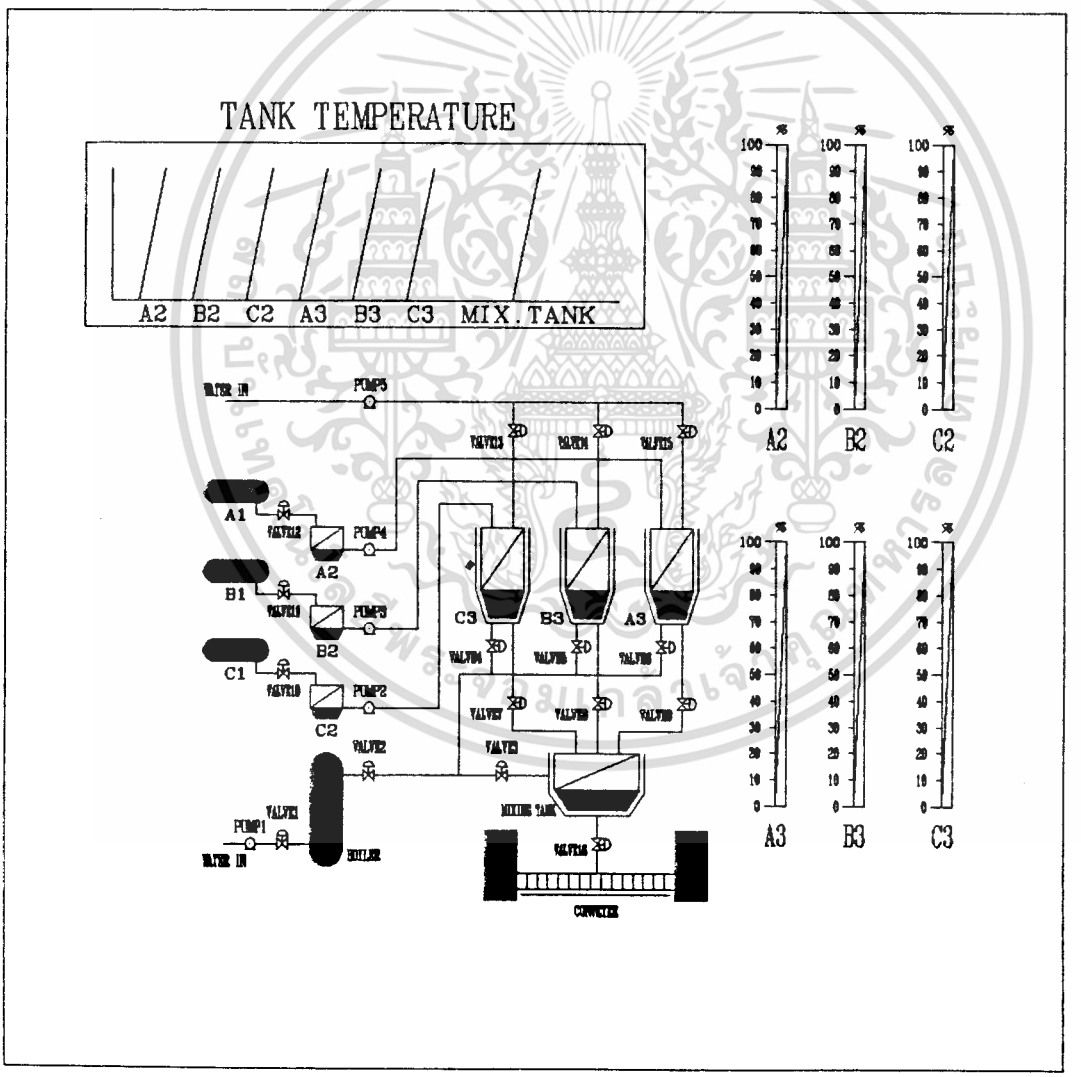
รูปที่ 6.7 แสดงการกำหนดระดับการเปลี่ยนแปลงของสภาวะ

การแยกอุปกรณ์ประเภทนี้ก็ใช้วิธีแยกทีละเส้นหรือ 2 คำสั่ง เนื่องจากอุปกรณ์แต่ละตัวจะประกอบด้วยเส้นตรงเพียงเส้นเดียว ดังในตัวอย่างจะได้ชุดคำสั่งดังนี้

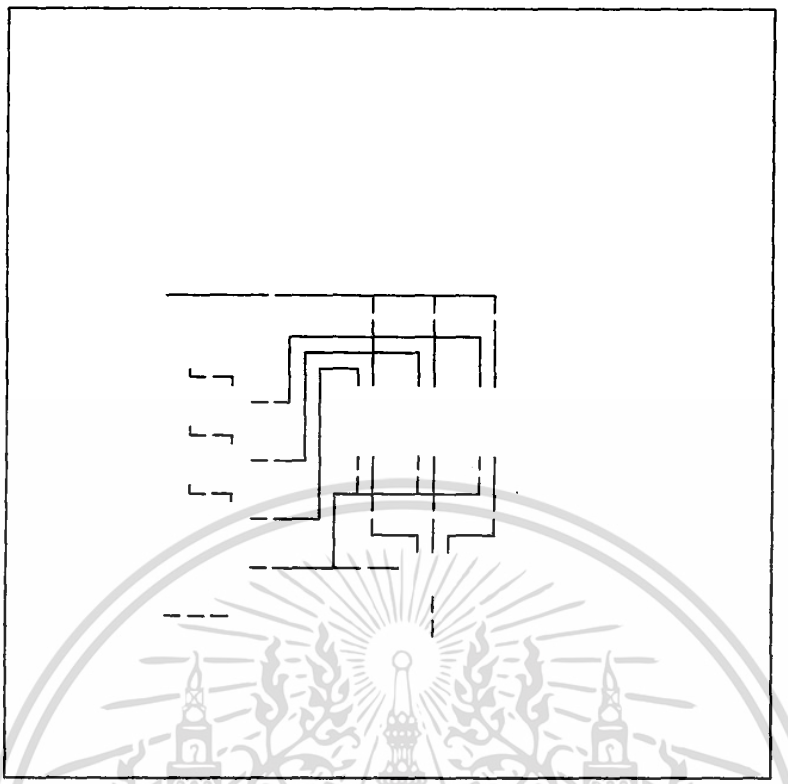
```
MOVE 309 529    <---------- เส้นหนึ่งเส้นคือระดับของอุปกรณ์หนึ่งตัว
SETC 1
LINE 341 551    <-----
MOVE 309 448    <---------- อุปกรณ์ตัวที่สอง
LINE 341 471    <-----
MOVE 309 368
     .
     .
MOVE 404 780
LINE 431 914
```

```
MOVE 512 780    <----------- อุปกรณ์ตัวสุดท้าย
LINE 539 914    <-----┘
MOVE 1 1
SETC 0
```

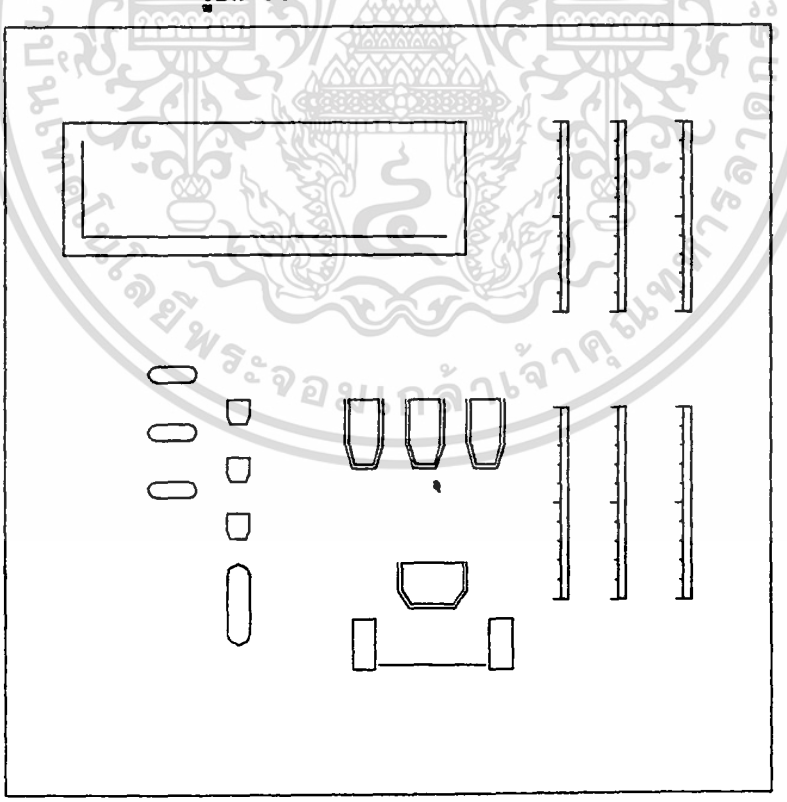
จะเห็นได้ว่าการแยกอุปกรณ์ประเภทนี้ทำได้ง่าย หรืออาจจะใช้โปรแกรม CNVRTLEV.EXE ในการแยกก็ได้ ส่วนวิธีการใช้งานโปรแกรม CNVRTDEV.EXE และ CNVRTLEV.EXE นั้นจะได้อธิบายโดยละเอียดต่อไป ดังนั้นหลังจากกระทำขั้นตอนที่ 3 เสร็จเรียบร้อยแล้วก็จะได้ภาพในชั้นต่างๆทั้งหมดดังในรูปที่ 6.8 ถึง 6.16



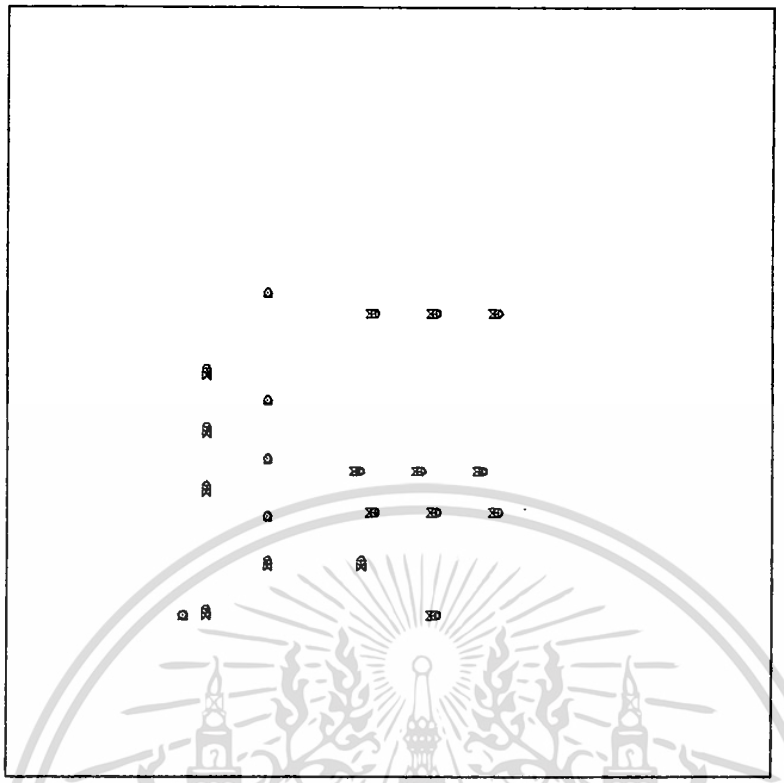
รูปที่ 6.8 แสดงภาพกระบวนการทั้งหมดซึ่งนำแต่ละชั้นมาทับซ้อนกัน



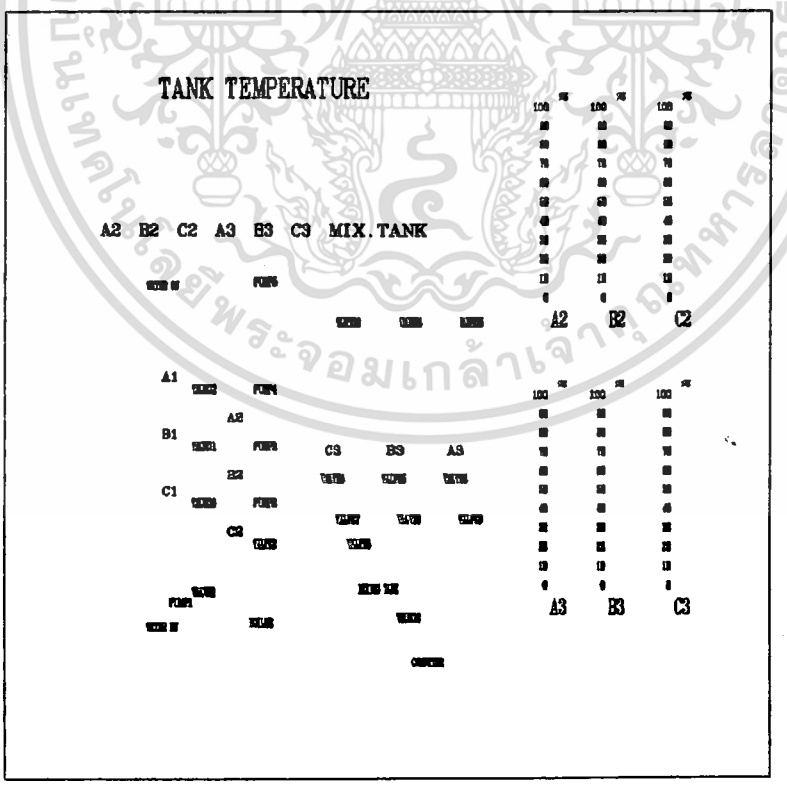
รูปที่ 6.9 แสดงภาพของชั้นที่ 1



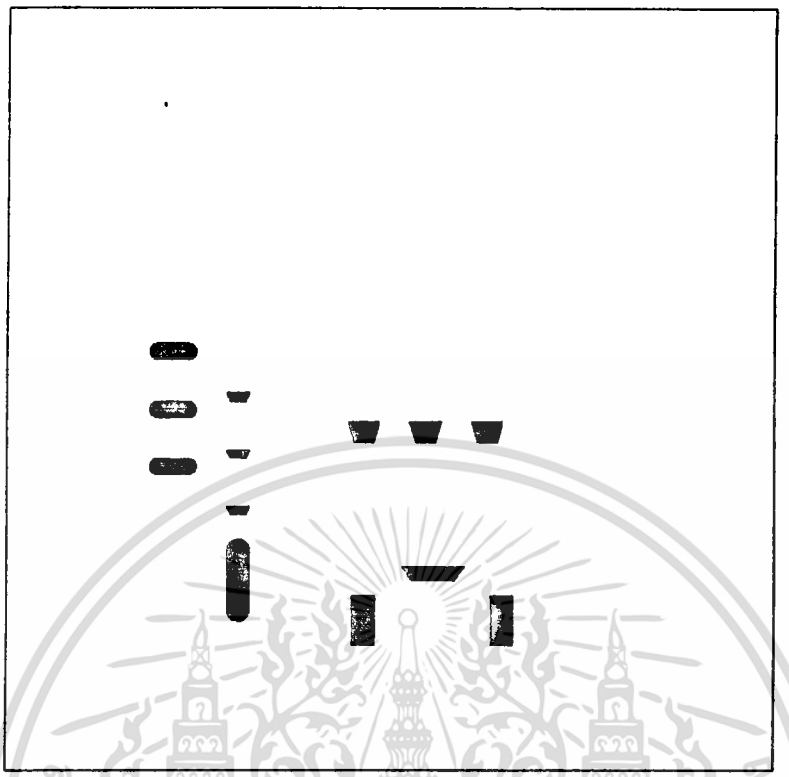
รูปที่ 6.10 แสดงภาพของชั้นที่ 2



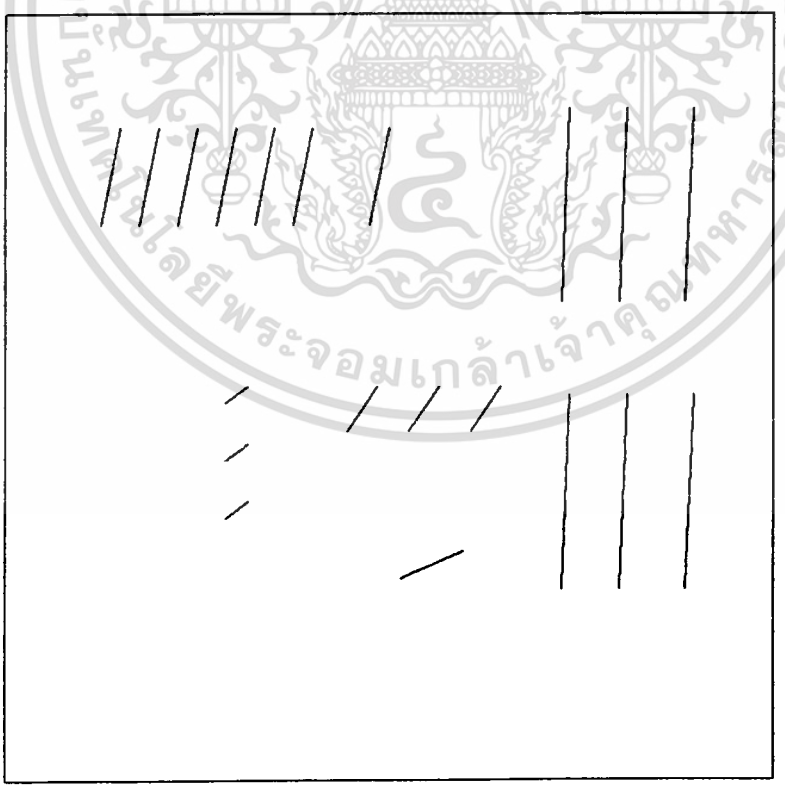
รูปที่ 6.11 แสดงภาพของชั้นที่ 3



รูปที่ 6.12 แสดงภาพของชั้นที่ 4



รูปที่ 6.13 แสดงภาพของชั้นที่ 5



รูปที่ 6.14 แสดงภาพของชั้นที่ 6

รูปที่ 6.15 แสดงภาพของชั้นที่ 7

รูปที่ 6.16 แสดงภาพของชั้นที่ 8

6.4 การสร้างชุดคำสั่งสำหรับระบบแสดงผล : หลังจากที่ได้สร้างภาพและเก็บภาพลงไฟล์แล้วขั้นตอนต่อมาคือเปลี่ยนภาพดังกล่าวให้เป็นชุดคำสั่งของระบบแสดงผล จากที่ได้กล่าวไว้ในตอนต้นของบทนี้แล้วว่าผู้ใช้จะใช้โปรแกรมใดในการสร้างภาพก็ได้แต่โปรแกรมนั้นจะต้องสามารถให้เอาท์พุทเป็น HPGL ได้ หรือมิฉะนั้นก็ต้องใช้โปรแกรม ProDesign II ในการสร้างภาพเลย ดังนั้นจะอธิบายแยกออกเป็น 2 ส่วนคือ การแปลงชุดคำสั่งที่สร้างด้วยโปรแกรมอื่น และการแปลงชุดคำสั่งที่สร้างด้วยโปรแกรม ProDesign II

6.4.1 การแปลงชุดคำสั่งที่สร้างด้วยโปรแกรม ProDesign II : จากภาพที่สร้างด้วย ProDesign II นั้นจะทำการแปลงให้เป็นชุดคำสั่งของระบบแสดงผลด้วยการใช้คำสั่ง PLOT หรือ [F6] ซึ่งจะเป็นการวาดภาพที่สร้างไว้ด้วยพล็อตเตอร์ (PLOTTER) แต่จะต้องกำหนดให้เป็นการวาดลงในไฟล์เพื่อที่จะเก็บชุดคำสั่งการวาดไว้ในจานแม่เหล็ก และเลือกชนิดของพล็อตเตอร์เป็นชนิดที่ผู้ใช้กำหนดคำสั่งได้เอง การกำหนดเหล่านี้กระทำได้โดยใช้ไฟล์ PDSETUP.EXE ซึ่งจะต้องกระทำก่อนที่จะเรียกโปรแกรม ProDesign II การเรียกไฟล์ PDSETUP.EXE กระทำได้โดยการพิมพ์ดังนี้

A>PDSETUP [Enter]

จอแสดงผลจะแสดงรายการการตั้งค่าพารามิเตอร์ต่างๆ ให้ทำการเลือกอุปกรณ์อื่นๆ ตามที่ต้องการ ยกเว้นพล็อตเตอร์ให้กำหนดดังนี้

- ชนิดของพล็อตเตอร์ให้เลือกหมายเลข 53 หรือที่โปรแกรมแสดงผลทางจอแสดงผลว่า

53 - A PLOTTER OTHER THAN THESE

- จอแสดงผลจะแสดงผลดังนี้

```
PLOTTER NAME:
SEPARATOR:
TERMINATOR:
NUMBER OF PENS:
X MINIMUM:
X MAXIMUM:
Y MINIMUM:
Y MAXIMUM:
MOVE COMMAND:
DRAW COMMAND:
SETUP COMMAND:
PEN CHANGE COMMAND:
RESOLUTION (DOT/INCH):
```

ความหมายของหัวข้อต่างๆมีดังนี้

- PLOTTER NAME : ชื่อของพล็อตเตอร์ใช้สำหรับอ้างอิงเท่านั้นไม่มีความหมาย
- SEPARATOR : เครื่องหมายแยกระหว่างข้อมูลตำแหน่งของ X และ Y
- TERMINATOR : เครื่องหมายแสดงการจบคำสั่ง 1 คำสั่ง เช่น ';'
- NUMBER OF PENS : จำนวนปากกาพล็อตเตอร์ หรือจำนวนสี
- X MINIMUM : ค่าเริ่มต้นของตำแหน่ง X
- X MAXIMUM : ค่าสุดท้ายของตำแหน่ง X
- Y MINIMUM : ค่าเริ่มต้นของตำแหน่ง Y
- Y MAXIMUM : ค่าสุดท้ายของตำแหน่ง Y
- MOVE COMMAND : คำสั่งในการเปลี่ยนตำแหน่งที่จะวาด
- DRAW COMMAND : คำสั่งในการวาด
- SETUP COMMAND : คำสั่งสำหรับการจัดการพล็อตเตอร์ก่อนการวาด
- PEN CHANGE COMMAND : คำสั่งเปลี่ยนปากกา หรือเปลี่ยนสี
- RESOLUTION (DOT/INCH) : ความละเอียดเป็นจุดต่อนิ้ว

ดังนั้นจากหัวข้อต่างๆ เหล่านี้จะทำการป้อนข้อมูล เพื่อให้ชุดคำสั่งการวาด เหมาะสมกับระบบแสดงผล ดังนี้

```
PLOTTER NAME:           HIGRAD
SEPARATOR:              ,
TERMINATOR:
NUMBER OF PENS:         7
X MINIMUM:              1
X MAXIMUM:              1000
Y MINIMUM:              1
Y MAXIMUM:              1000
MOVE COMMAND:           MOVE
DRAW COMMAND:           LINE
SETUP COMMAND:
PEN CHANGE COMMAND:     SETC
RESOLUTION (DOT/INCH): 100
```

- จากนั้นให้ เก็บการตั้งค่า เหล่านี้ลงในไฟล์ด้วยการกด [F1]

- จอแสดงผลจะแสดงอุปกรณ์ เอาท์พุทให้ เลือก ให้ เลือกอุปกรณ์ เอาท์พุท เป็นไฟล์หรือหมายเลข 5

- ออกจากโปรแกรม SETUP.EXE แล้วเรียกโปรแกรม ProDesign II ด้วยการพิมพ์

A>PD [Enter]

- เรียกข้อมูลภาพจากไฟล์โดยใช้คำสั่ง LOAD หรือ RETRIEVE [F9]

- ทำการแปลงให้ เป็นชุดคำสั่งของระบบแสดงผลด้วยการใช้คำสั่ง PLOT หรือ [F6] ซึ่งจะเป็นการวาดภาพที่สร้างไว้ด้วยพล็อตเตอร์ (PLOTTER) การใช้คำสั่งดังกล่าวให้แยกใช้ เป็นชั้นๆ โดยกำหนดให้ข้อมูลกลุ่มที่ 1 อยู่รวมกันทั้งหมดคือชั้นที่ 2,4,5 และ 7 ซึ่งเป็นอุปกรณ์ที่ไม่มีการ เปลี่ยนแปลงสภาวะ ส่วนชั้นอื่นๆคือ ชั้นที่ 1,3,6,8 จะแยกออกเป็นไฟล์ ดังนั้นจะมีไฟล์ชุดคำสั่งที่ถูกแปลงแล้วจำนวน 5 ไฟล์คือ

ไฟล์ที่ 1 เป็นไฟล์ชุดคำสั่งของอุปกรณ์ที่ไม่มีการเคลื่อนไหว

ไฟล์ที่ 2 เป็นไฟล์ชุดคำสั่งของท่อ (PIPE)

ไฟล์ที่ 3 เป็นไฟล์ชุดคำสั่งของวาล์ว (VALVE)

ไฟล์ที่ 4 เป็นไฟล์ชุดคำสั่งของระดับ (LEVEL)

ไฟล์ที่ 5 เป็นไฟล์ชุดคำสั่งของกระป๋อง (CAN)

6.4.2 การแปลงชุดคำสั่งที่สร้างด้วยโปรแกรมอื่นที่ไม่ใช่ ProDesign II : จากภาพที่สร้างไว้ด้วยโปรแกรมใดๆก็ตาม เมื่อสร้างเสร็จแล้วสั่งวาด (PLOT) ภาพนั้นโดยแยกการวาดออกเป็นชั้นๆตามที่กำหนด โดยกำหนดให้เอาท์พุทเป็น HPGL และเก็บลงไฟล์ หลังจากนั้นนำไฟล์ HPGL มาเปลี่ยนให้เป็นไฟล์ภาพของ ProDesign II ด้วยโปรแกรม HPGL.EXE หลังจากที่เปลี่ยนเป็นไฟล์ภาพของ ProDesign II แล้ว ก็ให้กลับไปทำตามขั้นตอนที่ 4.1 ต่อไป เนื่องจากไฟล์ภาพถูกเปลี่ยนให้เป็นไฟล์ภาพของ ProDesign II แล้ว

6.5 การแยกอุปกรณ์ในไฟล์ชุดคำสั่งการวาด : ขั้นตอนนี้จะเป็นการนำไฟล์ชุดคำสั่งการวาดภาพกระบวนการมาประมวลผลเพื่อแยกอุปกรณ์แต่ละตัวออกจากกันโดยใช้โปรแกรม CNVRTDEV.EXE และ CNVRTLEV.EXE

โปรแกรมทั้งสองจะทำการอ่านข้อมูลในไฟล์ซึ่งเก็บชุดคำสั่งการวาดอุปกรณ์ต่างๆแล้วแยกออกเป็นอุปกรณ์แต่ละตัวและกำหนดชื่อของอุปกรณ์ด้วย โดยชื่อของอุปกรณ์ผู้ใช้จะกำหนดขึ้นเองซึ่งจะต้องเรียงตามลำดับการวาดด้วยแล้วจึงเก็บข้อมูลชื่อนี้ไว้ในไฟล์ และจะต้องตั้งชนิดของไฟล์ (EXTENSION) เป็น .NDE ดังตัวอย่างไฟล์ PIPE.NDE ซึ่งเป็นชื่อของท่อต่างๆ

Pipe1

Pipe2

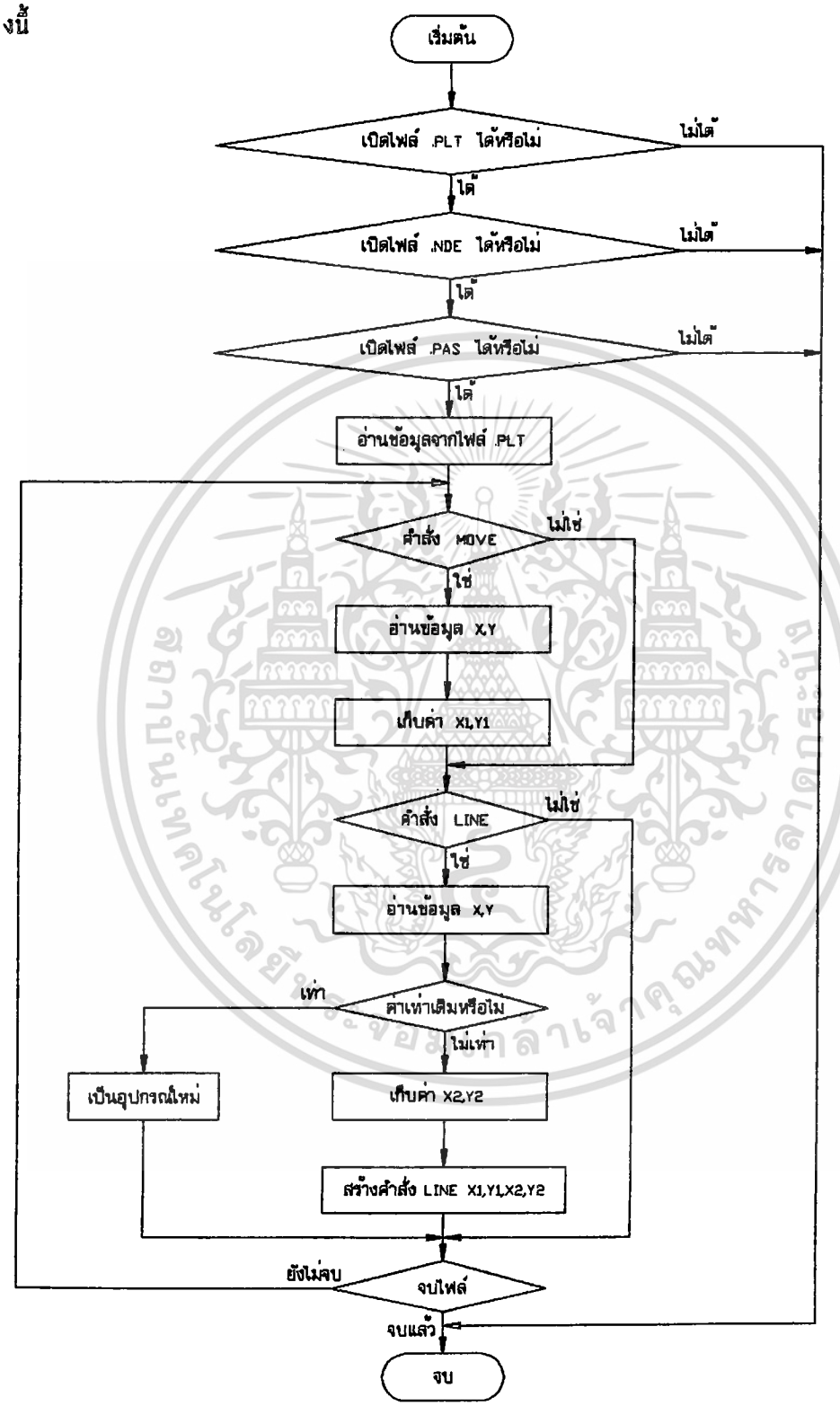
.

.

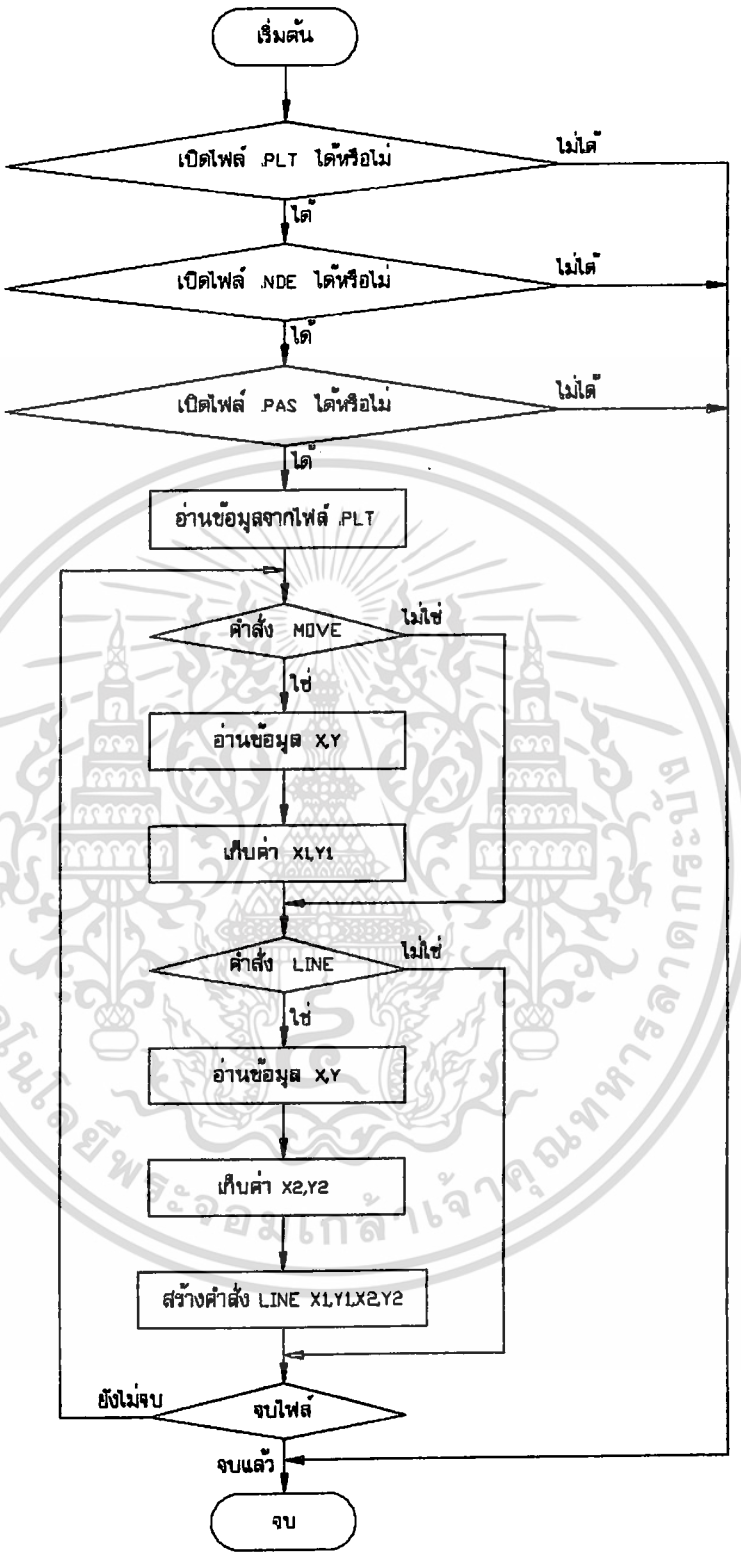
Pipe33

Pipe34

การทำงานของโปรแกรมทั้งสองจะสามารถเขียนเป็นโฟลว์ชาร์ท (FLOWCHART) ได้ดังนี้



รูปที่ 6.17 โฟลว์ชาร์ทของโปรแกรม CNVRTDEV.EXE

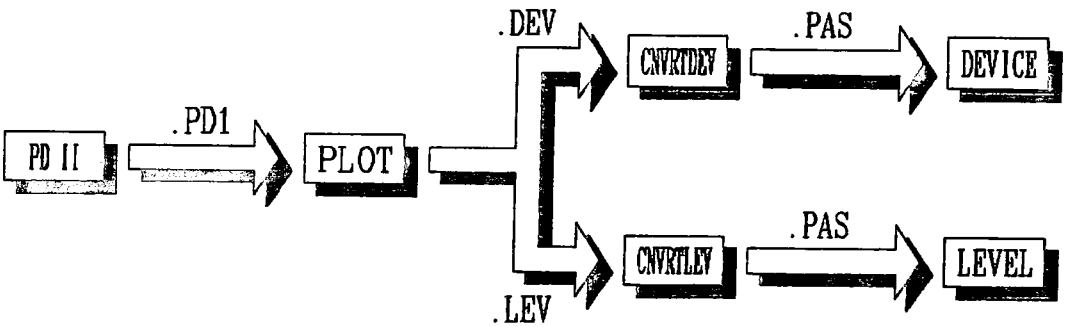


รูปที่ 6.18 โฟลว์ชาร์ทของโปรแกรม CNVRTLEV.EXE

เมื่อใช้โปรแกรมทั้งสองนี้แล้วชุดคำสั่งในการสร้างภาพกระบวนการ จะถูกเปลี่ยนเป็นโปรแกรมภาษาปาสคาลดังตัวอย่างจากการสร้างภาพท่อในกระบวนการ

```
unit Pipe;
interface
uses Grph7220;
procedure Pipe1(NodeColor : byte);
var Pipe1Status : boolean;
procedure Pipe2(NodeColor : byte);
var Pipe2Status : boolean;
          .
          .
procedure Pipe34(NodeColor : byte);
var Pipe34Status : boolean;
implementation
procedure Pipe1(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(223,770,245,770);
end;
          .
          .
procedure Pipe34(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(685,357,685,449);
end;
begin
end.
```

การสร้างภาพทั้งหมดที่กล่าวมาข้างต้นสามารถเขียนเป็นผังภาพ (BLOCK DIAGRAM) ของการส่งผ่านข้อมูลในขั้นตอนต่างๆได้ดังรูปที่ 6.19



รูปที่ 6.19 แสดงการส่งผ่านข้อมูลในขั้นตอนต่างๆ

ตัวอย่างการสร้างภาพกระบวนการและการกำหนดอุปกรณ์ในกระบวนการ

ในตัวอย่างนี้จะใช้โปรแกรม ProDesign II ในการอธิบายส่วนการใช้โปรแกรมอื่นๆ ก็จะมีลักษณะเดียวกันแตกต่างกันที่รายละเอียดของคำสั่งเท่านั้น

1.ร่างแบบของกระบวนการ สมมติให้ใช้แบบในรูปที่ 6.2

2.การแบ่งชั้นของภาพกระบวนการ ก็จะแบ่งตามตัวอย่างข้างบนคือแบ่งเป็น 8 ชั้น ชื่อในวงเล็บภาษาอังกฤษคือชื่อที่ใช้เรียกในการสร้างภาพกระบวนการ

- ชั้นที่ 1 เป็นชั้นของท่อ (PIPE)
- ชั้นที่ 2 เป็นชั้นของถัง (TANK)
- ชั้นที่ 3 เป็นชั้นของวาล์วและปั๊ม (VALVE)
- ชั้นที่ 4 เป็นตัวอักษรและข้อความ (TEXT)
- ชั้นที่ 5 เป็นสีที่ใช้ในการระบายเพื่อความสวยงาม (PAINT)
- ชั้นที่ 6 เป็นการกำหนดระดับในถัง (LEVEL)
- ชั้นที่ 7 เป็นขอบของรูป (BORDER)
- ชั้นที่ 8 เป็นการสร้างภาพกระป๋อง (CAN)

3.การสร้างภาพด้วย ProDesign II จะอธิบายเป็นขั้นๆ ดังนี้

3.1 เปิดเครื่องคอมพิวเตอร์ เมื่อเข้าสู่ DOS แล้ว ให้ใส่แผ่นโปรแกรม ProDesgin II แล้วกด

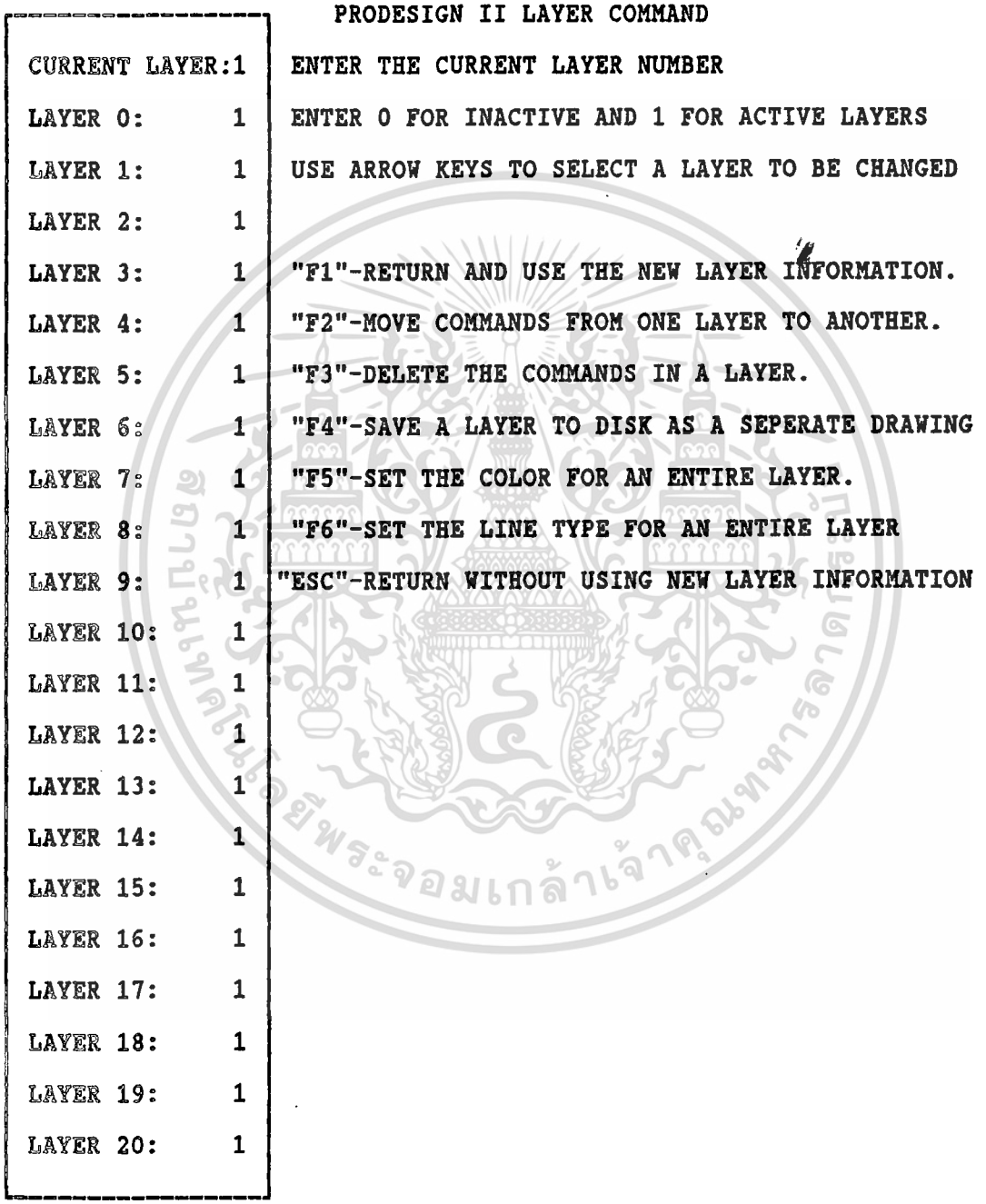
A>PD [Enter]

3.2 คอมพิวเตอร์จะทำโปรแกรม ProDesign II ซึ่งจะแสดงผลบนจอแสดงผลดังในรูปที่ 6.20

0 POINTS COLOR: 1 ZOOM: 1.00

รูปที่ 6.20 ภาพที่ปรากฏบนจอแสดงผลเมื่อเรียกโปรแกรม ProDesign II

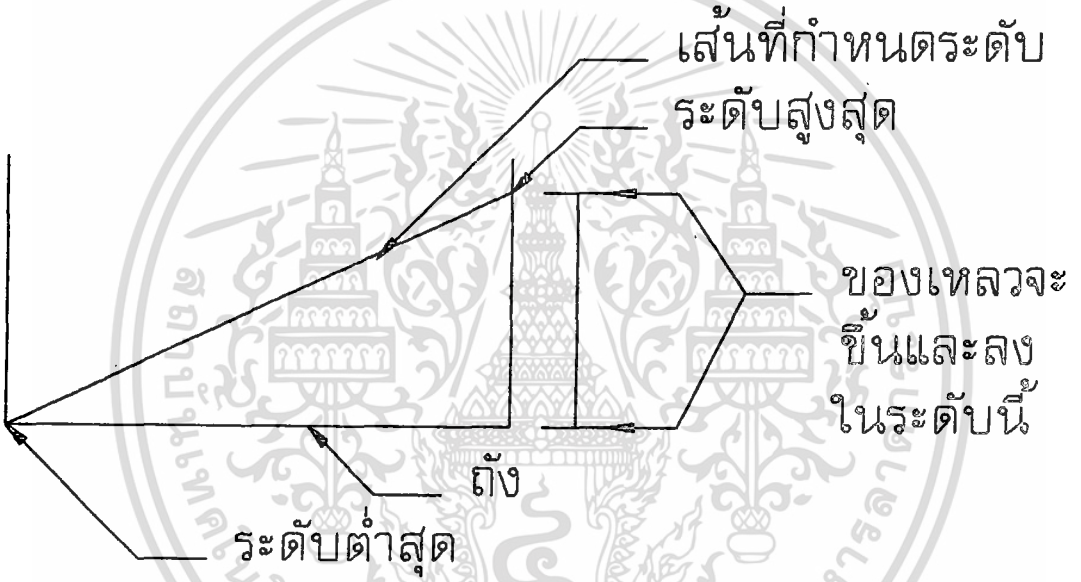
3.3 ในขั้นแรกจะทำการสร้างถังก่อน เนื่องจากเป็นโครงสร้างหลักของภาพ ซึ่งจะทำให้การจัดภาพทำได้สะดวกขึ้น ดังนั้นจะต้องเปลี่ยนชั้นในการวาดเป็นชั้นที่ 2 โดยการกด [L] จอแสดงผลจะแสดงรายการ ดังรูปที่ 6.21



รูปที่ 6.21 รายการสำหรับการจัดการเกี่ยวกับชั้น

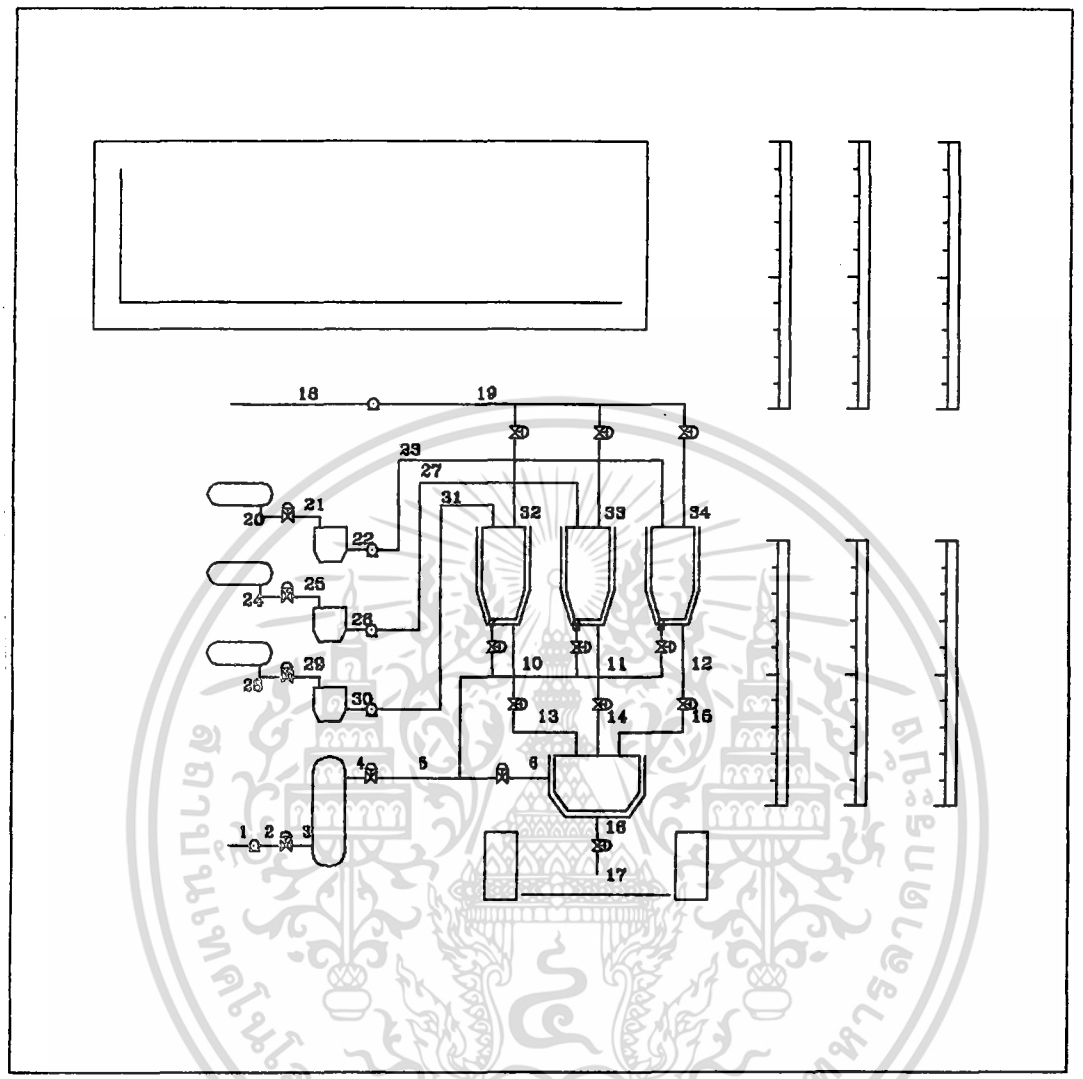
จะเห็นได้ว่าขณะนี้ชั้นที่ใช้งานอยู่คือชั้นที่ 1 ให้เลื่อนเคอร์เซอร์ (CURSOR) ไปที่เลข "1" ของบรรทัดที่มีข้อความ "CURRENT LAYER" แล้วเปลี่ยนให้เป็น "2" จากนั้นกด [F1] จอแสดงผลจะแสดงภาพในรูปที่ 6.19 อีกครั้งหนึ่ง จากนั้นก็สร้างภาพถังตามต้องการ ส่วนรายละเอียดการสร้างภาพในขั้นตอนนี้จะไม่กล่าวถึง เนื่องจากมีรายละเอียดมาก ผู้ใช้สามารถศึกษารายละเอียดของคำสั่งต่างๆได้จากคู่มือของ ProDesign II

3.4 เมื่อสร้างภาพถังเสร็จเรียบร้อยแล้วก็จะเป็นการกำหนดระดับสูงสุดและต่ำสุดในแต่ละถัง ให้กำหนดชั้นที่จะวาดใหม่ตามขั้นตอน 3.3 ให้เป็นชั้นที่ 6 แล้วลากเส้นทะแยงมุมตามรูป



รูปที่ 6.22 แสดงการกำหนดระดับสูงสุดต่ำสุดของของเหลวในถัง

3.5 หลังจากที่วาดถังและกำหนดระดับเสร็จเรียบร้อยแล้ว ต่อไปก็จะเป็นการวาดท่อ และอุปกรณ์อื่นๆต่อไป โดยสมมติว่าเริ่มจากการวาดท่อ ให้เปลี่ยนชั้นที่จะวาดเป็นชั้นที่ 1 ซึ่งเป็นชั้นของท่อ จากนั้นเริ่มวาดที่ท่อหมายเลข 1 (หมายเลขของท่อต่างๆจะต้องถูกกำหนดไว้แล้วตามรูปที่ 6.23)



รูปที่ 6.23 แสดงหมายเลขของท่อต่างๆ

ก่อนการวาดท่อที่ 1 ให้ทำการกำหนดจุดเริ่มต้นของชุดคำสั่งเสียก่อนตามหัวข้อที่ 3 ของบทนี้เรื่องการสร้างภาพลงในคอมพิวเตอร์ นั่นคือเลื่อนเคอร์เซอร์ไปที่จุดเริ่มต้นของท่อที่ 1 หรืออาจจะใช้ตำแหน่งอื่นก็ได้ แต่ควรจะใช้จุดเริ่มต้นของท่อที่ 1 เพื่อให้สะดวกต่อการตรวจสอบ จากนั้นกำหนดจุดลงไปด้วยคำสั่งต่อไปนี้

[0] <-- เป็นการกำหนดจุดสำหรับวาดจุดที่ 1

[0] <-- เป็นการกำหนดจุดสำหรับวาดจุดที่ 2

[V] <-- เป็นคำสั่งสำหรับวาดเส้นตรง จากจุดที่ 1 ไปยังจุดที่ 2

จากการใช้คำสั่งดังกล่าวจะเห็นได้ว่า คำสั่ง [V] นั้นเป็นการวาดเส้นตรงจากจุดที่ 1 ไปยังจุดที่ 2 แต่เมื่อจุดที่ 1 และจุดที่ 2 เป็นจุดเดียวกันเส้นที่ได้ก็คือจุดๆหนึ่งนั่นเอง

หลังจากนั้นก็ทำการวาดท่อที่ 1 และเมื่อเสร็จท่อที่ 1 แล้วก็เริ่มวาดท่อที่ 2 ด้วยวิธีการเดียวกันทำเช่นนี้จนกระทั่งเสร็จทั้งหมด

3.6 เมื่อวาดท่อเสร็จแล้วก็จะมีอุปกรณ์ที่จะต้องวาดในลักษณะเดียวกันนี้อีก 3 ชนิด คือ วาล์ว ปั๊ม และกระป๋อง แต่วาล์วและปั๊มได้รวมไว้ในชั้นเดียวกันเพราะฉะนั้นจะต้องสร้างอุปกรณ์อีก 2 ชั้น คือชั้น VALVE และ CAN ให้ใช้วิธีเดียวกับการวาดท่อ คือกำหนดจุดก่อนที่จะวาดอุปกรณ์ใดๆ และกำหนดชั้นที่จะวาดให้ถูกต้อง

3.7 หลังจากที่อุปกรณ์ต่างๆถูกสร้างและกำหนดเสร็จเรียบร้อยแล้ว ก็ทำการสร้างชั้นอื่นๆที่เหลืออยู่ ซึ่งชั้นเหล่านี้เป็นเพียงภาพที่นำไปซ้อนเท่านั้นไม่มีการเปลี่ยนแปลงสภาวะแต่อย่างใด ดังนั้นจึงเป็นการวาดแบบปกติไม่ต้องกำหนดจุดเริ่ม

3.8 ขั้นตอนสุดท้ายคือการเก็บภาพลงในแผ่นจานแม่เหล็ก ให้เก็บภาพทั้งภาพลงในไฟล์เดียวกัน แล้วพิมพ์ด้วยเครื่องพิมพ์หรือเครื่องวาดภาพ และแยกชั้นต่างๆออกเป็นแต่ละไฟล์โดยใช้คำสั่ง [F4] ในรายการการจัดการเกี่ยวกับชั้น ตามรูปที่ 6.20 จากนั้นให้พิมพ์ออกทางเครื่องพิมพ์แยกทีละชั้นเพื่อใช้ในการตรวจสอบ

4.การเปลี่ยนชุดคำสั่งการสร้างภาพ เมื่อได้ไฟล์ภาพที่แยกแต่ละชั้นแล้วให้เปลี่ยนคำสั่งในการวาดของ ProDesign II ให้เป็นไปตามหัวข้อที่ 4 ในบทนี้ จากนั้นทำการสร้างชุดคำสั่งเก็บลงในไฟล์ก็จะได้ไฟล์อีกชุดหนึ่ง ในไฟล์ชุดนี้ให้กำหนดชนิดของไฟล์ เป็น .DEV และ .LEV ซึ่งเป็นอุปกรณ์และระดับตามลำดับ จากนั้นใช้โปรแกรม CNVRTDEV สำหรับไฟล์ .DEV และ CNVRTLEV สำหรับไฟล์ .LEV ก็จะได้ไฟล์ออกมาอีกชุดหนึ่งซึ่งเป็นภาษาปาสคาล

จากนั้นก็จะเป็นการสร้างโปรแกรมเงื่อนไขการแสดงผลซึ่งจะกล่าวต่อไปในบทที่ 7

บทที่ 7
โปรแกรมเงื่อนไขการแสดงผล

โปรแกรมเงื่อนไขการแสดงผลนั้นในการใช้งานจริงแล้วผู้ใช้จะเป็นผู้พัฒนาขึ้นเอง ซึ่งวัตถุประสงค์ก็คือต้องการทำให้ระบบมีความยืดหยุ่นสูง ถึงแม้ว่าในขั้นแรกผู้ใช้จะต้องเสียเวลามากในการศึกษาและพัฒนาโปรแกรมส่วนนี้ขึ้นเองก็ตาม แต่ในระยะยาวแล้วผู้ใช้สามารถพัฒนาโปรแกรมให้ใช้งานได้อย่างมีประสิทธิภาพ โปรแกรมเงื่อนไขการแสดงผลสามารถแบ่งออกเป็นส่วนใหญ่ๆได้ 3 ส่วนคือ

7.1 การกำหนดช่วงเวลาการทำงานของแต่ละงาน

เนื่องจากเวลาส่วนใหญ่จะใช้ในการจัดการการแสดงผล และการกระทำทางตรรก (LOGIC) แต่เนื่องจากมีอุปกรณ์ที่จะต้องแสดงผลมาก รวมทั้งขณะที่แสดงผลจะต้องสามารถรับสภาวะจากกระบวนการ รับอินพุทจากแป้นพิมพ์ รับค่าจากคันโยก และอื่นๆ ดังนั้นการใช้โปรแกรมที่ทำงานแบบลำดับ (SEQUENCE) ตามปกติ จะทำให้เวลาในการตอบสนอง (RESPOND TIME) ของระบบช้าลงทำให้งานบางงานซึ่งมีความสำคัญมากๆต้องรองานอื่น เช่นการควบคุมทิศทางเลื่อนภาพ (SCROLLING) ด้วยคันโยกนั้นโปรแกรมจะต้องมาตรวจสอบอยู่เสมอว่ามีการบังคับคันโยกนั้นหรือไม่ เพื่อที่จะเลื่อนภาพได้ตามต้องการ ถ้าความถี่ของการทำงานในส่วนนี้น้อยจะทำให้ภาพที่เลื่อนไม่ราบเรียบ การใช้อินเตอร์รัพท์ (INTERRUPT) ก็เป็นวิธีหนึ่งซึ่งให้ผลดี แต่การพัฒนาโปรแกรมจะค่อนข้างยุ่งยากและซับซ้อนสำหรับผู้ใช้ที่ไม่มีความรู้ทางด้านฮาร์ดแวร์ (HARDWARE) อีกวิธีหนึ่งซึ่งให้ผลดีพอสมควรและเป็นวิธีการทางซอฟท์แวร์ทั้งหมดก็คือ การมัลติเพลกซ์ (MULTIPLEXING) จะเป็นการแบ่งเวลาของโปรแกรมออกเป็นส่วนๆแล้วกำหนดได้ว่าจะให้งานใดทำงานในช่วงใด ดังรูปที่ 7.1

รูปที่ 7.1 แสดงทฤษฎีการมัลติเพลกซ์งานหลายๆงาน

จากรูปที่ 7.1 จะเห็นได้ว่าการทำมัลติเพลกซ์จะสามารถกำหนดความถี่และช่วงเวลาการทำงานของแต่ละงานได้ สมมติให้ในโปรแกรมจะต้องมีงานที่ต้องทำอยู่ 8 งาน

งานที่ 1 : ตรวจสอบและรับค่าจากแป้นพิมพ์ ถ้ามีการกดแป้นพิมพ์ให้ทำงานตามคำสั่ง

งานที่ 2 : ตรวจสอบและรับค่าจากคันโยก ถ้ามีการโยกคันโยกในทิศทางใดให้เลื่อนภาพไปในทิศทางนั้น

งานที่ 3 : อ่านค่าสภาวะจากกระบวนการทางอุตสาหกรรมและประมวลผลทางคณิตศาสตร์

งานที่ 4 : แสดงสภาวะของอุปกรณ์ประเภทที่ 1

งานที่ 5 : แสดงสภาวะของอุปกรณ์ประเภทที่ 2

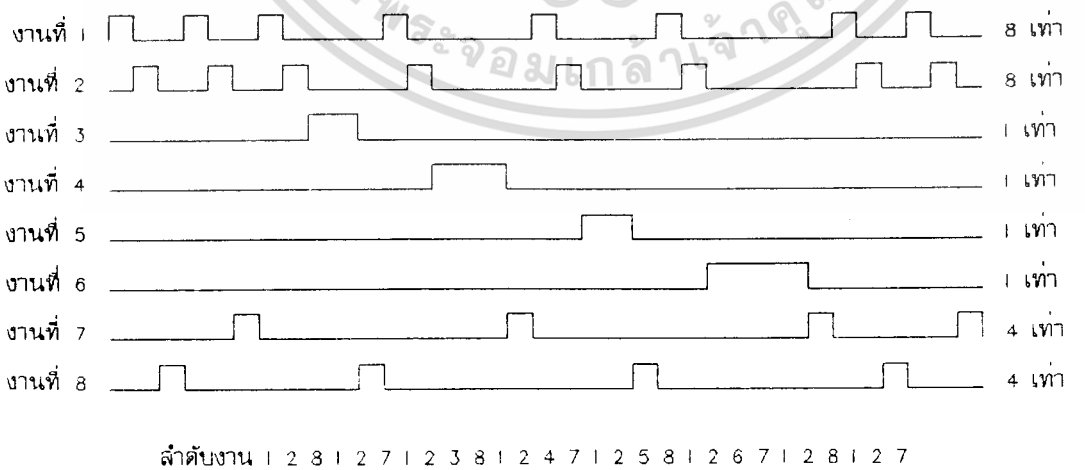
งานที่ 6 : แสดงสภาวะของอุปกรณ์ประเภทที่ 3

งานที่ 7 : ตรวจสอบเงื่อนไขการสั่งพิมพ์ และพิมพ์ค่าทางเครื่องพิมพ์

งานที่ 8 : ตรวจสอบเงื่อนไขการบันทึกข้อมูลลงจานแม่เหล็ก และบันทึกข้อมูล

งานทั้งหมดมีความสำคัญและความต้องการความถี่และช่วงเวลาการทำงานแตกต่างกัน วิธีการก็คือในขั้นแรกจัดลำดับการทำงานของแต่ละงาน เนื่องจากงานบางงานสามารถทำงานได้โดยอิสระ แต่บางงานจะต้องเป็นลำดับกันไป เช่น งานที่ 3,4,5,6 จะต้องทำงานเป็นลำดับกันโดยที่งานที่ 4,5,6 สามารถสลับกันได้ แต่จะต้องทำงานที่ 3 มาก่อน เนื่องจากงานที่ 3 เป็นการอ่านค่าจากกระบวนการ เพราะฉะนั้นข้อมูลจะต้องถูกปรับปรุงใหม่ (UPDATE) ก่อนแล้วจึงนำไปแสดงผล มิฉะนั้นข้อมูลที่นำไปแสดงผลจะเป็นข้อมูลเก่าซึ่งช้ากว่าความเป็นจริงอยู่ 1 รอบการทำงาน ดังในตัวอย่างนี้สมมติให้งานทุกงานถูกจัดลำดับมาโดยถูกต้องแล้ว

ในขั้นต่อไปเป็นการกำหนดความถี่ในการทำงานของแต่ละงาน หลักการของการกำหนดความถี่ส่วนใหญ่แล้วงานจะถูกทำเป็นลำดับ เช่นงานที่ 3,4,5,6 จะมีความถี่ในการทำงานเท่ากัน การกำหนดให้งานใดงานหนึ่งทำงานมากกว่างานอื่นก็ไม่มีผลการเปลี่ยนแปลงทำให้เสียเวลาไปโดยเปล่าประโยชน์ สมมติว่ากำหนดให้งานที่ 4 หรือ 5 หรือ 6 ทำงานมากกว่างานอื่นก็จะพบว่าไม่มีการเปลี่ยนแปลงใดๆ เลย เนื่องจากข้อมูลไม่มีการปรับปรุงใหม่นั่นเอง ส่วนการที่จะให้งานที่ 3 ทำงานมากกว่างานอื่นก็จะเป็นการปรับปรุงข้อมูลซ้ำซ้อนกันซึ่งข้อมูลที่อ่านเข้ามาในครั้งแรกยังไม่ถูกนำไปแสดงผลด้วยงาน 4,5 และ 6 ดังนั้นจะเห็นได้ว่าการกำหนดความถี่การทำงานจะสัมพันธ์กับการกำหนดช่วงเวลาด้วย สำหรับงานที่ต้องการความถี่ในการทำงานสูงก็คืองานการรับค่าอินพุทจากผู้ใช้ ในที่นี้คือแป้นพิมพ์และคันโยก การกำหนดให้ความถี่ของงานเหล่านี้ให้มีค่าสูงจะทำให้ความเร็วในการตอบสนองของระบบต่อผู้ใช้เป็นไปในลักษณะทันทีทันใด (INTERACTIVE) ซึ่งทำให้ผู้ใช้สามารถควบคุมระบบได้สะดวก ดังนั้นในตัวอย่างนี้จะกำหนดค่าความถี่และช่วงเวลาดังในรูปที่ 7.2



รูปที่ 7.2 แสดงช่วงเวลาและความถี่การทำงานที่กำหนด

ตัวอย่างการกำหนดช่วงเวลา ความถี่ และการทำงานของการมัลติเพลกซ์ก็จะแบ่งได้ดังนี้

- การกำหนดความถี่และช่วงเวลา : จะกำหนดด้วยเลขฐาน 2 (BINARY NUMBER) ตามช่วงเวลา เช่นถ้ากำหนดให้ช่วงเวลาต่อ 1 รอบเป็น 8 ช่วงเวลาก็จะใช้เพียง 1 ไบท์ แต่ถ้ากำหนดให้เป็น 16 ช่วงเวลาก็จะใช้ 2 ไบท์เป็นต้น โดยตั้งค่าคงที่ (CONSTANT) ตามขนาดดังกล่าวดังตัวอย่าง

```
const                {...:...:...:...:}
  Mask1  = $8080; {-_______-_______} {CheckKey}
  Mask2  = $FFFF; {-----------------} {JoyStick}
  Mask3  = $2020; {__-_____-_____} {ReadPort}
  Mask4  = $1010; {___-_____-____} {DisplayValue&Pipe}
  Mask5  = $0808; {____-_____-___} {DisplayGraph}
  Mask6  = $0404; {_____-_____-__} {DisplayLevel}
  Mask7  = $0202; {______-_____-_} {DisplayCanAnimation}
  Mask8  = $0101; {_______-_____-} {DisplayTemp}
  Mask9  = $8080; {-_______-_______} {DisplayHostScreen}
  Mask10 = $FFFF; {-----------------} {ReadDateAndTime}
  Mask11 = $2020; {__-_____-_____} {DisplayAlarm}
  Mask12 = $1010; {___-_____-____}
  Mask13 = $0808; {____-_____-___}
  Mask14 = $0404; {_____-_____-__}
  Mask15 = $0202; {______-_____-_}
  Mask16 = $0101; {_______-_____-}
```

จะเห็นได้ว่าการกำหนดจะใช้เลขฐาน 16 (HEXADECIMAL NUMBER) แต่การออกแบบจะถูกกำหนดมาจากเลขฐาน 2 เสียก่อน เช่นงานที่ 1 ใช้ชื่อว่า Mask1 จะมีช่วงเวลาเป็น 1000 0000 1000 0000 เป็นต้น

ในการกำหนดตามตัวอย่างในช่วงเวลาที่ 1 งานที่ทำคือ งานที่ 1,2,9,10
ช่วงเวลาที่ 2 งานที่ทำคือ งานที่ 2,10
ช่วงเวลาที่ 3 งานที่ทำคือ งานที่ 2,3,10,11
ช่วงเวลาที่ 4 งานที่ทำคือ งานที่ 2,4,10,12
ช่วงเวลาที่ 5 งานที่ทำคือ งานที่ 2,5,10,13
ช่วงเวลาที่ 6 งานที่ทำคือ งานที่ 2,6,10,14
ช่วงเวลาที่ 7 งานที่ทำคือ งานที่ 2,7,10,15
ช่วงเวลาที่ 8 งานที่ทำคือ งานที่ 2,8,10,16
ช่วงเวลาที่ 9 งานที่ทำคือ งานที่ 1,2,9,10
ช่วงเวลาที่ 10 งานที่ทำคือ งานที่ 2,10
ช่วงเวลาที่ 11 งานที่ทำคือ งานที่ 2,3,10,11
ช่วงเวลาที่ 12 งานที่ทำคือ งานที่ 2,4,10,12
ช่วงเวลาที่ 13 งานที่ทำคือ งานที่ 2,5,10,13
ช่วงเวลาที่ 14 งานที่ทำคือ งานที่ 2,6,10,14
ช่วงเวลาที่ 15 งานที่ทำคือ งานที่ 2,7,10,15
ช่วงเวลาที่ 16 งานที่ทำคือ งานที่ 2,8,10,16

- การเลื่อนลำดับของช่วงเวลา : จะตั้งตัวแปร (VARIABLE) ที่มีขนาดเท่ากับค่าคงที่ดังกล่าวขึ้นมา ในที่นี้ใช้ตัวแปรชื่อ Sequence ซึ่งค่าใน Sequence จะเป็นตัวบอกช่วงเวลา การเลื่อนลำดับของช่วงเวลากระทำได้โดยเลื่อนค่าใน Sequence ไป 1 ตำแหน่ง ดังตัวอย่าง

```
procedure RotateSequence;
begin {RotateSequence}
  Sequence := Sequence shr 1;
  if Sequence = $0000 then Sequence := $8000;
end {RotateSequence};
```

เนื่องจากในภาษาปาสคาลไม่มีคำสั่งหมุนข้อมูลจึงต้องใช้วิธี เลื่อนข้อมูล แล้วตั้งค่าใหม่ เมื่อ เลื่อนจนสุดความยาวแล้ว

- การตรวจสอบลำดับของงาน : เป็นการตรวจสอบลำดับของช่วงเวลาเปรียบเทียบกับลำดับของงานที่กำหนด ดังตัวอย่าง

```
procedure TaskPolling;
begin {TaskPolling}
  if (Sequence and Mask1 = Sequence) then Task1;
  if (Sequence and Mask2 = Sequence) then Task2;
  if (Sequence and Mask3 = Sequence) then Task3;
  if (Sequence and Mask4 = Sequence) then Task4;
  if (Sequence and Mask5 = Sequence) then Task5;
  if (Sequence and Mask6 = Sequence) then Task6;
  if (Sequence and Mask7 = Sequence) then Task7;
  if (Sequence and Mask8 = Sequence) then Task8;
  if (Sequence and Mask9 = Sequence) then Task9;
  if (Sequence and Mask10 = Sequence) then Task10;
  if (Sequence and Mask11 = Sequence) then Task11;
  if (Sequence and Mask12 = Sequence) then Task12;
  if (Sequence and Mask13 = Sequence) then Task13;
  if (Sequence and Mask14 = Sequence) then Task14;
  if (Sequence and Mask15 = Sequence) then Task15;
  if (Sequence and Mask16 = Sequence) then Task16;
end {TaskPolling};
```

7.2 การกำหนดสีของอุปกรณ์ในสภาวะต่างๆ

การกำหนดสีของอุปกรณ์ต่างๆซึ่งโดยปกติแล้วก็จะให้อุปกรณ์แต่ละชนิดมีสีที่แตกต่างกันเพื่อให้สังเกตุได้ง่าย ข้อสำคัญอีกประการหนึ่งก็คือการกำหนดสีในสภาวะที่แตกต่างกัน เช่น อุปกรณ์ที่มีสภาวะ เพียง 2 สภาวะคือปิดและ เปิดนั้นก็จะกำหนดให้สีของอุปกรณ์ขณะที่อยู่ในสภาวะทั้งสองให้แตกต่างกัน ส่วนอุปกรณ์ที่มีการรับค่า เป็นอนาลอกนั้นก็อาจจะใช้ เพียงสีเดียว เนื่องจากการ เปลี่ยนแปลงสามารถสังเกตุได้จากระดับ นอกจากกราฟที่อาจจะใช้หลายสีตามระดับหรือค่าที่ เปลี่ยนแปลงไป ดังตัวอย่าง

```
HideColor = _Black;
NormalColor = _White;
WaterPipeColor = _Blue;
BoilerPipeColor = _Green;
MixTankPipeColor = _Yellow;
TankAPipeColor = _Magenta;
TankBPipeColor = _Red;
TankCPipeColor = _Cyan;
CanOnColor = NormalColor;
CanOffColor = HideColor;
```

จากตัวอย่างได้กำหนดให้สีของท่อต่างๆนั้นแตกต่างกันตามชนิดของการใช้งาน โดยที่สีในขณะที่อยู่ในสภาวะปิด เป็นสี เดียวกันหมดคือสีขาว ส่วนระดับของของ เหลวในถังก็กำหนดให้แตกต่างกัน ส่วนค่าคงที่ 2 ตัวสุดท้ายนั้นในตัวอย่างการใช้งานจะแสดงผล เป็นการ เคลื่อนที่ของกระป๋องบนสายพานลำ เลียง (CONVEYER) เมื่อกระป๋อง เลื่อนไปอยู่ตำแหน่งใดก็จะมีการวาดกระป๋องที่ตำแหน่งนั้น ส่วนตำแหน่งอื่นก็จะลบออกด้วยการวาดด้วยสีดำ

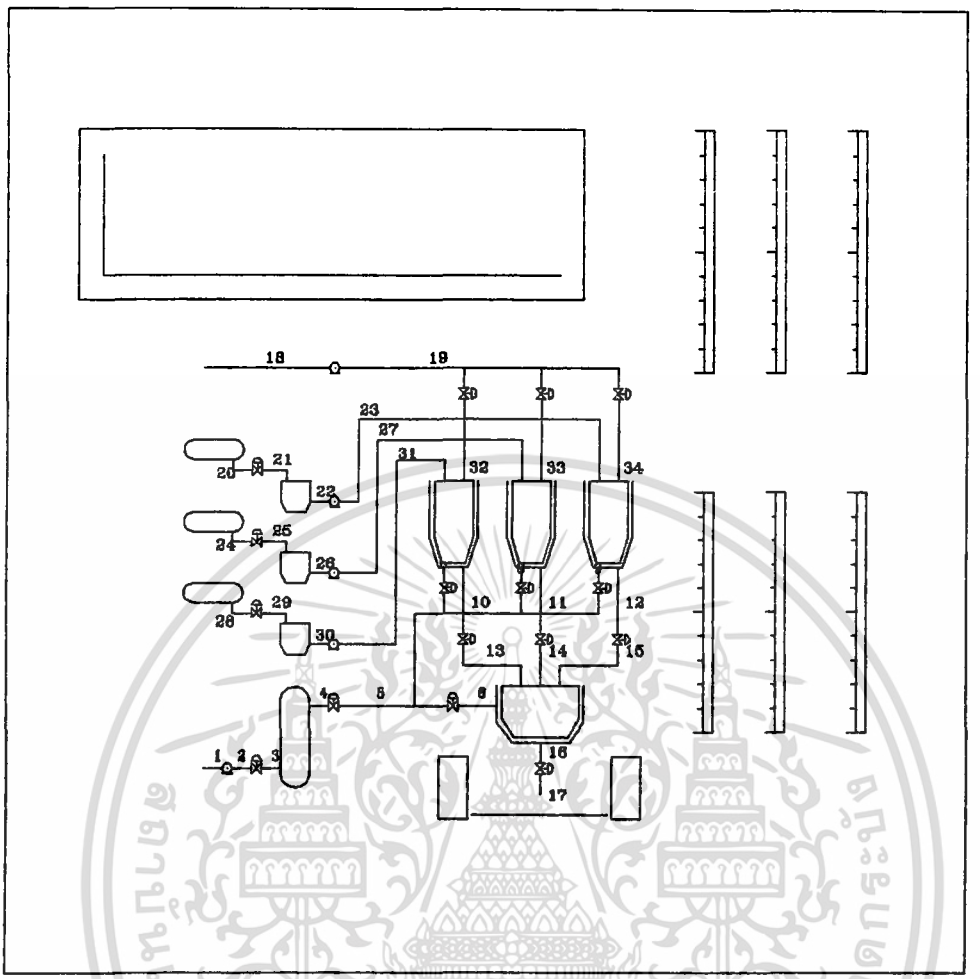
7.3 การกำหนดเงื่อนไขการแสดงผล

การกำหนดเงื่อนไขการแสดงผลจะเป็นส่วนที่สำคัญที่สุดในโปรแกรมส่วนนี้ เป็นการกำหนดเงื่อนไขการแสดงผลให้สอดคล้องกับการทำงานของกระบวนการ และเนื่องจากอุปกรณ์บางชนิดไม่มีการอ่านค่าหรือรับค่าสภาวะมาจากกระบวนการ เช่นท่อเป็นต้น ดังนั้นการที่จะทำให้ท่อดังกล่าวสามารถแสดงสภาวะได้ว่ามีการไหลในท่อหรือไม่ ก็จะนำสภาวะของวาล์วมาพิจารณาในกรณีที่วาล์วที่ต่ออยู่กับท่อนั้นมีสภาวะเปิดก็อาจจะสันนิษฐานได้ว่ามีการไหลในท่อนั้นหรืออาจจะมีเงื่อนไขอื่นๆมาเกี่ยวข้องอีกก็ได้ การสร้างเงื่อนไขต่างๆนั้นผู้ใช้จะต้องศึกษาการทำงานของกระบวนการ และรูปแบบการแสดงผลเป็นสำคัญ ดังในตัวอย่างที่ได้กล่าวถึงนั้นมีการทำเงื่อนไขเกี่ยวกับการแสดงผลอยู่มากจะขอยกตัวอย่างมาเพียงบางส่วนเท่านั้น ส่วนรายละเอียดของโปรแกรมทั้งหมดจะอยู่ในภาคผนวก 3

```
procedure DisplayDeviceNumber (Device : byte);
begin {DisplayDeviceNumber}
  case Device of
    1 : begin
          if Valve1Status xor DeviceStatus[1] then
          begin
            if Valve1Status then
            begin
              Valve1(WaterPipeColor);
              Pipe3(WaterPipeColor);
            end
            else
            begin
              Valve1(NormalColor);
              Pipe3(NormalColor);
            end;
            PrintStatus('Valve1',Valve1Status);
          end;
        end;
```

```
    2 : begin
          .
          .
          .
    21 : begin
           if Pump5Status xor DeviceStatus[21] then
           begin
             if Pump5Status then
             begin
               Pump5(WaterPipeColor);
               Pipe19(WaterPipeColor);
             end
             else
             begin
               Pump5(NormalColor);
               Pipe19(NormalColor);
             end;
             PrintStatus('Pump5',Pump5Status);
           end;
         end;
  end;
end {DisplayDeviceNumber};
```

ในตัวอย่างที่ได้ยกมานี้เป็นการทำเงื่อนไขในการแสดงผลของวาล์วและท่อ จะเห็นได้ว่าสภาวะของท่อนั้นขึ้นอยู่กับสภาวะของวาล์วโดยตรง ซึ่งในความเป็นจริงแล้วอาจจะมีเงื่อนไขอื่นมาเกี่ยวข้องอีกก็ได้ ส่วนตัวแปรที่นำมาตรวจสอบเงื่อนไขคือสภาวะของวาล์วนั้นๆ เช่น ValveStatus1 ก็คือสภาวะของวาล์วตัวที่ 1 ส่วนท่อหมายเลข 3 นั้นเป็นท่อที่ต่ออยู่กับวาล์วตัวที่ 1 ดังในรูปที่ 7.3



รูปที่ 7.3 แสดงหมายเลขของท่อต่างๆ

ส่วนตัวแปร DeviceStatus นั้นเป็นตัวแปรแบบอาเรย์ (ARRAY) ใช้สำหรับการเก็บสภาวะของอุปกรณ์ในช่วงเวลาที่ผ่านมา เพื่อให้โปรแกรมไม่ต้องวาดอุปกรณ์ทุกๆรอบการทำงานจะวาดก็ต่อเมื่อมีการเปลี่ยนแปลงสภาวะเท่านั้น ดังอัลกอริธึม (ALGORITHM) ต่อไปนี้

```
begin
  ReadStatus;
  if (CurrentStatus = ON) and (PreviousStatus = OFF) then Draw;
  if (CurrentStatus = OFF) and (PreviousStatus = ON) then Erase;
  PreviousStatus := Status;
end;
```

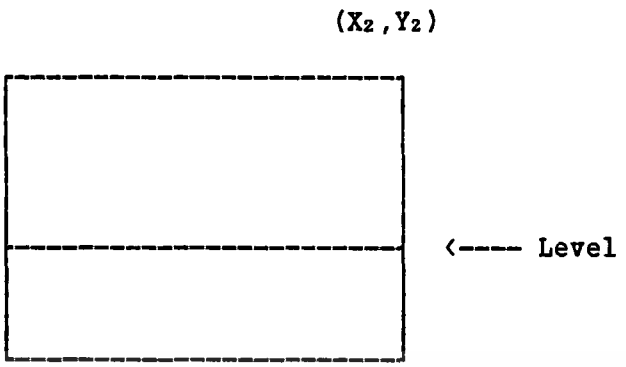
จากตัวอย่างจะเห็นได้ว่าโปรแกรมจะทำการวาดหรือลบ (Erase) ก็ต่อเมื่อมีการเปลี่ยนแปลงสภาวะจาก เปิด เป็นปิดหรือปิด เป็น เปิดนั่นเอง วิธีนี้จะช่วยให้ระบบทำงานได้เร็วขึ้น เนื่องจากไม่ต้องวาดใหม่ทุกครั้ง

เมื่อตรวจสอบสภาวะพบว่า เกิดการเปลี่ยนแปลงขึ้นและเปลี่ยนไปอย่างไรแล้ว ก็จะเป็นคำสั่งในการวาด จะเห็นได้ว่าถ้าวาล์วตัวใดมีสภาวะ เปิดก็จะทำการวาดวาล์วตัวนั้นและท่อที่เกี่ยวข้องด้วยสีที่กำหนด ส่วนกรณีที่วาล์วมีสภาวะปิดก็จะใช้สีอีกสีหนึ่งในที่นี้ใช้สีขาว ดังนั้นจะสามารถสังเกตุเห็นการเปลี่ยนแปลงสภาวะของกระบวนการได้อย่างชัดเจน

การแสดงการเปลี่ยนแปลงสภาวะโดยการเปลี่ยนระดับนั้น เป็นการเรียกใช้ชื่อของอุปกรณ์ที่ผู้ใช้ตั้งไว้ในขั้นตอนการสร้างภาพและแปลงชุดคำสั่ง (หัวข้อที่ 5 บทที่ 6) ตามด้วยระดับของของเหลวในถัง เช่น TankA2(Level); เป็นต้น ส่วนวิธีการสร้างภาพการเปลี่ยนแปลงระดับนั้น โปรแกรมที่กำหนดของอุปกรณ์แต่ละตัวจะถูกสร้างขึ้นเอง เมื่อใช้โปรแกรม CNVRTDEV และ CNVRTLEV ดังที่ได้อธิบายไว้ในบทที่ 6 ซึ่งมีลักษณะดังนี้

```
procedure TankA2(var Percent : integer);
begin
  SetColor(TankA2Color);
  CurrentLevel := TankA2Level;
  DrawLevel(309,471,341,449,Percent);
  TankA2Level := CurrentLevel;
end;
```

คำสั่ง SetColor เป็นการเปลี่ยนสีในการวาดซึ่งจะทำหน้าที่เปลี่ยนสีไปตามที่ผู้ใช้ได้กำหนดไว้ ส่วนตัวแปร CurrentLevel และ TankA2Level นั้นมีหน้าที่ตรวจสอบการเปลี่ยนแปลงของระดับโดยจะทำการวาดเมื่อมีการเปลี่ยนระดับไปจากเดิมเท่านั้น ซึ่งเหมือนกับตัวอย่างที่ยกมาข้างต้นนั่นเอง คำสั่ง DrawLevel เป็นคำสั่ง (PROCEDURE) ที่สร้างขึ้นสำหรับการวาดระดับโดยมีค่าพารามิเตอร์ที่ต้องการ 5 ตัวคือ X_1 Y_1 X_2 Y_2 และระดับที่ต้องการวาดดังในรูปที่ 7.4



รูปที่ 7.4 การกำหนดระดับสูงสุดและต่ำสุดของของเหลวในถัง

ค่าพารามิเตอร์ X_1 Y_1 X_2 Y_2 โปรแกรม CNVRTLEV จะเป็นตัวจัดการและนำมาใส่ไว้เอง

การแสดงการเปลี่ยนแปลงสภาวะอีกชนิดหนึ่งที่จะกล่าวถึงคือการแสดงภาพเคลื่อนไหว เช่นในกระบวนการตัวอย่างนั้นจะมีกระป๋องเลื่อนอยู่บนสายพานลำเลียง การทำให้กระป๋องหรือภาพเคลื่อนไหวก็ใช้วิธีสร้างภาพและลบภาพทีละภาพ โดยแต่ละภาพจะมีกระป๋องอยู่บนสายพานลำเลียง 5 กระป๋อง ภาพที่ต้องการสร้างมี 3 ภาพดังนั้นจะต้องมีการวาดกระป๋องทั้งสิ้น 15 กระป๋อง (ดังตัวอย่างในรูปที่ 6.16 ของบทที่ 6) นั่นคือ

ภาพที่ 1 กระป๋องอยู่ที่ตำแหน่งที่ 1,4,7,10,13

ภาพที่ 2 กระป๋องอยู่ที่ตำแหน่งที่ 2,5,8,11,14

ภาพที่ 3 กระป๋องอยู่ที่ตำแหน่งที่ 3,6,9,12,15

ดังนั้นเมื่อให้ทั้ง 3 ภาพแสดงออกมาอย่างต่อเนื่องกันก็จะเป็นภาพของกระป๋องที่เคลื่อนที่ไปบนสายพานลำเลียง การสร้างโปรแกรมเงื่อนไขก็จะมีลักษณะดังตัวอย่าง

```
procedure DisplayCanAnimation (CanPosition : byte);
begin {DisplayCanAnimation}
  case CanPosition of
    0 : begin
          Can3(CanOffColor);
          Can6(CanOffColor);
          Can9(CanOffColor);
          Can12(CanOffColor);
          Can15(CanOffColor);
          Can1(CanOnColor);
          Can4(CanOnColor);
          Can7(CanOnColor);
          Can10(CanOnColor);
          Can13(CanOnColor);
        end;
    1 : begin
          Can1(CanOffColor);
          Can4(CanOffColor);
          Can7(CanOffColor);
          Can10(CanOffColor);
          Can13(CanOffColor);
          Can2(CanOnColor);
          Can5(CanOnColor);
          Can8(CanOnColor);
          Can11(CanOnColor);
          Can14(CanOnColor);
        end;
    2 : begin
          Can2(CanOffColor);
          Can5(CanOffColor);
```

```
            Can8(CanOffColor);
            Can11(CanOffColor);
            Can14(CanOffColor);
            Can3(CanOnColor);
            Can6(CanOnColor);
            Can9(CanOnColor);
            Can12(CanOnColor);
            Can15(CanOnColor);
          end;
    end;
end {DisplayCanAnimation};
```

7.4 การประมาณค่าเวลาตอบสนองของระบบ

การประมาณค่าเวลาตอบสนองของระบบจะกระทำเพื่อใช้ในการหาประสิทธิภาพและประมาณการทำงานของระบบ เนื่องจากระบบมีการทำงานหลายประเภท เช่นการสร้างภาพกระบวนการ การอ่านค่าสภาวะและการส่งค่าเอาท์พุทสัญญาณเตือนเป็นต้น ดังนั้นเวลาที่ใช้ในการประมวลผลใน 1 รอบการทำงานก็คือค่าเวลาตอบสนองของระบบซึ่งจะมีค่าไม่เท่ากันทุกๆรอบการทำงาน เนื่องจากใน 1 รอบการทำงานจะมีเงื่อนไขการประมวลผลที่แตกต่างกัน การประมาณค่าเวลาดังกล่าวสามารถกระทำได้หลายวิธี เช่น

- **การประมาณค่าเวลาตอบสนองของระบบด้วยการคำนวณชุดคำสั่ง** วิธีนี้จะใช้การประมาณการทำงานของโปรแกรมว่าในแต่ละรอบการทำงานใช้ชุดคำสั่งจำนวนเท่าใด ถ้าโปรแกรมใช้ภาษาชั้นสูงการประมาณเวลาจะต้องประมาณจากเวลาในการประมวลผล 1 ชุดคำสั่งของภาษาชั้นสูงว่ามีกี่ชุดคำสั่งในภาษาเครื่องจากนั้นจึงทำการประมาณค่าเวลาของชุดคำสั่งในภาษาเครื่องว่าใช้จำนวนสัญญาณนาฬิกากี่ลูก จากนั้นจะนำจำนวนสัญญาณนาฬิกานี้มาคำนวณเป็นค่าเวลาซึ่งจะขึ้นอยู่กับความถี่สัญญาณนาฬิกาด้วย เมื่อได้ค่าเวลาโดยประมาณสำหรับ 1 ชุดคำสั่งในภาษาชั้นสูงก็จะต้องทำการประมาณจำนวนชุดคำสั่งใน 1 รอบการทำงานของโปรแกรมว่าใช้จำนวนกี่ชุดคำสั่งแล้วจึงนำมาคำนาณกับค่าเวลาที่หาได้ก็จะได้ค่าเวลาการตอบสนองของระบบโดยประมาณ เช่น

- 1 ชุดคำสั่งในภาษาปาสคาลจะมีค่าเฉลี่ยโดยประมาณเท่ากับ 20 ชุดคำสั่งในภาษาเครื่อง กำหนดให้เป็น n_p

- 1 ชุดคำสั่งในภาษาเครื่องจะใช้เวลาในการประมวลผลโดยเฉลี่ยประมาณเท่ากับสัญญาณนาฬิกาจำนวน 5 ลูก กำหนดให้เป็น n_a

- สัญญาณนาฬิกาจำนวน 1 ลูกจะใช้เวลา 0.1 ไมโครวินาที ที่ความถี่ 10 MHz กำหนดให้เป็น t_{clk}

- ในการประมวลผลใน 1 รอบการทำงานจะมีการทำงานโดยเฉลี่ยประมาณ 2000 ชุดคำสั่งในภาษาปาสคาล กำหนดให้เป็น n_s

ดังนั้นจากข้อมูลที่กล่าวข้างต้นจะได้ค่าเวลาในการตอบสนองของระบบ t_s จากสมการ

$$t_s = n_p \times n_a \times t_{clk} \times n_s$$

จะได้

$$t_s = 20 \times 5 \times 0.1 \times 10^{-6} \times 2000$$

$$= 0.02 \text{ วินาที หรือ } 20 \text{ มิลลิวินาที}$$

แต่การใช้วิธีนี้จะมีค่าผิดพลาดมากและยุ่งยากในการคำนวณ เนื่องจากการประมาณจำนวนชุดคำสั่งในภาษาชั้นสูง เช่นภาษาปาสคาลมีจำนวนมากและใช้เวลาในการประมวลผลแตกต่างกันมากโดยเฉพาะคำสั่งในการทำเงื่อนไขที่เป็นลอจิกและคณิตศาสตร์ นอกจากนั้นตัวแปลภาษาเทอร์โบปาสคาลที่ใช้ก็เป็นตัวแปลภาษาที่ไม่สามารถสร้างไฟล์ข้อมูลที่ใช้ในการตรวจสอบว่าใน 1 ชุดคำสั่งประกอบด้วยกี่ชุดคำสั่งในภาษาเครื่อง และโปรแกรมที่ใช้ก็มีการเปลี่ยนแปลงได้ด้วยผู้ใช้เองดังนั้นจำนวนชุดคำสั่งจะมีค่าเปลี่ยนแปลงอยู่เสมอ การใช้วิธีจึงค่อนข้างยุ่งยาก เสียเวลาในการคำนวณมากและจากการประมาณค่าเวลาในขั้นตอนต่างๆหลายขั้นตอนนี้เองทำให้เกิดค่าผิดพลาดสูง

- การประมาณค่าเวลาการตอบสนองของระบบด้วยเครื่องมือวัด วิธีนี้จะทำให้ได้ค่าเวลาที่ค่อนข้างแม่นยำ เนื่องจากเป็นการวัดค่าเวลาจากการทำงานของระบบจริง การวัดค่าดังกล่าวนี้กระทำได้โดยใช้ออสซิลโลสโคป (OSCILLOSCOPE) ทำการวัดสัญญาณใดๆที่จะเกิดขึ้นทุกๆรอบการทำงาน เช่นสัญญาณการติดต่อระหว่างคอมพิวเตอร์กับส่วนเชื่อมต่อกับกระบวนการทางอุตสาหกรรม สัญญาณนี้จะเกิดขึ้นทุกๆรอบการทำงาน เนื่องจากคอมพิวเตอร์จะทำการอ่านค่าสภาวะหรือส่งค่าสภาวะไปยังอุปกรณ์ภายนอก แต่ค่าเวลานี้จะไม่เท่ากันทุกๆรอบการทำงาน เนื่องจากโปรแกรมมีการทำเงื่อนไขมาก ดังนั้นค่าเวลาดังกล่าวจะเป็นค่าเวลาเฉลี่ยของการตอบสนองของระบบแต่ก็มีความผิดพลาดน้อยลง จากการทดลองพบว่าค่าเวลาตอบสนองของระบบประมาณ 30 - 40 มิลลิวินาที เหตุผลที่ไม่สามารถหาค่าที่แน่นอนได้ก็เนื่องจากค่าเวลานี้เปลี่ยนแปลงอยู่ตลอดเวลา ทำให้สัญญาณที่ปรากฏบนจอภาพออสซิลโลสโคปไม่อยู่นิ่ง ดังนั้นค่าที่ได้จึงได้มาจากการประมาณว่าอยู่ในช่วงดังกล่าว

- การประมาณค่าเวลาการตอบสนองของระบบด้วยโปรแกรม วิธีนี้จะใช้โปรแกรมสำหรับการคำนวณค่าเวลามาใช้ในการหาเวลาการตอบสนองของระบบ โดยใช้ค่าเวลาจากคอมพิวเตอร์ควบคุมระบบ เนื่องจากในคอมพิวเตอร์ควบคุมระบบจะต้องมีฐานเวลาหรือนาฬิกาซึ่งทำงานอยู่ตลอดเวลาในขณะที่ระบบทำงาน โดยโปรแกรมนี้จะเริ่มทำการจับเวลาเมื่อโปรแกรมเริ่มทำงานในรอบการทำงานหนึ่งๆและจะสิ้นสุดเมื่อโปรแกรมสิ้นสุดรอบการทำงานนั้นๆ ดังนั้นเวลาที่ได้ก็คือเวลาในการทำงานใน 1 รอบการทำงานนั่นเอง จากนั้นก็นำค่าเวลาดังกล่าวมาทำการหาค่าเฉลี่ยก็จะได้ค่าเวลาการตอบสนองของระบบ โดยโปรแกรมที่ใช้จะมีอัลกอริทึมดังนี้

```
procedure StopWatch (var Start,Stop : boolean; ElapsTime :
TimeType);
var
  StartTime,StopTime : TimeType;
begin
  if Start then
  begin
    ReadTime (StartTime);
    Start := False;
  end;
  if Stop then
  begin
    ReadTime (StopTime);
    Stop := False;
    ElapsTime := StopTime - StartTime;
    Saving (ElapsTime);
  end;
end;
```

จากอัลกอริทึมข้างต้นได้นำไปทดลองใช้ในโปรแกรมประมวลผล เพื่อใช้ในการหาค่าเวลาในการตอบสนองของระบบและเก็บค่าเวลาดังกล่าวไว้ในหน่วยความจำ เหตุผลที่ไม่พิมพ์ค่าเวลาดังกล่าวออกทางเครื่องพิมพ์เลยก็เนื่องจากว่าการส่งผลออกทางเครื่องพิมพ์จะเสียเวลามาก ดังนั้นอาจจะทำให้ค่าเวลาตอบสนองของระบบคลาดเคลื่อนไปจากเดิม เมื่อเก็บตัวอย่างค่าเวลาได้จำนวนหนึ่งแล้วก็ให้โปรแกรมหยุดทำงานแล้วให้พิมพ์ค่าที่หาได้ออกมาทางเครื่องพิมพ์ แต่ในระบบฐานเวลาของ IBM PC/XT นั้นมีความละเอียดเพียง 1/100 วินาทีดังนั้นจึงมีค่าผิดพลาดในหลัก 1/1000 วินาที เพื่อให้ค่าที่ได้มีความถูกต้องมากยิ่งขึ้นก็ควรจะหาค่าเฉลี่ยจากตัวอย่างจำนวนมากๆ ดังตัวอย่างนี้ใช้ค่าตัวอย่างจำนวน 1000 ค่า

รอบการทำงาน	เวลา
1	0:00:00.02
2	0:00:00.03
3	0:00:00.01
4	0:00:00.05
5	0:00:00.02
6	0:00:00.03
7	0:00:00.04
8	0:00:00.01
9	0:00:00.03
10	0:00:00.02
.	.
.	.
1000	0:00:00.04

จากการทดลองได้พิมพ์ค่าเวลาดังกล่าวออกมาจำนวน 1000 ค่าและให้คำนวณค่าเฉลี่ยซึ่งในการทดลองจะได้ค่าเฉลี่ยคือ 35 มิลลิวินาที

ในวิทยานิพนธ์ฉบับนี้ได้ทดลองหาค่าเวลาตอบสนองของระบบด้วยวิธีทั้ง 3 วิธีที่กล่าวมาจากการทดลองพบว่าเมื่อมีการเปลี่ยนแปลงแก้ไขโปรแกรมเงื่อนไขการแสดงผลหรือมีการเปลี่ยนภาพกระบวนการไป ก็พบว่าวิธีที่ 2 และ 3 มีค่าใกล้เคียงกัน ส่วนวิธีที่ 1 มีค่าผิดพลาดสูงมากส่วนใหญ่เนื่องมาจากการประมาณชุดคำสั่งในภาษาชั้นสูงเป็นภาษาเครื่องนั้นยุ่งยากและมีความผิดพลาดสูง เพราะชุดคำสั่งในภาษาปาสคาลมีเป็นจำนวนมากและในแต่ละชุดคำสั่งก็ใช้ชุดคำสั่งภาษาเครื่องไม่เท่ากันขึ้นอยู่กับเงื่อนไขและตัวแปรที่นำมาทำเงื่อนไข ดังนั้นในการประมาณค่าเวลาการตอบสนองของระบบการใช้วิธีที่ 3 เป็นวิธีที่เหมาะสมมากที่สุด เนื่องจากการประมาณค่าเวลาตอบสนองของระบบนี้จะใช้ในการพัฒนาโปรแกรมเงื่อนไขการแสดงผลและออกแบบระบบเท่านั้น ดังนั้นวิธีที่ 3 จะทำให้เสียเวลาในการหาหรือประมาณค่าดังกล่าวนี้น้อยที่สุด

ตัวอย่างการสร้างโปรแกรมเงื่อนไขการแสดงผล

โปรแกรมเงื่อนไขการแสดงผลจะแตกต่างกันไปตามกระบวนการ ในตัวอย่างนี้จะใช้กระบวนการตัวอย่างที่ใช้ในบทที่ 6

ก่อนที่จะเริ่มพัฒนาโปรแกรมเงื่อนไขการแสดงผลนั้นผู้ใช้จะต้องทำการแปลชุดคำสั่งการวาดภาพกระบวนการในแต่ละชั้นเสียก่อนตามตัวอย่างในบทที่ 6 ผลที่ได้จากการแปลจะเป็นชุดคำสั่งในภาษาปาสคาล

การพัฒนาโปรแกรมเงื่อนไขการแสดงผลนั้นจำเป็นจะต้องใช้ภาษาปาสคาล และใช้ตัวแปลภาษาเทอร์โบปาสคาล เวอร์ชั่น 4.0 (TURBO PASCAL V4.0) เนื่องจากโปรแกรมส่วนอื่นๆทั้งหมดเป็นภาษาปาสคาลทั้งหมด และคำสั่งที่ใช้ก็จำเป็นต้องใช้เทอร์โบปาสคาลในการแปล ดังนั้นผู้ใช้จะต้องสามารถเขียนโปรแกรมภาษาปาสคาลและรู้วิธีใช้เทอร์โบปาสคาล

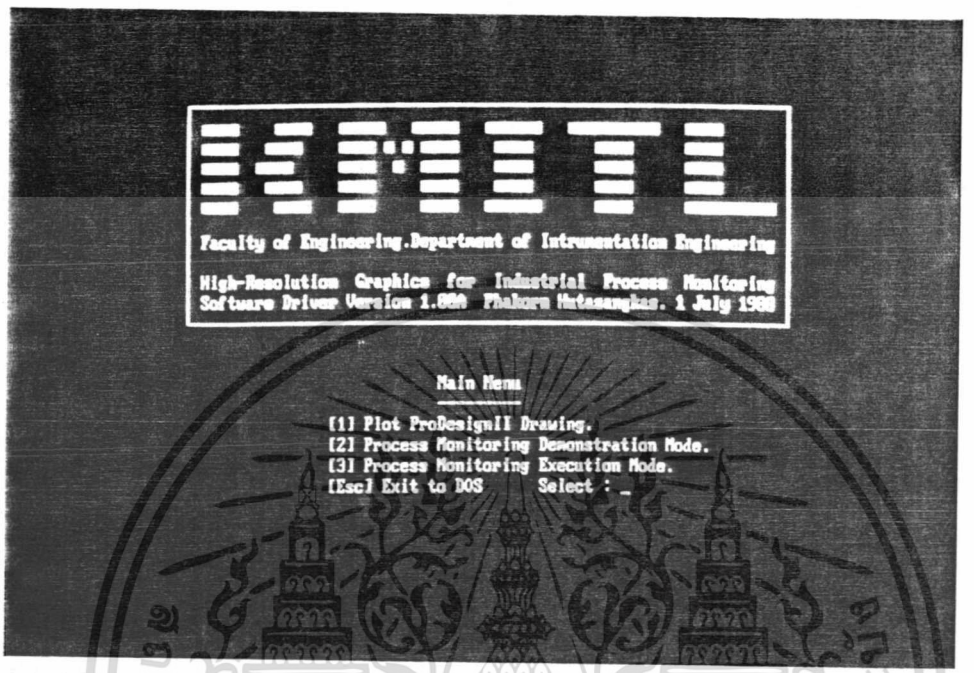
โปรแกรมเงื่อนไขการแสดงผลจะเป็นส่วนหนึ่งของโปรแกรมหลัก ซึ่งเรียกว่ายูนิท (UNIT) การตั้งชื่อยูนิทนี้จะต้องใช้ชื่อว่า user ถ้ากำหนดเป็นชื่ออื่นจะต้องแก้ไขในโปรแกรมหลักด้วย ส่วนการ uses ยูนิทใดนั้นขึ้นอยู่กับโปรแกรมว่าต้องเรียกใช้ยูนิทใดบ้าง ในตัวอย่างนี้จะต้องเรียกใช้ Dos,Crt,Printer,Grph7220 ซึ่ง Grph7220 เป็นยูนิทสำหรับคำสั่งควบคุมจอแสดงผลความละเอียดสูง นอกจากยูนิทเหล่านี้แล้วก็จะต้องมียูนิทซึ่งเกิดจากการเปลี่ยนชุดคำสั่งในการสร้างภาพกระบวนการ เช่น PIPE VALVE เป็นต้น ดังนั้นในตอนต้นของยูนิทนี้จะมีลักษณะดังนี้

```
unit User;
interface
uses Dos,Crt,Printer,Graph7220,Valve,Pipe,Tank,Can;
```

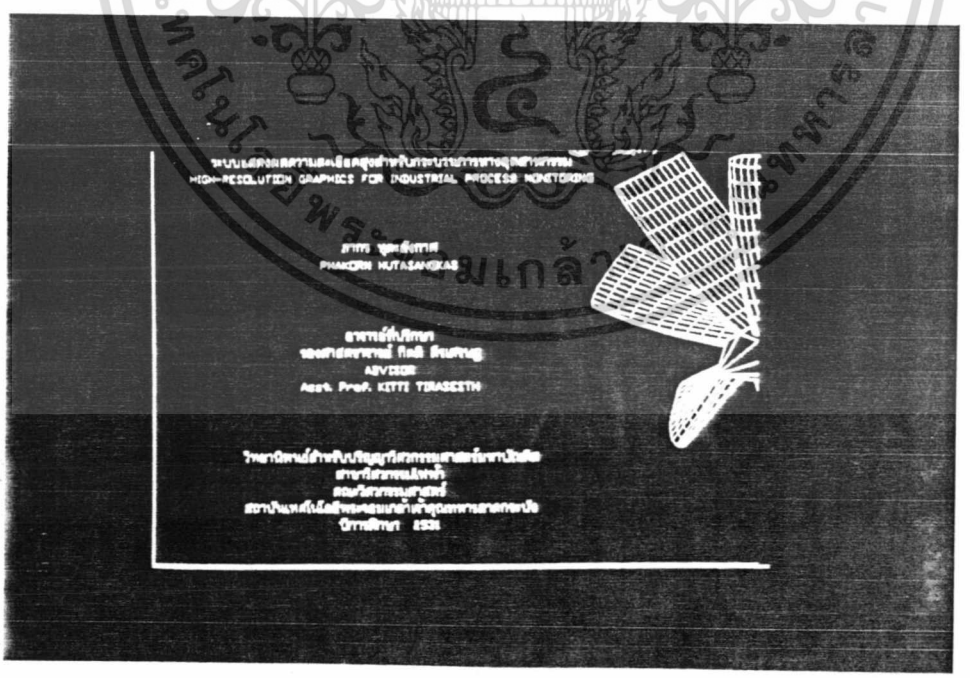
จากนั้นก็จะเป็นการกำหนดความถี่และช่วงเวลาในการทำงานของงานต่างๆ แต่ในครั้งแรกอาจจะยังไม่มีข้อมูลเพียงพอที่จะกำหนด เนื่องจากการกำหนดความถี่และช่วงเวลาจะขึ้นอยู่กับโปรแกรมและงานเป็นสำคัญ ดังนั้นจึงควรกำหนดให้ทุกงานมีความถี่และช่วงเวลาเท่ากันหมดก่อน เมื่อเขียนโปรแกรมไปช่วงหนึ่งแล้วสามารถทดสอบโปรแกรมได้ก็จะพิจารณาได้จากช่วงเวลาในการทำงานและการตอบสนองของระบบ แล้วจึงนำข้อมูลเหล่านี้มาพิจารณาในการกำหนดความถี่และช่วงเวลาดังกล่าว

ต่อจากนั้นก็จะเป็นส่วนของเงื่อนไขในการแสดงผลของกระบวนการซึ่งในส่วนนี้จะไม่อธิบายเนื่องจากจะแตกต่างกันตามกระบวนการและความต้องการของผู้ใช้ แต่จากตัวอย่างโปรแกรมในภาคผนวก ง. นั้นได้เขียนโปรแกรมย่อยที่จำเป็นสำหรับให้ผู้ใช้เรียกไปใช้งานได้ทันที หรือนำไปพัฒนาต่อ เช่น การอ่านค่าเวลาและวันที่ การสั่งพิมพ์ตามกำหนดเวลา การสั่งพิมพ์เมื่อมีการเปลี่ยนแปลงของสภาวะ การควบคุมการเลื่อนภาพด้วยคันโยก และอื่นๆ

จากรูปที่ 7.5 เป็นโปรแกรมเงื่อนไขการแสดงผลที่ได้ทดลองพัฒนาขึ้น โดยรูปดังกล่าวเป็นเมนูหลัก จะมีการทำงานให้เลือกอยู่ 3 แบบ คือ แบบที่ 1 จะเป็นการอ่านข้อมูลจากไฟล์มาทำการวาดอย่างเดียวไม่มีการประมวลผลหรือทำเงื่อนไขอื่น ดังในรูปที่ 7.6 - 7.9 ทั้ง 4 รูปเป็นข้อมูลที่อยู่ในไฟล์เดียวกันแต่ใช้วิธีเลื่อนภาพไปมาด้วยคันโยก ข้อมูลภาพเหล่านี้ได้มาจากโปรแกรม AUTOCAD ซึ่งถ้านำมาแสดงผลด้วยระบบแสดงผลความละเอียดสูงจะสามารถแสดงได้ครั้งละ 4 ภาพหรือมากกว่า สำหรับแบบที่ 2 เป็นการแสดงการติดตามค่าสภาวะของกระบวนการโดยค่าสภาวะต่างๆนั้นได้มาจากการควบคุมด้วยแป้นพิมพ์หรือทำงานตามเวลา เพื่อใช้ทดสอบหรือแสดงการทำงานของโปรแกรม หรือใช้ในการตรวจสอบความถูกต้องในการเปลี่ยนแปลงค่าสภาวะโดยไม่ต้องต่อกับกระบวนการจริง ดังในรูปที่ 7.10 - 7.13 ส่วนในแบบที่ 3 ก็มีลักษณะเดียวกันกับ แบบที่ 2 แต่เป็นการใช้งานจริงมีการอ่านค่าสภาวะมาจากกระบวนการจริง นอกจากการทำงานที่ได้กล่าวมาแล้วก็ยังมีการแสดงผลที่จอแสดงผลของคอมพิวเตอร์ควบคุมระบบด้วย ดังในรูปที่ 7.14 เป็นการแสดงค่าสภาวะโดยใช้ข้อความและแสดงค่าปริมาณออกมาเป็นตัวเลข นอกจากนั้นก็ยังมีการแสดงข้อความสำหรับสัญญาณเตือนอีกด้วย อนึ่งรูปแบบการทำงานเหล่านี้ผู้ใช้ไม่จำเป็นจะต้องใช้ในลักษณะนี้เสมอไป เพียงแต่เป็นแนวทางในการพัฒนาระบบของผู้ใช้เองเท่านั้น

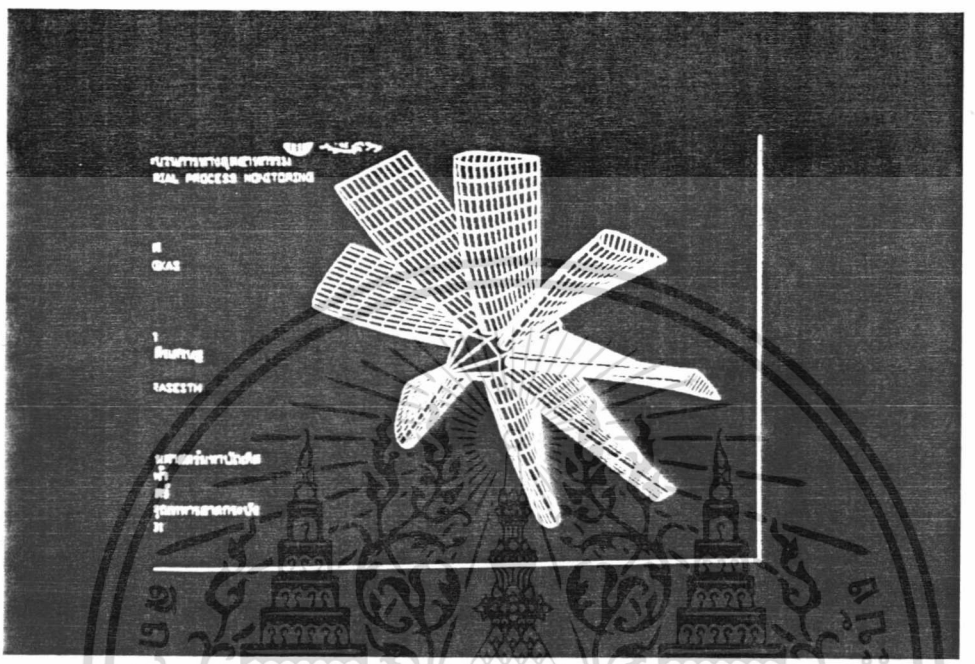


รูปที่ 7.5 เมนูหลักของโปรแกรม

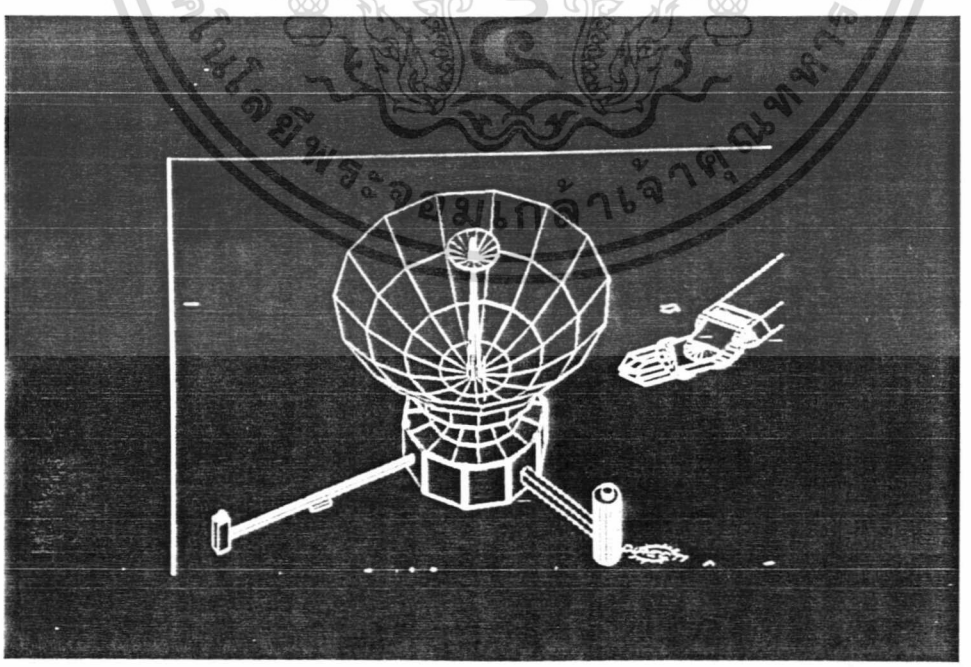


รูปที่ 7.6 ภาพจากไฟล์ข้อมูลของ AUTOCAD (1)

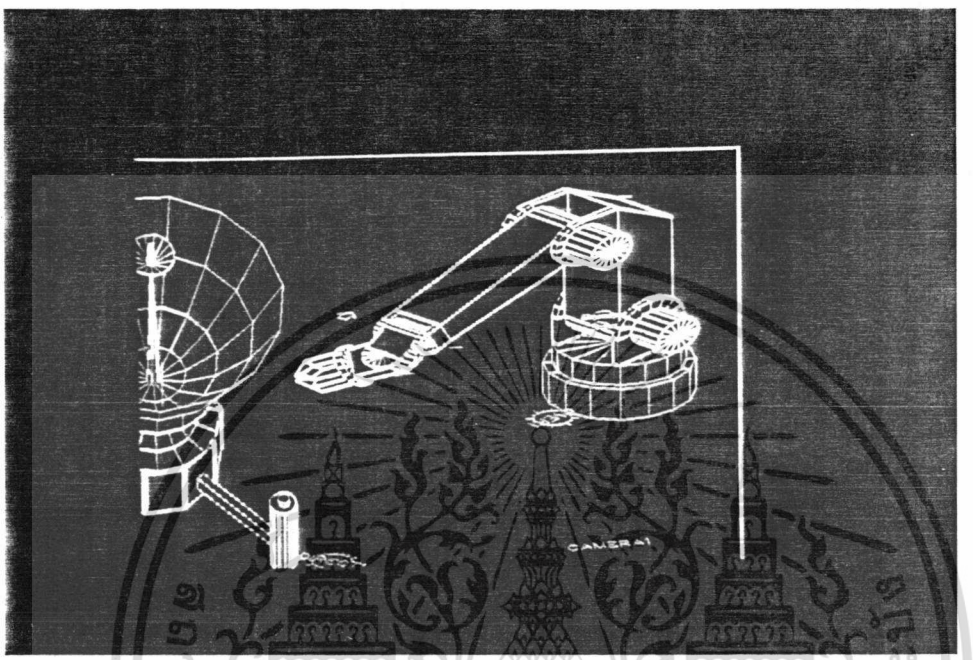
เมื่อนำมาแสดงในระบบฯจะแสดงได้ครั้งละ 4 ภาพหรือมากกว่า



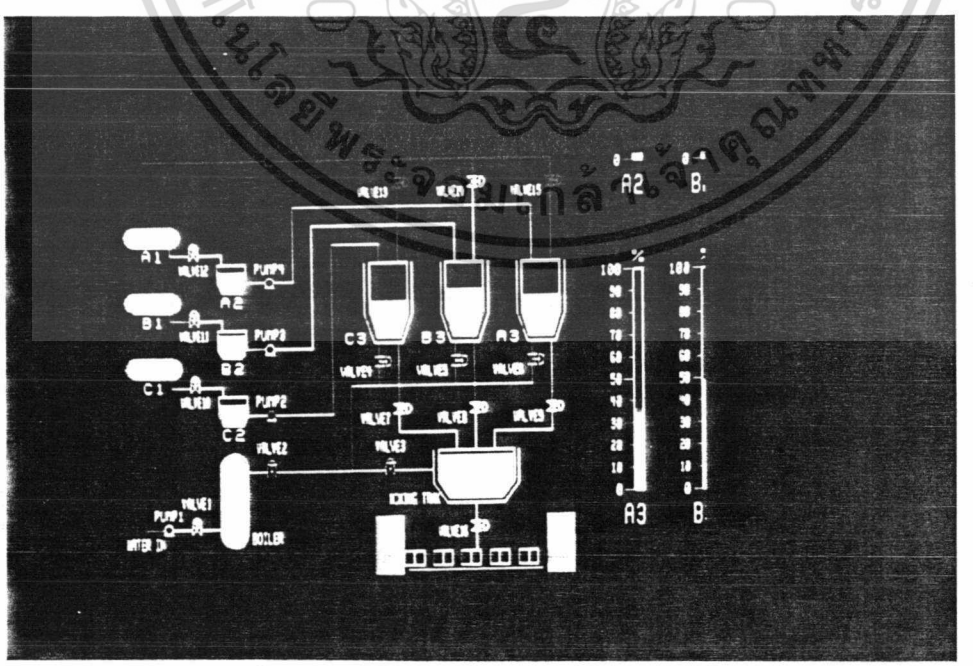
รูปที่ 7.7 ภาพจากไฟล์ข้อมูลของ AUTOCAD (2)



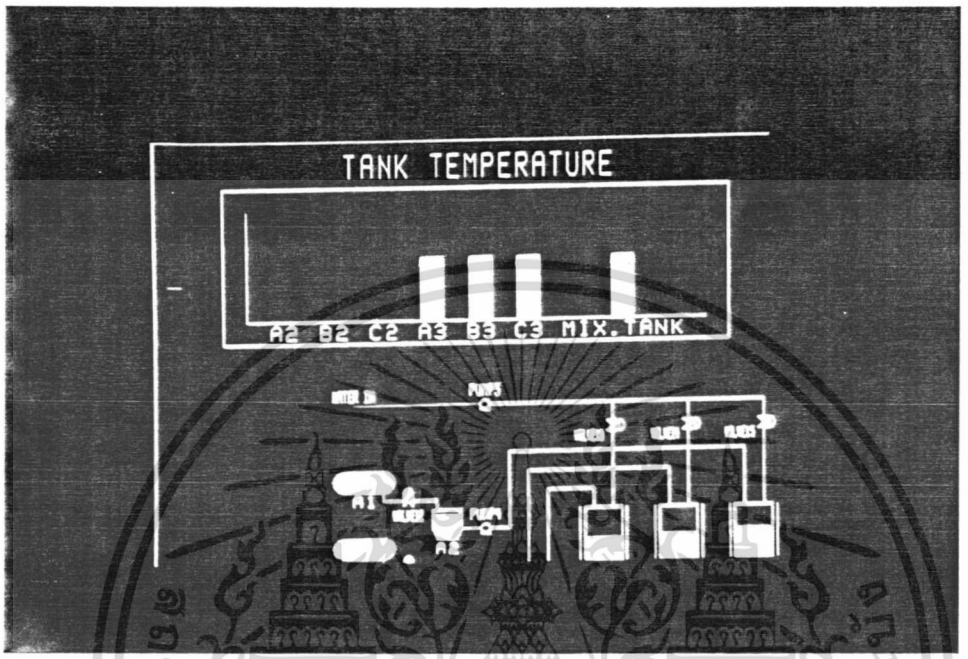
รูปที่ 7.8 ภาพจากไฟล์ข้อมูลของ AUTOCAD (3)



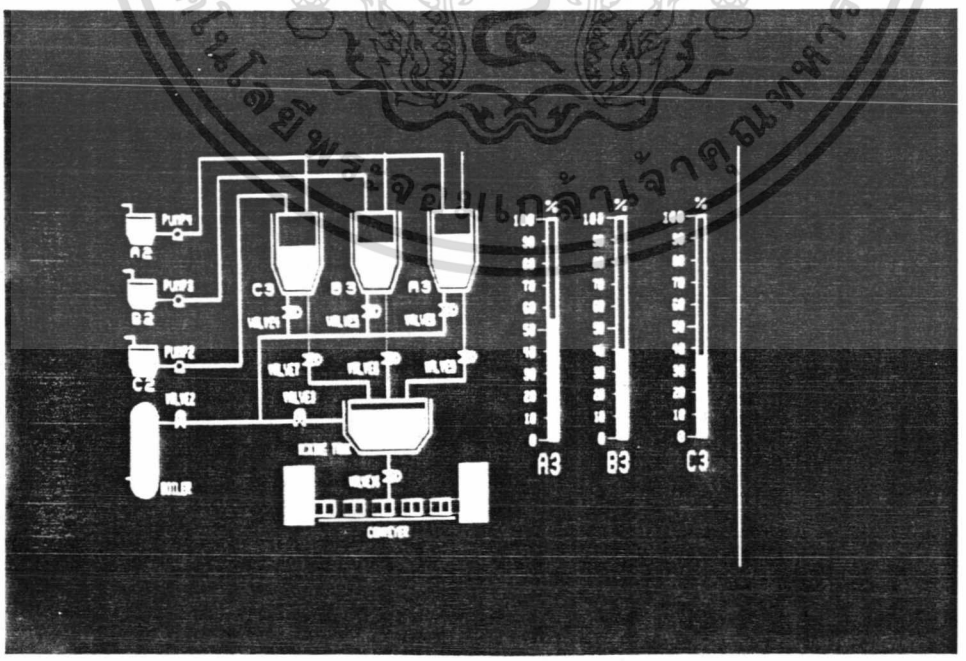
รูปที่ 7.9 ภาพจากไฟล์ข้อมูลของ AUTOCAD (4)



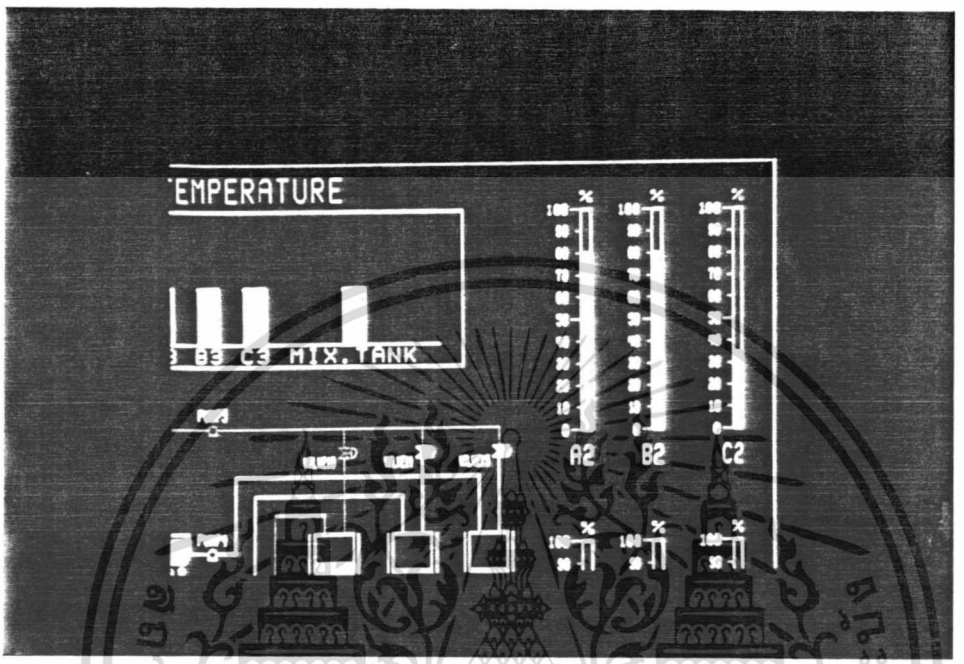
รูปที่ 7.10 การแสดงผลในการติดตามค่าสภาวะของกระบวนการ (1)



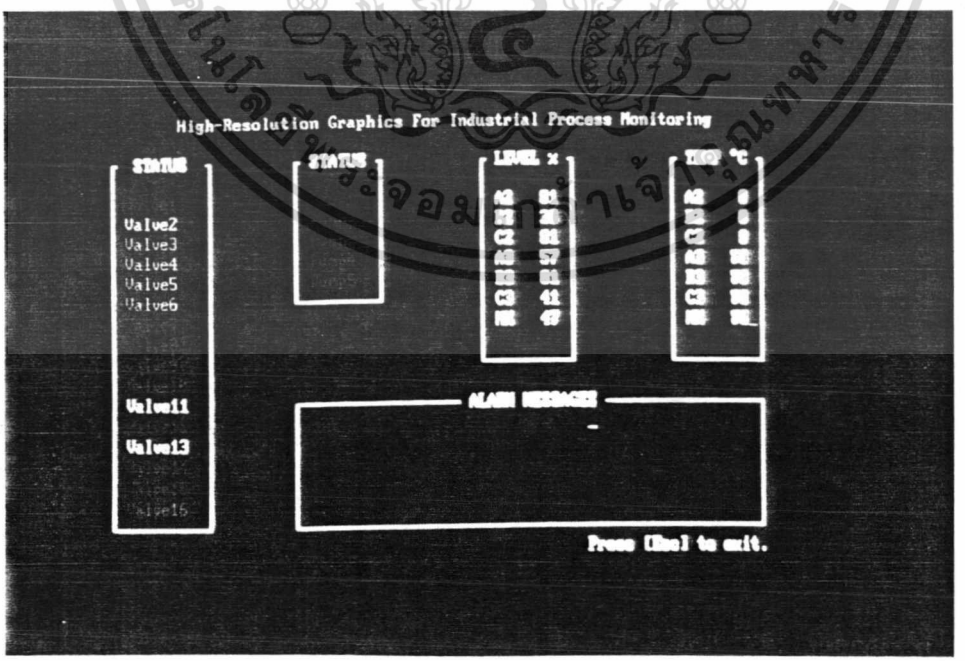
รูปที่ 7.11 การแสดงผลในการติดตามค่าสภาวะของกระบวนการ (2)



รูปที่ 7.12 การแสดงผลในการติดตามค่าสภาวะของกระบวนการ (3)



รูปที่ 7.13 การแสดงผลในการติดตามค่าสภาวะของกระบวนการ (4)



รูปที่ 7.14 การแสดงผลบนจอแสดงผลของคอมพิวเตอร์ควบคุมระบบ

บทที่ 8

บทสรุป

ระบบแสดงผลความละเอียดสูงสำหรับกระบวนการทางอุตสาหกรรมนี้ได้ออกแบบมาเพื่อให้นำไปประยุกต์ใช้งานในอุตสาหกรรมได้อย่างสะดวก โดยการรับค่าสภาวะต่างๆทั้งชนิดดิจิตอลซึ่งรับค่าแรงดันอินพุทได้ตั้งแต่ 0 ถึง 25 V. และอนาลอกซึ่งเป็นแบบลูปกระแส 4 ถึง 20 mA ก็จะเป็นมาตรฐานซึ่งอุปกรณ์การวัดและควบคุมในงานอุตสาหกรรมส่วนใหญ่จะมีรูปแบบของสัญญาณหรือข้อมูลในการวัดและควบคุมกระบวนการในลักษณะนี้ทั้งสิ้น ทางด้านการใช้งานนั้นอาจจะรู้สึกว่าใช้งานยากในส่วนการสร้างภาพ ซึ่งโปรแกรมลักษณะนี้ในท้องตลาดมีอยู่ด้วยกันหลายโปรแกรม จากการศึกษาแล้วพบว่าการใช้งานของโปรแกรมเหล่านี้ใช้งานได้ค่อนข้างง่าย แต่เมื่อนำไปใช้ในงานจริงแล้วการยืดหยุ่นของระบบต่ำ เช่น ภาพที่แสดงบนจอแสดงผลเป็นภาพกราฟฟิคความละเอียดต่ำ ดังนั้นเมื่อกระบวนการมีขนาดใหญ่ก็ไม่สามารถสร้างภาพได้ ภาพที่แสดงบนจอแสดงผลมักจะเป็นแบบหยาบๆ เนื่องจากความละเอียดของจอแสดงผลและส่วนควบคุมต่ำ และเมื่อต้องการให้ประมวลผลข้อมูลบางตัวที่รับเข้ามาจากกระบวนการก็ทำได้จำกัด บางโปรแกรมก็ทำไม่ได้เลย จากการสอบถามผู้ใช้ซึ่งใช้งานระบบติดตามและแสดงผลค่าสภาวะของกระบวนการทางอุตสาหกรรมแล้วส่วนใหญ่จะมีความเห็นว่าการสร้างภาพหรือผังภาพของกระบวนการนั้นผู้ใช้จะสร้างภาพของกระบวนการเพียงครั้งเดียวจากนั้นก็เป็นการใช้งานหรือการพัฒนาทางด้านอื่นๆ เช่นทางด้านเอาท์พุท การพิมพ์รายงาน การประมวลผลข้อมูลที่ได้ ดังนั้นความต้องการส่วนใหญ่คือความละเอียดของภาพกระบวนการบนจอแสดงผล ความเร็วและ ความสามารถในการแสดงผล ความสามารถทางด้านการประมวลผลทางคณิตศาสตร์ และ การควบคุมการพิมพ์ เป็นต้น ผู้ใช้ส่วนใหญ่มักจะพัฒนาโปรแกรมขึ้นเอง ดังนั้นปัญหาสำคัญของผู้ใช้ไม่ใช่ความยากง่ายในการใช้งานหรือการสร้างภาพ หากแต่ขึ้นอยู่กับความสามารถและความยืดหยุ่นของระบบเป็นสำคัญ เนื่องจากผู้ใช้สามารถออกแบบและพัฒนาระบบได้ด้วยตนเอง ทำให้ผลลัพธ์ที่ได้ตรงกับความต้องการใช้งานอย่างแท้จริง

ปัญหาที่พบในการทำวิทยานิพนธ์ครั้งนี้ เมื่อเริ่มต้นส่วนใหญ่เป็นปัญหาทางด้านการหาข้อมูล เนื่องจากระบบในลักษณะนี้ในประเทศไทยมีผู้ใช้อยู่ในวงจำกัด ทางด้านผู้แทนจำหน่ายส่วนใหญ่ก็เป็นระบบใหญ่ การใช้งานระบบเล็กในลักษณะเดียวกันมักจะเป็นการออกแบบเฉพาะงานและไม่มีการยึดมาตรฐานใด การสร้างภาพก็เป็นลักษณะการพัฒนาโปรแกรมซึ่งยุ่งยากและเสียเวลานาน ดังนั้นในวิทยานิพนธ์นี้จึงได้กำหนดเอามาตรฐานตามการควบคุมกระบวนการทางอุตสาหกรรมทั่วๆไป และใช้โปรแกรมสำเร็จรูปเป็นหลัก

กิตติกรรมประกาศ
(ACKNOWLEDGMENT)

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ก็ด้วยความร่วมมือและความช่วยเหลือจากหลายฝ่ายด้วยกัน โดยเฉพาะอย่างยิ่งคุณพ่อสัญญา คุณแม่สอาด หุตะสังกาศ รวมทั้งรองศาสตราจารย์กิตติ ตีรเศรษฐ ผู้ซึ่งเป็นอาจารย์ที่ปรึกษาและคุณสุทธินันท์ ปรมาธิกุล คุณวิริยะ กองรัตน์ คุณสุพรรณ กุลพาณิชย์ คุณประภาษ อุคคกิมาพันธุ์ และคุณบุษบง เจริญพันธ์โยธิน จึงขอขอบพระคุณท่านทั้งหลายไว้ ณ ที่นี้ด้วย

เอกสารอ้างอิง
(REFERENCES)

1.ภากร หุตะสังกาศ,กิตติ ตีรเศรษฐ, ไมโครคอมพิวเตอร์กับเครื่องควบคุมที่โปรแกรมได้ (MICROCOMPUTER LINK TO A PROGRAMMABLE CONTROLLER), การประชุมทางวิชาการวิศวกรรมไฟฟ้าสถาบันอุดมศึกษาแห่งประเทศไทย ครั้งที่ 9, 2529

2.ภากร หุตะสังกาศ,กิตติ ตีรเศรษฐ, การประยุกต์ใช้ไมโครคอมพิวเตอร์ในการจำลองการทำงานของเครื่องควบคุมที่โปรแกรมได้ (MICROCOMPUTER APPLICATION IN PROGRAMMABLE CONTROLLER SIMULATION), การประชุมวิชาการวิศวกรรมไฟฟ้า ครั้งที่ 10, 2530

3.Denis Jump, PROGRAMMER'S GUIDE TO MS-DOS, Reston Publishing Company, Inc., 1984

4.Lewis C. Eggebrecht, INTERFACING TO THE IBM PERSONAL COMPUTER, Howard W. Sams & Co., Inc., 1983

5.TURBO PASCAL VERSION 3.0 REFERENCE MANUAL, Borland International Inc., 1985

6.TURBO PASCAL VERSION 4.0 REFERENCE MANUAL, Borland International Inc., 1988

7.DISK OPERATING SYSTEM VERSION 3.20 REFERENCE, Programming Family, IBM Corp. & Microsoft Inc., 1986

8.IBM PERSONAL COMPUTER HARDWARE REFERENCE LIBRARY, Technical Reference, IBM Corp., 1983

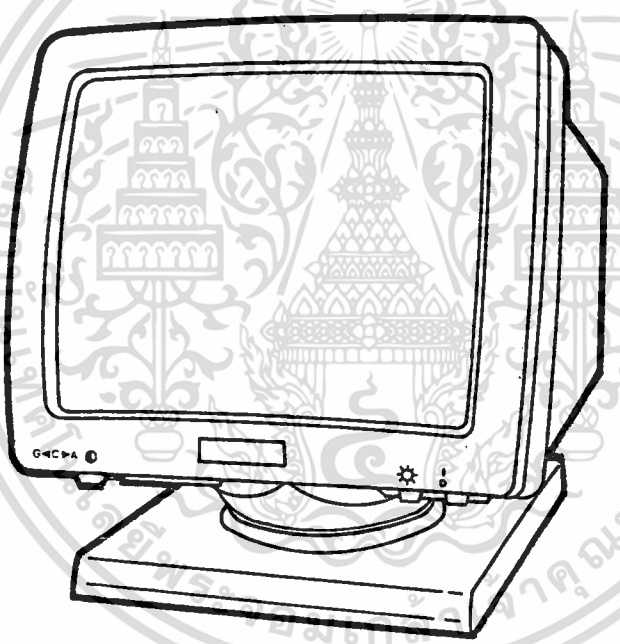
9.MICROSYSTEM COMPONENTS HANDBOOK VOLUME 2, Intel Corp., 1984

10.ARTIST'S OWNERS INSTRUCTION MANUAL, Control System, Inc., 1984

ภาคผนวก 1

รายละเอียดเกี่ยวกับจอแสดงผล ADI รุ่น DM-2214

COLOR MONITOR
USER MANUAL



MODEL DM-2214

I. INTRODUCTION

1.1 FCC Information

The Federal Communications Commission Radio Frequency Interference Statement includes the following warnings.

1. Use the attached specified shielded power supply cable and the shielded signal cable with DM-2214 so as not to interfere with radio and television reception.
2. This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, interference with radio and television may occur. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation.If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:
   —relocate the receiving antenna
   —relocate the computer with respect to the receiver
   —move the computer away from the receiver
   —plug the computer into a different outlet so that the computer and receiver are on different branch circuits.

   If necessary, the user should contact the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:
   "How to Indentify and Resolve Radio-TV Interference Problems." This booklet is available from the US Government Printing Office, Washington, D.C., 20402. Stock No. 004-000-00345-4.

1.2 General

DM-2214, a high resolution color monitor that designed for display graphic and textural output from IBM or IBM compatible personal computer. With two working frequencies of 15.75 KHz & 21.85 KHz, the monitor can meet the requirements of many color graphic adaptors such as IBM Color Graphic Adaptor and IBM Enhanced Graphic Adaptor. DM-2214, can also be used with other IBM compatible adaptors It provides users not only monochrome colors in amber and green but also 64full colors generated from RGB TTL signals in different intensities.

1.3 Features of DM-2214

—Displays "monochrome display quality" (8 x 14 character box) in text in color
—Provides much clearer & sharper picture in much more colors
—Operates with the IBM Enhanced Graphics Adaptor, and Color Graphics Adaptor
—Uses a high-contrast, non-glare larger screen
—A color/green/amber control knob is provided on the front panel
—Provides the ergonomic viewing by adjusting the tilt/swivel base

1.4 Cautions when using DM-2214

When setting up and using DM-2214, pay special attention to these points:

—The monitor carries high voltages in operation, the power cord should always be unplugged before the cover is removed. Some voltages may remain present in internal circuits after power-off. Please request a qualified electric technician taking care of inside monitor maintenance.
—Allow adequate ventilation all around DM-2214 so that heat from the monitor can properly dissipate.

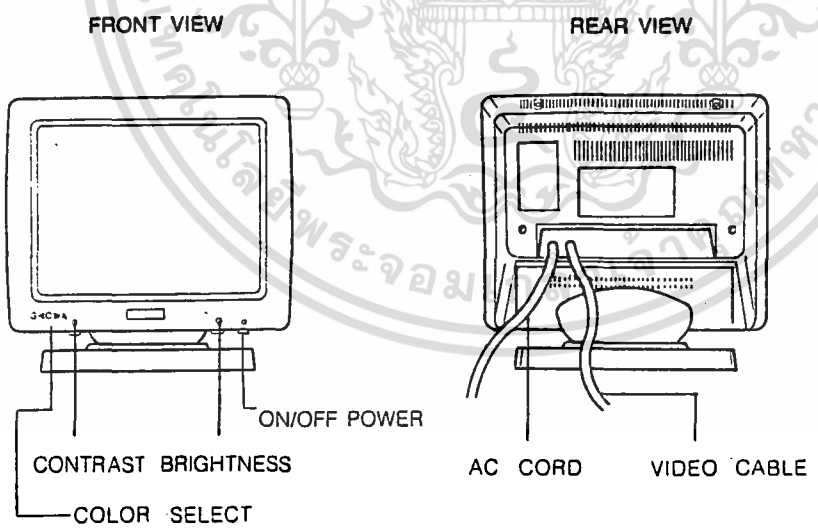
—Don't rest DM-2214 or other heavy objects on the power cord. A damaged power cord can cause fires or electrical shocks.
—Keep DM-2214 away from high capacity transformers, electric motors and other strong magnetic fields.
—Don't use DM-2214 in damp, dusty, or dirty places.

II. USING DM-2214

2.1 After opening the package carton, you will find a tilt and swivel base on top of the monitor. Be sure that your DM-2214 includes all the items as listing below:
a. DM-2214
b. Swivel/Tilt Base: Enables you to position DM-2214 at the best angle and tilt for easy viewing
c. User's Manual

2.2 Connecting DM-2214
1. Make sure the power of DM-2214 and the PC are turned off.
2. Connect the power cord, and plug the D shape signal input connector in PC/XT or AT. Make sure these connectors are well and tightly connected.
3. You may test the monitor by running a program on personal computer. First, select desired display color by turning color-selection switch. Then, turn both personal computer and monitor on, it takes a few seconds for monitor warm-up and showing screen display. After screen picture comes out, then you can turn the contrast adjustment knob to choose the most comfortable picture.

2.3 Adjusting DM-2214 Controls
Please familiarize yourself with the switches and controls that give DM-2214 all its capabilities locations of control knobs, switches and connectors are shown on the following front and rear panel diagrams:



- **LED LAMP**
  This is an indicator for power ON/OFF. When power of monitor is turned on, the LED lamp will lighten up.
- **COLOR-SELECTION**
  This switch is used to select desired display color. There are three alternatives, green, amber and color for choosing. You may slide the knob to desired position to select the wished color. Green is on the right side of the switch; color is in the middle and amber is on the left side.
- **CONTRAST ADJUSTMENT:**
  The contrast knob is pre-set by the manufacturers at switch off position. A standard color picture of full colors will be appeared on the screen to fit users for general room lighting condition. However, you can slide the knob to the right side to increase the contrast, to the left side to decrease it.
- **POWER ON/OFF BUTTON**
  This button is used for power connection. Push the button down for power on. and repush it up turns power off.
- **BRIGHTNESS CONTROL**
  This control is used for obtaining the most comfortable brightness on screen picture. Slide the control knob, to the right side to increase the picture brightness and to the left side to decrease the brightness.

SIGNAL-IN CONNECTOR

The 9 Pin D Shape Signal input connector connects the video card personal computer and monitor enabling effective delivery of RGB separate signal input. The monitor will not work properly unless the cable directs correct signal to each and every pin.

The connector model is illustrated as follows:

PIN 1 Chassis ground
PIN 2 Red Intensity
PIN 3 Red
PIN 4 Green
PIN 5 Blue
PIN 6 Green Intensity
PIN 7 Blue Intensity
PIN 8 Horizontal Sync.
PIN 9 Vertical Sync.

POWER CORD

The cord transmits AC power into the monitor for operation. Make sure to plug the cord into power connector before using.otherwise, monitor will not function.

III. TROUBLE SHOOTING

Before calling an authorized service center please check that the following items are properly connected and adjusted.

Problem	Check These Items Controls Switches	Location
No picture	Power switch D shape signal connector	Front Cable
Abnormal picture 1) Display scrolls up or down	V-Hold control	Rear
2) Display is too large or too small	V-Size control	Rear
3) Only one color appears	Color-Selection D shape signal connector	Front Cable
4) Brown color disappeared	Contrast control to normal position	Front

APPENDIX A SPECIFICATION

1. Cathode Ray Tube
   Pictch: 0.31 mm
   Size: 14 inch
   Deflection Angle: 90 degree
   Neck Diameter: 29.1 mm
   Face Treatment: Tint & Non-Glare
   Phosphor: P22, short persistence
2. Power Requirements
   Power Source: 110/220 VAC manual switch
   Power Consumption: 90 W max.
3. Deflection Characteristic
   Horizontal Frequency: 15.75/21.843 KHz Auto SW
4. Video Response
   Bandwidth: 20 MHz (−3dB)
   Rise Time: 20 ns
   Fall Time: 20 ns
   Colors: 64 plus amber and green monochrome switches
5. Display Format
   Horizontal resolution: 640 dots (maximum 770)
   Vertical resolution: 220/350 lines
   Character Format: 8 × 8/8 × 14 Matrix
   Capacity: 80 character × 25 Rows
   Total display characters: 2000
6. Input Signal
   Type: R.G.B. TTL LEVEL POSITIVE
   Synch: TTL separate

7 Display Performance
Picture Size: Horizontal-250mm + – 3mm
Vertical-180mm + – 3mm
Linearity: Character height or width will not vary more than 10% from average character size.
Geometry Distortion. (Pin cushion. trapezoidal. barrel)
Horizontal + – 3mm max full screen & + – 1.5mm max half screen
Vertical + – 3mm max
Centering: + – 3mm max

8 Misconvergence:
Center: 0.2mm for 130mm diameter
Corner: 0.4mm

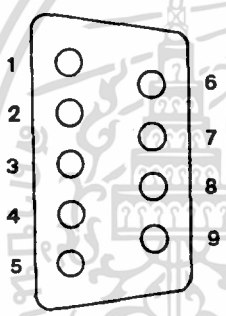
9 Operating Temperature Range: – 20°C to 40°C

10 Stroage Temperature Range: – 40°C to 45°C

11 Dimension: H375mm x L378mm x W361mm
(H14.8" x L14.9" x W14.2")

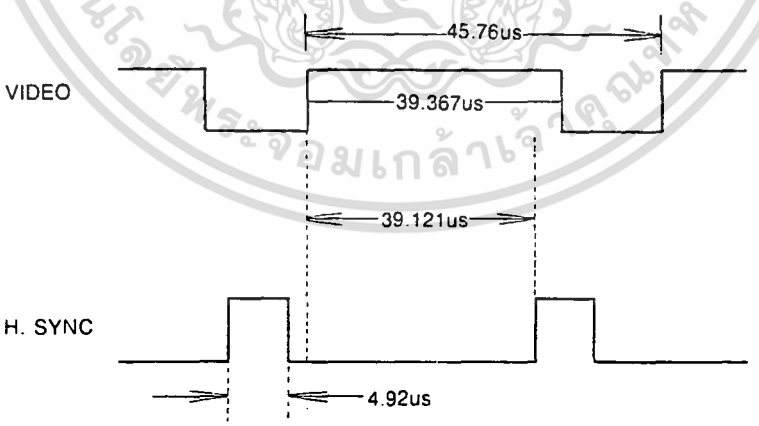
12 Net Weight: 16 Kg (35.2 Lbs)

APPENDIX B PIN ASSIGNMENTS AND SIGNAL LEVELS

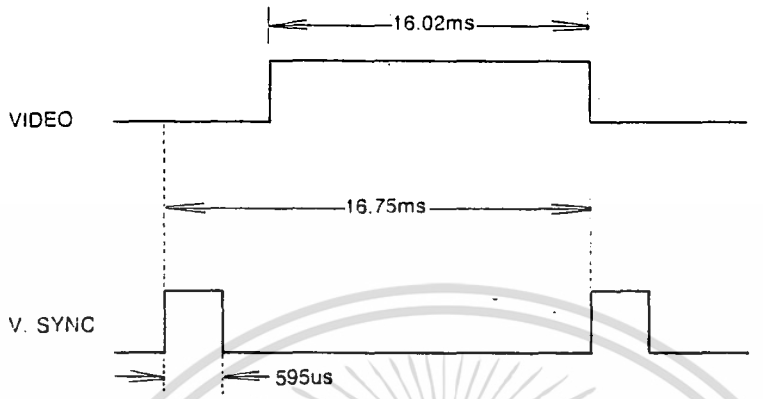


PIN	15.75 KHz	22 KHz
1	GND	GND
2	GND	2ND RED
3	RED	RED
4	GREEN	GREEN
5	BLUE	BLUE
6	INTENSITY	2ND GREEN
7	NC	2ND BLUE
8	H-SYNC	H-SYNC
9	V-SYNC	V-SYNC

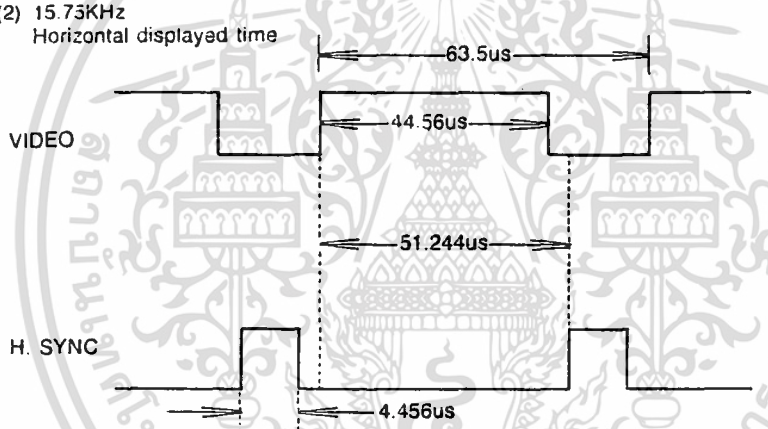
3. Timing chart
(1) 22KHZ
Horizontal displayed time



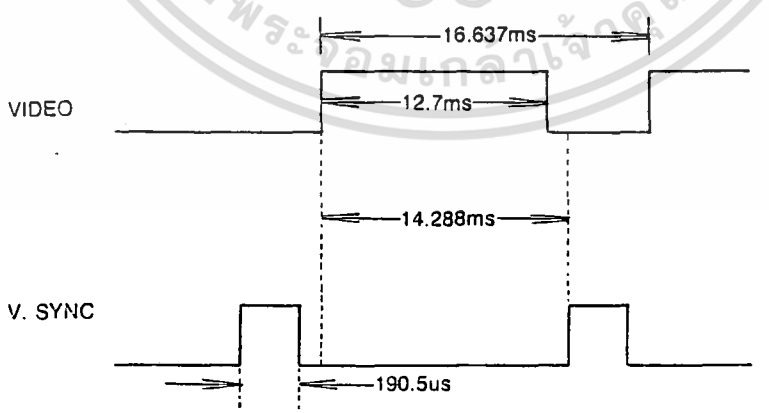
3. Timing chart (continue)
(1) 22KHz
Vertical displayed time



(2) 15.73KHz
Horizontal displayed time



3. Timing chart (continue)
(2) 15.75KHz
Vertical displayed time



ภาคผนวก 2

รายละเอียดเกี่ยวกับไอซี 82720

intel ARTICLE REPRINT AR-255

By managing tasks like graphics generation and CRT refreshing, a dedicated VLSI display controller simplifies the design of intelligent graphics work stations.

Dedicated VLSI chip lightens graphics display design load

The role of graphics is becoming increasingly important for unscrambling the communications traffic between people and computers. Thanks to microprocessors and dedicated control ICs, designing high-reliability graphics work stations is now easier and less expensive than in the days of small-scale integration and expensive discrete-circuit CRT technology. Microprocessors simplify workstation design by transferring some graphics control tasks from hardware to software. However, a dedicated VLSI controller such as the 82720—with an on-board graphics processor—can push another step forward toward fast and economical design of high-quality intelligent graphics systems.

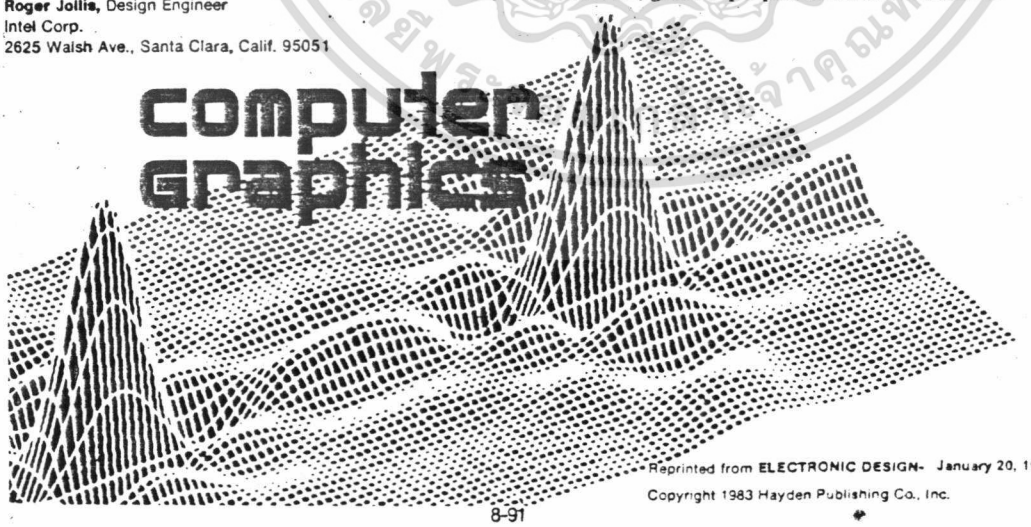
A typical application for the controller is a graphics work station aimed at high-end business and low-end engineering systems. Since such a station usually fits on the top of a desk, all of the electronics must be contained within a single printed-circuit board. This type of system requires a resolution of about 512 by 512 pixels and is frequently called on to display three-dimensional objects in various perspectives. To minimize the distortion of rotating objects, horizontal and vertical pixels should be equally spaced.

A typical display (500 vertices) must be drawn on the screen in less than 1 second to provide satisfactory interaction with the operator. The display may consist of lines, arcs, filled areas, and colors—seven colors are acceptable (see "A Look into Graphics Fundamentals").

Serial link interfaces station

An intelligent work station usually interfaces with a mainframe host via a serial communications link, a keyboard, and a serial link with an optional graphics tablet. This type of graphics input/output subsystem is diagrammed in Fig. 1. Two 5¼-in. floppy disks can satisfy the mass-storage needs of the system. Disk formatting must be compatible with the requirements of an IBM personal computer. Moreover, general-purpose software written for

Gary DePalma, Field Applications Engineer
Mark Olson, Product Marketing Engineer
Roger Jollis, Design Engineer
Intel Corp.
2625 Walsh Ave., Santa Clara, Calif. 95051



Reprinted from ELECTRONIC DESIGN- January 20, 1983

the 82720 has been initialized. Figure 4 shows the complete schematic for each plane of the bit-map interface.

The remainder of the hardware design interfaces the graphics display processor, the processor memory, and the other peripherals with the 80186 microprocessor. The task is simplified by the processor's on-board chip-selection logic and wait-state generators. Furthermore, because of the processor's highly integrated architecture, the size of the overall hardware is quite small.

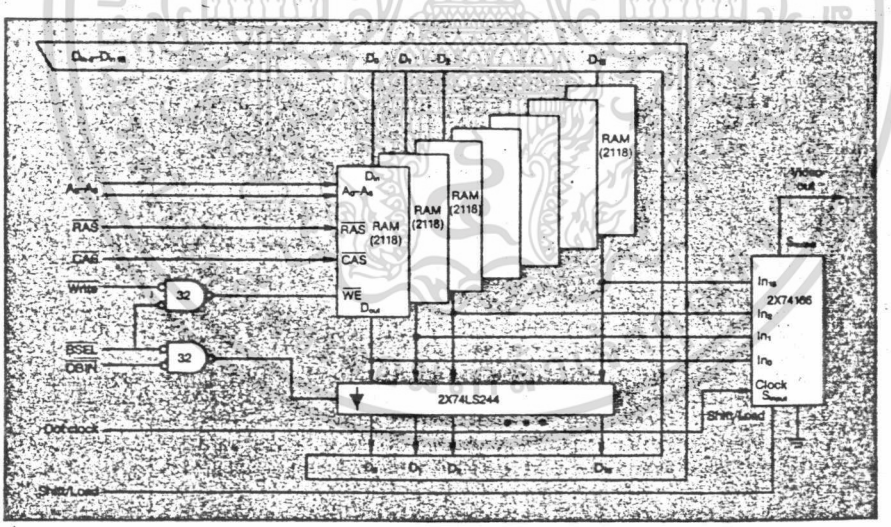
Joining processor and controller

Connecting the graphics display controller to the microprocessor is a simple task, as the processor's Data, Read, and Write signals are completely compatible with those of the 82720. However, because the controller has no chip-selection input, the Read or Write signals must be qualified through external hardware.

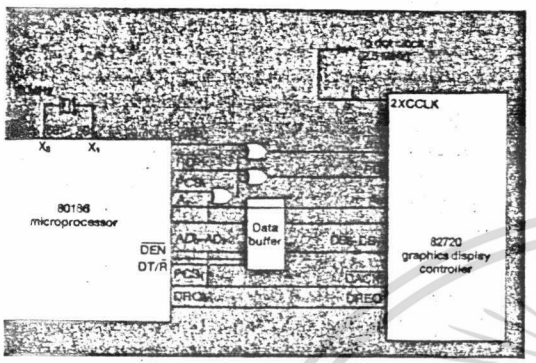
A number of chip-selection lines on the microprocessor can be programmed to place peripherals either in memory or in the processor's I/O space. Two gates are added to qualify the Read and Write signals. The DMA channel on the 80186 uses a second chip-select input as the Acknowledge signal, and data buffers are used to prevent bus contention at the end of a processor read cycle (Fig. 5).

Without buffers, the display controller must remove its data from the multiplexed address and data lines before the processor puts out the next address. At an 8-MHz clock rate, the processor requires that peripherals and memory vacate the bus in less than 85 ns; however, the standard speed of the controller is 100 ns. A faster version, the 82720-1, can be used, but it requires faster memory chips. A more cost-effective solution is simply adding buffers, if board space permits.

Serial communications to both the host and the optional tablet are handled by a multiprotocol serial controller (the 8274), which takes care of the host's synchronous and the tablet's asynchronous



4. The bit-map memory interface contains three address planes (one of which is shown here) to complete the graphics system. The RAS signal for the RAMs is generated by the graphics display controller.

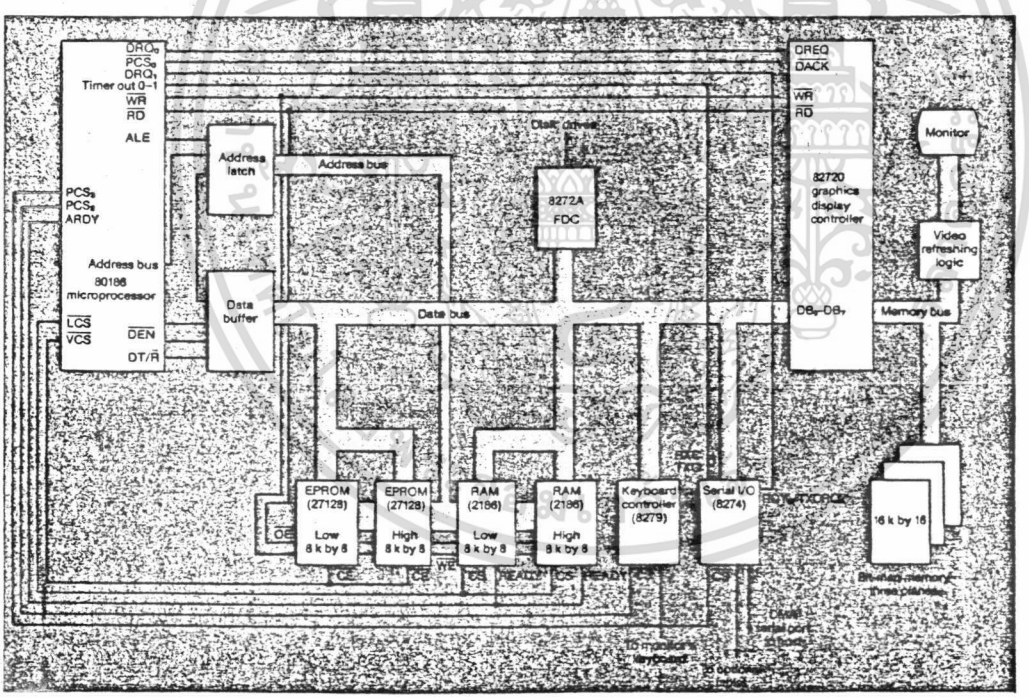


5. The interface between the 82720 and the system microprocessor is simple to implement because all of the processor's signals are compatible with the controller. It is necessary, however, to use external gates to qualify the RD or WR signals.

requirements. Interfacing is accomplished simply by connecting the buffered data bus, the latched lower-address lines, the Read and Write signals, and the chip-select. A final link brings the microprocessor's counter-timer output into the multiprotocol serial controller as a baud-rate clock. No buffering of the TTL support circuitry is necessary.

Universal chip interfaces keyboard

A universal peripheral interface chip (the UPI-42) serves as the keyboard interface and is programmed to scan the keyboard and interrupt the processor only on detection of a valid debounced keystroke. Mass-storage subsystems are managed by the 8272A floppy-disk controller. An external phase-locked loop circuit generates all of the timing signals reequired to connect a 5¼-in. drive to the system. On the microprocessor side, a DMA channel provides the link to the floppy-disk controller. Thus



6. A complete graphics control system is centered around an 80186 microprocessor and the 82720 controller. Local storage is provided by 32 kbytes of EPROM and 16 kbytes of RAM. The system comprises 85 chips and is housed on a single 12-by-12-in. printed-circuit board.

the processor has a high-speed disk interface, which loads it lightly.

To complete the graphics system illustrated in Fig. 6, 32 kbytes of EPROM and 16 kbytes of RAM support the microprocessor's program and display lists. The two EPROMs (27128s) come in 28-pin packages, thereby saving board space.

Hooking up the RAM chips is almost as straightforward. Since the microprocessor is a fully byte-addressable device, it can write bytes as well as words to the RAM. The chip-select input for the low (even) address RAM must be qualified with address A_0 at a logic zero, and the high (odd) address RAM must be qualified by the processor's Byte High Enable signal (BHE). The RAMs, designated 2186, have built-in controllers.

Since dynamic RAMs latch addresses on the leading edge of the chip-select signal, they must be qualified with the processor's Address Latch Enable signal to ensure that selection is made only after the address is valid. Then, a RAM latches the data to be written on the leading edge of the write pulse. The microprocessor's write signal must be delayed by one-half of a clock cycle to guarantee that data is valid at the correct time.

At this point, the design meets all of its performance goals. The system draws lines and circles at about 120,000 bits/s. That is approximately 82,000 pixels for a display consisting of even amounts of the three primary colors, as well as three secondary colors, and white. The 500 vectors of 25 pixels each can be drawn in about 0.15 s, six times faster than the 1-s requirement. The worst case—drawing all lines in white—can be accomplished in about one-third of a second. These specifications are satisfied when the graphics display controller is running from a 2.5-MHz clock. Drawing is performed only during retracing and the 82720 is programmed to use three memory cycles of each horizontal retrace for memory refreshing.

All of the components fit on a board measuring 12 by 12 in., so that the desktop size requirements are satisfied. The electronic components occupy about 100 in.^{2} of the low-cost, double-sided printed-circuit board. □

Bibliography:

Bresenham, J.E., "Algorithm for Computer Control of a Digital Plotter," IBM System Journal, 1965, 4(1) pp. 25-30.

intel

ARTICLE REPRINT

AR-298

Graphics Chip Makes Low-Cost, High Resolution Color Displays Possible

by Mark Olson and Brad May

The making of displays that are both high-resolution and low-cost is the key to producing equipment for both the automated office and the engineering workstation. Through the introduction of 16-bit μPs such as Intel's iAPX 8088, 80186 and 80286, the processing power has been made available to perform very sophisticated functions for the user while making the human interface very simple.

That processing power can be unnecessarily drained, however, if the μP is burdened with the entire task of graphics display. Such a burden can fill up a significant part of the processor's I/O bandwidth, slow down the refresh rate of the display, and decrease the computational power of the CPU.

Intelligent peripheral ICs offload processing tasks from the CPU.

The logical way to avoid such limitations is to dedicate a specialized processor to the handling of display function. It should be capable of accepting high-level commands to minimize the burden on the CPU, as well as optimizing the execution of such commands through raster operations implemented in hardware at the device level.

Such a chip is Intel's 82720 Graphics Display Controller (GDC). It has features that give systems a fast drawing speed while reducing graphics display costs by 60% or more. It achieves these results by taking over the drawing and refresh functions from the CPU, by allowing the use of dynamic RAM's instead of static RAM's, and by reducing the overall parts count needed to create a complete graphics system.

The implementation of the drawing task is a major feature of the GDC. Other graphics chips perform only the display refresh function, leaving the more complicated drawing function entirely to the CPU. Since the CPU is doing every pixel of the drawing function on these systems, they also require faster bit map RAM than with the GDC. The GDC, on the other hand, is capable of handling the drawing function itself, drawing such objects as characters, slanted characters, points, lines, arcs, rectangles, and slanted rectangles based only upon lengths, slopes, and arc centers supplied by the CPU. The GDC's processing, moreover, takes place concurrently with the processing of the CPU.

2048 × 2048 Resolution

With its 4 Megapixel addressability, one GDC can handle a monochrome display with resolution as high as 2048 × 2048, and multiple GDC's can be linked to provide even higher resolution, such as color displays at 2048 × 2048. The chances are, however, that the GDC's full power will not be used in most applications. The typical

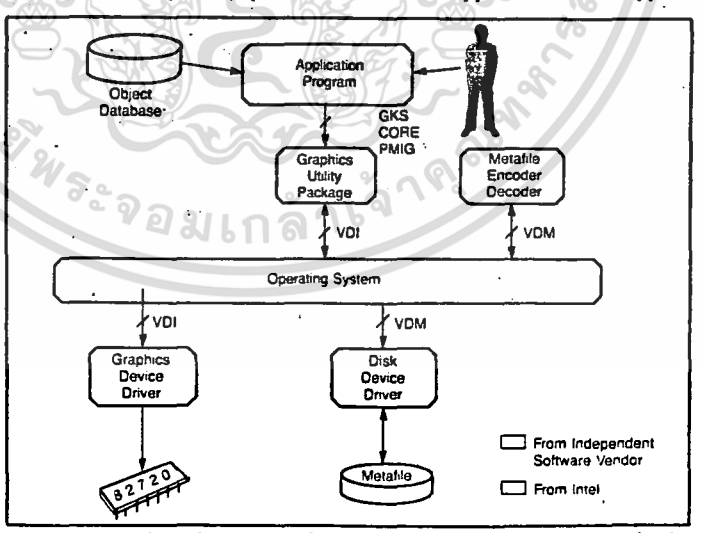


Figure 1: General graphics commands are translated into the VDI interface level and then into driver device commands.

Mark Olson and Brad May are Product Marketing Engineers for Peripheral Components Operation, Intel Corp., Santa Clara, CA 95051.

Reprinted from DIGITAL DESIGN © April 1983, Morgan-Grampian Publishing Company, Boston, MA 02215

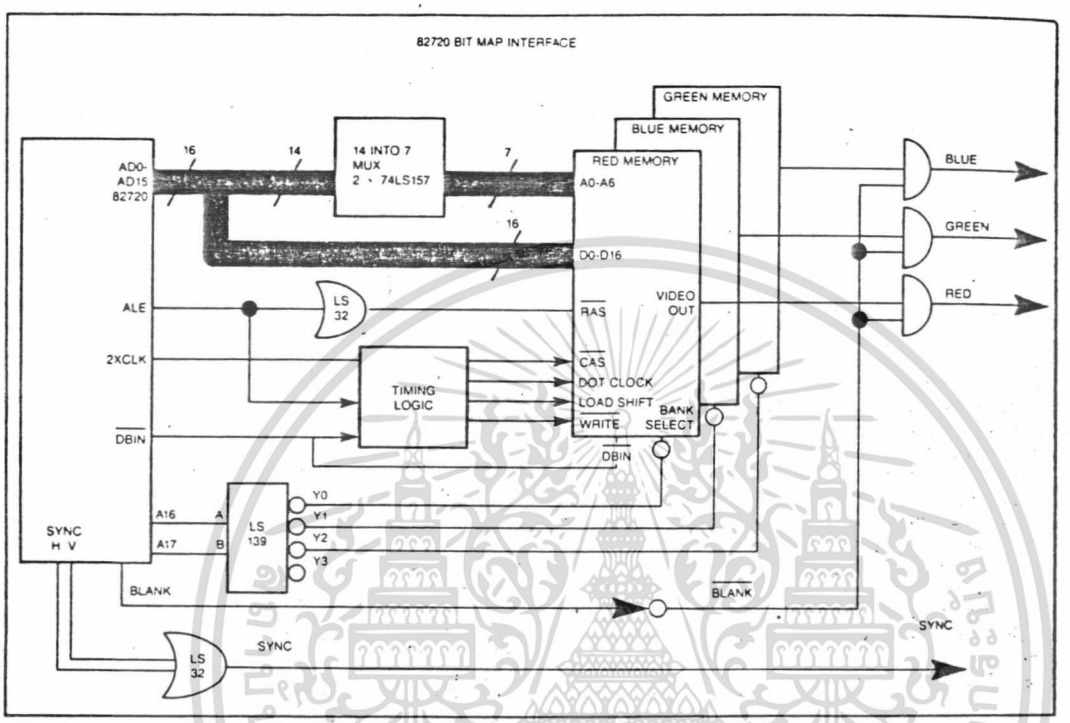


Figure 2: The memory is broken up into three planes, with each plane feeding one of the primary color guns of the CRT.

product considered high resolution for office automation applications is a 512 × 512 pixel monochrome or color display.

These latter restrictions are not imposed by the GDC, but rather have more to do with the cost of display monitors, the amount of RAM memory needed to support such displays, and the adequacy of such displays for most applications. It is possible to build "super graphics" boards with a GDC, such as the 1K by 1K pixel by 8 color plane graphics display designed by Phoenix Computer Graphics (Lafayette, LA). Such a display is capable of rendering 256 different colors on a high resolution screen.

Even higher performance can be achieved through the use of multiple GDC's to support multiple display windows, increased drawing speed, or increased bits per pixel. For multiple display windows, each GDC can be used to control one window of the display. For increased drawing speed, multiple GDC's can be operated in parallel. For increased bits/pixel, each GDC can contribute a portion of the number of bits necessary for a pixel.

Although the GDC is intended primarily for raster-scan graphics, it can also be used as a character display controller. It is capable of supporting up to four screens of data containing 25 rows by 80 columns, or one screen containing up to 100 rows by 256 characters.

Office Automation Display

High performance applications can stretch the usage of the GDC from low-end to high-end engineering displays, but research has shown that for office automation products, a 512 × 512 pixel display is quite acceptable, and that color is often a requirement. These requirements mesh with a major factor in display—the cost of the CRT. In OEM quantities, for example, one could expect to find a 512 × 512 monochrome display for under $100, a 256 × 256 color display (TV quality) for about $150, a 512 × 512 color CRT in the $300 range, and a 1K × 1K color display in the $800–$1000 category.

To give an example of the type of display that can be built for new office products using the GDC, consider a 512 × 512 pixel by 3 color plane combination CPU and graphics display on a single 12" by 12" board. Such a display is capable of generating 8 colors.

The list of parts (**Table 2**) comes to about $175 for 85 IC's taking up 104 square inches of board space. Even that parts count could be reduced by replacing the 48 16K DRAMs with 12 64K DRAMs—if a 4K × 16 bit DRAM were available. A very important note about the parts list is that the design is implemented with inexpensive 2118 dynamic RAMs. The design does

not require the faster, more expensive, and less dense static RAMs.

The parts count is low enough so that the processor and graphics controller can be placed together in a single 12" by 12" board. This is important because small overall size and footpad are selling points for desktop workstations. System speed is also enhanced when the graphics controller and CPU are on the same board, because their communication need not take up bus, inter-board bandwidth or experience any additional delays.

Pipelining Transformations

More important than putting the graphics display on the same board as the CPU is the level of communication between the CPU and graphics controller. If the burden of transformation processing is left entirely to the CPU while the graphics chip is used only as a CRT controller, then the CPU must communicate one bit per pixel to update a display. With the GDC, the CPU input takes higher level forms such as the slope and length of a line, the length and center point of an arc, or the key coordinates of a rectangle. Since the average line on a screen is about 25 pixels, that means that 25 times fewer CPU bus cycles are required to draw a graphical object with the GDC. These CPU cycles (an average of 50 μs each to calculate the graphical object and communicate it to the GDC) are the determining factor in drawing rate.

Viewed from a larger perspective, there are four tasks that must be performed by a CPU-graphics chip combination:

(1.) The CPU must calculate the higher-level graphics operations. This is done by the CPU and it involves the processing of macro-operations such as the CORE, GKS, PMIG or other graphics protocols. These general graphics commands are translated into an intermediate level, the VDI interface level (**Figure 1**) and then into device driver commands by software in the CPU.

(2.) Then, these lower-level graphical objects such as the key parameters for lines, arcs, characters, and rectangles, must be trans-

VLSI Takes Aim At Text Processing

The concept of co-processing is not a new one. Intended as a way of offloading computationally intensive tasks from a host CPU, it has been around at Intel since the introduction of the 8087 numerics processor and the 8089 I/O machine. A more recently developed product, the 82720 Graphics Display Controller is designed to bolster system performance by offloading graphics control chores from the CPU. The chip accepts high level commands from the CPU and, using its own drawing processor, accesses the required positions in the bit-map and handles the processing and display control functions.

Building on the success of these parts come two new co-processors designed to partition system intelligence even further. The 82586 is a communications co-processor designed to bridge the characteristics of CPU and network data rates. Its FIFO buffer and DMA facilities make it possible for a CPU to operate at the full Ethernet 10 Mbits/s transfer rate even in the face of continuous bursts of network data traffic.

Intel's most recent introduction is the 82730 text co-processor. Printers and other hard copy peripherals have supported additional text processing features such as proportional spacing and simultaneous superscript and subscript for some time. Implementing these features on the display screen has traditionally been a costly procedure. Thus, it is typically not done and screen displays often are not identical to their hard-copy printouts. Aimed to solve this designers headache, the 82730 has its own DMA capability and communicates asynchronously with the CPU via shared memory messages. It supports the generation of high quality text displays through features like proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. In addition, when coupled with the 82720 Graphics Display Controller (**Figure 1**) the 82730 provides flexible mixing of text and graphics simultaneously on the same display.

—*Wilson*

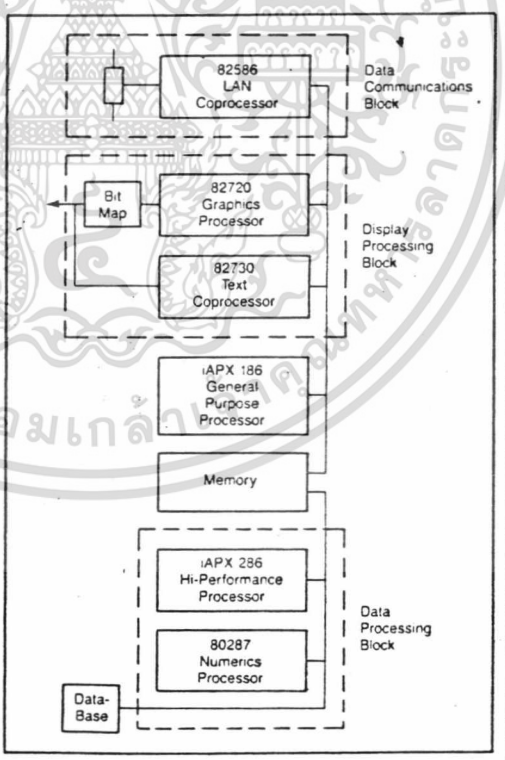


Figure 1: Offloading system tasks is simplified by new VLSI devices.

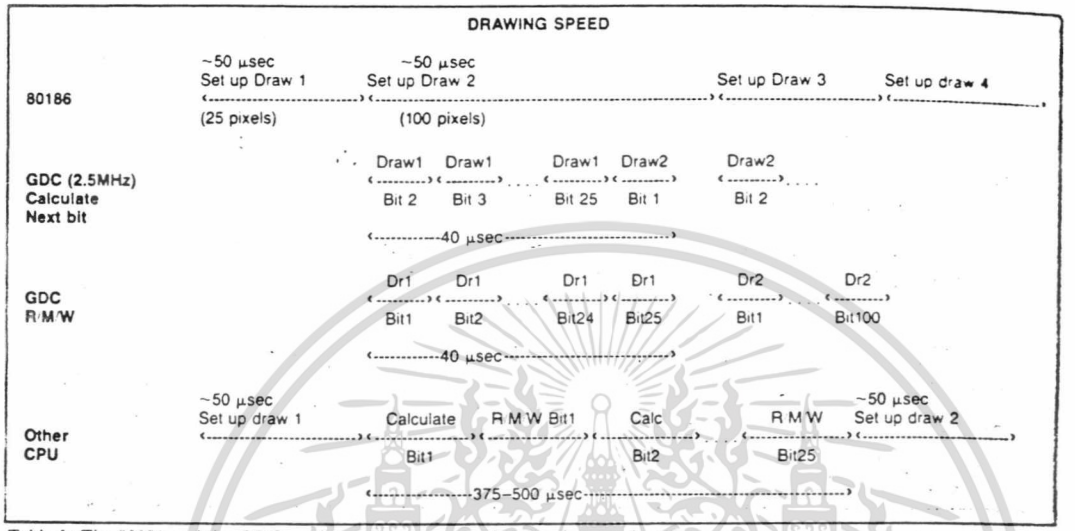


Table 1: The 80186 and the GDC work together to accomplish the drawing function.

formed into changes in the actual bits. This function is performed in hardware in the GDC concurrently with any level one processing done by the CPU. Other graphics controllers leave this task to the CPU to execute in software. The contrast is that, in such systems, the CPU must resolve the graphical object down to every point on a line, while with the GDC it need only designate the endpoints.

(3.) With the actual bits for the bit map calculated, they must be placed in the bit map memory. This involves a read-modify-write operation that requires three CPU cycles using other methods. With the GDC these operations are not the responsibility of the CPU. The GDC pipelines its execution so that it is calculating the next bit to change while it is executing the read-modify-write cycles.

(4.) Finally, the bit map memory must be dumped into the CRT. This is the refresh function performed by other graphics chips as well as the GDC.

The summation is that other systems require the CPU to process steps one to three serially, leaving only step four for the graphics controller. Systems with the GDC require the CPU to process only step one, with the GDC concurrently processing steps two through four. The GDC has another advantage in that during the transformation process in step three, the GDC executes the algorithms in hardware while a CPU must execute the algorithms in software. The algorithms are exactly the same in both cases. They are the Bresenam algorithms from IBM, in which the next pixel to be drawn becomes a binary decision between two pixels.

The execution of these algorithms is a crucial drawing time factor, because they are invoked many times for each updated screen. Consider that, in the inner loop of Bresenam's "line drawing algorithm," there are two or three additions, two comparisons or tests, and the masking of the proper value into the word for each pixel. The algorithms for drawing circles or filling areas are even more complex. In the inner loop of a fill algorithm, the old word must be read from the bit map, then tested to see if all, some, or none of the pixels are within the area to be filled. Next, it tests whether some or all of the pixels must be modified. Finally, the word must be returned to the bit map.

These algorithms are heavily used and the speed with which they can be executed has a direct effect upon the overall system efficiency. If they must be executed by a µP, the instruction fetching process slows down the calculations to a drawing rate of 15–20 µs per pixel. With a hardware implementation of these algorithms in the GDC, the calculations can be speeded up to achieve a drawing rate of 1600 ns (2.5 MHz version) or 800 ns (5 MHz version) per pixel.

Methods Of Refresh

In the fourth step, the dumping of bit map memory into the CRT, there are some differences between graphics controller chips. Motorola's MC6845 CRT controller, for example, uses a split-cycle refresh method in which each refresh cycle is alternated with a drawing cycle in which the µP updates the bit map. This gives the MC6845 a 50% drawing bandwidth.

With the GDC there are two drawing modes. The first is a "draw anytime" mode which replaces CRT refresh cycles with drawing cycles. This is the fastest mode, but it does result in on-screen disruptions. The second mode, which does not disrupt the on-screen display, draws only during the vertical and horizontal retracing periods. This gives the GDC about a 25%

1	80186	1	74LS04	1	20 MHz Clock
1	82720	1	74LS73	2	27128
2	74LS157	9	74LS244	2	2186
1	74LS139	8	74LS166	1	8274
1	74LS161	3	74LS32	1	8042
1	74LS11	2	8286	3	Connectors
1	74LS00	1	8 MHz Crystal	1	12×12 2 Layer PC

SUMMARY:

4 VLSI Controllers	80186	Processor
	82720	Graphics
	8274	Serial Link
	8042	Keyboard
4 VLSI Memory	27128	EPROM
	2186	IRAM
48 16K DRAMs	2118	DRAM
29 MSI/SSI	—	Buffers/Glue

TOTAL: 85 IC'S →104 Sq. Inches
Parts Cost →About $175

Table 2: Parts list for 512×512×4 Color Display.

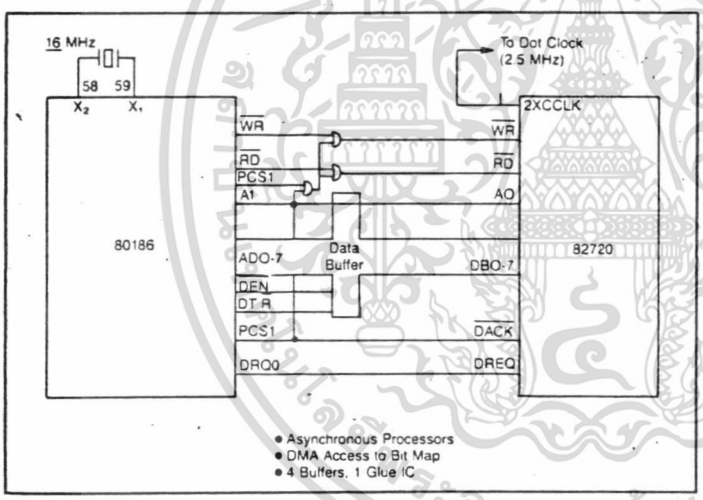


Figure 3: The two chip selects are OR'd together to qualify the R/W signals.

drawing bandwidth. At first glance that gives the GDC a disadvantage in drawing rate, but the fact is, with its pipelining and hardware execution of transformations, the GDC makes much more efficient use of its bandwidth. The critical timing factor is the amount of CPU participation in the drawing process, not the refresh bandwidth of the graphics controller. Another tradeoff is that, with its split-cycle architecture, the MC6845 requires RAM memory that is twice as fast as that required by the GDC in the same application.

Inexpensive RAM Is Fast Enough

Applying this perspective, one can begin to build the display with parts listed in **Table 2**. First one notes that a square display, as indicated by the 512×512 pixel initial specification, is not pleasing to the eye. It is much more appealing to have an aspect ratio of about 4:3, in which the number of pixels horizontally is 4/3 the number vertically. If the resolution is such that the total number of pixels is not a power of two, it will be necessary to round up to the next power of two and waste the extra bits.

The pixel arrangement which best meets this requirement is one with a 432 × 576 pixel format. It also meets the requirement that the number of pixels horizontally be an even number of 16-bit words. With three color bits per pixel (red, blue, and green), the total display memory is then about 500×500×3, or 750k bits.

It makes the most sense to break the memory up into three planes, with each plane feeding one of the primary color guns of the CRT (**Figure 2**). This leads to a memory arrangement of 16K × 16 × 3, using 16K dynamic RAMs with a 1K × 16 architecture. When drawing graphics figures, the memory can be treated as one large plane, split into the three primary colors. Drawing in low-order memory could represent red, middle-order could be used for green, and high-order for blue.

One advantage of this 3D memory is that drawing with a primary color requires setting only one bit per pixel. Drawing with a secondary color such as cyan, yellow, or magenta would take two GDC cycles, and creating white from all three colors would take three GDC cycles. If this were an issue, additional hardware could be used to draw more than one plane at a time. As the results will show, however, the drawing speed requirements can be exceeded without any added hardware.

Calculate The Drawing Rate

To see if the proposed design is practical, one should first calculate the drawing rate to see what the user interface will be like. Then one should check the refresh rate to make sure the design is uninterrupted and without flicker.

The proof of the assumption that CPU participation is the dominating factor lies in the 50 μs average time that it takes the CPU to calculate a graphical object and communicate its key parameters to the GDC. Assume that the graphical object is an average line containing

25 pixels, and that there are about 500 vectors on the average screen display.

The GDC's normal clock rate is 2.5 MHz, giving it a 400 ns period (the maximum clock rate is 5 MHz, with a 200 ns period.) It takes four GDC cycles to execute a read-modify-write on a bit (because two read cycles are required), so that the GDC's normal drawing rate is one pixel per 1600 ns. To draw the 25 pixels involved in the average line, then, would take 25 × 1600 ns, or 40 μs. Since this operation is done concurrently with CPU processing, the GDC will be waiting for the next graphical object by the time the CPU is ready.

If the screen were filled with nothing but 25-pixel vectors, then the drawing rate would be determined by the 50 μs average CPU calculation and transfer cycle, averaging about 2 μs per pixel. If all the vectors were white (worst case), then it would take 1.5 secs of drawing time to update the white screen. Since, in the undisturbed-screen mode, drawing is only done during the 25% of the time that the screen is undergoing horizontal or vertical blanking, this would mean 6 secs between updates.

In reality, however, the screen will not be filled with vectors. It will have an average of 500 vectors, and the color distribution could be presumed to be evenly distributed as one-third primary colors, one-third secondary colors, and one-third white. The 500 vectors will require the drawing of 12.5K pixels in monochrome, or 25K pixels with distributed colors. At a drawing rate of 2 μs per pixel, this takes 50 ms to draw. Drawing only during blanking, the screen would be updated in 200 ms.

Under these conditions, it would not help to use the maximum clock rate GDC (5 MHz), but if in some applications the average vector length is 100 pixels, then the CPU calculation-and-bus cycle (50 μs) would remain the same and the GDC's drawing cycle (1600 ns × 100 = 160 μs) would become a limiting factor. Using the 5 MHz GDC would cut that drawing time down to 800 ns/pixel, or 80 μs/vector. The 500 vector average screen would then contain 100K pixels with distributed colors and could be drawn in 80 ms. Multiplying by four because the drawing is done during blanking (25% of the time), that is 320 ms. That is a screen update in less than one-third second for a "busy" screen.

Calculate The Refresh Rate

These calculations are of little importance if the display flickers due to lack of refresh. This exercise is actually a demonstration of how the basic GDC clock rate was derived. Assume a non-interlaced display that must be refreshed 60 times per second. That gives a screen refresh rate of 16.67 ms, but on a typical CRT some 4.27 ms of that is blanked, leaving 12.4 ms of active display time. The dot sweep period is the 12.4 ms divided by the number of pixels (432 × 576 = 248.8K), or 49.8 ns. The inverse gives a 20.07 MHz dot clock.

Since the GDC dumps 16 bits from the bit map memory into the

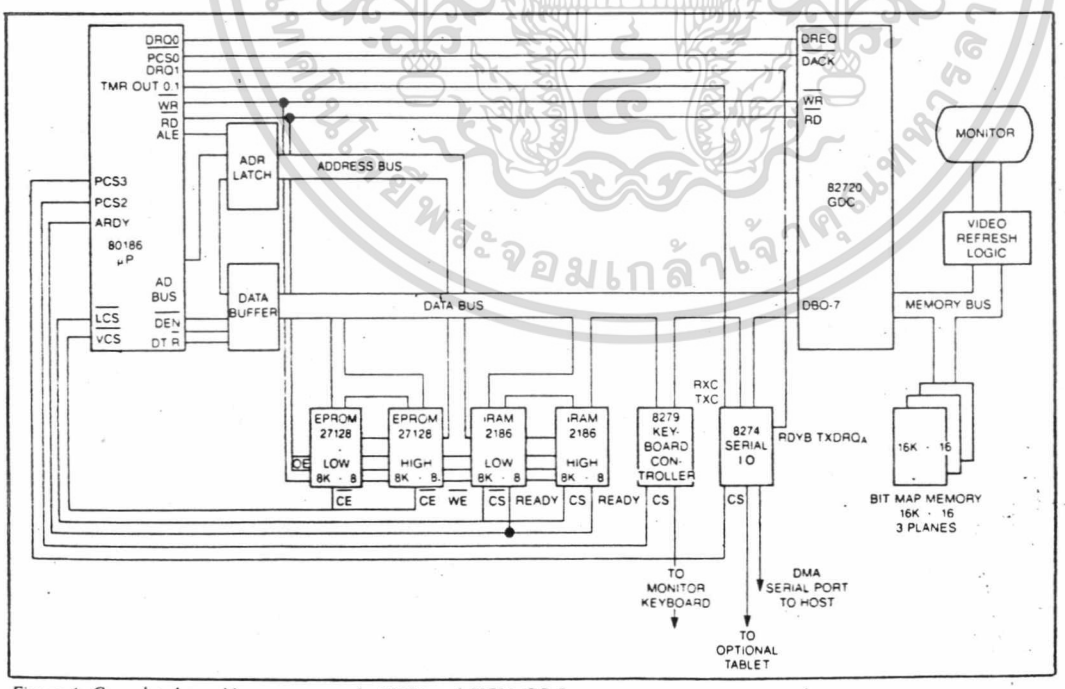


Figure 4: Completed graphics system uses the 80186 and 82720 GDC.

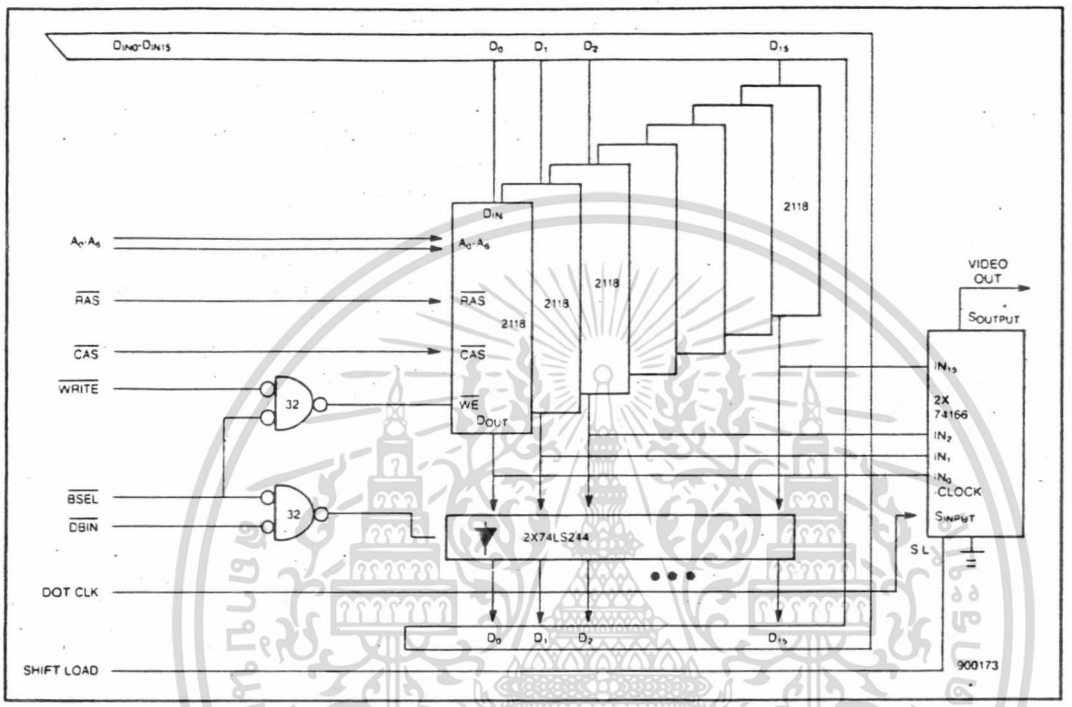


Figure 5: Since the 186 is a fully byte addressable machine, it is possible to write bytes as well as words into the RAMs.

16-bit shift register during each read, and since the shift register then feeds these bits out serially to the CRT, it makes sense that the GDC's read period should be 16 times the dot sweep period. That gives a GDC read period of about 800 ns. With each GDC read taking two cycles, the basic GDC clock period is then 400 ns, or 2.5 MHz. This gives a rock-solid display, and one would only want to go to the 5 MHz GDC to improve drawing rate.

For those who want to examine the blanking intervals to see if the CRT is indeed "typical," the blanking can be further broken down. The vertical blanking interval is 1.25 ms, leaving 15.42 ms to scan the 432 lines on the active portion of the display. Dividing 15.42 ms by 432 lines gives a 35.7 μs period per line, or a horizontal sweep rate of 28 KHz. Time is also needed for horizontal retrace, in this case, 7 μs of horizontal blanking per line. This leaves 28.7 μs to scan the 576 pixels on each line, resulting in the dot sweep period of 49.8 ns. Using a 20 MHz CRT helps keep the costs down, but the GDC can use CRT displays as fast as 80 MHz when higher resolution is required.

Mixed Mode

While it is possible to generate 8×8 characters and slanted characters in the graphics mode, the GDC also offers a mixed mode memory organization to display both characters and graphics drawn from separate windows in the display memory. The advantage of this mode is that it allows characters to be manipulated as 8-bit entities instead of the 64 bits that each would require in graphics mode. Of necessity, the graphics window display memory is reduced in this mode (64K 16-bit words instead of 256K), but even the reduced maximum graphics memory is still a megapixel and quite sufficient for both office automation and engineering display purposes.

In the character window, the GDC operates as it does in the pure character mode, with the exception that the line counter must be implemented externally. In addition to the two windows used for graphics and characters in the mixed mode, two other windows can be supported. These can be designated as either character or graphics windows by a selection on the A17 line.

Panning, Zooming, Light Pen

As special features, the GDC allows both panning and zooming in either graphics, character, or mixed modes. The zoom is accomplished by effectively increasing the size of the dots on the screen. Vertically, this is done by repeating the same display line. The number of repeat times is determined by the display zoom parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor.

PRELIMINARY

82720
GRAPHICS DISPLAY CONTROLLER

- Displays Low-to-High Resolution Images
- Draws Characters, Points, Lines, Arcs, and Rectangles
- Supports Monochrome, Gray Scale, or Color Displays
- Zooms, Pans and Windows Through a 4 Mpixel Display Memory
- Extremely Flexible Programmable Screen Display, Blanking, and Sync Formats
- Compatible with Intel's Microprocessor Families
- High-Level Commands Off Load Host Processor from Bit Map Loading and Screen Refresh Tasks
- Supports Graphics, Character, and Mixed Display Modes

FUNCTIONAL DESCRIPTION

Introduction

The 82720 Graphics Display Controller (GDC) is an intelligent microprocessor peripheral designed to drive high-performance raster-scan computer graphics and character CRT displays. Positioned between the video display memory and Intel microprocessor bus, the GDC performs the tasks needed to generate the raster display and manage the display memory. Processor software overhead is minimized by the GDC's sophisticated instruction set, graphics figure drawing, and DMA transfer capabilities. The display memory directly supported by the GDC can be configured in any number of formats and sizes up to 256K 16-bit words. The display can be zoomed and partitioned screen areas can be independently scrolled and panned. With its light pen input and multiple controller capability, the GDC is ideal for most computer graphics applications. Systems implemented with the GDC can be designed to be compatible with standards such as VDI, NAPLPS, GKS, Core, or custom implementations.

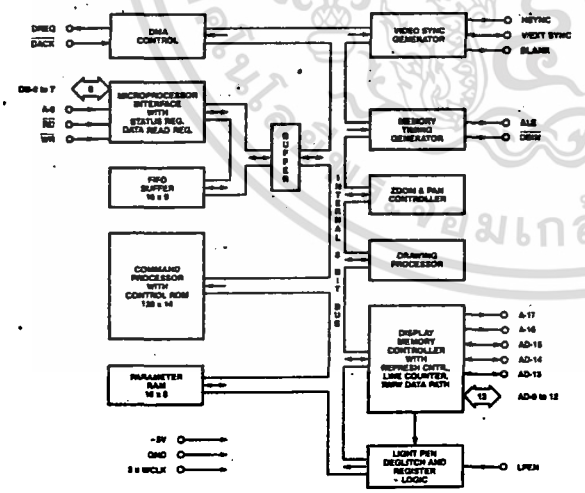


Figure 1. Block Diagram

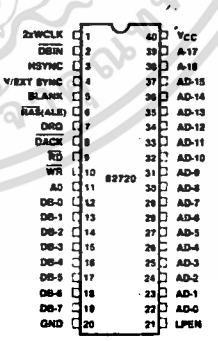


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Description
2XWCLK	1	I	**Clock Input**
$\overline{DBIN}$	2	O	**Display Bus Input:** Read strobe output used to read display memory data into the GDC.
HSYNC	3	O	**Horizontal Sync:** Output used to initiate the horizontal retrace of the CRT display.
V/EXT SYNC	4	I/O	**Vertical Sync:** Output used to initiate the vertical retrace of the CRT display. In slave mode, this pin is an input used to synchronize the GDC with the master raster timing device.
BLANK	5	O	**Blank:** Output used to suppress the video signal.
$\overline{RAS}$ (ALE)	6	O	**Row Address Strobe (Address Latch Enable):** Output used to start the control timing chain when used with dynamic RAMs. When used with static RAMs, this signal is used to demultiplex the display address/data bus.
DRQ	7	O	**DMA Request:** Output used to request a DMA transfer from a DMA controller (8237) or I/O processor (8089).
$\overline{DACK}$	8	I	**DMA Acknowledge:** Input used to acknowledge a DMA transfer from a DMA controller or I/O processor.
$\overline{RD}$	9	I	**Read:** Input used to strobe GDC Data into the microprocessor.
$\overline{WR}$	10	I	**Write:** Input used to strobe microprocessor data into the GDC.
A0	11	I	**Register Address:** Input used to select between commands and data read or written.
DB0	12	I/O	**Bidirectional Microprocessor Data Bus Line:** Input enabled by $\overline{WR}$. Output enabled by $\overline{RD}$.
DB1	13		
DB2	14		
DB3	15		
DB4	16		
DB5	17		
DB6	18		
DB7	19		
GND	20		**Ground.**
V_{CC}	40		**+5V Power Supply**
A_{17}	39	O	**Graphics Mode:** Display Address Bit 17 Output **Character Mode:** Cursor and Line Counter Bit 4 Output **Mixed Mode:** Cursor and Image Mode Flag
A_{16}	38	O	**Graphics Mode:** Display Address Bit 16 Output **Character Mode:** Line Counter Bit 3 Output **Mixed Mode:** Attribute Blink and Line Counter Reset
AD_{15}	37	I/O	**Graphics Mode:** Display Address/Data Bits 13–15 **Character Mode:** Line Counter Bits 0–2 Output **Mixed Mode:** Display Address/Data Bits 13–15
AD_{14}	36		
AD_{13}	35		
AD_{12}	34	I/O	**Display Address/Data Bits 0–12**
AD_{11}	33		
AD_{10}	32		
AD_9	31		
AD_8	30		
AD_7	29		
AD_6	28		
AD_5	27		
AD_4	26		
AD_3	25		
AD_2	24		
AD_1	23		
AD_0	22		
LPEN	21	I	**Light Pen Detect Input**

FUNCTIONAL DESCRIPTION (Continued)

Microprocessor Bus Interface

Control of the GDC by the system microprocessor is achieved through an 8-bit bidirectional interface. The status register is readable at any time. Access to the FIFO buffer is coordinated through flags in the status register.

Command Processor

The contents of the FIFO are interpreted by the command processor. The command bytes are decoded, and the succeeding parameters are distributed to their proper destinations within the GDC. The bus interface has priority over the command processor when both access the FIFO simultaneously.

DMA Control

The DMA Control circuitry in the GDC coordinates data transfers when using an external DMA controller. The DMA Request and Acknowledge handshake lines interface with an 8257 or 8237 DMA controller or 8089 I/O processor, so that display data can be moved between the microprocessor memory and the display memory.

Parameter RAM

The 16-byte RAM stores parameters that are used repetitively during the display and drawing processes. In character mode, the RAM holds the partitioned display area parameters. In graphics mode, the RAM also holds the drawing pattern and graphics character.

Video Sync Generator

Based on the clock input, the sync logic generates the raster timing signals for almost any interlaced, non-interlaced, or "repeat field" interlaced video format. The generator is programmed during the idle period following a reset. In video sync slave mode, it coordinates timing between the GDC and another video source.

Memory Timing Generator

The memory timing circuitry provides two memory cycle types: a two-clock period refresh cycle and the read-modify-write (RMW) cycle which takes four clock periods. The memory control signals needed to drive the display memory devices are easily generated from the GDC's $\overline{RAS}$(ALE) and $\overline{DBIN}$ outputs.

Zoom and Pan Controller

Based on the programmable zoom display factor and the display area parameters in the parameter RAM, the zoom and pan controller determines when to advance to the next memory address for display refresh and when to go on to the next display area. A horizontal zoom is produced by slowing down the display refresh rate while maintaining the video sync rates. Vertical zoom is accomplished by repeatedly accessing each line a number of times equal to the horizontal repeat. Once the line count for a display area is exhausted, the controller accesses the starting address and line count of the next display area from the parameter RAM. The system microprocessor, by modifying a display area starting address, allows panning in any direction, independent of the other display areas.

Drawing Processor

The drawing processor contains the logic necessary to calculate the addresses and positions of the pixels of the various graphics figures. Given a starting point and the appropriate drawing parameters, the drawing processor needs no further assistance to complete the figure drawing.

Display Memory Controller

The display memory controller's tasks are numerous. Its primary purpose is to multiplex the address and data information in and out of the display memory. It also contains the 16-bit logic units used to modify the display memory contents during RMW cycles, the character mode line counter, and the refresh counter for dynamic RAMs. The memory controller apportions the video field time between the various types of cycles.

Light Pen Debouncer

Only if two rising edges on the light pen input occur at the same point during successive video fields are the pulses accepted as a valid light pen detection. A status bit indicates to the system microprocessor that the light pen register contains a valid address.

System Operation

The GDC is designed to work with Intel microprocessors to implement high-performance computer graphics systems. System efficiency is maximized through partitioning and a pipelined architecture. At the lowest level, the GDC generates the basic video

raster timing, including sync and blanking signals. Partitioned areas on the screen and zooming are also accomplished at this level. At the next level, video display memory is modified during the figure drawing operations and data moves. Third, display memory address are calculated pixel by pixel as drawing progresses. Outside the GDC at the next level, preliminary calculations are done to prepare drawing parameters. At the fifth level, the picture must be represented as a list of graphics figures drawable by the GDC. Finally, this representation must be manipulated, stored and communicated. The GDC takes care of the high-speed and repetitive tasks required to implement graphics systems.

GENERAL OVERVIEW

In order to minimize system bus loading, the 82720 uses a private video memory for storage of the video image. Up to 512K bytes of video memory can be directly supported. For example, this is sufficient capacity to store a 2048 × 2048 pixel × 1 bit image. Images can be generated on the screen by:

—Drawing Commands
—Program-Controlled Transfers
—DMA Transfers from System Memory

The 82720 can be configured to support a wide variety of graphics applications. It can support:

—High Dot Rates
—Color Planes
—Horizontal Split Screen
—Character-oriented Displays
—Multiplexed Graphic and Character Display

GRAPHIC DISPLAY CONFIGURATIONS

The 82720 provides the flexibility to handle a wide variety of graphic applications. This flexibility results from having its own private video memory for storage of the graphics image. The organization of this memory determines the performance, the number of bits/pixel and the size of the display. Several different video memory organizations are examined in the following paragraphs.

In the simplest 82720 system, the memory can store up to a 2048 × 2048 × 1 bit image. It can display a 1024 × 1024 × 1 bit section of the image at a maximum dot rate of 44 MHz, or 88 MHz in wide mode. In this configuration, only 1 bit/pixel is used.

By partitioning the memory into multiple banks, color, gray scale and higher bandwidth displays can be supported. By adding various amounts of external logic, many cost/performance tradeoffs for both display and drawing are realizable.

The video memory can be partitioned into 4 banks, each 1024 x 1024 bits. By selecting all 4 memory banks during display, 4 bits/pixel can be provided by a single 82720. Each bank of video memory contributes 1 bit to each pixel. This configuration can support color monitors, again with a maximum dot shift rate of 44 or 88 MHz.

Higher performance may be achieved by using multiple 82720s. Multiple 82720s can be used to support mutliple display windows, increased drawing speed, or increased bits per pixel. For display windows, each 82720 controls one window of the display. For increased drawing speed, multiple 82720s are operated in parallel. For increased bits/pixel, each 82720 contributes a portion of the number of bits necessary for a pixel.

CHARACTER DISPLAY CONFIGURATION

Although the 82720 is intended primarily for raster-scan graphics, it can be used as a character display controller. The 82720 can support up to 8K by 13 bits of private video memory in this configuration (1 character = 13 bits). This is sufficient memory to store 4 screens of data containing 25 rows by 80 characters. The 82720 can display up to 256 characters per row. Smooth vertical scrolling of each of 4 independent display partitions is also supported.

MIXED DISPLAY CONFIGURATION

The GDC can support a mixed display system for both graphic and character information. This capability allows the display screen to be partitioned between graphic and character data. It is possible to switch between one graphic display window and one character display window with raster line resolution. A maximum of 256K bytes of video memory is supported in this mode: half is for graphic data, half is for character data. In graphic mode, a one megapixel image can be stored and displayed. In character mode, 64K, 16-bit characters can be stored.

DETAILED OPERATIONAL DESCRIPTION

The GDC can be used in one of three basic modes —Graphics Mode, Character Mode and Mixed Mode. This section of the data sheet describes the following for each mode:

1. Memory organization
2. Display timing
3. Special Display functions
4. Drawing and writing

Graphics Mode Memory Organization

The Display Memory is organized into 16-bit words (32-bit words in wide mode). Since the display memory can be larger than the CRT display itself, two width parameters must be specified: display memory width and display width. The Display width (in words) is selected by a parameter of the Reset command. The Display memory width (in words) is selected by a parameter of the Pitch command. The height of the Display memory can be larger than the display itself. The height of the Display is selected by a parameter of the Reset command. The GDC can directly address up to 4Mbits (0.5Mbytes) of display RAM in graphics mode.

Graphics Mode Display Timing

All raster blanking and display timings of the GDC are a function of the input clock frequency. Sixteen or 32 bits of data are read from the RAM and loaded into a shift register in each two clock period display cycle. The Address and Data busses of the GDC are multiplexed. In the first part of the cycle, the address of the word to be read is latched into an external demultiplexer. In the second part of the cycle the data is read from the RAM and loaded into the shift register. Since all 16 (32) bits of data are to be displayed, the dot clock is 8 × (16 ×) the GDC clock or 16 × (32 ×) the Read cycle rate.

Parameters of the Reset or Sync command determine the horizontal and vertical front porch, sync pulse, and back porch timings. Horizontal parameters are specified as multiples of the display cycle time, and vertical parameters as a multiple of the line time.

Another Reset command parameter selects interlaced or non-interlaced mode. A bit in the parameter RAM can define Wide Display Mode. In this mode, while data is being sent to the screen, the display address counter is incremented by two rather than one. This allows the display memory to be configured to deliver 32 bits from each display read cycle.

The V Sync command specifies whether the V Sync Pin is an input or an output. If the V Sync Pin is an output, the GDC generates the raster timing for the display and other CRT controllers can be synchronized to it. If the V Sync pin is an input, the GDC can be synchronized to any external vertical Sync signal.

Graphics Mode Special Display Functions:

WINDOWING

The GDC's Graphics Mode Display can be divided into two windows on the screen, upper and lower. The windows are defined by parameters written into the GDC's parameter RAM. Each window is specified by a starting address and a window length in lines. If the second window is not used, the first window parameters should be specified to be the same as the active display length.

ZOOMING

A parameter of the GDC's zoom command allows zooming by effectively increasing the size of the dots on the screen. This is accomplished vertically by repeating the same display line. The number of times it is repeated is determined by the display zoom factor parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor. It is the responsibility of the microprocessor controlling the GDC to provide the shift register clock circuitry with the zoom factor required to slow down the shift registers to the appropriate speed. The frequency of the 2XWCLK should not be changed. The zoom factor must be set to a known state upon initialization.

PANNING

Panning is accomplished by changing the starting address of the display window. In this way, panning is possible in any direction, vertically on a line by line basis and horizontally on a word by word basis.

Graphics Mode Drawing and Writing

The GDC can draw solid or patterned lines, arcs, circles, rectangles, slanted rectangles, characters, slanted characters, filled rectangles. Direct access to the bit map is also provided via the DMA Commands and the Read or Write data commands.

MEMORY MODIFICATION

All drawing and writing functions take place at the location in the display RAM specified by the cursor. The cursor is not displayed in Graphics Mode. The cursor location is modified by the execution of drawing, reading or writing commands. The cursor will move to the bit following the last bit accessed.

Each bit is drawn by executing a Read-Modify-Write cycle on the display RAM. These R/M/W cycles normally require four 2XWCLK cycles to execute. If the display zoom factor is greater than two, each R/M/W cycle will be extended to the width of a display cycle. Write Data (WDAT), Read Data (RDAT), DMA write (DMAW) and DMA read (DMAR) commands can be used to examine or modify one to 16 bits in each word during each R/M/W cycle. All other graphics drawing commands modify one bit per R/M/W cycle.

An internal 16-bit Mask register determines which bit(s) in the accessed word are to be modified. A one in the Mask register allows the corresponding bit in the display RAM to be modified by the R/M/W cycle. A zero in the Mask register prevents the GDC from modifying the corresponding bit in the display RAM.

The mask must be set by the Mask Command prior to issuing the WDAT or DMAW command. The Mask register is automatically set by the CURS command and manipulated by the graphics commands.

The display RAM bits can be modified in one of four ways. They can be set to 1, reset to 0, complemented or replaced by a pattern.

When replace by a pattern mode is selected, lines, arcs and rectangles will be drawn using the 16-bit pattern in parameter RAM bytes 8 and 9.

In set, reset, or complement mode, parameter RAM bytes 8 and 9 act as another level of masking for line arc and rectangle drawing. As each 16-bit segment of the line or arc is drawn, it is checked against the pattern in the parameter RAM. If the pattern RAM bit is a one, the display RAM bit will be set, reset, or complemented per the proper modes. If the pattern RAM bit is a zero, the display RAM bit won't be modified.

When replace by pattern mode is selected, the graphics character and fill commands will cause the 8 x 8 pattern in parameter RAM bytes 8 to 15 to be written directly into the display RAM in the appropriate locations.

In set, reset, or complement mode, the 8 x 8 pattern in parameter RAM bytes 8 to 15 act as a mask pattern for graphics character or fill commands. If the appropriate parameter RAM bit is set, the display RAM bit will be modified. If the parameter RAM bit is zero, the display RAM bit will not be modified. These modes are selected by issuing a WDAT command without parameters before issuing graphics commands. The pattern in the parameter RAM has no effect on WDAT, RDAT, DMAW, or DMAR operations.

READING AND DRAWING COMMANDS

After the modification mode has been set and the parameter RAM has been loaded, the final drawing parameters are loaded via the figure specify (FIGS) command. The first parameter specifies the direction in which drawing will occur and the figure type to be drawn. This parameter is followed by one to five more parameters depending on the type of character to be drawn.

The direction parameter specifies one of eight octants in which the drawing or reading will occur. The effect of drawing direction on the various figure types is shown in Figure 9.

RDAT, WDAT, DMAR, and DMAW Operations move through the Display memory as shown in the "DMA" Column.

The other parameters required to set up figure reading or drawing are shown in Figure 3.

DRAWING TYPE	DC	D	D2	D1	DM
INITIAL VALUE*	0	0	0	−1	−1
LINE	|Δl|	2(|ΔD| − |Δl|)	2(|ΔD| − |Δl|)	2|ΔD|	−
ARC**	rsin θ↑	r−1	2(r−1)	−1	rsin θ↑
RECTANGLE	3	A−1	B−1	−1	A−1
AREA FILL	B−1	A	A	−	−
GRAPHIC CHARACTER***	B−1	A	A	−	−
WRITE DATA	W−1	−	−	−	−
DMAW	D−1	C−1	−	−	−
DMAR	D−1	C−2	(C−2)/2†	−	−
READ DATA	W	−	−	−	−

*INITIAL VALUES FOR THE VARIOUS PARAMETERS ARE LOADED WHEN THE FIGS COMMAND BYTE IS PROCESSED.
**CIRCLES ARE DRAWN WITH 8 ARCS, EACH OF WHICH SPAN 45°, SO THAT SIN ϕ = 1/√2 AND SIN θ = 0.
***GRAPHIC CHARACTERS ARE A SPECIAL CASE OF BIT-MAP AREA FILLING IN WHICH B AND A ≤ 8. IF A = 8 THERE IS NO NEED TO LOAD D AND D2.

WHERE:
−1 = ALL ONES VALUE.
ALL NUMBERS ARE SHOWN IN BASE 10 FOR CONVENIENCE. THE GDC ACCEPTS BASE 2 NUMBERS (2s COMPLEMENT NOTATION WHERE APPROPRIATE).
− = NO PARAMETER BYTES SENT TO GDC FOR THIS PARAMETER.
Δl = THE LARGER OF Δx OR Δy.
ΔD = THE SMALLER OF Δx OR Δy.
r = RADIUS OF CURVATURE, IN PIXELS.
ϕ = ANGLE FROM MAJOR AXIS TO END OF THE ARC. ϕ ≤ 45°.
θ = ANGLE FROM MAJOR AXIS TO START OF THE ARC. θ ≤ 45°.
↑ = ROUND UP TO THE NEXT HIGHER INTEGER.
↓ = ROUND DOWN TO THE NEXT LOWER INTEGER.
A = NUMBER OF PIXELS IN THE INITIALLY SPECIFIED DIRECTION.
B = NUMBER OF PIXELS IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
W = NUMBER OF WORDS TO BE ACCESSED.
C = NUMBER OF BYTES TO BE TRANSFERRED IN THE INITIALLY SPECIFIED DIRECTION. (TWO BYTES PER WORD IF WORD TRANSFER MODE IS SELECTED.)
D = NUMBER OF WORDS TO BE ACCESSED IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
DC = DRAWING COUNT PARAMETER WHICH IS ONE LESS THAN THE NUMBER OF RMW CYCLES TO BE EXECUTED.
DM = DOTS MASKED FROM DRAWING DURING ARC DRAWING.
† = NEEDED ONLY FOR WORD READS.

Figure 3. Drawing Parameter Details

After the parameters have been set, line, arc, circle, rectangle or slanted rectangle drawing operations are initiated by the Figure Draw (FIGD) command. Character, slanted character, area fill and slanted area fill drawing operations are initiated by the Graphics Character Draw (GCHRD) command. DMA transfers are initiated by the DMA Read or Write (DMAR or DMAW) commands. Data Read Operations are initiated by the Read Data (RDAT) Command. Data Write Operations are initiated by writing a parameter after the WDAT command.

The area fill operation steps and repeats the 8 × 8 graphics character pattern draw operation to fill a rectangular area. If the size of the rectangle is not an integral number of 8 x 8 pixels, the GDC will automatically truncate the pattern at the edges furthest from the starting point.

The Graphics Character Drawing capability can be modified by the Graphics Character Write Zoom Factor (GCHR) parameter of the zoom command. The zoom write factor may be set from 1 to 16 (by using from 0 to 15 in the parameter). Each dot will be repeated in memory horizontally and vertically (adjusted for drawing direction) the number of times specified by the zoom factor.

The WDAT command can be used to rapidly fill large areas in memory with the same value. The mask is set to all 1's, and the least significant bit of the WDAT parameter replaces all bits of each word written.

Character Mode Memory Organization

In character mode, the Display memory is organized into up to 8K characters of up to 13 bits each. Wide mode is also available for characters of up to 26 bits.

The display memory can be larger than the display itself. The display width (in characters) is a parameter of the reset command. The display memory width (in characters) is a parameter of the Pitch Command. The height of the display (in lines) is a parameter of the Reset Command. The display memory height is determined by dividing the number of display memory words by the pitch.

In character mode, the display works almost exactly as it does in graphics mode. The differences lie in the fact that data read from the display RAM is used to drive a character generator as well as attribute logic if desired. In Character mode, address bits 13–16 become line counter outputs used to select the proper line of the character generator, and the address 17 output becomes the cursor and line counter MSB output.

Character Mode Display Timing

In character mode, the display timing works as it does in graphics mode. In addition, the Address 17 output becomes cursor output. The characteristics of the cursor are defined by parameters of the cursor and Character Characteristics (CCHAR) command. One bit allows the cursor output to be enabled or disabled. The height of the cursor is programmable by selecting the top and bottom line between which the cursor will appear. The blink rate is also programmable. The parameter selects the number of frame times that the cursor will be inactive and active, resulting in a 50% duty cycle cursor blinking at 2× the period specified by the parameter.

The cursor output pin also provides the line counter bit 4 signal, which is valid 10 clocks after the trailing edge of HSYNC.

Character Mode Special Display Functions

WINDOWING

The GDC's Character Mode display can be partitioned into one to four windows on the screen. The windows are defined by parameters written into the GDC's Parameter RAM. Each window is specified by a starting address and a window length in lines.

If windowing is not required, the first window length should be specified to be the same as the active display length.

ZOOMING AND PANNING

In character mode, zooming and pan handling commands function the same way as in Graphics Mode.

Character Mode Drawing and Writing

The GDC can read or write characters of up to 13 bits into or out of the Display RAM.

All reading and writing functions take place at the display RAM location specified by the cursor. The cursor location can be read by issuing the CURD command. The cursor can be moved anywhere within the display memory by the CURS command. The cursor location is also modified by the execution of character read or write commands.

Each character is written or read via a Read/Modify/Write cycle. The mask register contents determine which bit(s) in the character are modified. The mask register can be used to change character codes without modifying attribute bits or vice-versa. The Replace with pattern, Set, Reset and Complement

modes work exactly as they do in graphics mode, with the exception that the parameter RAM Pattern is not used. The pattern used is a parameter of the WDAT command.

The Figure Specify (FIGS) command must be set to Character Display mode, as well as specify the direction the cursor will be moved by read or write data commands.

In character mode, the FIGD and GCHRD commands are not used.

Mixed Mode Memory Organization

In mixed mode, the display memory is organized into two banks of up to 64K words of 16 bits each (32 bits in wide mode).

The display height and width are programmable by the same Reset or Sync command parameters as in the graphics and character modes. The display memory width (in words) is a parameter of the Pitch Command and the height of the display memory is determined by dividing the number of display memory words by the pitch.

An image mode signal is used to switch the external circuitry between graphics and character modes in two display windows.

In a graphics window, the GDC works as it does in pure graphics mode, but on a smaller total memory space (64K words vs 512K words).

In a character window, the GDC works as it does in pure character mode, but the line counter must be implemented externally. The counter is clocked by the horizontal sync pulse and reset by a signal supplied by the GDC.

In mixed mode, the GDC provides both a cursor and an attribute blink timing signal.

Mixed Mode Display Timing

In mixed mode, each word in a graphic area is accessed twice in succession. The AW parameter of the Reset or Sync command should be set to twice its normal value, and the video shift register load signal must be suppressed during the extra access cycle.

In addition, A16 becomes a Multiplexed Attribute and Clear Line Counter signal and A17 becomes a multiplexed cursor and image mode signal. A16 provides an active high line counter reset signal which is valid 10 clocks after the trailing edge of HSYNC. During the active display line time, A16 provides blink timing for external attribute circuitry. This signal blinks at 1/2 the blink rate of the cursor with a 75% on, 25% off duty cycle. A17 provides a signal which selects between graphics or character display, which is also valid 10 clocks after the trailing edge of HSYNC. During the active display time, A17 provides the cursor signal. The cursor timing and characteristics are defined in exactly the same way as in pure character mode.

Mixed Mode Special Display Functions

WINDOWING

The GDC supports two display windows in mixed mode. They can independently be programmed into either graphics or character mode determined by the state of two bits in the parameter RAM. The window location in display memory and size are also determined by parameters in the parameter RAM.

ZOOMING AND PANNING

In mixed mode, zooming and panning commands function the same as in graphics and character mode.

Mixed Mode Drawing and Writing

In mixed mode, the GDC can write or draw in exactly the same ways as in both graphics and character modes. In addition, the FIGS command has a parameter GD (Graphics Drawing Flag) which sets the image mode signal to select the proper RAM bank.

DEVICE PROGRAMMING

The GDC occupies two addresses on the system microprocessor bus through which the GDC's status register and FIFO are accessed. Commands and parameters are written into the GDC FIFO and are differentiated by address bit A0. The status register or the FIFO can be read as selected by the address line.

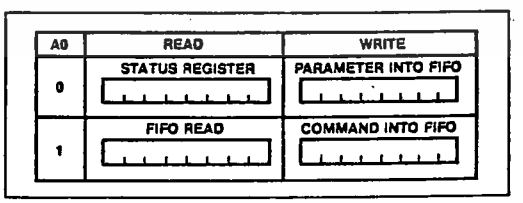


Figure 4. GDC Microprocessor Bus Interface Registers

which are handled by the GDC. Display memory is organized as a linearly addressed space of these words. Addressing of individual pixels is handled by the GDC's internal RMW logic.

During the drawing process, the GDC finds the next pixel of the figure which is one of the eight nearest neighbors of the last pixel drawn. The GDC assigns each of these eight directions a number from 0 to 7, starting with straight down and proceeding counterclockwise.

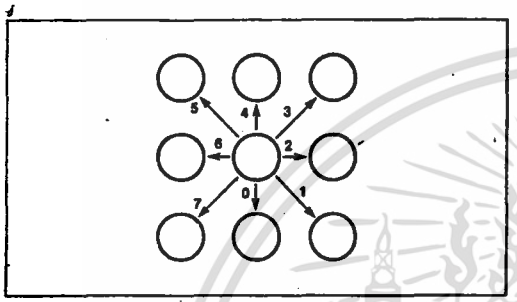


Figure 7. Drawing Directions

Figure drawing requires the proper manipulation of the address and the pixel bit position according to the drawing direction to determine the next pixel of the figure. To move to the word above or below the current one, it is necessary to subtract or add the number of words per line in display memory. This parameter is called the pitch. To move to the word to either side, the Execute word address cursor, EAD, must be incremented or decremented as the dot address pointer bit reaches the LSB or the MSB of the Mask register. To move to a pixel within the same word, it is necessary to rotate the dot address pointer register to the right or left.

Figure 8 summarizes these operations for each direction.

Whole word drawing is useful for filling areas in memory with a single value. By setting the Mask register to all 1s with the MASK command, both the LSB and MSB of the dAD will always be 1, so that the EAD value will be incremented or decremented for each cycle regardless of direction. One RMW cycle will be able to affect all 16 bits of the word for any drawing type. One bit in the Pattern register is used per RMW cycle to write all the bits of the word to the same value. The next Pattern bit is used for the word, etc.

DIR	ADDRESS OPERATION(S)
0	EAD = EAD + P
1	EAD = EAD + P If dAD.MSB = 1 then EAD = EAD + 1 dAD = LR(dAD)
2	If dAD.MSB = 1 then EAD = EAD + 1 dAD = LR(dAD)
3	EAD = EAD – P If dAD.MSB = 1 then EAD = EAD + 1 dAD =LR(dAD)
4	EAD = EAD – P
5	EAD = EAD – P If dAD.LSB = 1 then EAD = EAD – 1 dAD = RR(dAD)
6	If dAD.LSB = 1 then EAD = EAD – 1 dAD = RR(dAD)
7	EAD = EAD + P If dAD.LSB = 1 then EAD = EAD – 1 dAD = RR(dAD)

WHERE
P = PITCH, LR = LEFT ROTATE, RR = RIGHT ROTATE
CAD = CURSOR ADDRESS
dAD = DOT ADDRESS
LSB = LEAST SIGNIFICANT BIT
MSB = MOST SIGNIFICANT BIT

Figure 8. Address Calculation Details

For the various figures, the effect of the initial direction upon the resulting drawing is shown in figure 9.

Note that during line drawing, the angle of the line may be anywhere within the shaded octant defined by the DIR value. Arc drawing starts in the direction initially specified by the DIR value and veers into an arc as drawing proceeds. An arc may be up to 45 degrees in length. DMA transfers are done on word boundaries only, and follow the arrows indicated in the table to find successive word addresses. The slanted paths for DMA transfers indicate the GDC changing both the X and Y components of the word address when moving to the next word. It does not follow a 45 degree diagonal path by pixels.

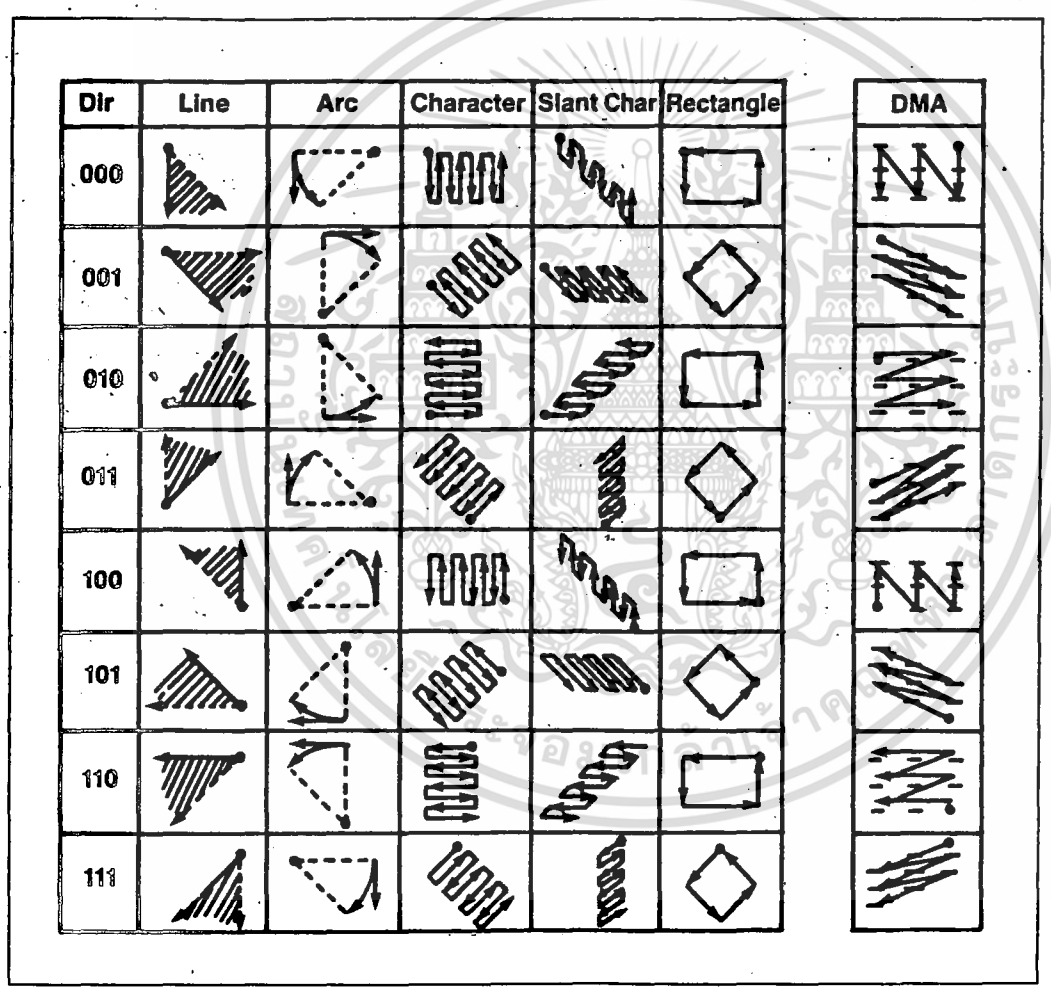


Figure 9. Effect of the Direction Parameter

Drawing Parameters

In preparation for graphics figure drawing, the GDC's Drawing Processor needs the figure type, direction and drawing parameters, the starting pixel address, and the pattern from the microprocessor. Once these are in place within the GDC, the Figure Draw command, FIGD, initiates the drawing operation. From that point on, the system microprocessor is not involved in the drawing process. The GDC Drawing Processor coordinates the RMW circuitry and address registers to draw the specified figure pixel by pixel.

The algorithms used by the processor for figure drawing are designed to optimize its drawing speed. To this end, the specific details about the figure to be drawn are reduced by the microprocessor to a form conducive to high-speed address calculations within the GDC. In this way the repetitive, pixel-by-pixel calculations can be done quickly, thereby minimizing the overall figure drawing time. Figure 3 summarizes the parameters.

Graphics Character Drawing

Graphics characters can be drawn into display memory pixel-by-pixel. The up to 8-by-8 character is loaded into the GDC's parameter RAM by the system microprocessor. Consequently, there are no limitations on the character set used. By varying the drawing parameters and drawing direction, numerous drawing options are available. In area fill applications, a character can be written into display memory as many times as desired without reloading the parameter RAM.

Once the parameter RAM has been loaded with up to eight graphics character bytes by the appropriate PRAM command, the GCHRD command can be used to draw the bytes into display memory starting at the cursor. The zoom magnification factor for writing, set by the zoom command, controls the size of the character written into the display memory in integer multiples of 1 through 16. The bit values in the PRAM are repeated horizontally and vertically the number of times specified by the zoom factor.

The movement of these PRAM bytes to the display memory is controlled by the parameters of the FIGS command. Based on the specified height and width of the area to be drawn, the parameter RAM is scanned to fill the required area.

For an 8-by-8 graphics character, the first pixel drawn uses the LSB of RA-15, the second pixel uses bit 1 of RA-15, and so on, until the MSB of RA-15 is reached. The GDC jumps to the corresponding bit in RA-14 to continue the drawing. The progression then advances toward the LSB of RA-14. This snaking sequence is continued for the other 6 PRAM bytes. This progression matches the sequence of display memory addresses calculated by the drawing processor as shown in figure 9. If the area is narrower than 8 pixels wide, the snaking will advance to the next PRAM byte before the MSB is reached. If the area is less than 8 lines high, fewer bytes in the parameter RAM will be scanned. If the area is larger than 8 by 8, the GDC will repeat the contents of the parameter RAM in two dimensions.

Parameter RAM Contents

The parameters stored in the parameter RAM, PRAM, are available for the GDC to refer to repeatedly during figure drawing and raster-scanning. In each mode of operation the values in the PRAM are interpreted by the GDC in a predetermined fashion. The host microprocessor must load the appropriate parameters into the proper PRAM locations. PRAM loading command allows the host to write into any location of the PRAM and transfer as many bytes as desired. In this way any stored parameter byte or bytes may be changed without influencing the other bytes.

The PRAM stores two types of information. For specifying the details of the display area partitions, blocks of four bytes are used. The four parameters stored in each block include the starting address in display memory of each display area, and its length. In addition, there are two mode bits for each area which specify whether the area is a bit-mapped graphics area or a coded character area, and whether a normal or wide display cycle is to be used for that area.

The other use for the PRAM contents is to supply the pattern for figure drawing when in a bit-mapped graphics area or mode. In these situations, PRAM bytes 8 through 16 are reserved for this patterning information. For line, arc, and rectangle drawing (linear figures) locations 8 and 9 are loaded into the Pattern register to allow the GDC to draw dotted, dashed, etc. lines. For area filling and graphics bit-mapped character drawing locations 8 through 15 are referenced for the pattern or character to be drawn.

Details of the bit assignments are shown on the following pages for the various modes of operation.

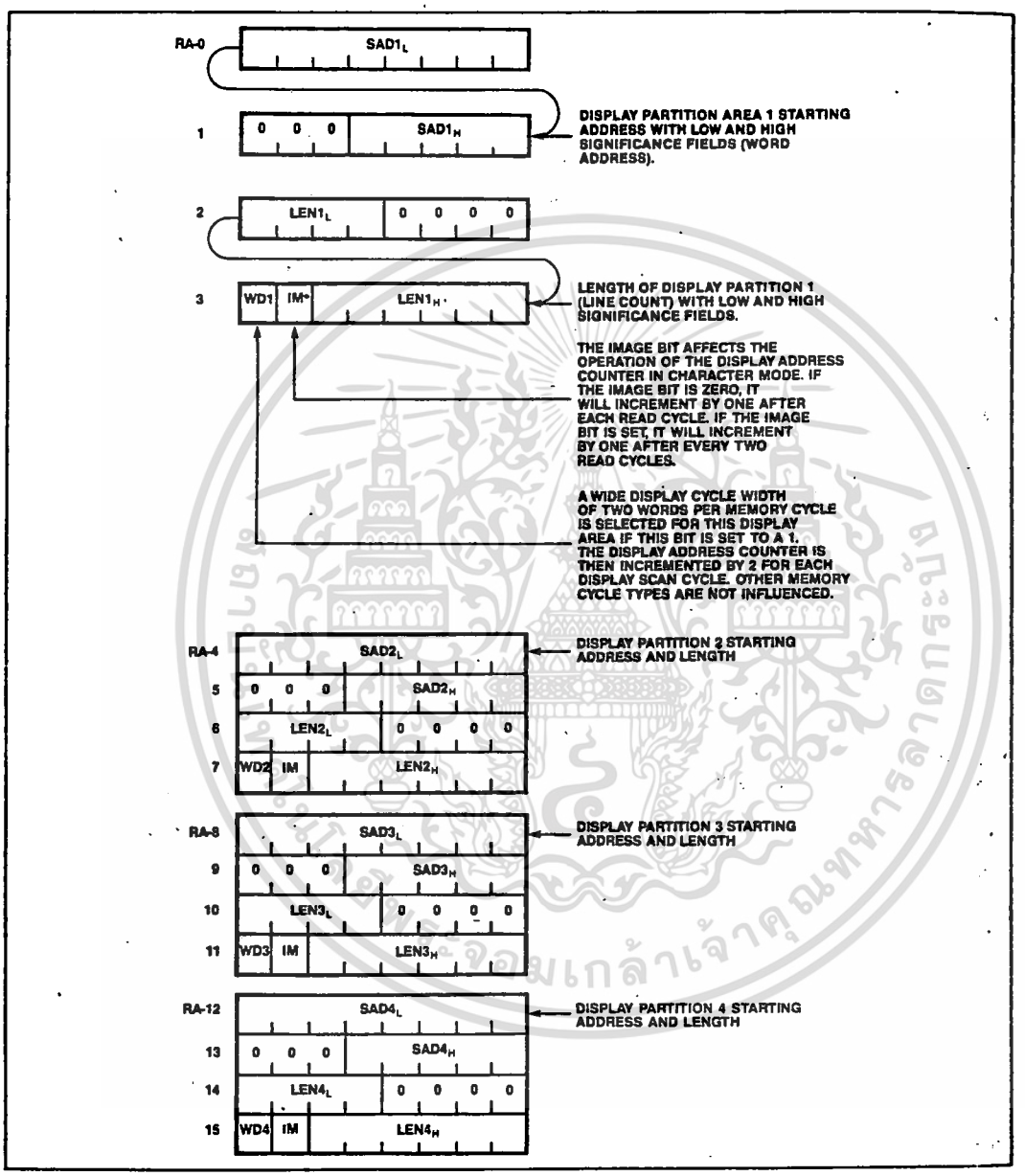


Figure 10. Parameter RAM Contents—Character Mode

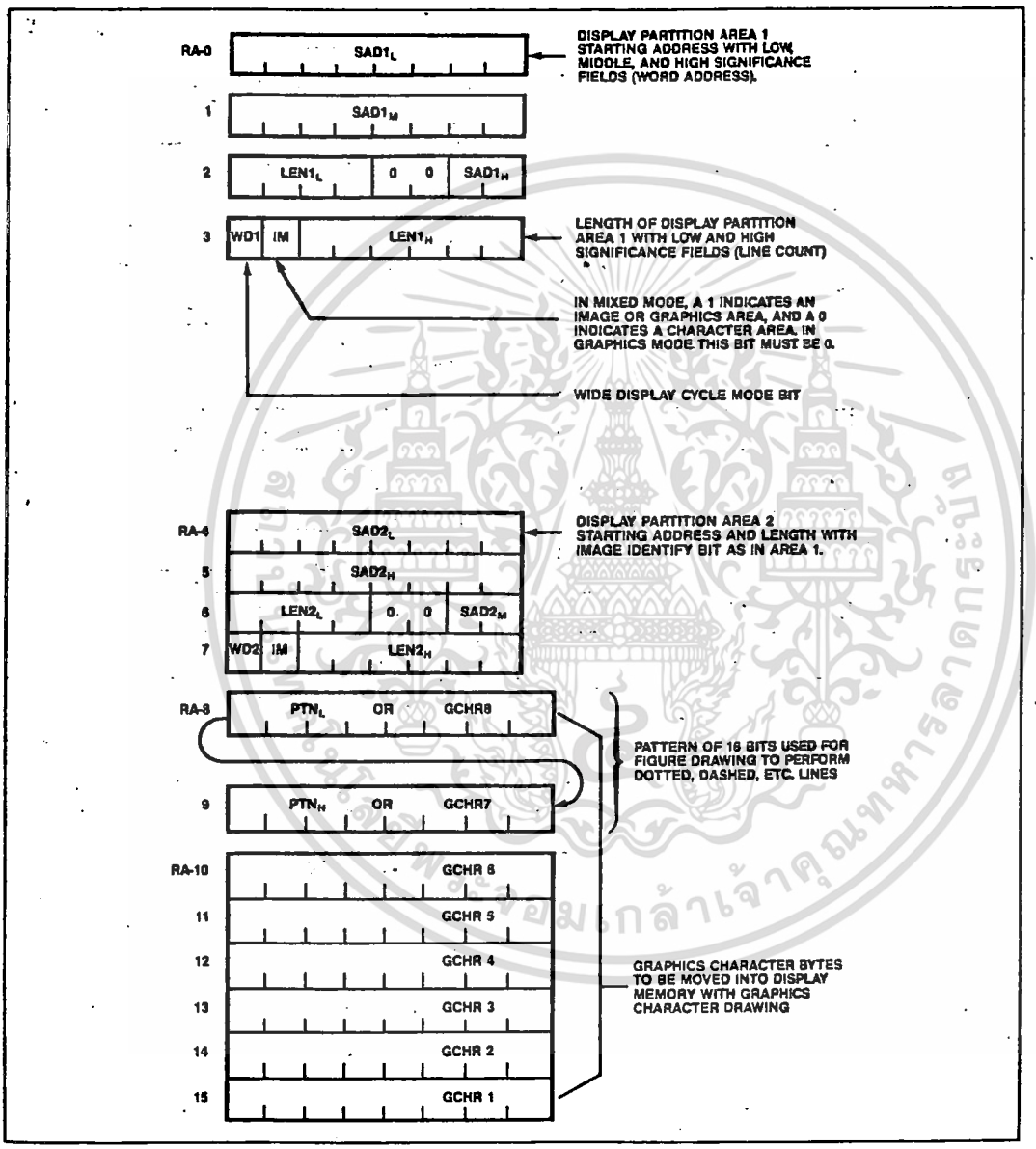


Figure 11. Parameter RAM Contents—Graphics and Mixed Graphics and Character Modes

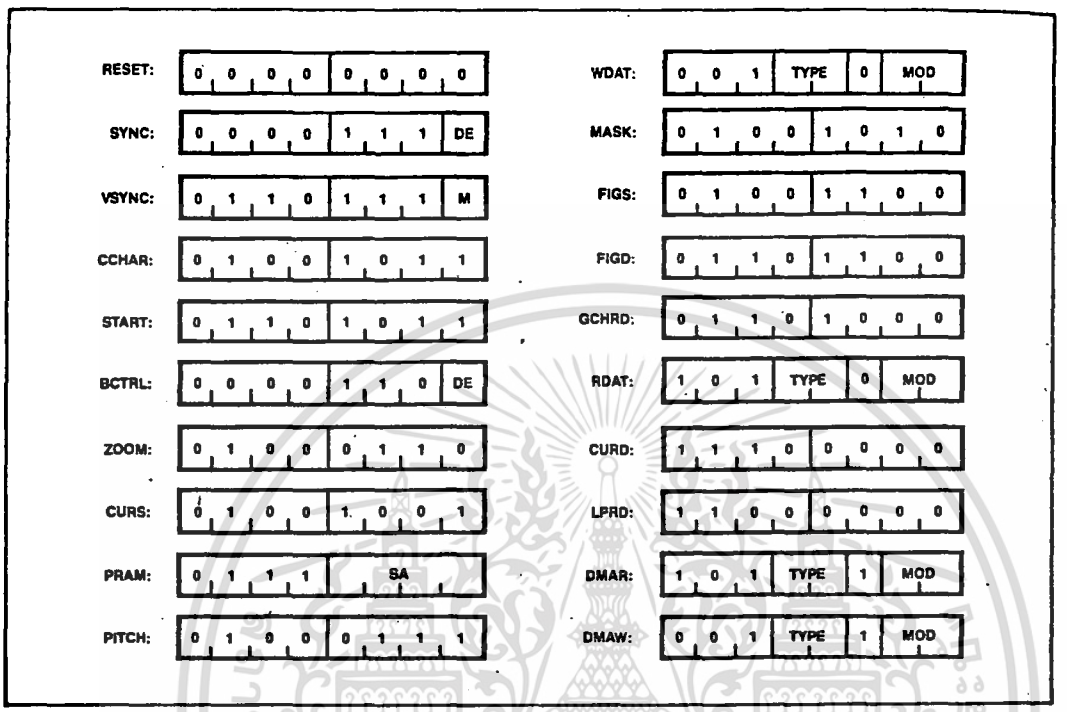


Figure 12. Command Bytes Summary

VIDEO CONTROL COMMANDS

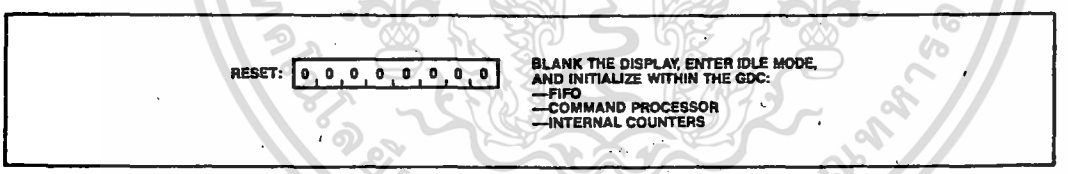


Figure 13. Reset Command

RESET COMMAND

This command can be executed at any time and does not modify any of the parameters already loaded into the GDC.

If followed by parameter bytes, this command also sets the sync generator parameters as described below. Idle mode is exited with the START command.

C G	DISPLAY MODE
0 0	MIXED GRAPHICS & CHARACTER
0 1	GRAPHICS MODE
1 0	CHARACTER MODE
1 1	INVALID

I S	VIDEO FRAMING
0 0	NONINTERLACED
0 1	INVALID
1 0	INTERLACED REPEAT FIELD FOR CHARACTER DISPLAYS
1 1	INTERLACED

D	DYNAMIC RAM REFRESH CYCLES ENABLE
0	NO REFRESH—STATIC RAM
1	REFRESH—DYNAMIC RAM

F	DRAWING TIME WINDOW
0	DRAWING DURING ACTIVE DISPLAY TIME AND RETRACE BLANKING
1	DRAWING ONLY DURING RETRACE BLANKING

Figure 15. Mode Control Bits

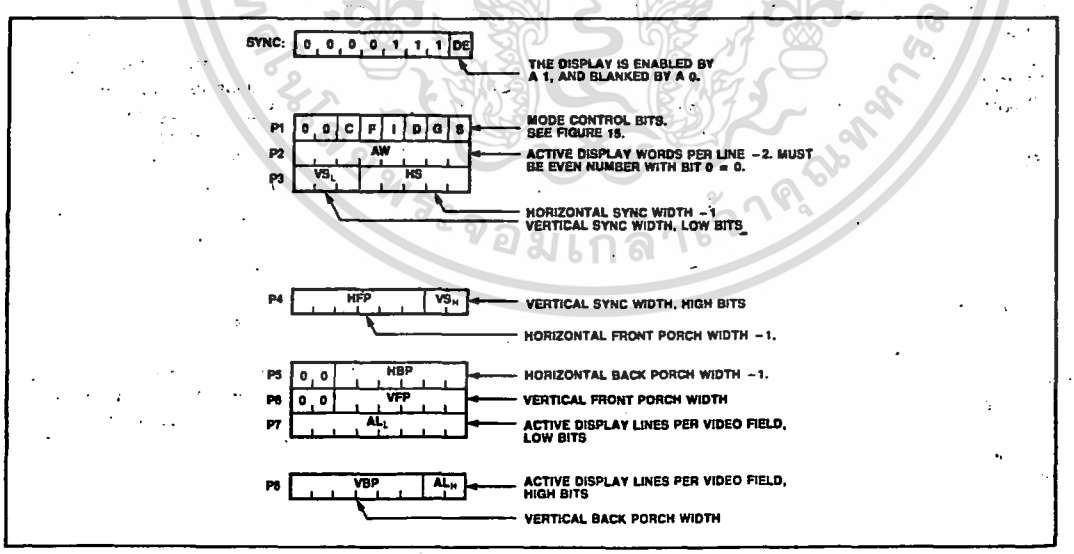


Figure 16. Sync Command

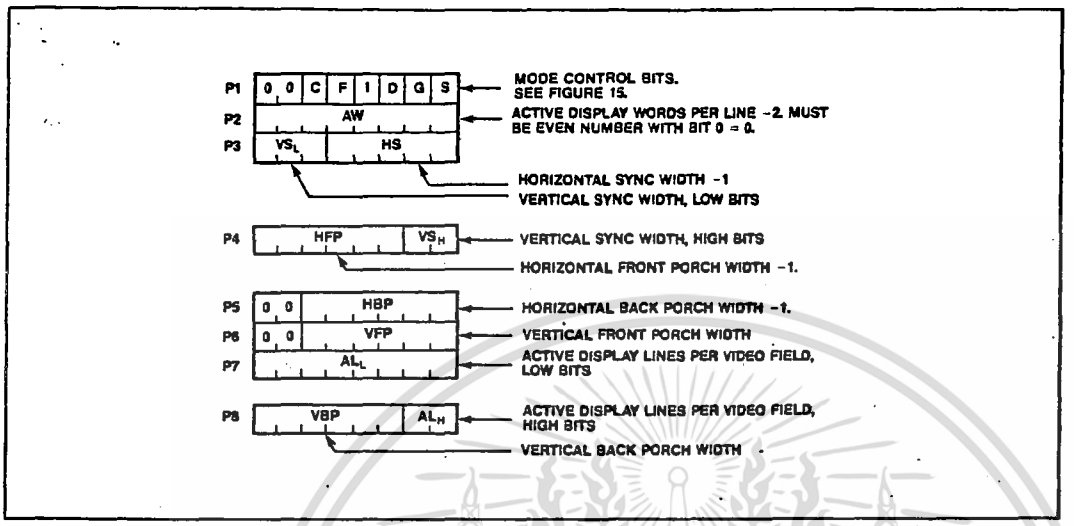


Figure 14. Optional Reset Parameters

In graphics mode, a word is a group of 16 pixels. In character mode, a word is one character code and its attributes, if any.

The number of active words per line must be an even number from 2 to 256.

An all-zero parameter value selects a count equal to 2^n where n = number of bits in the parameter field for vertical parameters.

All horizontal widths are counted in display words. All vertical intervals are counted in lines.

Sync Parameter Constraints

HORIZONTAL FRONT PORCH CONSTRAINTS

1. In general:
   HFP ≥2 words
2. If DMA is used, or the display zoom factor is greater than one in interlaced display mode:
   HFP ≥3 words
3. If the GDC is used in slave mode:
   HFP ≥4 words
4. If the light pen input is used:
   HFP ≥6 words

HORIZONTAL Sync CONSTRAINTS

1. If dynamic RAM refresh is used:
   HS ≥2 words
2. If interlaced display mode is used:
   HS ≥5 words

HORIZONTAL BACK PORCH CONSTRAINTS

1. In general:
   HBP ≥3 words
2. If interlaced display mode is used, or the IMAGE or WIDE mode bits change within one video field:
   HBP ≥ 5 words

MODE CONTROL BITS (FIGURE 15)

Repeat Field Framing:	2 Field Sequence with ½ line offset between otherwise identical fields.
Interlaced Framing:	2 Field Sequence with ½ line offset. Each field displays alternate lines.
Noninterlaced Framing:	1 field brings all of the information to the screen.

Total scanned lines in interlace mode is odd. The sum of VFP + VS + VBP + AL should equal one less than the desired odd number of lines.

Dynamic RAM refresh is important when high display zoom factors or DMA are used in such a way that not all of the rows in the RAMs are regularly accessed during display raster generation and for otherwise inactive display memory.

Access to display memory can be limited to retrace blanking intervals only, so that no disruptions of the image are seen on the screen.

SYNC Format Specify Command

This command loads parameters into the sync generator. The various parameter fields and bits are identical to those at the RESET command. The GDC is not reset nor does it enter idle mode.

Vertical Sync Mode Command

When using two or more GDCs to contribute to one image, one GDC is defined as the master sync generator, and the others operate as its slaves. The VSYNC pins of all GDCs are connected together.

Slave Mode Operation

A few considerations should be observed when synchronizing two or more GDCs to generate overlayed video via the VSYNC INPUT/OUTPUT pin. As mentioned above, the Horizontal Front Porch (HFP) must be 4 or more display cycles wide. This is equivalent to eight or more clock cycles. This gives the slave GDCs time to initialize their internal video sync generators to the proper point in the video field to match the incoming vertical sync pulse (VSYNC). This resetting of the generator occurs just after the end of the incoming VSYNC pulse, during the HFP interval. Enough time during HFP is required to allow the slave GDC to complete the operation before the start of the HSYNC interval.

Once the GDCs are initialized and set up as Master and Slaves, they must be given time to synchronize. It is a good idea to watch the VSYNC status bit of the Master GDC and wait until after one or more VSYNC pulses have been generated before the display process is started. The START command will begin the active display of data and will end the video synchronization process, so be sure there has been at least one VSYNC pulse generated for the Slaves to synchronize to.

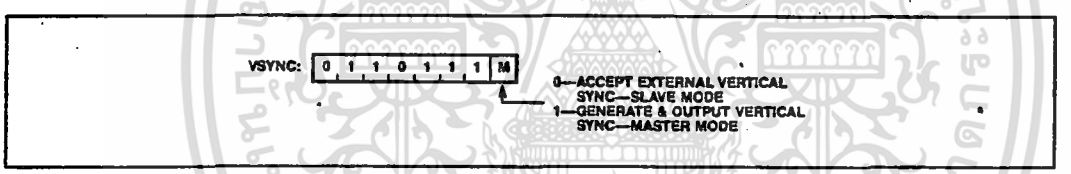


Figure 17. Vertical Sync Mode Command

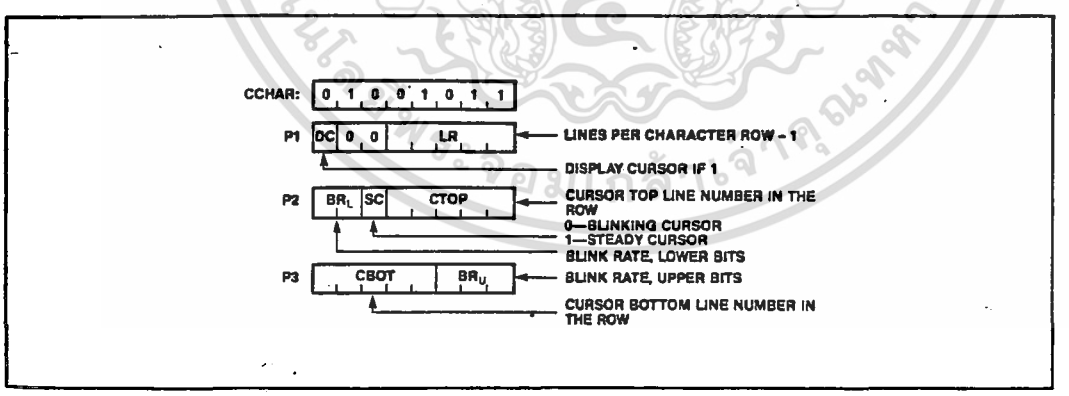


Figure 18. Cursor & Character Characteristics Command

Cursor and Character Characteristics Command

In graphics mode, LR should be set to 0. For interlaced displays in graphics mode, BR should be set to 3. The blink rate parameter controls both the cursor and attribute blink rates. The cursor blink-on-time = blink-off-time = 2 × BR (video frames). The attribute blink rate is always ½ the cursor rate but with a ¾ on-¼ off duty cycle.

DISPLAY CONTROL COMMANDS

Zoom Factors Specify Command

Zoom magnification factors of 1 through 16 are available using codes 0 through 15, respectively.

Cursor Position Specify Command

In character mode, the third parameter byte is not needed. The cursor is displayed for the word time in which the display scan address (DAD) equals the cursor address. In graphics mode, the cursor word address specifies the word containing the starting pixel of the drawing; the dot address value specifies the pixel within that word.

Parameter RAM Load Command

From the starting address, SA, any number of bytes may be loaded into the parameter RAM at incrementing addresses, up to location 15. The sequence of parameter bytes is terminated by the next command byte entered into the FIFO. The parameter RAM stores 16 bytes of information in predefined locations which differ for graphics and character modes. See the parameter RAM discussion for bit assignments.

Pitch Specification Command

This value is used during drawing by the drawing processor to find the word directly above or below the current word, and during display to find the start of the next line.

The Pitch parameter (width of display memory) is set by two different commands. In addition to the PITCH command, the RESET (or SYNC) command also sets the pitch value. The "active words per line" parameter, which specifies the width of the raster-scan display, also sets the Pitch of the display memory. In situations in which these two values are equal there is no need to execute a PITCH command.

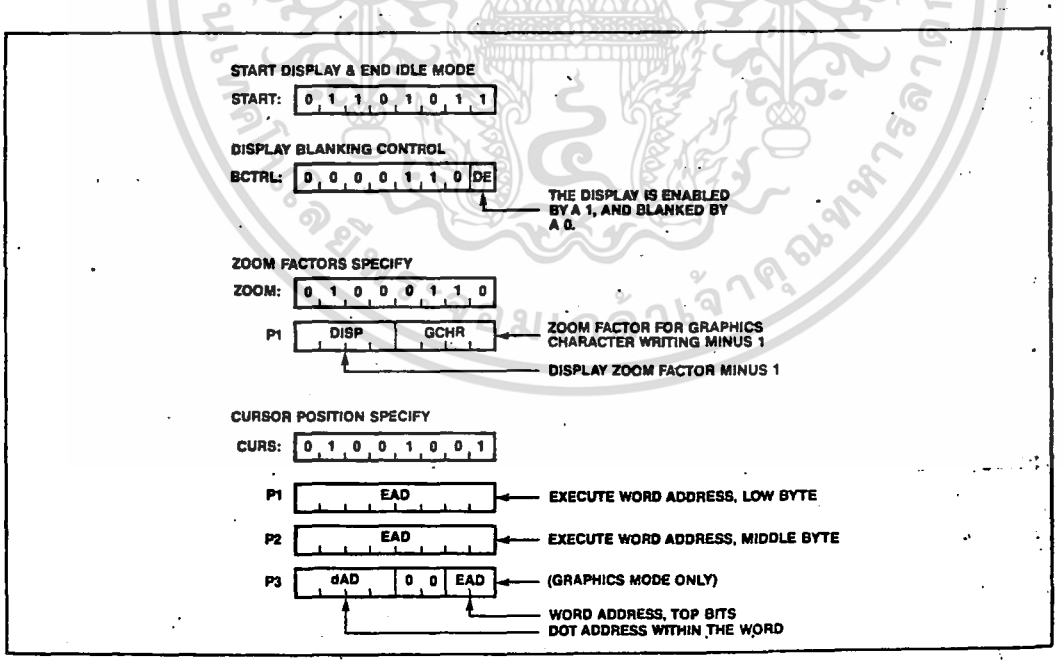


Figure 19. Display Control Commands

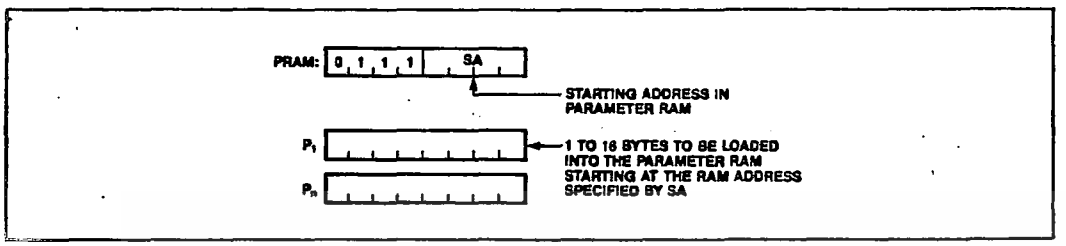


Figure 20. Parameter RAM Load Command

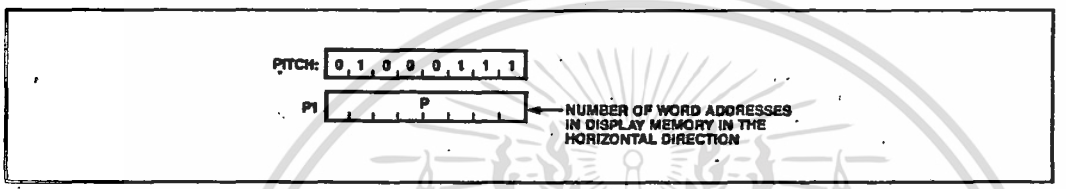


Figure 21. Pitch Specification Command

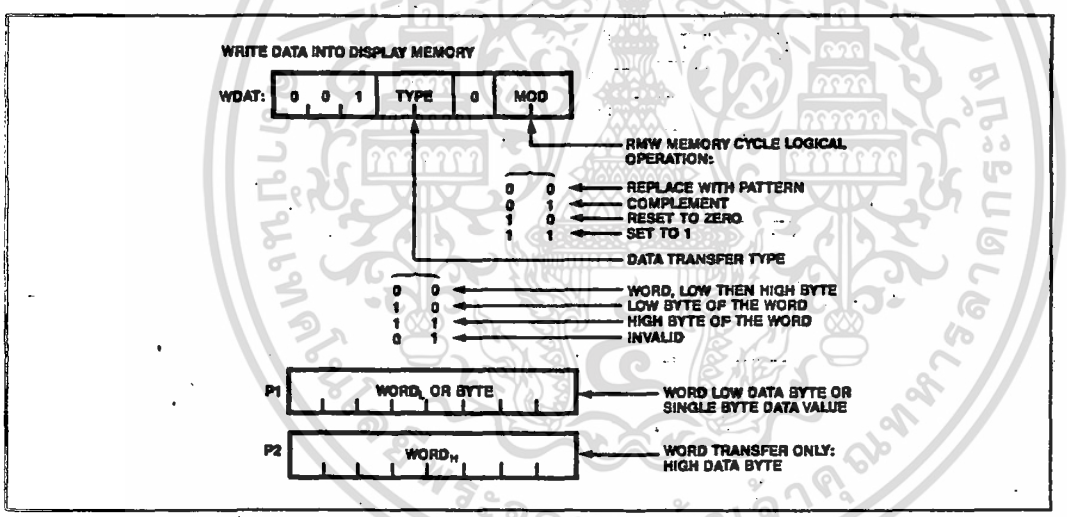


Figure 22. Write Data Command

DRAWING CONTROL COMMANDS

Write Data Command

Upon receiving a set of parameters (two bytes for a word transfer, one for a byte transfer), one RMW cycle into Video Memory is done at the address pointed to by the cursor EAD. The EAD pointer is advanced to the next word, according to the previously specified direction. More parameters can then be accepted.

For byte writes, the unspecified byte is treated as all zeros during the RMW memory cycle.

In graphics bit-map situations, only the LSB of the WDAT parameter bytes is used as the pattern in the RMW operations. Therefore it is possible to have only an all ones or all zeros pattern. In coded character applications all the bits of the WDAT parameters are used to establish the drawing pattern.

The WDAT command operates differently from the other commands which initiate RMW cycle activity. It requires parameters to set up the Pattern register while the other commands use the stored values in the parameter RAM. Like all of these commands, the

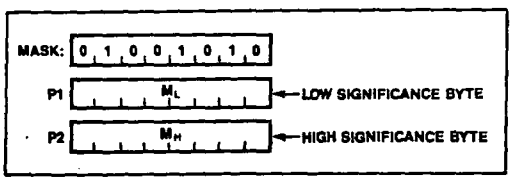
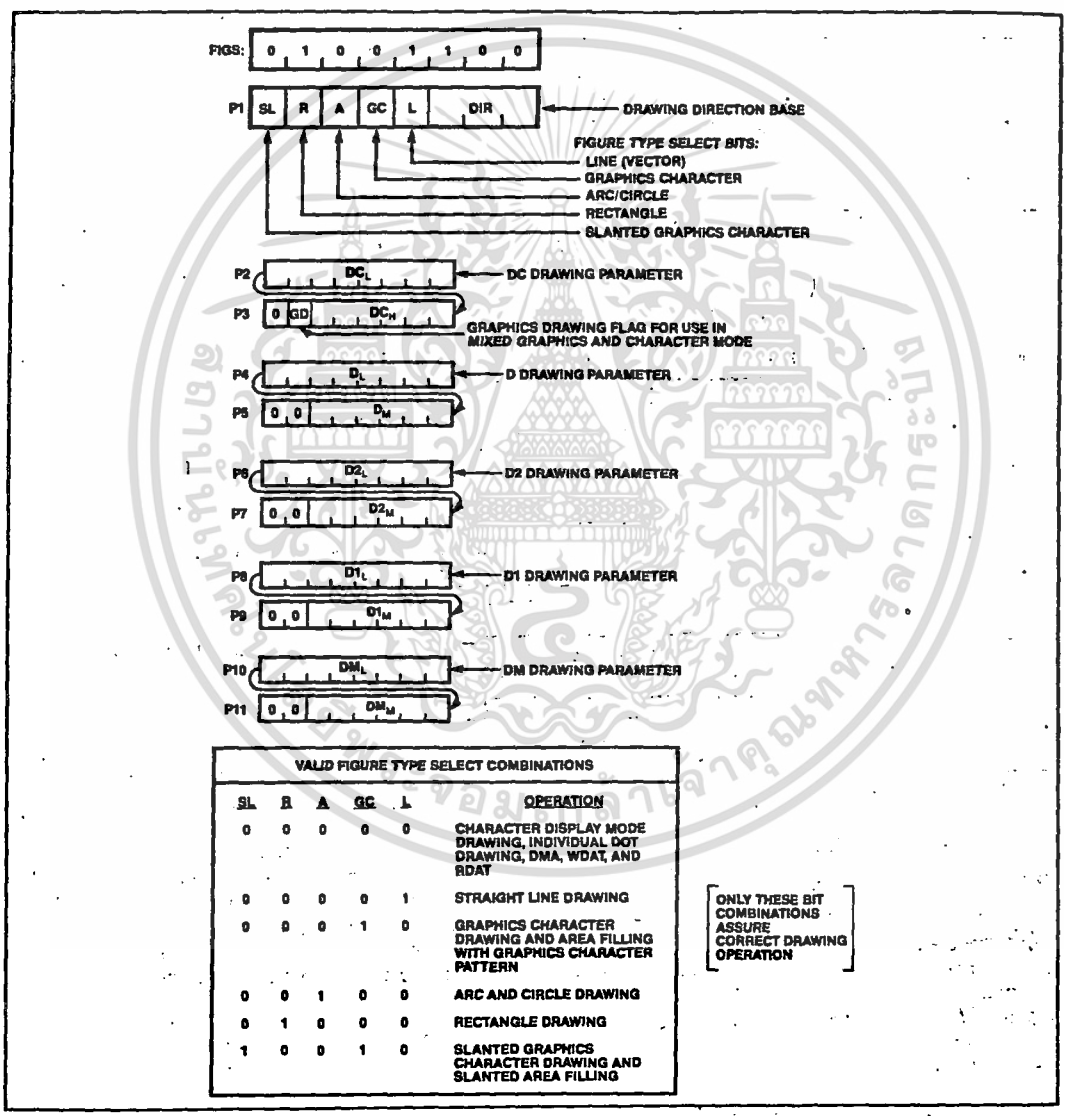


Figure 23. Mask Register Load Command

WDAT command must be preceded by a FIGS command and its parameters. Only the first three parameters need be given following the FIGS opcode, to set up the type of drawing, the DIR direction, and the DC value. The DC parameter +1 will be the number of RMW cycles done by the GDC with the first set of WDAT parameters. Additional sets of WDAT parameters will see a DC value of 0 which will cause only one RMW cycle to be executed.



VALID FIGURE TYPE SELECT COMBINATIONS

SL	R	A	GC	L	OPERATION
0	0	0	0	0	CHARACTER DISPLAY MODE DRAWING, INDIVIDUAL DOT DRAWING, DMA, WDAT, AND RDAT
0	0	0	0	1	STRAIGHT LINE DRAWING
0	0	0	1	0	GRAPHICS CHARACTER DRAWING AND AREA FILLING WITH GRAPHICS CHARACTER PATTERN
0	0	1	0	0	ARC AND CIRCLE DRAWING
0	1	0	0	0	RECTANGLE DRAWING
1	0	0	1	0	SLANTED GRAPHICS CHARACTER DRAWING AND SLANTED AREA FILLING

ONLY THESE BIT COMBINATIONS ASSURE CORRECT DRAWING OPERATION

Figure 24. Figure Drawing Parameters Specify Command

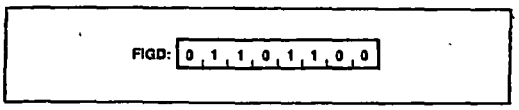


Figure 25. Figure Draw Start Command

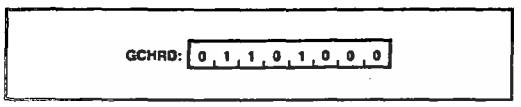


Figure 26. Graphics Character Draw and Area Filling Start Command

Mask Register Load Command

This command sets the value of the 16-bit Mask register of the figure drawing processor. The Mask register controls which bits can be modified in the display memory during a read-modify-write cycle.

The Mask register is loaded both by the MASK command and the third parameter byte of the CURS command. The MASK command accepts two parameter bytes to load a 16-bit value into the MASK register. All 16 bits can be individually one or zero, under program control. The CURS command on the other hand, puts a "1 of 16" pattern into the Mask register based on the value of the Dot Address value, dAD. If normal single-pixel-at-a-time graphics figure drawing is desired, there is no need to do a MASK command at all since the CURS command will set up the proper pattern to address the proper pixels as drawing progresses. For coded character DMA, and screen setting and clearing operations using the WDAT command, the MASK command should be used after the CURS command if its third parameter byte has been output. The Mask register should be set to all ones for any "word-at-a-time" operation.

Figure Draw Start Command

On execution of this instruction, the GDC loads the parameters from the parameter RAM into the drawing processor and starts the drawing process at the pixel pointed to by the cursor, EAD, and the dot address, dAD.

Graphics Char. Draw and Area Fill Start Command

Based on parameters loaded with the FIGS command, this command initiates the drawing of the graphics character or area filling pattern stored in Parameter RAM. Drawing begins at the address in display memory pointed to by the EAD and dAD values.

DATA READ COMMANDS

Read Data Command

Using the DIR and DC parameters of the FIGS command to establish direction and transfer count, multiple RMW cycles can be executed without specification of the cursor address after the initial load (DC = number of words or bytes).

As this instruction begins to execute, the FIFO buffer direction is reversed so that the data read from display memory can pass to the microprocessor. Any commands or parameters in the FIFO at this time will be lost. A command byte sent to the GDC will immediately reverse the buffer direction back to write mode, and all RDAT information not yet read from the FIFO will be lost. MOD should be set to 00.

Cursor Address Read Command

The Execute Address, EAD, points to the display memory word containing the pixel to be addressed.

The Dot Address, dAD, within the word is represented as a 1-of-16 code.

Light Pen Address Read Command

The light pen address, LAD, corresponds to the display word address, DAD, at which the light pen input signal is detected and deglitched.

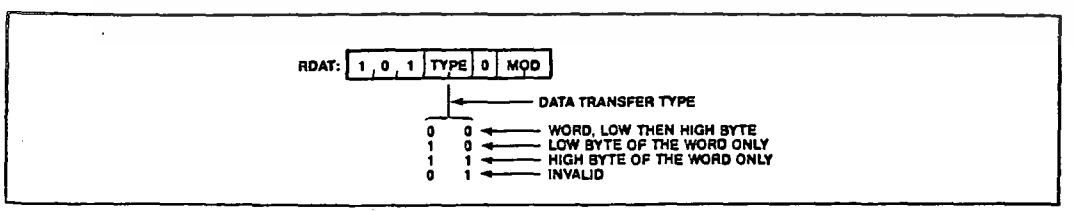


Figure 27. Read Data from Display Memory Command

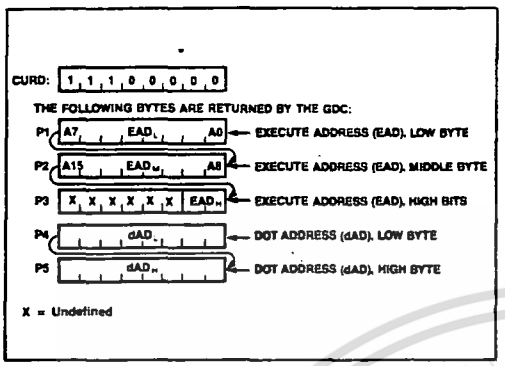


Figure 28. Cursor Address Read Command

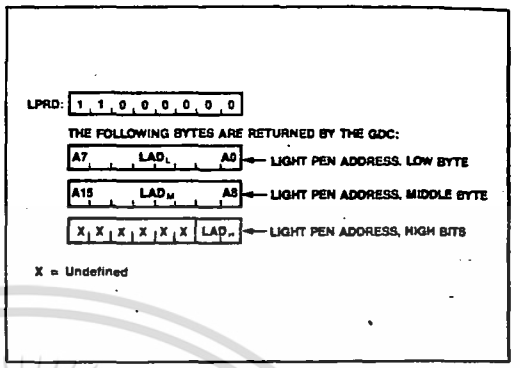


Figure 29. Light Pen Address Read Command

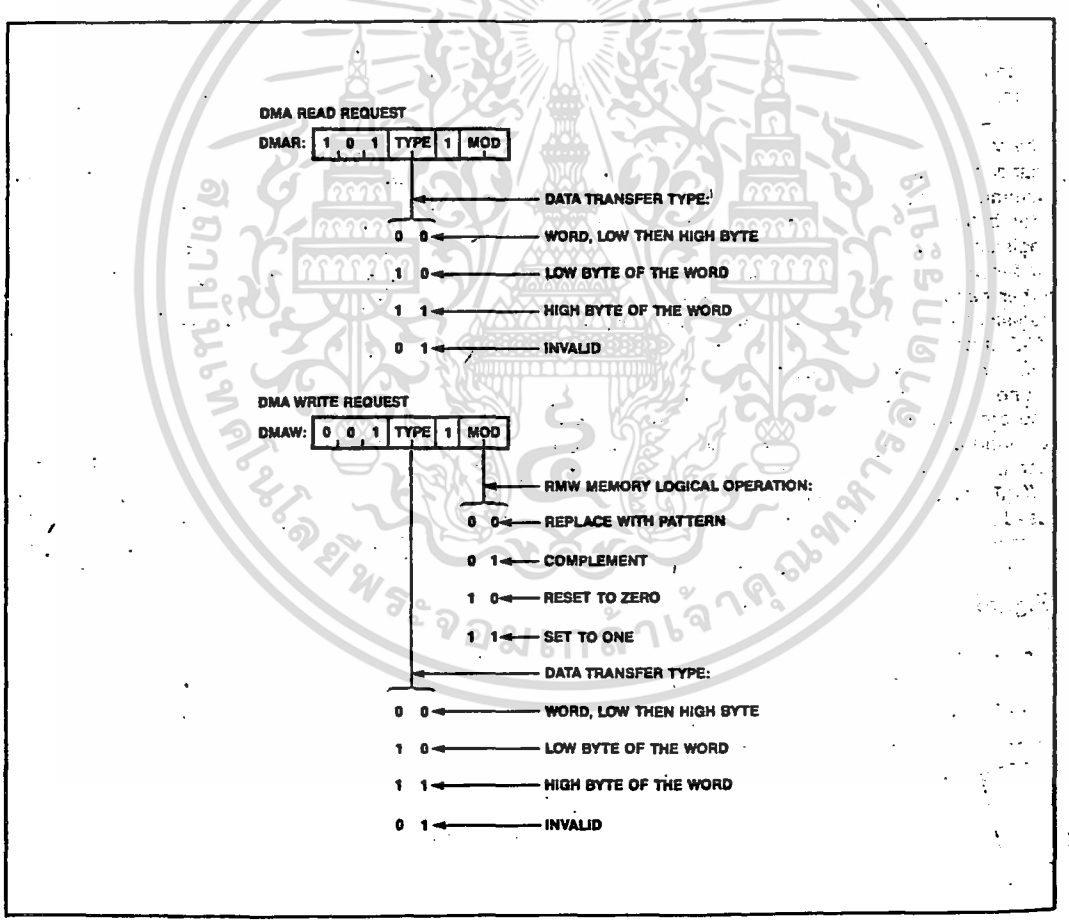


Figure 30. DMA Control Commands

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 ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watt

COMMENT: Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS

T_A = 0°C to 70° C; V_{CC} = 5V ± 10%; GND = 0V

Symbol	Parameter	Limits		Unit	Conditions
		Min.	Max.		
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{OZ}	Output Leakage Current		±10	μA	$V_{SS}+0.45 \leq V_I \leq V_{CC}$
I_{IL}	Input Leakage Current		±10	μA	$V_{SS} \leq V_I \leq V_{CC}$
V_{CL}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CH}	Clock Input High Voltage	3.5	V_{CC} + 0.5	V	
I_{CC}	V_{CC} Supply Current		270	mA	Typical = 150 mA

CAPACITANCE

T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Limits		Unit	Conditions
		Min.	Max.		
C_{IN}	Input Capacitance		10	pF	fc = 1 MHz V = 0
C_{IO}	I/O Capacitance		20	pF	
C_{OUT}	Output Capacitance		20	pF	
C_O	Clock Input Capacitance		20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{SS} = 0V, V_{CC} = +5V ±10%)

DATA BUS READ CYCLE

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{AR}	A_0 setup to $\overline{RD}$ ↓	0		0		0		ns	
T_{RA}	A_0 hold after $\overline{RD}$ ↑	0		0		0		ns	
T_{RR}	$\overline{RD}$ Pulse Width	$T_{RD}+20$		$T_{RD}+20$		$T_{RD}+20$		ns	
T_{RD}	$\overline{RD}$ ↓ to Data Out Delay		120		80		70	ns	CL = 50pF
T_{DF}	$\overline{RD}$ ↑ to Data Float Delay	0	120	0	100	0	90	ns	
T_{RV}	$\overline{RD}$ Recovery Time	4 T_{CY}		4 T_{CY}		4 T_{CY}		ns	

DATA BUS WRITE CYCLE

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{AW}	A_0 Setup to $\overline{WR}$ ↓	0		0		0		ns	
T_{WA}	A_0 Hold after $\overline{WR}$ ↑	0		0		10		ns	
T_{WW}	$\overline{WR}$ Pulse Width	120		100		90		ns	
T_{DW}	Data Setup to $\overline{WR}$ ↑	100		80		70		ns	
T_{WD}	Data Hold after $\overline{WR}$ ↑	0		0		10		ns	
T_{RV}	$\overline{WR}$ Recovery Time	4 T_{CY}		4 T_{CY}		4 T_{CY}		ns	

DISPLAY MEMORY TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{CA}	Address/Data Delay from 2XWCLK ↑	30	160	30	130	30	110	ns	CL = 50pF
T_{AC}	Address/Data Hold Time	30	160	30	130	30	110	ns	CL = 50pF
T_{DC}	Data Setup to 2XWCLK ↓	0		0		0		ns	
T_{CD}	Data Hold Time	$T_{IE}+20$		$T_{IE}+20$		$T_{IE}+20$		ns	
T_{IE}	2XWCLK ↑ to $\overline{DBIN}$	30	120	30	90	30	80	ns	CL = 50pF
T_{CAH}	2XWCLK ↑ to ALE ↑	30	125	30	100	30	90	ns	CL = 50pF
T_{CAL}	2XWCLK ↑ to ALE ↓	30	100	30	80	30	70	ns	CL = 50pF
T_{AL}	ALE Low Time	$T_{CY}+30$		$T_{CY}+30$		$T_{CY}+30$		ns	
T_{AH}	ALE High Time	$T_{CH}-20$		$T_{CH}-20$		$T_{CH}-20$		ns	
T_{CO}	Video Signal Delay from 2XWCLK ↑		150		120		100	ns	

A.C. CHARACTERISTICS (Continued)

OTHER TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{PC}	LPEN or VSYNC Input Setup to 2XWCLK↑	30		20		15		ns	
T_{PP}	LPEN or VSYNC Input Pulse Width	T_{CY}		T_{CY}		T_{CY}		ns	

CLOCK TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{CY}	Clock Period	250	2000	200	2000	180	2000	ns	
T_{CH}	Clock High Time	105		80		70		ns	
T_{CL}	Clock Low Time	105		80		70		ns	
T_R	Rise Time		20		20		20	ns	
T_F	Fall Time		20		20		20	ns	

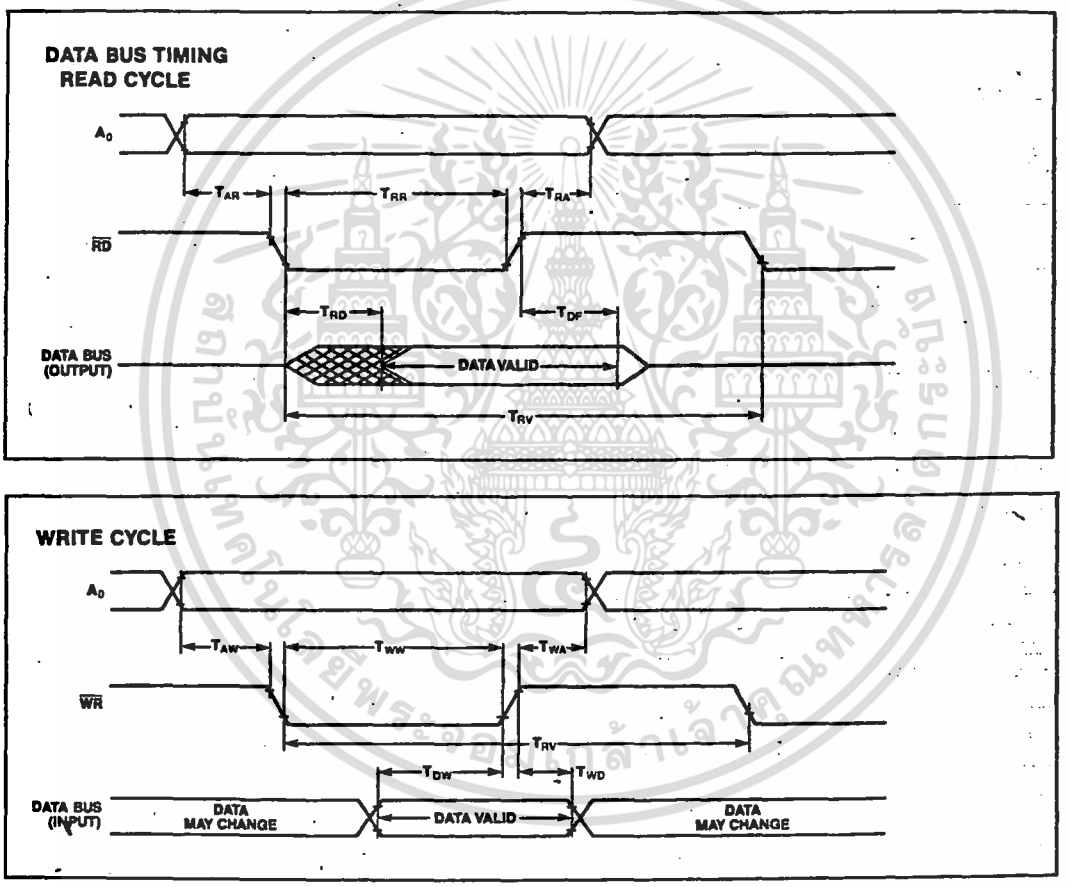
DMA TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{ACC}	$\overline{DACK}$ Setup to $\overline{RD}$↓ or $\overline{WR}$↓	0		0		0		ns	
T_{CAC}	$\overline{DACK}$ Hold from $\overline{RD}$↑ or $\overline{WR}$↑	0		0		0		ns	
T_{RR1}	$\overline{RD}$ Pulse Width	$T_{RD1}+20$		$T_{RD1}+20$		$T_{RD1}+20$		ns	
T_{RD1}	$\overline{RD}$↓ to Data Out Delay		$1.5\,T_{CY}+120$		$1.5\,T_{CY}+80$		$1.5\,T_{CY}+70$	ns	CL = 50pF
T_{KQ}	2XWCLK↑ to DREQ Delay		150		120		100	ns	CL = 50pF
T_{RQAK}	DREQ Setup to DACK↓	0		0		0		ns	
T_{AKRQ}	$\overline{DACK}$↓ to DREQ↓ Delay		$T_{CY}+150$		$T_{CY}+120$		$T_{CY}+100$	ns	CL = 50pF
T_{AKH}	$\overline{DACK}$ High Time	T_{CY}		T_{CY}		T_{CY}		ns	
T_{AK1}	$\overline{DACK}$ Cycle Time, Word Mode	$4\,T_{CY}$		$4\,T_{CY}$		$4\,T_{CY}$		ns	
T_{AK2}	$\overline{DACK}$ Cycle Time, Byte Mode	$5\,T_{CY}$		$5\,T_{CY}$		$5\,T_{CY}$		ns	

A.C. TEST CONDITIONS

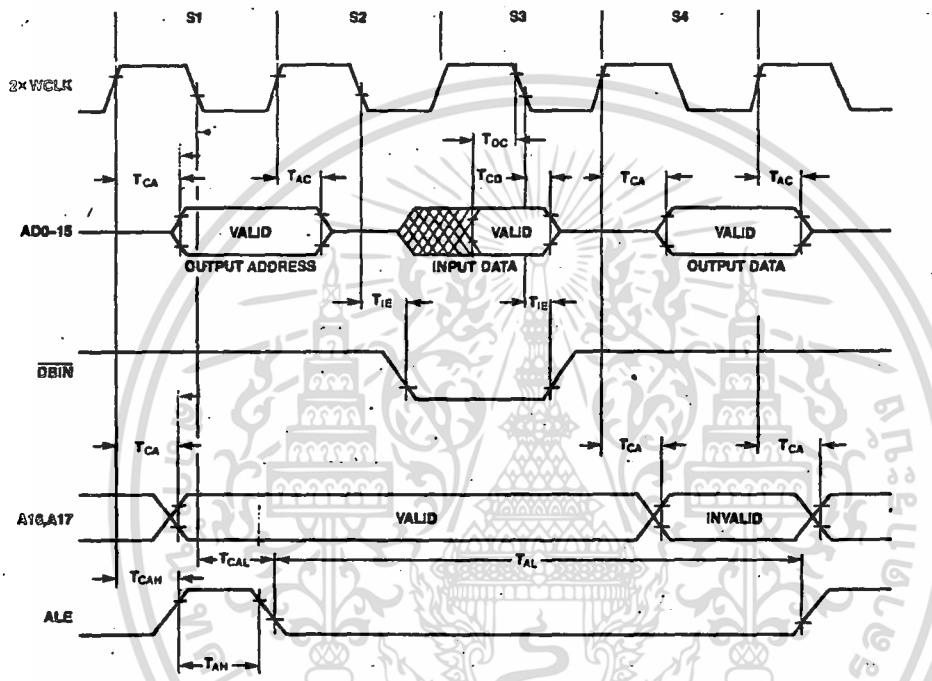
Input Pulse Levels (except 2XWCLK) ..... 0.45V to 2.4V
Input Pulse Levels (2XWCLK) ..... 0.3V to 3.9V
Timing Measurement Reference Levels (except 2XWCLK) ..... 0.8V to 2.0V
Timing Measurement Reference Levels (2XWCLK) ..... 0.6V to 3.5V

WAVEFORMS



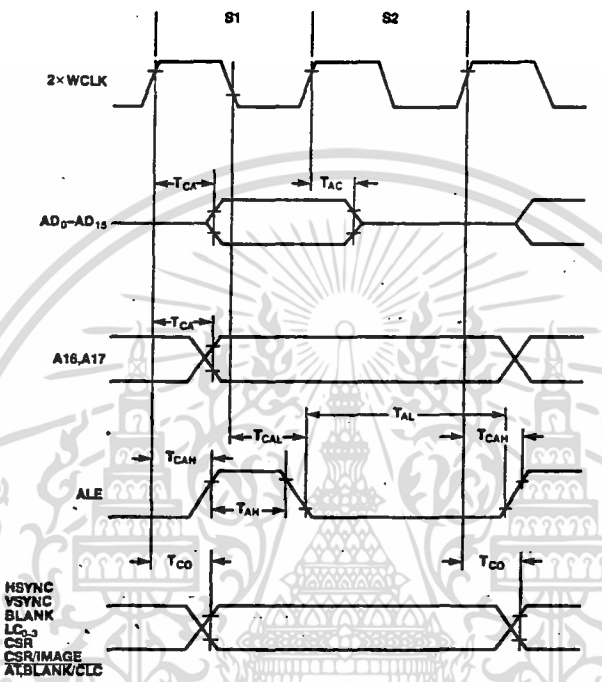
WAVEFORMS (Continued)

DISPLAY MEMORY TIMING
READ/MODIFY/WRITE CYCLE

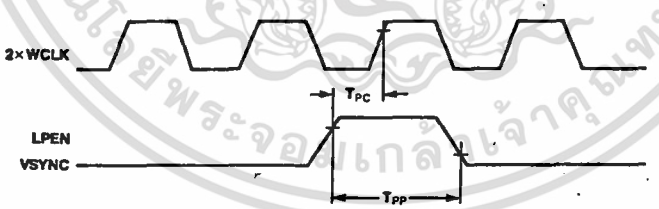


WAVEFORMS (Continued)

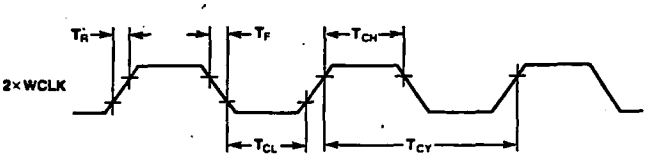
DISPLAY MEMORY TIMING (Continued)
READ CYCLE



OTHER TIMING

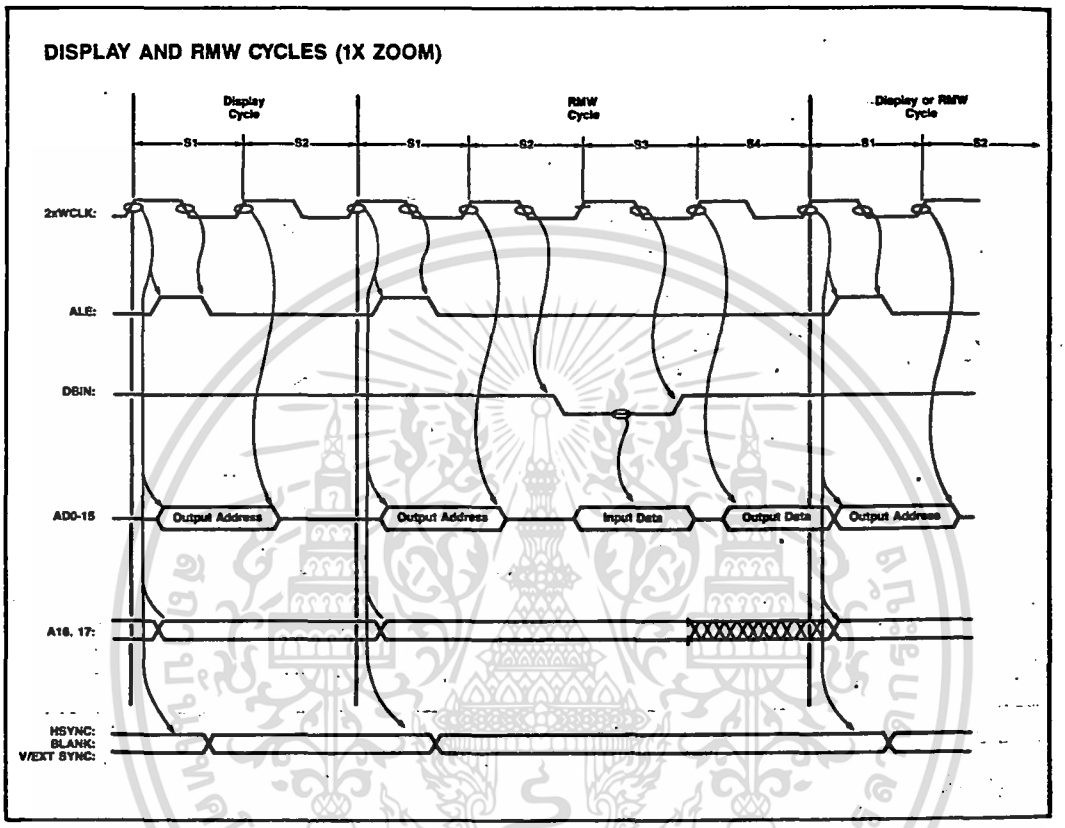


CLOCK TIMING



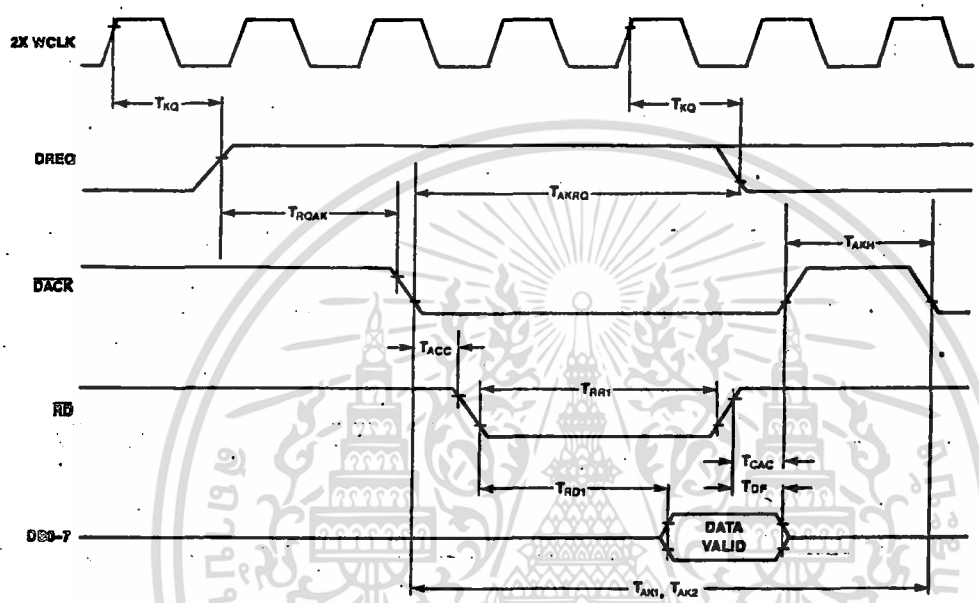
WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (1X ZOOM)

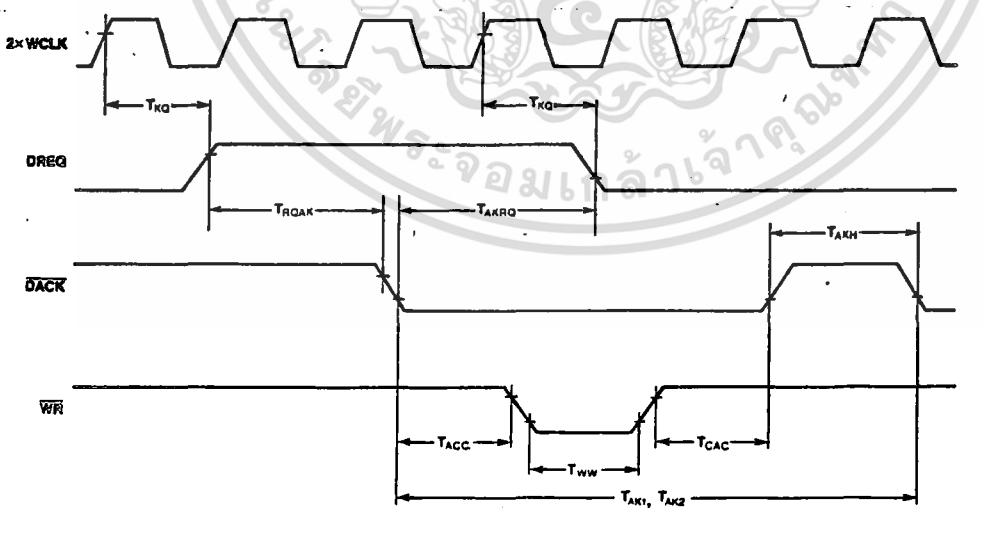


WAVEFORMS (Continued)

DMA TIMING READ

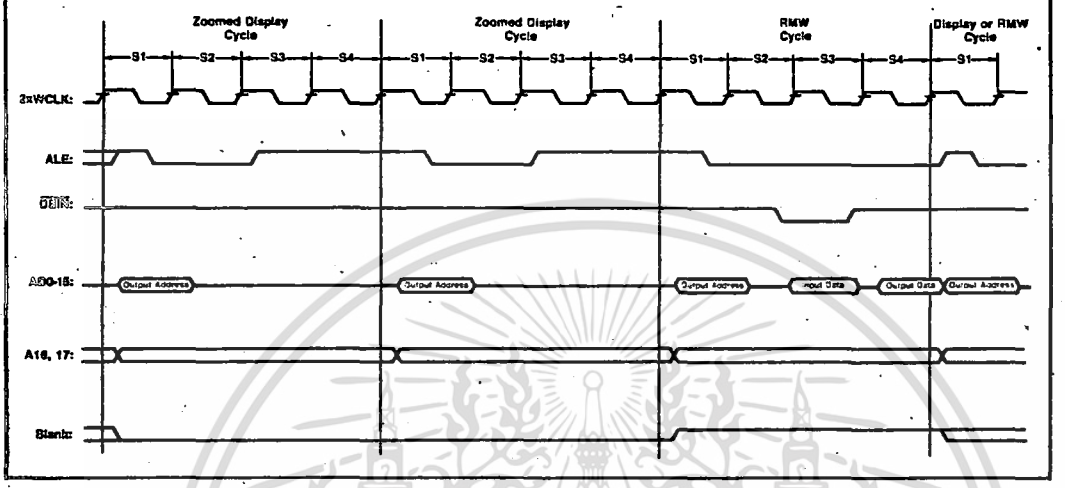


WRITE

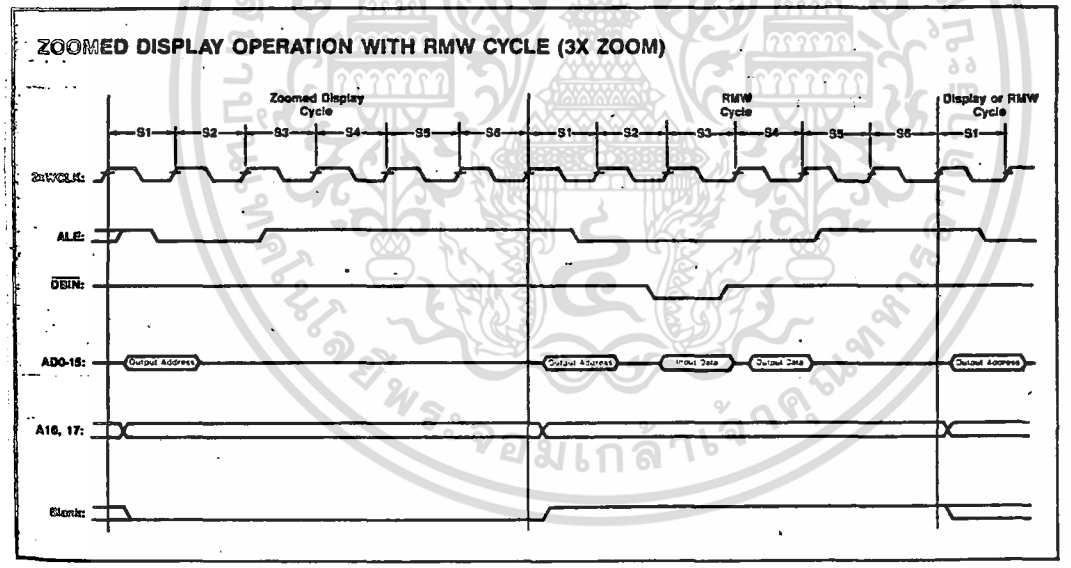


WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (2X ZOOM)

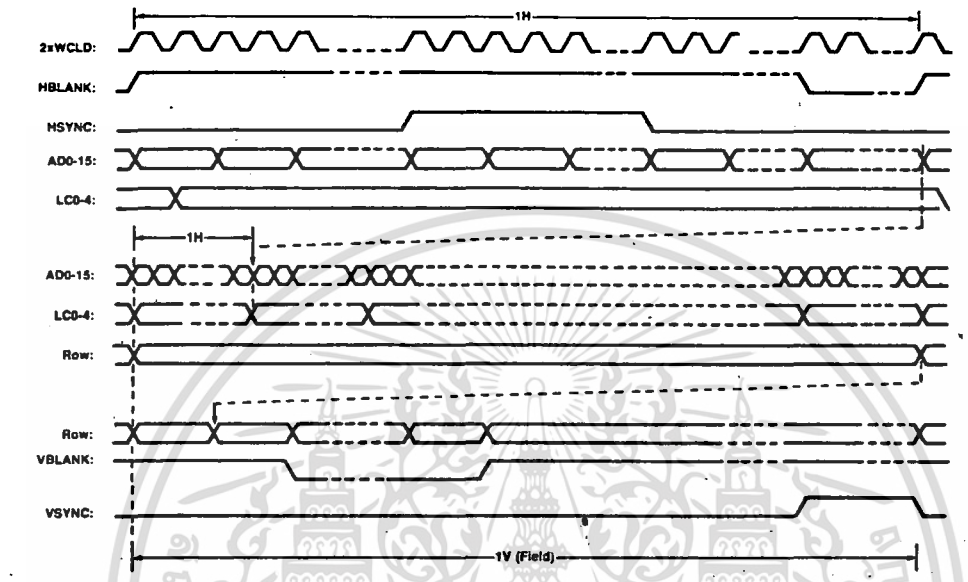


ZOOMED DISPLAY OPERATION WITH RMW CYCLE (3X ZOOM)

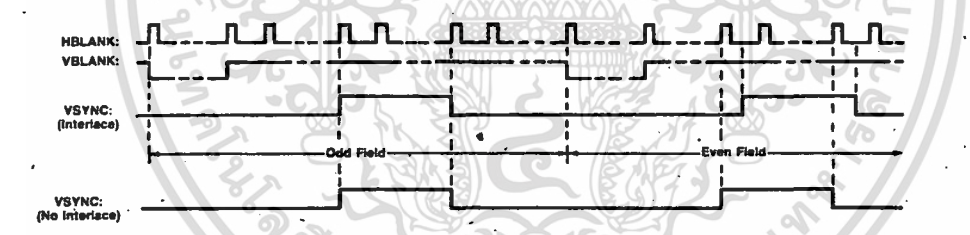


WAVEFORMS (Continued)

VIDEO SYNC SIGNALS TIMING

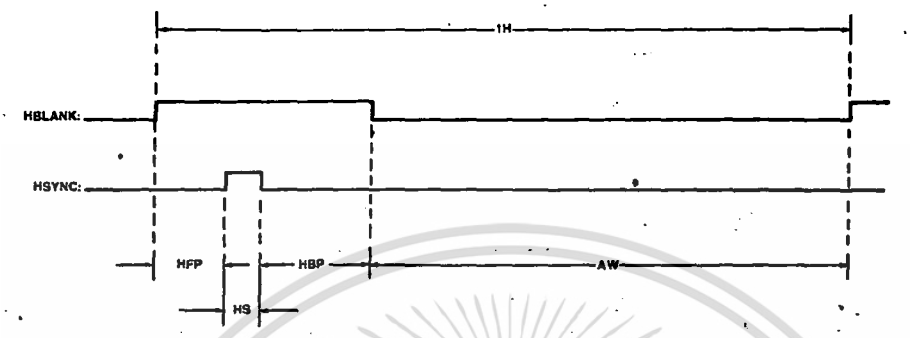


INTERLACED VIDEO TIMING

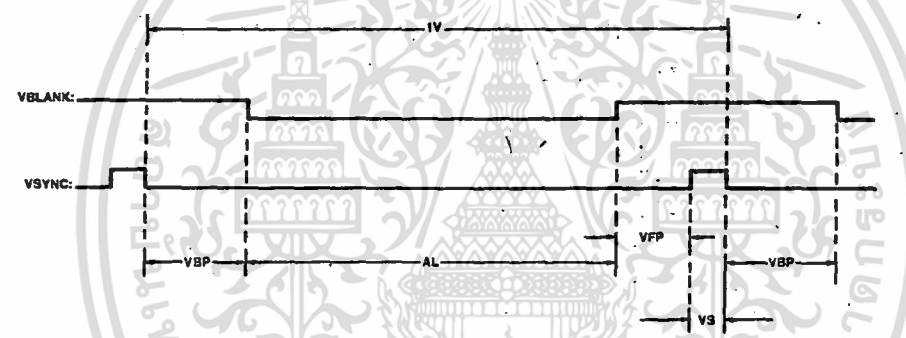


WAVEFORMS (Continued)

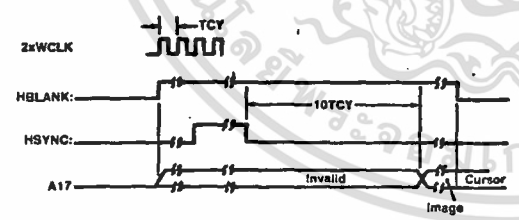
VIDEO HORIZONTAL SYNC GENERATOR PARAMETERS



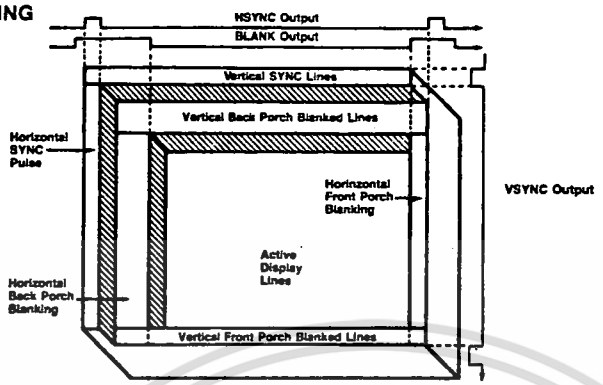
VIDEO VERTICAL SYNC GENERATOR PARAMETERS



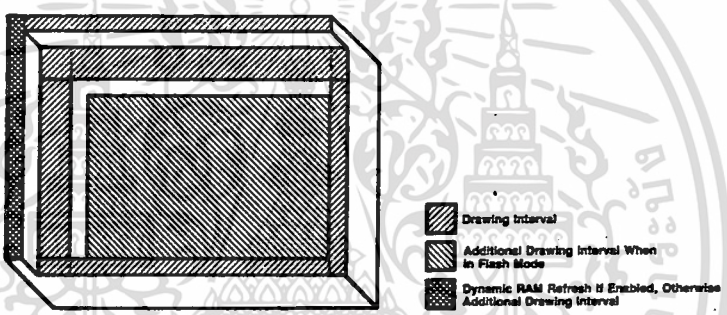
CURSOR—IMAGE BIT FLAG



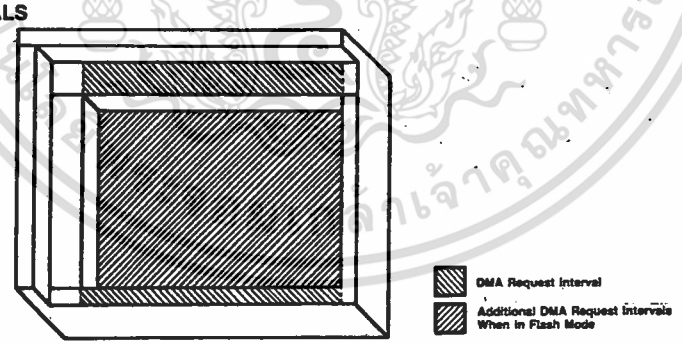
VIDEO FIELD TIMING



DRAWING INTERVALS



DMA REQUEST INTERVALS



ภาคผนวก 3

โปรแกรมที่ใช้ในการทดลอง

```
Text File List Processing Program V4.00C Page : 1
File : PROMON.PAS
Current Date : Thursday September 1, 1988
Current Time : 4:34 PM

Program : Process Monitoring Main Program
Programmer : Phakorn Hutasangkas.

Program ProcessMonitoring;

uses Crt,Grph7220,ReadPDII,Utility;

var
  Ch : char;

begin
  SetConfig;
  PrintTitle;
  repeat
    Menu;
    if not ExitToDOS then
    begin
      TextColor(LightMagenta+Blink);
      Write('System Initialize....');
      InitGraph;
      SetColor(_White);
      DrawBorder;
      TextColor(LightCyan);
      GotoXY(1,WhereY);
      PlotProDesign;
      if (Mode = Demo) or (Mode = Execute) then
        ProcessMonitor
      else
      begin
        repeat
          if Keypressed then Ch := ReadKey;
          ReadJoyStick;
        until Ch = Esc;
      end;
    end;
    Title;
    TextColor(LightMagenta+Blink);
    Write('System Initialize....');
    InitGraph;
    SetColor(_White);
    DrawBorder;
  until ExitToDOS;
  ClrScr;
  TextColor(White);
  TextBackGround(Black);
  Writeln('Program Terminated.');
end.
```

```
Program : GDC Driver Software
Programmer : Phakorn Hutasangkas.

Unit Grph7220;

interface

uses Crt;

const
  _Black = 0;
  _Blue = 1;
  _Green = 2;
  _Cyan = 3;
  _Red = 4;
  _Magenta = 5;
  _Yellow = 6;
  _White = 7;

const
{Key control Code }
  Esc              =   #27;
  Up               =   #72;
  Down             =   #80;
  Right            =   #77;
  Left             =   #75;
  LeftUp           =   #71;
  LeftDown         =   #79;
  RightUp          =   #73;
  RightDown        =   #81;
{Screen set data}
  Pix              =   $40;{ 40 X=1024} { 78 (1920*480)}
  HorWord          =   $40;
  ClrMark          =   $23;
  ComplMark        =   $21;
  Bst              =   $23;
  HorMark          =   $40;
  VerMark          =   $20;
  FIFOEmptyMark    =   $04;
  DRMark           =   $01;
  DrawMark         =   $03;
{Filling area Code}
  Slant            =   $80;
  GrChr            =   $10;
  SlChr            =   $90;
{Line type Code}
  DotLine0         =   $0000;
  DotLine1         =   $AAAA;
```

```
  DotLine2            =    $8888;
  DotLine3            =    $8080;
  DotLine4            =    $CCCC;
  DotLine5            =    $EEEE;
  DotLine6            =    $FCFC;
  DotLine7            =    $FEFE;
  DotLine8            =    $FFFF;
  Page0               =    0;
  Page1               =    1;
  Page2               =    2;
  Page3               =    3;

type
  Table    = record
               Para : array [0..31,0..7] of byte;
             end;
  DFormat  = record
               Code : byte;
               Para : array [0..3] of byte;
             end;
  PFormat  = record
               Code : byte;
               Para : array [0..2] of byte;
             end;
  LFormatT =record
               Code : byte;
               Para : byte;
             end;
  EFormat = record
               Code : byte;
               Para : array [0..7] of byte;
             end;
  CharD   = array [0..31] of byte;
  CircleDir =  array [0..9] of byte;
  CData     =  array [0..3,0..1] of integer;

const
  pramc     :  DFormat =
              (Code   :  $70;
               Para   : ($00,
                         $00,
                         $00,     {len1}
                         $10));   {len2}

  {720*512 set Lenght of display 100}
  {720*720  -    -    -    -    168}

const
  sync      :  EFormat =
              (Code   :  $0F; {DO=1: DISPLAY IS ENABLE}
               Para   : ($1F, { [0 0 C F I D G S] }
                              { [0 0 0 1 1 1 1 1] }
                              { GRAPHIC MODE      }
                              { INTERACED MODE    }
```

```
                        { NO REFRESH (SRAM) }
                        { DRAW DURING BLANKING }
                  $2B, { [  26 AW        ] }
                  $65, { [ VS  :   HS    ] }
                       { [1 0 0 0 0 1 0 1] }
            {17}  $0F, { [  HFP      :VS ] }
                       { [0 0 0 0 1 1 0 0] }
                  $03, { [0 0:    HBP    ] }
                  $0A, { [0 0:    VFP    ] }
                  $00, { [      AL       ] }
                  $19  { [ VBP       :AL ] }));
                       { [0 1 0 0 0 1 0 0] }
resetT   : EFormat =
          (Code  :  $00; {DO=1: DISPLAY IS ENABLE}
           Para  : ($1F, { [0 0 C F I D G S] }
                         { [0 0 0 1 1 0 1 1] }
                         { GRAPHIC MODE      }
                         { INTERACED MODE    }
                         { NO REFRESH (SRAM) }
                         { DRAW DURING BLANKING }
                  $2b, { [  26 AW        ] }
                  $85, { [ VS  :   HS    ] }
            {17}  $0F, { [  HFP      :VS ] }
                       { [0 0 0 1 0 1 1 1] }
                  $03, { [0 0:    HBP    ] }
                  $06, { [0 0:    VFP    ] }
                  $90, { [      AL       ] }
                  $41  { [ VBP       :AL ] })); {500line}
                       { [0 1 0 0 0 1 0 0] }

                { 800*720 } { AW  = 32H }
                            { HFP = 5   }
                            { HBP = 3   }
                            { VFP = 1   }
                            { VBP = 9   }
                            { AL  = 168H}
                            { 720*720 } { AW  = 2CH }
                            { HFP = 5   }
                            { HBP = 5   }
                            { VFP = 1   }
                            { VBP = 9   }
                            { AL  = 168H}

                 { 720*340 } { AW  = 2CH }
                             { HFP = 5   }
                             { HBP = 5   }
                             { VFP = 5   }
                             { VBP = 11  }
                             { AL  = AA  }

                { 720*512 } { AW  = 2CH }
                            { HFP = 5   }
                            { HBP = 5   }
                            { VFP = 5   }
```

```
                        { VBP = 9   }
                        { AL  = 100H}

var
  NameCode       : byte;
  Num            : integer;
  line_base_addr : integer;
  Filevar        : Text;
  Ratio          : real;
  i,j,x,y        : integer;
  addr,r         : integer;
  m,n            : byte;
  buffer         : byte;
  FIFO_FULL      : boolean;
  NumD           : integer;
  EAD            : word;
  daD            : byte;
  Lformat        : LFormatT;
  ch             : CHAR;
  dir            : byte;
  MasterScreenLine : integer;
  SlaveScreenLine  : integer;
  Page             : integer;
  ChCount    : integer;
  Col      : byte;
  DrawColor : byte;
  ZoomFactor : real;
  Button1,Button2 : boolean;
  Color : byte;
  CurrentLevel : byte;
  Year,Month,Day,DayOfWeek : word;
  Hour,Min,Sec,Tic : word;

procedure InitGraph;

procedure Check_DFIFO;

procedure Screen(setype:byte;Page:integer);

procedure WDAT(form:byte;datamask:word);

procedure SetColor(ScreenColor : byte);

procedure Line(X1,Y1,X2,Y2 : word);

procedure Zoom(size:byte);

procedure DrawBorder;

procedure Scrolling(direction:CHAR);

procedure DrawLevel(X1,Y1,X2,Y2 : word; var Percent : integer);

implementation
```

```
procedure Reset82720;
begin
  Port[$301] := resetT.Code;
  for i := 0 to 7 do
  begin
    Port[$300] := resetT.Para[i];
  end;
end;

procedure Sync82720;
begin
  Port[$301] := sync.Code;
  for i := 0 to 7 do
  begin
    Port[$300] := sync.Para[i];
  end;
end;

procedure CheckFIFO;
begin
  repeat until (Port[$300] and FIFOEmptyMark > 0);
end;

procedure Check_DFIFO;
begin
  repeat until (Port[$300] and DrawMark <= 0);
end;

procedure Pram;
begin
  CheckFIFO;
  Port[$301] := pramc.Code;
  for i := 0 to 3 do
  begin
    CheckFIFO;
    Port[$300] := pramc.Para[i];
  end;
end;

procedure Zoom(size:byte);
begin
  CheckFIFO;
  Port[$301] :=  $46;
  Port[$300] :=  size;
end;

procedure Bctrl1;
begin
  CheckFIFO;
  Port[$301] :=  $0D;
end;

procedure WDAT(form:byte;datamask:word);
```

```
begin
  CheckFIFO;
  Port[$301] := form;
  Port[$300] := Lo(datamask);
  Port[$300] := Hi(datamask);
end;

procedure Cchar;
begin
  CheckFIFO;
  Port[$301] := $4B;
  Port[$300] := $00{81};
  Port[$300] := $C0{00};
  Port[$300] := $00{30};
end;

procedure Curs(daD:byte;address:integer);
begin
  CheckFIFO;
  Port[$301] := $49;
  Port[$300] := Lo(address);
  Port[$300] := Hi(address);
  Port[$300] := daD;
end;

procedure Figs(com:byte;DC,D,D2,D1,DM:integer);
begin
  Port[$301] := $4C;
  Port[$300] := com;
  Port[$300] := Lo(DC);  { DC L }
  Port[$300] := Hi(DC);  { DC H }
  Port[$300] := Lo(D) ;  { D  L }
  Port[$300] := Hi(D) ;  { D  H }
  Port[$300] := Lo(D2);  { D2 L }
  Port[$300] := Hi(D2);  { D2 H }
  Port[$300] := Lo(D1);  { D1 L }
  Port[$300] := Hi(D1);  { D1 H }
  Port[$300] := Lo(DM);  { DM L }
  Port[$300] := Hi(DM);  { DM H }
end;

procedure LFigs(com:byte;DC:integer);
begin
  Port[$301] := $4C;
  Port[$300] := com;
  Port[$300] := Lo(DC);  { DC L }
  Port[$300] := Hi(DC);  { DC H }
end;

procedure Mask(dat:word);
begin
  Port[$301] := $4A;
  Port[$300] := Lo(dat);
  Port[$300] := Hi(dat);
```

```
end;

procedure byteMask(Lodat,Hidat:byte);
begin
  Port[$301] := $4A;
  Port[$300] := Lodat;
  Port[$300] := Hidat;
end;

procedure Figd;
begin
  Port[$301] := $6C;
end;

procedure Pitch;
begin
  Port[$301] :=$47;
  Port[$300] :=Pix;
end;

procedure ConvertAddr(x,y:integer;var daD:byte;var EAD:word);
begin
  line_base_addr := Pix * y;
  EAD := line_base_addr + trunc (x/16);
  daD := x mod 16;
  daD := daD shl 4;
end;

procedure SetLinetype(linetype:word);
begin
  Port[$301] := $78;
  Port[$300] := Lo(linetype);
  Port[$300] := Hi(linetype);
  Port[$300] := $00;
  Port[$300] := $00;
  Port[$300] := $00;
  Port[$300] := $00;
  Port[$300] := $00;
  Port[$300] := $00;
end;

procedure Screen(setype:byte;Page:integer);
var
  i,j : integer;
begin
  for i := 0 to 3 do
  begin
    for j := 0 to 4 do
    begin
      {---------------}
      {clear memory #1}
      Check_DFIFO;
      Curs($00+i,($3FFF-$04FF)*j);
      CheckFIFO;
```

```
      Mask($FFFF);
      CheckFIFO;
      WDAT(Setype,$0000);
      CheckFIFO;
      LFigs($02,$3FFF);
      CheckFIFO;
      Figd;
    end;
  end;
end;

procedure MappingScreen(master,slave:integer;sel:byte);
begin
  CheckFIFO;
  case sel of
    1 : begin
          MasterScreenLine := MasterScreenLine*16;
          Port[$301] := $70;
          Port[$300] := Lo(master);
          Port[$300] := Hi(master);
          Port[$300] := $00{Lo(MasterScreenline)};
          Port[$300] := $10{Hi(MasterScreenLine)};
        end;
    2 : begin
          SlaveScreenLine  := SlaveScreenLine*16;
          Port[$301] := $74;
          Port[$300] := Lo(slave);
          Port[$300] := Hi(slave);
          Port[$300] := Lo(SlaveScreenLine);
          Port[$300] := Hi(SlaveScreenLine);
        end;
  end;
end;

procedure Start;
begin
  Port[$301] := $6B;
end;

procedure SetColor(ScreenColor : byte);
begin { SetColor }
  DrawColor := ScreenColor;
end { SetColor };

procedure DrawLine(X1,Y1,X2,Y2 : word);
var
  Quadrant,Octant  : byte;
  dX,dY : integer;
  dI,dD : integer;
  D,D1,D2,DC : integer;
begin { DrawLine }
  dX := X2-X1;
  dY := Y2-Y1;
  if (dX >= 0) and (dY < 0)  then Quadrant := 1 else
```

```
  if (dX < 0)  and (dY < 0)  then Quadrant := 2 else
  if (dX < 0)  and (dY >= 0) then Quadrant := 3 else
  if (dX >= 0) and (dY >= 0) then Quadrant := 4;
  case Quadrant of
    1 : begin
          if abs(dX) > abs(dY) then Octant := 2 else
          if abs(dX) < abs(dY) then Octant := 3 else
          if abs(dX) = abs(dY) then Octant := 2;
        end;
    2 : begin
          if abs(dX) > abs(dY) then Octant := 5 else
          if abs(dX) < abs(dY) then Octant := 4 else
          if abs(dX) = abs(dY) then Octant := 4;
        end;
    3 : begin
          if abs(dX) > abs(dY) then Octant := 6 else
          if abs(dX) < abs(dY) then Octant := 7 else
          if abs(dX) = abs(dY) then Octant := 6;
        end;
    4 : begin
          if abs(dX) > abs(dY) then Octant := 1 else
          if abs(dX) < abs(dY) then Octant := 0 else
          if abs(dX) = abs(dY) then Octant := 0;
        end;
  end;
  if Octant in [0,3,4,7] then
  begin
    dI := dY;
    dD := dX;
  end
  else { Octant in [1,2,5,6] }
  begin
    dI := dX;
    dD := dY;
  end;
  Check_DFIFO;
  ConvertAddr(X1,Y1,daD,EAD);
  Check_DFIFO;
  Curs(daD+Page,EAD);
  DC := abs(dI);
  D  := (2 * abs(dD)) - abs(dI);
  D2 := 2 * (abs(dD) - abs(dI));
  D1 := 2 * abs(dD);
  Quadrant := $08 OR Octant;
  CheckFIFO;
  Figs(Quadrant,DC,D,D2,D1,00);
  CheckFIFO;
  Figd;
end { DrawLine };

procedure Line (X1,Y1,X2,Y2 : word);
begin { Line }
  SetLineType(DotLine8);
  case DrawColor of
```

```
    0{Black} : begin { ---- }
          {Set to Black}
          SetLineType(DotLine0);
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
          Page := 2;
          DrawLine(X1,Y1,X2,Y2);
        end;
    1{Blue} : begin { B--- }
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
          {Set to Black}
          SetLineType(DotLine0);
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
          Page := 2;
          DrawLine(X1,Y1,X2,Y2);
        end;
    2{Green} : begin { -G-- }
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
          {Set to Black}
          SetLineType(DotLine0);
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
          Page := 2;
          DrawLine(X1,Y1,X2,Y2);
        end;
    3{Cyan} : begin { BG-- }
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
          {Set to Black}
          SetLineType(DotLine0);
          Page := 2;
          DrawLine(X1,Y1,X2,Y2);
        end;
    4{Red} : begin { ---R }
          Page := 2;
          DrawLine(X1,Y1,X2,Y2);
          {Set to Black}
          SetLineType(DotLine0);
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
        end;
    5{Magenta} : begin { B--R }
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
          Page := 2;
```

```
          DrawLine(X1,Y1,X2,Y2);
          {Set to Black}
          SetLineType(DotLine0);
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
        end;
    6{Yellow} : begin { -GR }
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
          Page := 2;
          DrawLine(X1,Y1,X2,Y2);
          {Set to Black}
          SetLineType(DotLine0);
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
        end;
    7{White} : begin { BGR }
          Page := 0;
          DrawLine(X1,Y1,X2,Y2);
          Page := 1;
          DrawLine(X1,Y1,X2,Y2);
          Page := 2;
          DrawLine(X1,Y1,X2,Y2);
        end;
  end;
end { DrawLine };

procedure DrawBorder;
begin { DrawBorder }
  Line(1,1,1000,1);
  Line(1000,1,1000,1000);
  Line(1000,1000,1,1000);
  Line(1,1000,1,1);
end { DrawBorder };

procedure Scrolling(direction:CHAR);
var
  count : byte;
begin
  case direction of
    Up : begin
           for count := 1 to 8 do
             addr := addr + Pix;
         end;
    Down  : begin
              for count := 1 to 8 do
                addr := addr - Pix;
            end;
    Right : begin
              addr := addr - 1;
            end;
    Left  : begin
              addr := addr + 1;
            end;
```

```
    LeftUp  : begin
                addr := addr + 1;
                for count := 1 to 8 do
                  addr := addr + Pix;
              end;
    LeftDown: begin
                addr := addr + 1;
                for count := 1 to 8 do
                  addr := addr - Pix;
              end;
    RightUp  : begin
                 addr := addr - 1;
                 for count := 1 to 8 do
                   addr := addr + Pix;
               end;
    RightDown: begin
                 addr := addr - 1;
                 for count := 1 to 8 do
                   addr := addr - Pix;
               end;
  end;
  if Direction in [Up,Down,Left,Right,LeftUp,LeftDown,RightUp,RightDown] then
    MappingScreen(addr,addr,Page1);
end;

procedure InitGraph;
begin
  Reset82720;
  Sync82720;
  Cchar;
  Pitch;
  Zoom($00);
  Pram;
  start;
  Port[$31c]:=1;
  Bctrl1;
  Screen($22,Page);
  Check_DFIFO;
  wdat($20,$FFFF);
  addr := $7C00;
  MappingScreen(addr,addr,Page1);
end;

procedure DrawLevel(X1,Y1,X2,Y2 : word; var Percent : integer);
var
  deltaY,NextLevel : word;
  Count : word;
  deltaLevel : integer;
begin {DrawLevel}
  if Percent < 0 then Percent := 0;
  if Percent > 100 then Percent := 100;
  X1 := X1 + 1;
  Y1 := Y1 - 1;
  X2 := X2 - 1;
```

```
  Y2 := Y2 + 1;
  deltaY := Y1 - Y2;
  NextLevel := deltaY * Percent div 100;
  deltaLevel := CurrentLevel - NextLevel;
  Y := Y1 - CurrentLevel;
  if deltaLevel > 0 then { Down }
  begin
    Count := 0;
    SetColor(_Black);
    repeat
      Line(X1,Y,X2,Y);
      Inc(Y);
      Inc(Count);
    until Count = deltaLevel;
  end;
  if deltaLevel < 0 then { Up }
  begin
    deltaLevel := abs(deltaLevel);
    Count := 0;
    repeat
      Line(X1,Y,X2,Y);
      Dec(Y);
      Inc(Count);
    until Count = deltaLevel;
  end;
  CurrentLevel := NextLevel;
end {DrawLevel};

begin
  CurrentLevel := 0;
end.
```

```
Program : ProDesign II Format Conversion
Programmer : Phakorn Hutasangkas.

Unit ReadPDII;

interface

uses Crt,OpenFile,Grph7220;

procedure PlotProDesign;

implementation

const
  MaxY = 1000;

var
  SrceFile : FileType;
  FileName : string;
  Command : string[4];
  BufLine : string;
  XPosition : byte;
  XStr,YStr : string;
  XValue,YValue : word;
  ErrorCode : integer;
  X,Y : word;

procedure ReadDataAndPlot;
begin {ReadDataAndPlot}
  while not EOF(SrceFile) do
  begin
    Readln(SrceFile,BufLine);
    Command := Copy(BufLine,1,4);
    Delete(BufLine,1,Pos(' ',BufLine));
    XPosition := Pos(' ',BufLine);
    if XPosition = 0 then
      XStr := Copy(BufLine,1,Length(BufLine)) { Single Parameter }
    else
      XStr := Copy(BufLine,1,XPosition - 1);  { Double Parameter }
    Delete(BufLine,1,Pos(' ',BufLine));
    YStr := Copy(BufLine,1,Length(BufLine));
    Val(XStr,XValue,ErrorCode);
    Val(YStr,YValue,ErrorCode);
    if ErrorCode = 0 then
      YValue := Abs(MaxY - YValue);
    if Command = 'MOVE' then
    begin
      X := XValue;
```

```
      Y := YValue;
    end;
    if Command = 'LINE' then
    begin
      Line(X,Y,XValue,YValue);
      X := XValue;
      Y := YValue;
    end;
    if Command = 'SETC' then
      SetColor(XValue);
  end;
end {ReadDataAndPlot};

procedure PlotProDesign;
begin
  if OpenReadFile('Enter Prodesign Filename',SrceFile,FileName,'.PLT') then
    ReadDataAndPlot;
end;

begin
end.
```

```
Program : File Handling Program
Programmer : Phakorn Hutasangkas.

{
              +-----------------------------------+
              |                                   |
              | Unit  : Open File I               |
              | Programmer : Phakorn Hutasangkas. |
              | Version    : 1.00A                |
              | Last Updated :  24 Mar 1988       |
              |                                   |
              +-----------------------------------+
}

unit OpenFile;

interface

uses Crt;

type
  FileType = Text;

function OpenReadFile (Title : string; var Filevar : FileType; FileName,Extension : string) : boolean;

function OpenWriteFile (Title : string; var Filevar: FileType; FileName,Extension : String) : boolean;

implementation

const
  Esc = #27;

var
  Buf : string;
  Ok       : boolean;
  FilenameBuffer : String[30];
  ChCount  : byte;
  Ch : char;
  Exist    : boolean;

function OpenReadFile (Title : string; var Filevar : FileType; FileName,Extension : string) : boolean;
begin { OpenReadFile }
  repeat
    FilenameBuffer := '';
    Write (Title,' [',Extension,'] : ');
    readln (Filename);
    if Pos('.',Filename) = 0 then
      Filename := concat (Filename,Extension);
```

```
    for ChCount := 1 to Length(Filename) do
    begin
      Buf := Copy(Filename,ChCount,1);
      Ch := Upcase(Buf[1]);
      FilenameBuffer := FilenameBuffer + Ch;
    end;
    Filename := FilenameBuffer;
    Assign (Filevar,Filename);
    {$I-} Reset (Filevar); {$I+}
    Ok := (IOresult = 0);
    if not Ok then
    begin
      GotoXY(1,WhereY-1);
      Write (#$0D); ClrEol;
      TextColor (LightRed);
      Write (^G,'File [',Filename,'] not found');
      TextColor (LightCyan);
      Write (' [Enter] : re-enter [Esc] : Quit');
      Ch := ReadKey;
      Write (#$0D); ClrEol;
      OpenReadFile := False;
    end
    else
      OpenReadFile := True;
  until (Ok or (Ch = Esc));
end { OpenReadFile };

function OpenWriteFile (Title : string; var Filevar : FileType; FileName,Extension : String) : boolean;
begin { OpenWriteFile }
  repeat
    FilenameBuffer := '';
    Write (Title,' [',Extension,'] : ');
    readln (Filename);
    if Pos('.',Filename) = 0 then
      Filename := concat (Filename,Extension);
    for ChCount := 1 to Length(Filename) do
    begin
      Buf := Copy(Filename,ChCount,1);
      Ch := Upcase(Buf[1]);
      FilenameBuffer := FilenameBuffer + Ch;
    end;
    Filename := FilenameBuffer;
    Assign (Filevar,Filename);
    {$I-} Reset (Filevar); {$I+}
    Ok := (IOresult = 0);
    if Ok then
    begin
      GotoXY (1,WhereY-1);
      Write (#$0D); ClrEol;
      TextColor (LightRed);
      Write (^G,'File [',Filename,'] exist');
      TextColor (LightCyan);
      Write (' [SpaceBar] Overwrite [Enter] : re-enter [Esc] : Quit');
      Ch := ReadKey;
```

```
      Write (#$0D); ClrEol;
      if Ch = Esc then
      begin
        Writeln('Open file process terminated.');
        OpenWriteFile := false;
      end;
    end;
    if (not OK) or (Ch = ' ') then
    begin
      Writeln('Overwriting to ',FileName);
      OpenWriteFile := true;
      ReWrite (Filevar);
    end;
  until (Ch = Esc) or (Ch = ' ') or (not OK);
end { OpenWriteFile };

begin
end.
```

```
Program : User Define Program
Programmer : Phakorn Hutasangkas.

unit Utility;

interface

uses Dos,Crt,Printer,Grph7220,Valve,Pipe,Tank,Can;

type
  ModeType = (Plot,Demo,Execute);

const
  JoyStick = $0201;
  AddrPortA = $280;
  AddrPortB = $283;

const                {...:...:...:...:}
  Mask1  = $0101; {_______-_______-} {CheckKey}
  Mask2  = $FFFF; {----------------} {JoyStick}
  Mask3  = $0404; {_____-_____-__} {ReadPort}
  Mask4  = $0808; {____-____-____} {DisplayValue&Pipe}
  Mask5  = $1010; {___-____-____} {DisplayGraph}
  Mask6  = $2020; {__-____-____} {DisplayLevel}
  Mask7  = $4040; {_-____-____} {DisplayCanAnimation}
  Mask8  = $8080; {-____-____} {DisplayTemp}
  Mask9  = $0101; {_______-_______-} {DisplayHostScreen}
  Mask10 = $FFFF; {----------------} {ReadDateAndTime}
  Mask11 = $0404; {_____-_____-__} {DisplayAlarm}
  Mask12 = $0808; {____-____-____}
  Mask13 = $1010; {___-____-____}
  Mask14 = $2020; {__-____-____}
  Mask15 = $4040; {_-____-____}
  Mask16 = $8080; {-____-____}

const
  MaxDevice = 21;
  LowLevel = 20;
  HighLevel = 80;
  AlarmLow = 10;
  AlarmHigh = 90;
  On = True;
  Off = False;
  HideColor = _Black;
  NormalColor = _White;
  WaterPipeColor = _Blue;
  BoilerPipeColor = _Green;
  MixTankPipeColor = _Yellow;
```

```
  TankAPipeColor = _Magenta;
  TankBPipeColor = _Red;
  TankCPipeColor = _Cyan;
  CanOnColor = NormalColor;
  CanOffColor = HideColor;

var
  CanPosition : byte;
  DeviceStatus : array [1..MaxDevice] of boolean;

var
  Sequence : word;
  Mode : ModeType;
  ExitToDOS : boolean;
  PreviousMin : word;
  PreviousPosition : byte;
  PrintOnTime : boolean;
  PrintingTime : byte;

procedure TransferData;

function DateAndTime : string;

procedure PrintStatus (Message : string; Status : boolean);

procedure PrintLevel (Message : string; Percent : byte);

procedure PrintTitle;

procedure Title;

procedure SetConfig;

procedure ProcessMonitor;

procedure Menu;

procedure ReadJoyStick;

implementation

procedure DisplayDeviceNumber (Device : byte);
begin {DisplayDeviceNumber}
  case Device of
    1 : begin
          if Valve1Status xor DeviceStatus[1] then
          begin
            if Valve1Status then
            begin
              Valve1(WaterPipeColor);
              Pipe3(WaterPipeColor);
            end
            else
            begin
```

```
            Valve1(NormalColor);
            Pipe3(NormalColor);
          end;
          PrintStatus('Valve1',Valve1Status);
        end;
      end;
  2 : begin
        if Valve2Status xor DeviceStatus[2] then
        begin
          if Valve2Status then
          begin
            Valve2(BoilerPipeColor);
            Pipe5(BoilerPipeColor);
          end
          else
          begin
            Valve2(NormalColor);
            Pipe5(NormalColor);
          end;
          PrintStatus('Valve2',Valve2Status);
        end;
      end;
  3 : begin
        if Valve3Status xor DeviceStatus[3] then
        begin
          if Valve3Status then
          begin
            Valve3(BoilerPipeColor);
            Pipe6(BoilerPipeColor);
          end
          else
          begin
            Valve3(NormalColor);
            Pipe6(NormalColor);
          end;
          PrintStatus('Valve3',Valve3Status);
        end;
      end;
  4 : begin
        if Valve4Status xor DeviceStatus[4] then
        begin
          if Valve4Status then
          begin
            Valve4(BoilerPipeColor);
            Pipe7(BoilerPipeColor);
          end
          else
          begin
            Valve4(NormalColor);
            Pipe7(NormalColor);
          end;
          PrintStatus('Valve4',Valve4Status);
        end;
      end;
```

```
    5 : begin
          if Valve5Status xor DeviceStatus[5] then
          begin
            if Valve5Status then
            begin
              Valve5(BoilerPipeColor);
              Pipe8(BoilerPipeColor);
            end
            else
            begin
              Valve5(NormalColor);
              Pipe8(NormalColor);
            end;
            PrintStatus('Valve5',Valve5Status);
          end;
        end;
    6 : begin
          if Valve6Status xor DeviceStatus[6] then
          begin
            if Valve6Status then
            begin
              Valve6(BoilerPipeColor);
              Pipe9(BoilerPipeColor);
            end
            else
            begin
              Valve6(NormalColor);
              Pipe9(NormalColor);
            end;
            PrintStatus('Valve6',Valve6Status);
          end;
        end;
    7 : begin
          if Valve7Status xor DeviceStatus[7] then
          begin
            if Valve7Status then
            begin
              Valve7(TankCPipeColor);
              Pipe13(TankCPipeColor);
            end
            else
            begin
              Valve7(NormalColor);
              Pipe13(NormalColor);
            end;
            PrintStatus('Valve7',Valve7Status);
          end;
        end;
    8 : begin
          if Valve8Status xor DeviceStatus[8] then
          begin
            if Valve8Status then
            begin
              Valve8(TankBPipeColor);
```

```
            Pipe14(TankBPipeColor);
          end
          else
          begin
            Valve8(NormalColor);
            Pipe14(NormalColor);
          end;
          PrintStatus('Valve8',Valve8Status);
        end;
      end;
  9 : begin
        if Valve9Status xor DeviceStatus[9] then
        begin
          if Valve9Status then
          begin
            Valve9(TankAPipeColor);
            Pipe15(TankAPipeColor);
          end
          else
          begin
            Valve9(NormalColor);
            Pipe15(NormalColor);
          end;
          PrintStatus('Valve9',Valve9Status);
        end;
      end;
 10 : begin
        if Valve10Status xor DeviceStatus[10] then
        begin
          if Valve10Status then
          begin
            Valve10(TankCPipeColor);
            Pipe29(TankCPipeColor);
          end
          else
          begin
            Valve10(NormalColor);
            Pipe29(NormalColor);
          end;
          PrintStatus('Valve10',Valve10Status);
        end;
      end;
 11 : begin
        if Valve11Status xor DeviceStatus[11] then
        begin
          if Valve11Status then
          begin
            Valve11(TankBPipeColor);
            Pipe25(TankBPipeColor);
          end
          else
          begin
            Valve11(NormalColor);
            Pipe25(NormalColor);
```

```
          end;
          PrintStatus('Valve11',Valve11Status);
        end;
      end;
  12 : begin
        if Valve12Status xor DeviceStatus[12] then
        begin
          if Valve12Status then
          begin
            Valve12(TankAPipeColor);
            Pipe21(TankAPipeColor);
          end
          else
          begin
            Valve12(NormalColor);
            Pipe21(NormalColor);
          end;
          PrintStatus('Valve12',Valve12Status);
        end;
      end;
  13 : begin
        if Valve13Status xor DeviceStatus[13] then
        begin
          if Valve13Status then
          begin
            Valve13(WaterPipeColor);
            Pipe32(WaterPipeColor);
          end
          else
          begin
            Valve13(NormalColor);
            Pipe32(NormalColor);
          end;
          PrintStatus('Valve13',Valve13Status);
        end;
      end;
  14 : begin
        if Valve14Status xor DeviceStatus[14] then
        begin
          if Valve14Status then
          begin
            Valve14(WaterPipeColor);
            Pipe33(WaterPipeColor);
          end
          else
          begin
            Valve14(NormalColor);
            Pipe33(NormalColor);
          end;
          PrintStatus('Valve14',Valve14Status);
        end;
      end;
  15 : begin
        if Valve15Status xor DeviceStatus[15] then
```

```
          begin
            if Valve15Status then
            begin
              Valve15(WaterPipeColor);
              Pipe34(WaterPipeColor);
            end
            else
            begin
              Valve15(NormalColor);
              Pipe34(NormalColor);
            end;
            PrintStatus('Valve15',Valve15Status);
          end;
        end;
    16 : begin
          if Valve16Status xor DeviceStatus[16] then
          begin
            if Valve16Status then
            begin
              Valve16(MixTankPipeColor);
              Pipe17(MixTankPipeColor);
            end
            else
            begin
              Valve16(NormalColor);
              Pipe17(NormalColor);
            end;
            PrintStatus('Valve16',Valve16Status);
          end;
        end;
    17 : begin
          if Pump1Status xor DeviceStatus[17] then
          begin
            if Pump1Status then
            begin
              Pump1(WaterPipeColor);
              Pipe2(WaterPipeColor);
            end
            else
            begin
              Pump1(NormalColor);
              Pipe2(NormalColor);
            end;
            PrintStatus('Pump1',Pump1Status);
          end;
        end;
    18 : begin
          if Pump2Status xor DeviceStatus[18] then
          begin
            if Pump2Status then
            begin
              Pump2(TankCPipeColor);
              Pipe31(TankCPipeColor);
            end
```

```
            else
            begin
              Pump2(NormalColor);
              Pipe31(NormalColor);
            end;
            PrintStatus('Pump2',Pump2Status);
          end;
        end;
    19 : begin
          if Pump3Status xor DeviceStatus[19] then
          begin
            if Pump3Status then
            begin
              Pump3(TankBPipeColor);
              Pipe27(TankBPipeColor);
            end
            else
            begin
              Pump3(NormalColor);
              Pipe27(NormalColor);
            end;
            PrintStatus('Pump3',Pump3Status);
          end;
        end;
    20 : begin
          if Pump4Status xor DeviceStatus[20] then
          begin
            if Pump4Status then
            begin
              Pump4(TankAPIpeColor);
              Pipe23(TankAPipeColor);
            end
            else
            begin
              Pump4(NormalColor);
              Pipe23(NormalColor);
            end;
            PrintStatus('Pump4',Pump4Status);
          end;
        end;
    21 : begin
          if Pump5Status xor DeviceStatus[21] then
          begin
            if Pump5Status then
            begin
              Pump5(WaterPipeColor);
              Pipe19(WaterPipeColor);
            end
            else
            begin
              Pump5(NormalColor);
              Pipe19(NormalColor);
            end;
            PrintStatus('Pump5',Pump5Status);
```

```
          end;
        end;
  end;
end {DisplayDeviceNumber};

procedure DisplayDevice;
var Device : byte;
begin {DisplayDevice}
  for Device := 1 to MaxDevice do
    DisplayDeviceNumber(Device);
end {DisplayDevice};

procedure DisplayLevel;
begin {DisplayLevel}
  TankA2(TankA2Percent);
  TankB2(TankB2Percent);
  TankC2(TankC2Percent);
  TankA3(TankA3Percent);
  TankB3(TankB3Percent);
  TankC3(TankC3Percent);
  MixTank(MixTankPercent);
  if PrintOnTime then
  begin
    PrintLevel('Tank A2 : ',TankA2Percent);
    PrintLevel('Tank B2 : ',TankB2Percent);
    PrintLevel('Tank C2 : ',TankC2Percent);
    PrintLevel('Tank A3 : ',TankA3Percent);
    PrintLevel('Tank B3 : ',TankB3Percent);
    PrintLevel('Tank C3 : ',TankC3Percent);
    PrintLevel('Mixing Tank : ',MixTankPercent);
    PrintOnTime := false;
  end;
end {DisplayLevel};

procedure DisplayGraph;
begin {DisplayGraph}
  case TankA2Percent of
     0.. 50 : GraphA2Color := _Green;
    51.. 80 : GraphA2Color := _Magenta;
    81..100 : GraphA2Color := _Red;
  end;
  case TankB2Percent of
     0.. 50 : GraphB2Color := _Green;
    51.. 80 : GraphB2Color := _Magenta;
    81..100 : GraphB2Color := _Red;
  end;
  case TankC2Percent of
     0.. 50 : GraphC2Color := _Green;
    51.. 80 : GraphC2Color := _Magenta;
    81..100 : GraphC2Color := _Red;
  end;
  case TankA3Percent of
     0.. 50 : GraphA3Color := _Green;
    51.. 80 : GraphA3Color := _Magenta;
```

```
    81..100 : GraphA3Color := _Red;
  end;
  case TankB3Percent of
     0.. 50 : GraphB3Color := _Green;
    51.. 80 : GraphB3Color := _Magenta;
    81..100 : GraphB3Color := _Red;
  end;
  case TankC3Percent of
     0.. 50 : GraphC3Color := _Green;
    51.. 80 : GraphC3Color := _Magenta;
    81..100 : GraphC3Color := _Red;
  end;
  GraphA2(TankA2Percent);
  GraphB2(TankB2Percent);
  GraphC2(TankC2Percent);
  GraphA3(TankA3Percent);
  GraphB3(TankB3Percent);
  GraphC3(TankC3Percent);
end {DisplayGraph};

procedure DisplayTemp;
begin {DisplayTemp}
  case TempA2Percent of
     0.. 50 : TempA2Color := _Green;
    51.. 80 : TempA2Color := _Magenta;
    81..100 : TempA2Color := _Red;
  end;
  case TempB2Percent of
     0.. 50 : TempB2Color := _Green;
    51.. 80 : TempB2Color := _Magenta;
    81..100 : TempB2Color := _Red;
  end;
  case TempC2Percent of
     0.. 50 : TempC2Color := _Green;
    51.. 80 : TempC2Color := _Magenta;
    81..100 : TempC2Color := _Red;
  end;
  case TempA3Percent of
     0.. 50 : TempA3Color := _Green;
    51.. 80 : TempA3Color := _Magenta;
    81..100 : TempA3Color := _Red;
  end;
  case TempB3Percent of
     0.. 50 : TempB3Color := _Green;
    51.. 80 : TempB3Color := _Magenta;
    81..100 : TempB3Color := _Red;
  end;
  case TempC3Percent of
     0.. 50 : TempC3Color := _Green;
    51.. 80 : TempC3Color := _Magenta;
    81..100 : TempC3Color := _Red;
  end;
  case TempMixPercent of
     0.. 50 : TempMixColor := _Green;
```

```
    51.. 80 : TempMixColor := _Magenta;
    81..100 : TempMixColor := _Red;
  end;
  TempA2(TempA2Percent);
  TempB2(TempB2Percent);
  TempC2(TempC2Percent);
  TempA3(TempA3Percent);
  TempB3(TempB3Percent);
  TempC3(TempC3Percent);
  TempMix(TempMixPercent);
end {DisplayTemp};

procedure DisplayCanAnimation (CanPosition : byte);
begin {DisplayCanAnimation}
  case CanPosition of
    0 : begin
          Can3(CanOffColor);
          Can6(CanOffColor);
          Can9(CanOffColor);
          Can12(CanOffColor);
          Can15(CanOffColor);
          Can1(CanOnColor);
          Can4(CanOnColor);
          Can7(CanOnColor);
          Can10(CanOnColor);
          Can13(CanOnColor);
        end;
    1 : begin
          Can1(CanOffColor);
          Can4(CanOffColor);
          Can7(CanOffColor);
          Can10(CanOffColor);
          Can13(CanOffColor);
          Can2(CanOnColor);
          Can5(CanOnColor);
          Can8(CanOnColor);
          Can11(CanOnColor);
          Can14(CanOnColor);
        end;
    2 : begin
          Can2(CanOffColor);
          Can5(CanOffColor);
          Can8(CanOffColor);
          Can11(CanOffColor);
          Can14(CanOffColor);
          Can3(CanOnColor);
          Can6(CanOnColor);
          Can9(CanOnColor);
          Can12(CanOnColor);
          Can15(CanOnColor);
        end;
  end;
end {DisplayCanAnimation};
```

```
procedure TransferData;
begin {TransferData}
  DeviceStatus[1] := Valve1Status;
  DeviceStatus[2] := Valve2Status;
  DeviceStatus[3] := Valve3Status;
  DeviceStatus[4] := Valve4Status;
  DeviceStatus[5] := Valve5Status;
  DeviceStatus[6] := Valve6Status;
  DeviceStatus[7] := Valve7Status;
  DeviceStatus[8] := Valve8Status;
  DeviceStatus[9] := Valve9Status;
  DeviceStatus[10] := Valve10Status;
  DeviceStatus[11] := Valve11Status;
  DeviceStatus[12] := Valve12Status;
  DeviceStatus[13] := Valve13Status;
  DeviceStatus[14] := Valve14Status;
  DeviceStatus[15] := Valve15Status;
  DeviceStatus[16] := Valve16Status;
  DeviceStatus[17] := Pump1Status;
  DeviceStatus[18] := Pump2Status;
  DeviceStatus[19] := Pump3Status;
  DeviceStatus[20] := Pump4Status;
  DeviceStatus[21] := Pump5Status;
end {TransferData};

procedure InitialDevice;
var
  Device : byte;
begin {InitialDevice}
  Pipe1(WaterPipeColor);
  Pipe2(WaterPipeColor);
  Pipe3(WaterPipeColor);
  Pipe4(BoilerPipeColor);
  Pipe10(TankCPipeColor);
  Pipe11(TankBPipeColor);
  Pipe12(TankAPipeColor);
  Pipe16(MixTankPipeColor);
  Pipe18(WaterPipeColor);
  Pipe20(TankAPipeColor);
  Pipe22(TankAPipeColor);
  Pipe24(TankBPipeColor);
  Pipe26(TankBPipeColor);
  Pipe28(TankCPipeColor);
  Pipe30(TankCPipeColor);
  TankA2Percent := 0;
  TankB2Percent := 0;
  TankC2Percent := 0;
  TankA3Percent := 0;
  TankB3Percent := 0;
  TankC3Percent := 0;
  MixTankPercent := 0;
  TempA2Percent := 0;
  TempB2Percent := 0;
  TempC2Percent := 0;
```

```
  TempA3Percent := 0;
  TempB3Percent := 0;
  TempC3Percent := 0;
  TempMixPercent := 0;
  for Device := 1 to MaxDevice do
    DeviceStatus[Device] := On;
  Valve1Status := not DeviceStatus[1];
  Valve2Status := not DeviceStatus[2];
  Valve3Status := not DeviceStatus[3];
  Valve4Status := not DeviceStatus[4];
  Valve5Status := not DeviceStatus[5];
  Valve6Status := not DeviceStatus[6];
  Valve7Status := not DeviceStatus[7];
  Valve8Status := not DeviceStatus[8];
  Valve9Status := not DeviceStatus[9];
  Valve10Status := not DeviceStatus[10];
  Valve11Status := not DeviceStatus[11];
  Valve12Status := not DeviceStatus[12];
  Valve13Status := not DeviceStatus[13];
  Valve14Status := not DeviceStatus[14];
  Valve15Status := not DeviceStatus[15];
  Valve16Status := not DeviceStatus[16];
  Pump1Status := not DeviceStatus[17];
  Pump2Status := not DeviceStatus[18];
  Pump3Status := not DeviceStatus[19];
  Pump4Status := not DeviceStatus[20];
  Pump5Status := not DeviceStatus[21];
  for Device := 1 to MaxDevice do
    DisplayDeviceNumber(Device);
end {InitialDevice};

procedure ProcessDemoLogic;
var DeviceNo : byte;
begin {ProcessDemoLogic}

  if CanPosition = 1 then
  begin
    if not Valve16Status then
    begin
      Valve16Status := On;
    end;
  end
  else
  begin
    if Valve16Status then
    begin
      Valve16Status := Off;
    end;
  end;

  if Valve3Status or Valve4Status or Valve5Status or Valve6Status then
  begin
    Valve2Status := On;
  end
```

```
  else
  begin
    Valve2Status := Off;
  end;

  if TankA2Percent < LowLevel then
  begin
    Valve12Status := On;
  end;
  if TankA2Percent > HighLevel then
  begin
    Valve12Status := Off;
  end;

  if TankB2Percent < LowLevel then
  begin
    Valve11Status := On;
  end;
  if TankB2Percent > HighLevel then
  begin
    Valve11Status := Off;
  end;

  if TankC2Percent < LowLevel then
  begin
    Valve10Status := On;
  end;
  if TankC2Percent > HighLevel then
  begin
    Valve10Status := Off;
  end;

  if TankA3Percent < LowLevel then
  begin
    Valve15Status := On;
    Pump4Status := On;
  end;
  if TankA3Percent > HighLevel then
  begin
    Valve15Status := Off;
    Pump4Status := Off;
  end;

  if TankB3Percent < LowLevel then
  begin
    Valve14Status := On;
    Pump3Status := On;
  end;
  if TankB3Percent > HighLevel then
  begin
    Valve14Status := Off;
    Pump3Status := Off;
  end;
```

```
  if TankC3Percent < LowLevel then
  begin
    Valve13Status := On;
    Pump2Status := On;
  end;
  if TankC3Percent > HighLevel then
  begin
    Valve13Status := Off;
    Pump2Status := Off;
  end;

  if Valve13Status or Valve14Status or Valve15Status then
  begin
    Pump5Status := On;
  end
  else
  begin
    Pump5Status := Off;
  end;

  if MixTankPercent < LowLevel then
  begin
    Valve7Status := On;
    Valve8Status := On;
    Valve9Status := On;
  end;
  if MixTankPercent > HighLevel then
  begin
    Valve7Status := Off;
    Valve8Status := Off;
    Valve9Status := Off;
  end;

  if TempA3Percent < LowLevel then
  begin
    Valve6Status := On;
  end;
  if TempA3Percent > HighLevel then
  begin
    Valve6Status := Off;
  end;

  if TempB3Percent < LowLevel then
  begin
    Valve5Status := On;
  end;
  if TempMixPercent > HighLevel then
  begin
    Valve5Status := Off;
  end;

  if TempC3Percent < LowLevel then
  begin
    Valve4Status := On;
```

```
  end;
  if TempC3Percent > HighLevel then
  begin
    Valve4Status := Off;
  end;

  if TempMixPercent < LowLevel then
  begin
    Valve3Status := On;
  end;
  if TempMixPercent > HighLevel then
  begin
    Valve3Status := Off;
  end;

  if Valve10Status then Inc(TankC2Percent,1);
  if Pump2Status   then Dec(TankC2Percent,2);

  if Valve11Status then Inc(TankB2Percent,1);
  if Pump3Status   then Dec(TankB2Percent,2);

  if Valve12Status then Inc(TankA2Percent,1);
  if Pump4Status   then Dec(TankA2Percent,2);

  if Valve13Status or Pump2Status then Inc(TankC3Percent,2);
  if Valve7Status  then Dec(TankC3Percent,5);

  if Valve14Status or Pump3Status then Inc(TankB3Percent,2);
  if Valve8Status  then Dec(TankB3Percent,1);

  if Valve15Status or Pump4Status then Inc(TankA3Percent,2);
  if Valve9Status  then Dec(TankA3Percent,3);

  if Valve7Status or Valve8Status or Valve9Status then Inc(MixTankPercent,9);
  if Valve16Status then Dec(MixTankPercent,2);

  if Valve6Status then Inc(TempA3Percent) else Dec(TempA3Percent);

  if Valve5Status then Inc(TempB3Percent) else Dec(TempB3Percent);

  if Valve4Status then Inc(TempC3Percent) else Dec(TempC3Percent);

  if Valve3Status then Inc(TempMixPercent) else Dec(TempMixPercent);

end {ProcessDemoLogic};

procedure ReadDateAndTime;
begin {ReadDateAndTime}
  GetDate(Year,Month,Day,DayOfWeek);
  GetTime(Hour,Min,Sec,Tic);
  if PreviousMin <> Min then
  begin
    if (not PrintOnTime) and (Min mod PrintingTime = 0) then
      PrintOnTime := true;
```

```
    end;
    PreviousMin := Min;
  end {ReadDateAndTime};

  function DateAndTime : string;
  var
    YY : string[4];
    MM,DD,HH,MN,SC : string[2];
  begin {DateAndTime}
    ReadDateAndTime;
    Str(Year,YY);
    Str(Month,MM);
    Str(Day,DD);
    Str(Hour,HH);
    Str(Min,MN);
    Str(Sec,SC);
    if Length(MM) = 1 then MM := '0' + MM;
    if Length(DD) = 1 then DD := '0' + DD;
    if Length(HH) = 1 then HH := '0' + HH;
    if Length(MN) = 1 then MN := '0' + MN;
    if Length(SC) = 1 then SC := '0' + SC;
    DateAndTime := MM + '/' + DD + '/' + YY + ' ' + HH + ':' + MN + ':' + SC;
  end {DateAndTime};

  procedure PrintStatus (Message : string; Status : boolean);
  begin {PrintStatus}
    Write (Lst,DateAndTime,' ');
    Write (Lst,Message);
    if Status = True then
    begin
      Writeln(Lst,' On');
    end
    else
    begin
      Writeln(Lst,' Off');
    end;
  end {PrintStatus};

  procedure PrintLevel (Message : string; Percent : byte);
  begin {PrintLevel}
    Write (Lst,DateAndTime,' ');
    Writeln (Lst,Message,Percent,'%');
  end {PrintLevel};

  procedure PrintTitle;
  begin {PrintTitle}
    Writeln(Lst,'      +-------------------------------------------------------------------+');
    Writeln(Lst,'      | ****   ****   *****   *****   ******   **********   ****        |');
    Writeln(Lst,'      | ****   ****   **** * * ****   ****       ****       ****        |');
    Writeln(Lst,'      | ****  ****    ****   *  ****   ****       ****       ****        |');
    Writeln(Lst,'      | ****   ****   ****     ****   ****       ****       ****        |');
    Writeln(Lst,'      | ****    ****  ****     ****   ******     ****       **********  |');
    Writeln(Lst,'      |                                                                   |');
    Writeln(Lst,'      | Faculty of Engineering.Department of Intrumentation Engineering |');
```

```
  Writeln(Lst,'       |                                                                |');
  Writeln(Lst,'       | High-Resolution Graphics for Industrial Process Monitoring |');
  Writeln(Lst,'       | Software Driver Version 1.00A  Phakorn Hutasangkas. 1 July 1988 |');
  Writeln(Lst,'       +----------------------------------------------------------------+');
  Writeln(Lst);
  Writeln(Lst,'        CURRENT DATE : ',DateAndTime);
  Writeln(Lst);
end {PrintTitle};

procedure Title;
begin {Title}
  ClrScr;
  TextColor(LightCyan);
  Writeln('       +----------------------------------------------------------------+');
  Writeln('       | *****    *****   ******   ******   *******   ***********   *****  |');
  Writeln('       | *****   *****    ***** * * *****    *****       *****     *****  |');
  Writeln('       | *****  *****     *****   *  *****    *****       *****     *****  |');
  Writeln('       | *****  *****     *****      *****    *****       *****     *****  |');
  Writeln('       | *****   *****    *****      *****    *******     *****     ********** |');
  Writeln('       |                                                                |');
  Writeln('       | Faculty of Engineering.Department of Intrumentation Engineering |');
  Writeln('       |                                                                |');
  Writeln('       | High-Resolution Graphics for Industrial Process Monitoring |');
  Writeln('       | Software Driver Version 1.00A  Phakorn Hutasangkas. 1 July 1988 |');
  Writeln('       +----------------------------------------------------------------+');
  Writeln;
  Writeln;
end {Title};

procedure SetConfig;
var Password : string;
begin {SetConfig}
  Title;
  GotoXY(30,15);
  Write('Password : ');
  Password := '';
  repeat
    Ch := ReadKey;
    Write('*');
    if Ch <> #$0D then Password := Password + Ch;
  until Ch = #$0D;
  Password := '@12@4' + Password + '@4@5';
  if Password <> '@12@4585007080@4@5' then
  begin
    GotoXY(30,15);
    Write(' Access Code Denied. ');
    Write(^G);
    Halt;
  end
  else
  begin
    GotoXY(30,15);
    Write('Access Code Approved.');
    Delay(1000);
```

```
  end;
  GotoXY(24,15);
  Write('Enter Printing time in minute : ');
  Readln(PrintingTime);
end {SetConfig};

procedure Menu;
begin {Menu}
  Title;
  TextColor(LightRed);
  Writeln('                                    Main Menu');
  Writeln('                                    ---------');
  Writeln('                          [1] Plot ProDesignII Drawing.');
  Writeln('                          [2] Process Monitoring Demonstration Mode.');
  Writeln('                          [3] Process Monitoring Execution Mode.');
  Write  ('                          [Esc] Exit to DOS      Select : ');
  Ch := ReadKey;
  case Ch Of
    '1' : begin
            Title;
            TextColor(LightRed);
            Writeln('Plot ProDesignII Drawing Mode.');
            Writeln('------------------------------');
            Writeln;
            TextColor(LightCyan);
            Mode := Plot;
          end;
    '2' : begin
            Title;
            TextColor(LightRed);
            Writeln('Process Monitoring Demonstration Mode.');
            Writeln('--------------------------------------');
            Writeln;
            TextColor(LightCyan);
            Mode := Demo;
          end;
    '3' : begin
            Title;
            TextColor(LightRed);
            Writeln('Process Monitoring Execution Mode.');
            Writeln('----------------------------------');
            Writeln;
            TextColor(LightCyan);
            Mode := Execute;
          end;
    Esc : ExitToDOS := True;
  end;
end {Menu};

procedure CheckKey;
begin {CheckKey}
  if KeyPressed then
  begin
    Ch := ReadKey;
```

```
    if (Ch = #0) and (keypressed) then
    begin
      { Function Key }
      Sound(1000); NoSound;
      Ch := ReadKey;
      Scrolling(Ch);
    end
    else
    begin
      if Ch <> #27 then
      begin
        {User Task}
      end;
    end;
  end;
end {CheckKey};

procedure ReadJoyStick;
var
  CountAX,CountAY : byte;
begin {ReadJoyStick}
  if (Port[JoyStick] and $10 = $0) then
    Button1 := On
  else
    Button1 := Off;
  if (Port[JoyStick] and $20 = $0) then
    Button2 := On
  else
    Button2 := Off;
  CountAX := 0;
  CountAY := 0;
  Port[JoyStick] := $00; {Trig the mono-stable}
  repeat
    Inc(CountAX);
  until (Port[JoyStick] and $01 = $00);
  Port[JoyStick] := $00; {Trig the mono-stable}
  repeat
    Inc(CountAY);
  until (Port[JoyStick] and $02 = $00);
  case CountAX of
    0..40 : begin
              Scrolling(Left);
            end;
    41..150 : {Center};
    151..200 : begin
                 Scrolling(Right);
               end;
  end;
  case CountAY of
    0..30 : begin
              Scrolling(Up);
            end;
    31..108 : {Center};
    109..200 : begin
```

```
                Scrolling(Down);
              end;
  end;
  if Mode = Plot then Delay(65) else Delay(40);
end {ReadJoyStick};

procedure Initial8255;
begin {Initial8255}
  Port[$383] := $83;
end {Initial8255};

procedure ResetADC;
begin {ResetADC}
  Port[$382] := $30;
end {ResetADC};

procedure SelectCH(CH : byte);
begin {SelectCH}
  Port[$380] := CH;
end {SelectCH};

procedure StartConvert;
begin {StartConvert}
  Port[$382] := $70;
  repeat until (Port[$382] and $01) = $01;
  repeat until (Port[$382] and $01) = $00;
  Port[$382] := $30;
  Port[$382] := $20;
end {StartConvert};

procedure ReadADC(CH : byte; var Percent : integer);
var
  HighByte,LowByte : byte;
  Data : word;

begin {ReadADC}
  SelectCH(CH);
  StartConvert;
  LowByte := Port[$381];
  Port[$382] := $30;
  Port[$382] := $10;
  HighByte := Port[$381];
  Port[$382] := $30;
  Data := (HighByte shl 8) or LowByte;
  Data := Data and $0FFF;
  Percent := Trunc((Data * 6.209434926E-3 - 1.010635) * 100 / 16 - 25);
end {ReadADC};

procedure ReadPort;
var
  DataA,DataB : byte;
begin {ReadPort}
  DataA := Port[AddrPortA];
  DataB := Port[AddrPortB];
```

```
  Valve1Status := (DataA and $01) = $01;
  Valve2Status := (DataA and $02) = $02;
  Valve3Status := (DataA and $04) = $04;
  Valve4Status := (DataA and $08) = $08;
  Valve5Status := (DataA and $10) = $10;
  Valve6Status := (DataA and $20) = $20;
  Valve7Status := (DataA and $40) = $40;
  Valve8Status := (DataA and $80) = $80;

  Valve9Status  := (DataB and $01) = $01;
  Valve10Status := (DataB and $02) = $02;
  Valve11Status := (DataB and $04) = $04;
  Valve12Status := (DataB and $08) = $08;
  Valve13Status := (DataB and $10) = $10;
  Valve14Status := (DataB and $20) = $20;
  Valve15Status := (DataB and $40) = $40;
  Valve16Status := (DataB and $80) = $80;

  ReadADC(0,TankA2Percent);
  ReadADC(1,TankB2Percent);
  ReadADC(2,TankC2Percent);
  ReadADC(3,TankA3Percent);
  ReadADC(4,TankB3Percent);
  ReadADC(5,TankC3Percent);
  ReadADC(6,MixTankPercent);
  ReadADC(7,TempMixPercent);

  CanPosition := 0;
end {ReadPort};

procedure DrawOverlay;
begin {DrawOverlay}
  TextColor(LightGreen);
  Writeln;
  Writeln('        High-Resolution Graphics For Industrial Process Monitoring');
  Writeln;
  TextColor(White);
  Writeln('+  STATUS  +        + STATUS +        + LEVEL % +         + TEMP ¢C +');
  Writeln('|          |        |        |        |         |         |         |');
  Writeln('|          |        |        |        | A2      |         | A2      |');
  Writeln('|          |        |        |        | B2      |         | B2      |');
  Writeln('|          |        |        |        | C2      |         | C2      |');
  Writeln('|          |        |        |        | A3      |         | A3      |');
  Writeln('|          |        |        |        | B3      |         | B3      |');
  Writeln('|          |        +--------+        | C3      |         | C3      |');
  Writeln('|          |                          | MX      |         | MX      |');
  Writeln('|          |                          |         |         |         |');
  Writeln('|          |                          +---------+         +---------+');
  Writeln('|          |');
  Writeln('|          |        +------------------ ALARM MESSAGES ------------------+');
  Writeln('|          |        |                                                    |');
  Writeln('|          |        |                                                    |');
  Writeln('|          |        |                                                    |');
```

```
  Writeln('|         |       |                                                  |');
  Writeln('|         |       |                                                  |');
  Writeln('+---------+       +--------------------------------------------------+');
  Writeln('                                                  Press [Esc] to exit.');
end {DrawOverlay};

procedure Display (X,Y : byte; Message : string; Status : boolean);
begin {Display}
  GotoXY(X,Y);
  if Status = On then TextColor(LightMagenta) else TextColor(DarkGray);
  Write(Message);
end {Display};

procedure DisplayValue (X,Y : byte; Percent : word);
var Value : string;
begin {DisplayValue}
  Str(Percent,Value);
  GotoXY(X,Y);
  TextColor(White);
  case Length(Value) of
    1 : Write('  ',Value);
    2 : Write(' ',Value);
    3 : Write(Value);
  else Write('OFL');
  end;
end {DisplayValue};

procedure DisplayHostScreen;
begin {DisplayHostScreen}
  if Valve1Status xor DeviceStatus[1] then Display(3,6,'Valve1',Valve1Status);
  if Valve2Status xor DeviceStatus[2] then Display(3,7,'Valve2',Valve2Status);
  if Valve3Status xor DeviceStatus[3] then Display(3,8,'Valve3',Valve3Status);
  if Valve4Status xor DeviceStatus[4] then Display(3,9,'Valve4',Valve4Status);
  if Valve5Status xor DeviceStatus[5] then Display(3,10,'Valve5',Valve5Status);
  if Valve6Status xor DeviceStatus[6] then Display(3,11,'Valve6',Valve6Status);
  if Valve7Status xor DeviceStatus[7] then Display(3,12,'Valve7',Valve7Status);
  if Valve8Status xor DeviceStatus[8] then Display(3,13,'Valve8',Valve8Status);
  if Valve9Status xor DeviceStatus[9] then Display(3,14,'Valve9',Valve9Status);
  if Valve10Status xor DeviceStatus[10] then Display(3,15,'Valve10',Valve10Status);
  if Valve11Status xor DeviceStatus[11] then Display(3,16,'Valve11',Valve11Status);
  if Valve12Status xor DeviceStatus[12] then Display(3,17,'Valve12',Valve12Status);
  if Valve13Status xor DeviceStatus[13] then Display(3,18,'Valve13',Valve12Status);
  if Valve14Status xor DeviceStatus[14] then Display(3,19,'Valve14',Valve13Status);
  if Valve15Status xor DeviceStatus[15] then Display(3,20,'Valve15',Valve14Status);
  if Valve16Status xor DeviceStatus[16] then Display(3,21,'Valve16',Valve16Status);
  if Pump1Status xor DeviceStatus[17] then Display(23,6,'Pump1',Pump1Status);
  if Pump2Status xor DeviceStatus[18] then Display(23,7,'Pump2',Pump2Status);
  if Pump3Status xor DeviceStatus[19] then Display(23,8,'Pump3',Pump3Status);
  if Pump4Status xor DeviceStatus[20] then Display(23,9,'Pump4',Pump4Status);
  if Pump5Status xor DeviceStatus[21] then Display(23,10,'Pump5',Pump5Status);

  DisplayValue(47,6,TankA2Percent);
  DisplayValue(47,7,TankB2Percent);
  DisplayValue(47,8,TankC2Percent);
```

```
  DisplayValue(47,9,TankA3Percent);
  DisplayValue(47,10,TankB3Percent);
  DisplayValue(47,11,TankC3Percent);
  DisplayValue(47,12,MixTankPercent);
  DisplayValue(68,6,TempA2Percent);
  DisplayValue(68,7,TempB2Percent);
  DisplayValue(68,8,TempC2Percent);
  DisplayValue(68,9,TempA3Percent);
  DisplayValue(68,10,TempB3Percent);
  DisplayValue(68,11,TempC3Percent);
  DisplayValue(68,12,TempMixPercent);
end {DisplayHostScreen};

procedure AlarmMessage (Message : string);
begin {AlarmMessage}
  GotoXY(23,17);
  TextColor(LightRed+Blink);
  Write(Message);
end {AlarmMessage};

procedure AlarmOff;
begin {AlarmOff}
  GotoXY(23,17);
  TextColor(Black);
  Write('                    ');
end {AlarmOff};

procedure DisplayAlarm;
begin {DisplayAlarm}
  if TankA3Percent >= AlarmHigh then
    AlarmMessage('Tank A3 High Level Alarm')
  else
    if TankA3Percent <= AlarmLow then
      AlarmMessage('Tank A3 Low Level Alarm')
    else
      AlarmOff;
end {DisplayAlarm};

procedure Task1;
begin {Task1}
  CheckKey;
end {Task1};

procedure Task2;
begin {Task2}
  ReadJoyStick;
end {Task2};

procedure Task3;
begin {Task3}
  if Mode <> Plot then TransferData;
  case Mode of
    Demo : ProcessDemoLogic;
    Execute : ReadPort;
```

```
  end;
  if Mode <> Plot then DisplayDevice;
end {Task3};

procedure Task4;
begin {Task4}
end {Task4};

procedure Task5;
begin {Task5}
  DisplayGraph;
end {Task5};

procedure Task6;
begin {Task6}
  DisplayLevel;
end {Task6};

procedure Task7;
begin {Task7}
  case Mode of
    Demo : begin
             if (Sec mod 5 = 0) then
             begin
               Inc(CanPosition);
               if CanPosition = 3 then CanPosition := 0;
                 DisplayCanAnimation(CanPosition);
             end;
           end;
    Execute : begin
                if CanPosition <> PreviousPosition then
                  DisplayCanAnimation(CanPosition);
                PreviousPosition := CanPosition;
              end;
  end;
end {Task7};

procedure Task8;
begin {Task8}
  DisplayTemp;
end {Task8};

procedure Task9;
begin {Task9}
  DisplayHostScreen;
end {Task9};

procedure Task10;
begin {Task10}
  ReadDateAndTime;
end {Task10};

procedure Task11;
begin {Task11}
```

```
  DisplayAlarm;
end {Task11};

procedure Task12;
begin {Task12}
end {Task12};

procedure Task13;
begin {Task13}
end {Task13};

procedure Task14;
begin {Task14}
end {Task14};

procedure Task15;
begin {Task15}
end {Task15};

procedure Task16;
begin {Task16}
end {Task16};

procedure RotateSequence;
begin {RotateSequence}
  Sequence := Sequence shl 1;
  if Sequence = $0000 then Sequence := $0001;
end {RotateSequence};

procedure TaskPolling;
begin {TaskPolling}
  if (Sequence and Mask1 = Sequence) then Task1;
  if (Sequence and Mask2 = Sequence) then Task2;
  if (Sequence and Mask3 = Sequence) then Task3;
  if (Sequence and Mask4 = Sequence) then Task4;
  if (Sequence and Mask5 = Sequence) then Task5;
  if (Sequence and Mask6 = Sequence) then Task6;
  if (Sequence and Mask7 = Sequence) then Task7;
  if (Sequence and Mask8 = Sequence) then Task8;
  if (Sequence and Mask9 = Sequence) then Task9;
  if (Sequence and Mask10 = Sequence) then Task10;
  if (Sequence and Mask11 = Sequence) then Task11;
  if (Sequence and Mask12 = Sequence) then Task12;
  if (Sequence and Mask13 = Sequence) then Task13;
  if (Sequence and Mask14 = Sequence) then Task14;
  if (Sequence and Mask15 = Sequence) then Task15;
  if (Sequence and Mask16 = Sequence) then Task16;
end {TaskPolling};

procedure ProcessMonitor;
begin {ProcessMonitor}
  MasterScreenLine := 480;
  SlaveScreenLine  := 480;
  InitialDevice;
```

```
  ClrScr;
  DrawOverlay;
  Ch := ' ';
  Sequence := $01;
  repeat
    TaskPolling;
    RotateSequence;
  until Ch = #27;
end {ProcessMonitor};

begin
  ExitToDOS := False;
  CanPosition := 0;
  TankA2Color := TankAPipeColor;
  TankA3Color := TankAPipeColor;
  TankB2Color := TankBPipeColor;
  TankB3Color := TankBPipeColor;
  TankC2Color := TankCPipeColor;
  TankC3Color := TankCPipeColor;
  MixTankColor := MixTankPipeColor;
  Initial8255;
  ResetADC;
  PrintOnTime := False;
end.
```

```
Program : Pipe Routine
Programmer : Convert from ProDesign II

unit Pipe;

interface

uses Grph7220;

procedure Pipe1(NodeColor : byte);
var Pipe1Status : boolean;

procedure Pipe2(NodeColor : byte);
var Pipe2Status : boolean;

procedure Pipe3(NodeColor : byte);
var Pipe3Status : boolean;

procedure Pipe4(NodeColor : byte);
var Pipe4Status : boolean;

procedure Pipe5(NodeColor : byte);
var Pipe5Status : boolean;

procedure Pipe6(NodeColor : byte);
var Pipe6Status : boolean;

procedure Pipe7(NodeColor : byte);
var Pipe7Status : boolean;

procedure Pipe8(NodeColor : byte);
var Pipe8Status : boolean;

procedure Pipe9(NodeColor : byte);
var Pipe9Status : boolean;

procedure Pipe10(NodeColor : byte);
var Pipe10Status : boolean;

procedure Pipe11(NodeColor : byte);
var Pipe11Status : boolean;

procedure Pipe12(NodeColor : byte);
var Pipe12Status : boolean;

procedure Pipe13(NodeColor : byte);
var Pipe13Status : boolean;
```

```
procedure Pipe14(NodeColor : byte);
var Pipe14Status : boolean;

procedure Pipe15(NodeColor : byte);
var Pipe15Status : boolean;

procedure Pipe16(NodeColor : byte);
var Pipe16Status : boolean;

procedure Pipe17(NodeColor : byte);
var Pipe17Status : boolean;

procedure Pipe18(NodeColor : byte);
var Pipe18Status : boolean;

procedure Pipe19(NodeColor : byte);
var Pipe19Status : boolean;

procedure Pipe20(NodeColor : byte);
var Pipe20Status : boolean;

procedure Pipe21(NodeColor : byte);
var Pipe21Status : boolean;

procedure Pipe22(NodeColor : byte);
var Pipe22Status : boolean;

procedure Pipe23(NodeColor : byte);
var Pipe23Status : boolean;

procedure Pipe24(NodeColor : byte);
var Pipe24Status : boolean;

procedure Pipe25(NodeColor : byte);
var Pipe25Status : boolean;

procedure Pipe26(NodeColor : byte);
var Pipe26Status : boolean;

procedure Pipe27(NodeColor : byte);
var Pipe27Status : boolean;

procedure Pipe28(NodeColor : byte);
var Pipe28Status : boolean;

procedure Pipe29(NodeColor : byte);
var Pipe29Status : boolean;

procedure Pipe30(NodeColor : byte);
var Pipe30Status : boolean;

procedure Pipe31(NodeColor : byte);
var Pipe31Status : boolean;
```

```
procedure Pipe32(NodeColor : byte);
var Pipe32Status : boolean;

procedure Pipe33(NodeColor : byte);
var Pipe33Status : boolean;

procedure Pipe34(NodeColor : byte);
var Pipe34Status : boolean;

implementation

procedure Pipe1(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(223,770,245,770);
end;

procedure Pipe2(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(255,770,277,770);
end;

procedure Pipe3(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(287,770,309,770);
end;

procedure Pipe4(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(341,701,363,701);
end;

procedure Pipe5(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(373,701,497,701);
  Line(459,701,459,598);
  Line(459,598,663,598);
  Line(663,598,663,575);
  Line(577,575,577,598);
  Line(491,598,491,575);
end;

procedure Pipe6(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(507,701,550,701);
end;

procedure Pipe7(NodeColor : byte);
begin
```

```
  SetColor(NodeColor);
  Line(491,546,491,563);
end;

procedure Pipe8(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(577,563,577,546);
end;

procedure Pipe9(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(663,546,663,563);
end;

procedure Pipe10(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(513,546,513,621);
end;

procedure Pipe11(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(599,546,599,621);
end;

procedure Pipe12(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(685,546,685,621);
end;

procedure Pipe13(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(513,632,513,655);
  Line(513,655,577,655);
  Line(577,655,577,678);
end;

procedure Pipe14(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(599,632,599,678);
end;

procedure Pipe15(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(685,632,685,655);
  Line(685,655,620,655);
  Line(620,655,620,678);
```

```
end;

procedure Pipe16(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(599,741,599,764);
end;

procedure Pipe17(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(599,775,599,798);
end;

procedure Pipe18(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(223,322,363,322);
end;

procedure Pipe19(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(373,322,685,322);
  Line(685,322,685,345);
  Line(599,322,599,345);
  Line(513,345,513,322);
end;

procedure Pipe20(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(255,426,255,437);
  Line(255,437,277,437);
end;

procedure Pipe21(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(287,437,314,437);
  Line(314,437,314,449);
end;

procedure Pipe22(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(341,471,363,471);
end;

procedure Pipe23(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(373,471,395,471);
  Line(395,471,395,380);
```

```
  Line(395,380,663,380);
  Line(663,380,663,449);
end;

procedure Pipe24(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(255,506,255,517);
  Line(255,517,277,517);
end;

procedure Pipe25(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(287,517,314,517);
  Line(314,517,314,529);
end;

procedure Pipe26(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(341,552,363,552);
end;

procedure Pipe27(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(373,552,416,552);
  Line(416,552,416,403);
  Line(416,403,577,403);
  Line(577,403,577,449);
end;

procedure Pipe28(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(255,586,255,598);
  Line(255,598,277,598);
end;

procedure Pipe29(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(287,598,314,598);
  Line(314,598,314,609);
end;

procedure Pipe30(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(341,632,363,632);
end;

procedure Pipe31(NodeColor : byte);
```

```
begin
  SetColor(NodeColor);
  Line(373,632,438,632);
  Line(438,632,438,426);
  Line(438,426,491,426);
  Line(491,426,491,449);
end;

procedure Pipe32(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(513,357,513,449);
end;

procedure Pipe33(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(599,357,599,449);
end;

procedure Pipe34(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(685,357,685,449);
end;

begin
end.
```

```
Program : Valve Routine
Programmer : Convert from ProDesign II

unit Valve;

interface

uses Grph7220;

procedure Pump1(NodeColor : byte);
var Pump1Status : boolean;

procedure Pump2(NodeColor : byte);
var Pump2Status : boolean;

procedure Pump3(NodeColor : byte);
var Pump3Status : boolean;

procedure Pump4(NodeColor : byte);
var Pump4Status : boolean;

procedure Pump5(NodeColor : byte);
var Pump5Status : boolean;

procedure Valve1(NodeColor : byte);
var Valve1Status : boolean;

procedure Valve2(NodeColor : byte);
var Valve2Status : boolean;

procedure Valve3(NodeColor : byte);
var Valve3Status : boolean;

procedure Valve4(NodeColor : byte);
var Valve4Status : boolean;

procedure Valve5(NodeColor : byte);
var Valve5Status : boolean;

procedure Valve6(NodeColor : byte);
var Valve6Status : boolean;

procedure Valve7(NodeColor : byte);
var Valve7Status : boolean;

procedure Valve8(NodeColor : byte);
var Valve8Status : boolean;
```

```
procedure Valve9(NodeColor : byte);
var Valve9Status : boolean;

procedure Valve10(NodeColor : byte);
var Valve10Status : boolean;

procedure Valve11(NodeColor : byte);
var Valve11Status : boolean;

procedure Valve12(NodeColor : byte);
var Valve12Status : boolean;

procedure Valve13(NodeColor : byte);
var Valve13Status : boolean;

procedure Valve14(NodeColor : byte);
var Valve14Status : boolean;

procedure Valve15(NodeColor : byte);
var Valve15Status : boolean;

procedure Valve16(NodeColor : byte);
var Valve16Status : boolean;

implementation

procedure Pump1(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(245,770,246,766);
  Line(246,766,247,765);
  Line(247,765,250,764);
  Line(250,764,253,765);
  Line(253,765,254,766);
  Line(254,766,255,770);
  Line(255,770,254,773);
  Line(254,773,253,774);
  Line(253,774,250,775);
  Line(250,775,247,774);
  Line(247,774,246,773);
  Line(246,773,245,770);
  Line(246,774,245,775);
  Line(245,775,255,775);
  Line(255,775,254,774);
end;

procedure Pump2(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(363,632,364,629);
  Line(364,629,365,628);
  Line(365,628,368,626);
  Line(368,626,371,628);
  Line(371,628,372,629);
```

```
  Line(372,629,373,632);
  Line(373,632,372,635);
  Line(372,635,371,637);
  Line(371,637,368,638);
  Line(368,638,365,637);
  Line(365,637,364,635);
  Line(364,635,363,632);
  Line(364,636,363,638);
  Line(363,638,373,638);
  Line(373,638,372,636);
end;

procedure Pump3(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(363,552,364,549);
  Line(364,549,365,548);
  Line(365,548,368,546);
  Line(368,546,371,548);
  Line(371,548,372,549);
  Line(372,549,373,552);
  Line(373,552,372,555);
  Line(372,555,371,557);
  Line(371,557,368,558);
  Line(368,558,365,557);
  Line(365,557,364,555);
  Line(364,555,363,552);
  Line(364,556,363,558);
  Line(363,558,373,558);
  Line(373,558,372,556);
end;

procedure Pump4(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(363,471,364,468);
  Line(364,468,365,467);
  Line(365,467,368,466);
  Line(368,466,371,467);
  Line(371,467,372,468);
  Line(372,468,373,471);
  Line(373,471,372,475);
  Line(372,475,371,476);
  Line(371,476,368,477);
  Line(368,477,365,476);
  Line(365,476,364,475);
  Line(364,475,363,471);
  Line(364,476,363,477);
  Line(363,477,373,477);
  Line(373,477,372,476);
end;

procedure Pump5(NodeColor : byte);
begin
```

```
  SetColor(NodeColor);
  Line(363,322,364,319);
  Line(364,319,365,318);
  Line(365,318,368,317);
  Line(368,317,371,318);
  Line(371,318,372,319);
  Line(372,319,373,322);
  Line(373,322,372,326);
  Line(372,326,371,327);
  Line(371,327,368,328);
  Line(368,328,365,327);
  Line(365,327,364,326);
  Line(364,326,363,322);
  Line(364,327,363,328);
  Line(363,328,373,328);
  Line(373,328,372,327);
end;

procedure Valve1(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(277,764,277,775);
  Line(277,775,287,764);
  Line(287,764,287,775);
  Line(287,775,277,764);
  Line(282,770,282,761);
  Line(277,761,287,761);
  Line(277,761,278,758);
  Line(278,758,279,757);
  Line(279,757,282,756);
  Line(282,756,285,757);
  Line(285,757,286,758);
  Line(286,758,287,761);
end;

procedure Valve2(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(363,695,363,707);
  Line(363,707,373,695);
  Line(373,695,373,707);
  Line(373,707,363,695);
  Line(368,701,368,692);
  Line(363,692,373,692);
  Line(363,692,364,689);
  Line(364,689,365,688);
  Line(365,688,368,687);
  Line(368,687,371,688);
  Line(371,688,372,689);
  Line(372,689,373,692);
end;

procedure Valve3(NodeColor : byte);
begin
```

```
  SetColor(NodeColor);
  Line(497,695,497,706);
  Line(497,706,507,695);
  Line(507,695,507,706);
  Line(507,706,497,695);
  Line(502,701,502,692);
  Line(497,692,507,692);
  Line(497,692,498,689);
  Line(498,689,499,688);
  Line(499,688,502,687);
  Line(502,687,505,688);
  Line(505,688,506,689);
  Line(506,689,507,692);
end;

procedure Valve4(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(497,563,486,563);
  Line(486,563,497,574);
  Line(497,574,486,574);
  Line(486,574,497,563);
  Line(491,569,499,569);
  Line(499,563,499,574);
  Line(499,563,503,564);
  Line(503,564,504,565);
  Line(504,565,505,569);
  Line(505,569,504,572);
  Line(504,572,503,573);
  Line(503,573,499,574);
end;

procedure Valve5(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(583,563,572,563);
  Line(572,563,583,575);
  Line(583,575,572,575);
  Line(572,575,583,563);
  Line(577,569,585,569);
  Line(585,563,585,575);
  Line(585,563,589,564);
  Line(589,564,590,565);
  Line(590,565,591,569);
  Line(591,569,590,573);
  Line(590,573,589,574);
  Line(589,574,585,575);
end;

procedure Valve6(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(668,563,658,563);
  Line(658,563,668,575);
```

```
  Line(668,575,658,575);
  Line(658,575,668,563);
  Line(663,569,671,569);
  Line(671,563,671,575);
  Line(671,563,674,564);
  Line(674,564,676,565);
  Line(676,565,677,569);
  Line(677,569,676,573);
  Line(676,573,674,574);
  Line(674,574,671,575);
end;

procedure Valve7(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(518,621,507,621);
  Line(507,621,518,632);
  Line(518,632,507,632);
  Line(507,632,518,621);
  Line(513,626,521,626);
  Line(521,621,521,632);
  Line(521,621,524,622);
  Line(524,622,525,623);
  Line(525,623,527,626);
  Line(527,626,525,630);
  Line(525,630,524,631);
  Line(524,631,521,632);
end;

procedure Valve8(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(604,621,593,621);
  Line(593,621,604,632);
  Line(604,632,593,632);
  Line(593,632,604,621);
  Line(599,626,607,626);
  Line(607,621,607,632);
  Line(607,621,610,622);
  Line(610,622,611,623);
  Line(611,623,612,626);
  Line(612,626,611,630);
  Line(611,630,610,631);
  Line(610,631,607,632);
end;

procedure Valve9(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(690,621,679,621);
  Line(679,621,690,632);
  Line(690,632,679,632);
  Line(679,632,690,621);
  Line(685,626,693,626);
```

```
  Line(693,621,693,632);
  Line(693,621,696,622);
  Line(696,622,697,623);
  Line(697,623,698,626);
  Line(698,626,697,630);
  Line(697,630,696,631);
  Line(696,631,693,632);
end;

procedure Valve10(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(277,592,277,603);
  Line(277,603,287,592);
  Line(287,592,287,603);
  Line(287,603,277,592);
  Line(282,598,282,589);
  Line(277,589,287,589);
  Line(277,589,278,586);
  Line(278,586,279,585);
  Line(279,585,282,584);
  Line(282,584,285,585);
  Line(285,585,286,586);
  Line(286,586,287,589);
end;

procedure Valve11(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(277,512,277,523);
  Line(277,523,287,512);
  Line(287,512,287,523);
  Line(287,523,277,512);
  Line(282,517,282,509);
  Line(277,509,287,509);
  Line(277,509,278,506);
  Line(278,506,279,505);
  Line(279,505,282,503);
  Line(282,503,285,505);
  Line(285,505,286,506);
  Line(286,506,287,509);
end;

procedure Valve12(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(277,431,277,443);
  Line(277,443,287,431);
  Line(287,431,287,443);
  Line(287,443,277,431);
  Line(282,437,282,428);
  Line(277,428,287,428);
  Line(277,428,278,425);
  Line(278,425,279,424);
```

```
  Line(279,424,282,423);
  Line(282,423,285,424);
  Line(285,424,286,425);
  Line(286,425,287,428);
end;

procedure Valve13(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(518,345,507,345);
  Line(507,345,518,357);
  Line(518,357,507,357);
  Line(507,357,518,345);
  Line(513,351,521,351);
  Line(521,345,521,357);
  Line(521,345,524,346);
  Line(524,346,525,347);
  Line(525,347,527,351);
  Line(527,351,525,355);
  Line(525,355,524,356);
  Line(524,356,521,357);
end;

procedure Valve14(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(604,345,593,345);
  Line(593,345,604,357);
  Line(604,357,593,357);
  Line(593,357,604,345);
  Line(599,351,607,351);
  Line(607,345,607,357);
  Line(607,345,610,346);
  Line(610,346,611,347);
  Line(611,347,612,351);
  Line(612,351,611,355);
  Line(611,355,610,356);
  Line(610,356,607,357);
end;

procedure Valve15(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(690,345,679,345);
  Line(679,345,690,357);
  Line(690,357,679,357);
  Line(679,357,690,345);
  Line(685,351,693,351);
  Line(693,345,693,357);
  Line(693,345,696,346);
  Line(696,346,697,347);
  Line(697,347,698,351);
  Line(698,351,697,355);
  Line(697,355,696,356);
```

```
  Line(696,356,693,357);
end;

procedure Valve16(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(604,764,593,764);
  Line(593,764,604,775);
  Line(604,775,593,775);
  Line(593,775,604,764);
  Line(599,770,607,770);
  Line(607,764,607,775);
  Line(607,764,610,765);
  Line(610,765,611,766);
  Line(611,766,612,770);
  Line(612,770,611,773);
  Line(611,773,610,774);
  Line(610,774,607,775);
end;

begin
end.
```

```
Program : Tank Routine
Programmer : Convert from ProDesign II

unit Tank;

interface

uses Grph7220;

procedure TankA2(var Percent : integer);
var TankA2Level : integer;
    TankA2Percent : integer;
    TankA2Color : byte;

procedure TankB2(var Percent : integer);
var TankB2Level : integer;
    TankB2Percent : integer;
    TankB2Color : byte;

procedure TankC2(var Percent : integer);
var TankC2Level : integer;
    TankC2Percent : integer;
    TankC2Color : byte;

procedure TankA3(var Percent : integer);
var TankA3Level : integer;
    TankA3Percent : integer;
    TankA3Color : byte;

procedure TankB3(var Percent : integer);
var TankB3Level : integer;
    TankB3Percent : integer;
    TankB3Color : byte;

procedure TankC3(var Percent : integer);
var TankC3Level : integer;
    TankC3Percent : integer;
    TankC3Color : byte;

procedure MixTank(var Percent : integer);
var MixTankLevel : integer;
    MixTankPercent : integer;
    MixTankColor : byte;

procedure GraphA2(var Percent : integer);
var GraphA2Level : integer;
    GraphA2Percent : integer;
    GraphA2Color : byte;
```

```
procedure GraphB2(var Percent : integer);
var GraphB2Level : integer;
    GraphB2Percent : integer;
    GraphB2Color : byte;

procedure GraphC2(var Percent : integer);
var GraphC2Level : integer;
    GraphC2Percent : integer;
    GraphC2Color : byte;

procedure GraphA3(var Percent : integer);
var GraphA3Level : integer;
    GraphA3Percent : integer;
    GraphA3Color : byte;

procedure GraphB3(var Percent : integer);
var GraphB3Level : integer;
    GraphB3Percent : integer;
    GraphB3Color : byte;

procedure GraphC3(var Percent : integer);
var GraphC3Level : integer;
    GraphC3Percent : integer;
    GraphC3Color : byte;

procedure TempA2(var Percent : integer);
var TempA2Level : integer;
    TempA2Percent : integer;
    TempA2Color : byte;

procedure TempB2(var Percent : integer);
var TempB2Level : integer;
    TempB2Percent : integer;
    TempB2Color : byte;

procedure TempC2(var Percent : integer);
var TempC2Level : integer;
    TempC2Percent : integer;
    TempC2Color : byte;

procedure TempA3(var Percent : integer);
var TempA3Level : integer;
    TempA3Percent : integer;
    TempA3Color : byte;

procedure TempB3(var Percent : integer);
var TempB3Level : integer;
    TempB3Percent : integer;
    TempB3Color : byte;

procedure TempC3(var Percent : integer);
var TempC3Level : integer;
    TempC3Percent : integer;
```

```
    TempC3Color : byte;

procedure TempMix(var Percent : integer);
var TempMixLevel : integer;
    TempMixPercent : integer;
    TempMixColor : byte;

implementation

procedure TankA2(var Percent : integer);
begin
  SetColor(TankA2Color);
  CurrentLevel := TankA2Level;
  DrawLevel(309,471,341,449,Percent);
  TankA2Level := CurrentLevel;
end;

procedure TankB2(var Percent : integer);
begin
  SetColor(TankB2Color);
  CurrentLevel := TankB2Level;
  DrawLevel(309,552,341,529,Percent);
  TankB2Level := CurrentLevel;
end;

procedure TankC2(var Percent : integer);
begin
  SetColor(TankC2Color);
  CurrentLevel := TankC2Level;
  DrawLevel(309,632,341,609,Percent);
  TankC2Level := CurrentLevel;
end;

procedure TankA3(var Percent : integer);
begin
  SetColor(TankA3Color);
  CurrentLevel := TankA3Level;
  DrawLevel(652,512,695,449,Percent);
  TankA3Level := CurrentLevel;
end;

procedure TankB3(var Percent : integer);
begin
  SetColor(TankB3Color);
  CurrentLevel := TankB3Level;
  DrawLevel(566,512,609,449,Percent);
  TankB3Level := CurrentLevel;
end;

procedure TankC3(var Percent : integer);
begin
  SetColor(TankC3Color);
  CurrentLevel := TankC3Level;
  DrawLevel(481,512,524,449,Percent);
```

```
  TankC3Level := CurrentLevel;
end;

procedure MixTank(var Percent : integer);
begin
  SetColor(MixTankColor);
  CurrentLevel := MixTankLevel;
  DrawLevel(556,715,642,678,Percent);
  MixTankLevel := CurrentLevel;
end;

procedure GraphA2(var Percent : integer);
begin
  SetColor(GraphA2Color);
  CurrentLevel := GraphA2Level;
  DrawLevel(780,327,791,59,Percent);
  GraphA2Level := CurrentLevel;
end;

procedure GraphB2(var Percent : integer);
begin
  SetColor(GraphB2Color);
  CurrentLevel := GraphB2Level;
  DrawLevel(860,327,871,59,Percent);
  GraphB2Level := CurrentLevel;
end;

procedure GraphC2(var Percent : integer);
begin
  SetColor(GraphC2Color);
  CurrentLevel := GraphC2Level;
  DrawLevel(952,327,962,59,Percent);
  GraphC2Level := CurrentLevel;
end;

procedure GraphA3(var Percent : integer);
begin
  SetColor(GraphA3Color);
  CurrentLevel := GraphA3Level;
  DrawLevel(780,730,791,461,Percent);
  GraphA3Level := CurrentLevel;
end;

procedure GraphB3(var Percent : integer);
begin
  SetColor(GraphB3Color);
  CurrentLevel := GraphB3Level;
  DrawLevel(860,730,871,461,Percent);
  GraphB3Level := CurrentLevel;
end;

procedure GraphC3(var Percent : integer);
begin
  SetColor(GraphC3Color);
```

```
  CurrentLevel := GraphC3Level;
  DrawLevel(952,730,962,461,Percent);
  GraphC3Level := CurrentLevel;
end;

procedure TempA2(var Percent : integer);
begin
  SetColor(TempA2Color);
  CurrentLevel := TempA2Level;
  DrawLevel(136,220,163,86,Percent);
  TempA2Level := CurrentLevel;
end;

procedure TempB2(var Percent : integer);
begin
  SetColor(TempB2Color);
  CurrentLevel := TempB2Level;
  DrawLevel(190,220,217,86,Percent);
  TempB2Level := CurrentLevel;
end;

procedure TempC2(var Percent : integer);
begin
  SetColor(TempC2Color);
  CurrentLevel := TempC2Level;
  DrawLevel(243,220,270,86,Percent);
  TempC2Level := CurrentLevel;
end;

procedure TempA3(var Percent : integer);
begin
  SetColor(TempA3Color);
  CurrentLevel := TempA3Level;
  DrawLevel(297,220,324,86,Percent);
  TempA3Level := CurrentLevel;
end;

procedure TempB3(var Percent : integer);
begin
  SetColor(TempB3Color);
  CurrentLevel := TempB3Level;
  DrawLevel(351,220,378,86,Percent);
  TempB3Level := CurrentLevel;
end;

procedure TempC3(var Percent : integer);
begin
  SetColor(TempC3Color);
  CurrentLevel := TempC3Level;
  DrawLevel(404,220,431,86,Percent);
  TempC3Level := CurrentLevel;
end;

procedure TempMix(var Percent : integer);
```

```
begin
  SetColor(TempMixColor);
  CurrentLevel := TempMixLevel;
  DrawLevel(512,220,539,86,Percent);
  TempMixLevel := CurrentLevel;
end;

begin
  TankA2Level := 0;
  TankB2Level := 0;
  TankC2Level := 0;
  TankA3Level := 0;
  TankB3Level := 0;
  TankC3Level := 0;
  MixTankLevel := 0;
  GraphA2Level := 0;
  GraphB2Level := 0;
  GraphC2Level := 0;
  GraphA3Level := 0;
  GraphB3Level := 0;
  GraphC3Level := 0;
  TempA2Level := 0;
  TempB2Level := 0;
  TempC2Level := 0;
  TempA3Level := 0;
  TempB3Level := 0;
  TempC3Level := 0;
  TempMixLevel := 0;
end.
```

```
Program : Can Routine
Programmer : Convert from ProDesign II

unit Can;

interface

uses Grph7220;

procedure Can1(NodeColor : byte);
var Can1Status : boolean;

procedure Can2(NodeColor : byte);
var Can2Status : boolean;

procedure Can3(NodeColor : byte);
var Can3Status : boolean;

procedure Can4(NodeColor : byte);
var Can4Status : boolean;

procedure Can5(NodeColor : byte);
var Can5Status : boolean;

procedure Can6(NodeColor : byte);
var Can6Status : boolean;

procedure Can7(NodeColor : byte);
var Can7Status : boolean;

procedure Can8(NodeColor : byte);
var Can8Status : boolean;

procedure Can9(NodeColor : byte);
var Can9Status : boolean;

procedure Can10(NodeColor : byte);
var Can10Status : boolean;

procedure Can11(NodeColor : byte);
var Can11Status : boolean;

procedure Can12(NodeColor : byte);
var Can12Status : boolean;

procedure Can13(NodeColor : byte);
var Can13Status : boolean;
```

```
procedure Can14(NodeColor : byte);
var Can14Status : boolean;

procedure Can15(NodeColor : byte);
var Can15Status : boolean;

implementation

procedure Can1(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(517,799,517,816);
  Line(517,816,528,816);
  Line(528,816,528,799);
  Line(528,799,517,799);
end;

procedure Can2(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(528,799,528,816);
  Line(528,816,539,816);
  Line(539,816,539,799);
  Line(539,799,528,799);
end;

procedure Can3(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(539,799,539,816);
  Line(539,816,549,816);
  Line(549,816,549,799);
  Line(549,799,539,799);
end;

procedure Can4(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(549,799,549,816);
  Line(549,816,560,816);
  Line(560,816,560,799);
  Line(560,799,549,799);
end;

procedure Can5(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(560,799,560,816);
  Line(560,816,571,816);
  Line(571,816,571,799);
  Line(571,799,560,799);
end;

procedure Can6(NodeColor : byte);
```

```
begin
  SetColor(NodeColor);
  Line(571,799,571,816);
  Line(571,816,581,816);
  Line(581,816,581,799);
  Line(581,799,571,799);
end;

procedure Can7(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(581,799,581,816);
  Line(581,816,592,816);
  Line(592,816,592,799);
  Line(592,799,581,799);
end;

procedure Can8(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(592,799,592,816);
  Line(592,816,603,816);
  Line(603,816,603,799);
  Line(603,799,592,799);
end;

procedure Can9(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(603,799,603,816);
  Line(603,816,614,816);
  Line(614,816,614,799);
  Line(614,799,603,799);
end;

procedure Can10(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(614,799,614,816);
  Line(614,816,624,816);
  Line(624,816,624,799);
  Line(624,799,614,799);
end;

procedure Can11(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(624,799,624,816);
  Line(624,816,635,816);
  Line(635,816,635,799);
  Line(635,799,624,799);
end;

procedure Can12(NodeColor : byte);
```

```
begin
  SetColor(NodeColor);
  Line(635,799,635,816);
  Line(635,816,646,816);
  Line(646,816,646,799);
  Line(646,799,635,799);
end;

procedure Can13(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(646,799,646,816);
  Line(646,816,657,816);
  Line(657,816,657,799);
  Line(657,799,646,799);
end;

procedure Can14(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(657,799,657,816);
  Line(657,816,667,816);
  Line(667,816,667,799);
  Line(667,799,657,799);
end;

procedure Can15(NodeColor : byte);
begin
  SetColor(NodeColor);
  Line(667,799,667,816);
  Line(667,816,678,816);
  Line(678,816,678,799);
  Line(678,799,667,799);
end;

begin
end.
```

```
Program : CNVRTDEV & CNVRTLEV Variable Declaration
Programmer : Phakorn Hutasangkas.

unit CnvrtDcl; { Convert's Declaration }

interface

const
  MaxLineInProcedure = 200;
  MaxX = 1000; { Hercules }
  MaxY = 1000;
type
  FileType = Text;

var
  SrceFile : FileType;
  DestFile : FileType;
  FileName : string;

implementation

begin
end.
```

```
Program : Device Conversion Utility
Programmer : Phakorn Hutasangkas.

Program ConvertDevice;

uses Dos,Crt,CnvrtDcl,OpenFile;

var
  Line1,Line2 : string;
  BufLine : string;
  NodeCount : word;
  UnitName : string;
  NodeName : string;
  Command : string[4];
  XPosition,YPosition : byte;
  XStr,YStr : string;
  YValue : word;
  ErrorCode : integer;
  LengthY : byte;
  StartUp : boolean;
  X,Y : string;
  Parameter : string;
  LineCount : byte;
  NodeFile,BufferFile1,BufFile : FileType;

procedure SearchParameter(BufLine : string);
begin {SearchParameter}
  Delete(BufLine,1,Pos(' ',BufLine));
  XPosition := Pos(' ',BufLine);
  if XPosition = 0 then
    XStr := Copy(BufLine,1,Length(BufLine)) { Single Parameter }
  else
    XStr := Copy(BufLine,1,XPosition - 1);  { Double Parameter }
  Delete(BufLine,1,Pos(' ',BufLine));
  YStr := Copy(BufLine,1,Length(BufLine));
  Val(YStr,YValue,ErrorCode);
  if ErrorCode = 0 then
    YValue := Abs(MaxY - YValue);
  Str(YValue,YStr);
end {SearchParameter};

procedure RejectSETCCommand;
begin {RejectSETCCommand}
  Assign(BufFile,'BUFFER1.$$$');
  Rewrite(BufFile);
  while not EOF(SrceFile) do
  begin
    Readln(SrceFile,Line1);
```

```
    Command := Copy(Line1,1,4);
    if Command <> 'SETC' then
      Writeln(BufFile,Line1);
  end;
end {RejectSETCCommand};

procedure ReadNode;
begin {ReadNode}
  Assign(DestFile,'BUFFER2.$$$');
  Rewrite(DestFile);
  Reset(BufFile);
  Readln(BufFile,Line1);
  SearchParameter(Line1);
  while not EOF(BufFile) do
  begin
    X := XStr;
    Y := YStr;
    Readln(BufFile,Line2);
    SearchParameter(Line2);
    if ((X = XStr) and (Y = YStr)) then
    begin
      Writeln(DestFile,'*',NodeCount);
      Inc(NodeCount);
      StartUp := True;
      LineCount := 1;
    end;
    if LineCount <> 2 then
      Writeln(DestFile,Line1)
    else
    begin
      StartUp := False;
      LineCount := 1;
    end;
    if StartUp then Inc(LineCount);
    Line1 := Line2;
  end;
end {ReadNode};

procedure ConvertDataFile;
begin {ConvertDataFile}
  Assign(SrceFile,'BUFFER2.$$$');
  Reset(SrceFile);
  Write('Enter Unit Name : ');
  Readln(UnitName);
  Reset(NodeFile);
  while not EOF(SrceFile) do
  begin
    Readln(SrceFile,BufLine);
    if Copy(BufLine,1,1) = '*' then
    begin
      if not StartUp then
      begin
        Writeln(BufferFile1,'end;');
        Writeln(BufferFile1);
```

```
      end
      else
        StartUp := False;
      Readln(NodeFile,NodeName);
      Writeln(BufferFile1,'procedure ',NodeName,'(NodeColor : byte);');
      Writeln(BufferFile1,'begin');
      Writeln(BufferFile1,'  SetColor(NodeColor);');
      Readln(SrceFile,BufLine);
    end;
    Command := Copy(BufLine,1,4);
    if Command = 'MOVE' then
    begin
      SearchParameter(BufLine);
      X := XStr;
      Y := YStr;
    end;
    if Command = 'LINE' then
    begin
      SearchParameter(BufLine);
      Parameter := 'Line(' + X + ',' + Y + ',' + XStr + ',' + YStr + ');';
      Writeln(BufferFile1,'  ',Parameter);
      X := XStr;
      Y := YStr;
    end;
  end;
  Writeln(BufferFile1,'end;');
end {ConvertDataFile};

procedure DataFileProcessing;
begin {DataFileProcessing}
  Assign(BufferFile1,'BUFFER1.$$$');
  Rewrite(BufferFile1);
  ConvertDataFile;
  Writeln(DestFile,'unit ',UnitName,';');
  Writeln(DestFile);
  Writeln(DestFile,'interface');
  Writeln(DestFile);
  Writeln(DestFile,'uses Grph7220;');
  Writeln(DestFile);
  Reset(NodeFile);
  while not EOF(NodeFile) do
  begin
    Readln(NodeFile,NodeName);
    Writeln(DestFile,'procedure ',NodeName,'(NodeColor : byte);');
    Writeln(DestFile,'var ',NodeName,'Status : boolean;');
    Writeln(DestFile);
  end;
  Writeln(DestFile,'implementation');
  Writeln(DestFile);
  Reset(BufferFile1);
  while not EOF(BufferFile1) do
  begin
    Readln(BufferFile1,BufLine);
    Writeln(DestFile,BufLine);
```

```
  end;
  Writeln(DestFile);
  Writeln(DestFile,'begin');
  Writeln(DestFile,'end.');
  Close(BufferFile1);
  Close(DestFile);
end {DataFileProcessing};

begin
  Writeln;
  Writeln('CNVRTDEV Version 1.00A by Phakorn Hutasangkas. 26 June 1988');
  Writeln;
  NodeCount := 1;
  if OpenReadFile('Enter Plot Source Filename',SrceFile,FileName,'.PLT') then
  begin
    RejectSETCCommand;
    ReadNode;
  end;
  Close(DestFile);
  Close(SrceFile);
  StartUp := True;
  NodeCount := 1;
  if OpenReadFile('Enter Node Filename',NodeFile,FileName,'.NDE') then
  begin
    if OpenWriteFile('Enter Destination Filename',DestFile,FileName,'.PAS') then
      DataFileProcessing;
    Close(SrceFile);
  end;
end.
```

```
Program : Level Conversion Utility
Programmer : Phakorn Hutasangkas.

Program ProDesignToPASCAL;

uses Dos,Crt,CnvrtDcl,OpenFile;

var
  BufLine : string;
  NodeCount : word;
  UnitName : string;
  NodeName : string;
  Command : string;
  XPosition,YPosition : byte;
  XStr,YStr : string;
  YValue : word;
  ErrorCode : integer;
  LengthY : byte;
  StartUp : boolean;
  X,Y : string;
  Parameter : string;
  NodeFile,BufferFile1 : FileType;

procedure SearchParameter;
begin {SearchParameter}
  Delete(BufLine,1,Pos(' ',BufLine));
  XPosition := Pos(' ',BufLine);
  if XPosition = 0 then
    XStr := Copy(BufLine,1,Length(BufLine)) { Single Parameter }
  else
    XStr := Copy(BufLine,1,XPosition - 1);  { Double Parameter }
  Delete(BufLine,1,Pos(' ',BufLine));
  YStr := Copy(BufLine,1,Length(BufLine));
  Val(YStr,YValue,ErrorCode);
  if ErrorCode = 0 then
    YValue := Abs(MaxY - YValue);
  Str(YValue,YStr);
end {SearchParameter};

procedure ConvertDataFile;
begin {ConvertDataFile}
  Write('Enter Unit Name : ');
  Readln(UnitName);
  Reset(NodeFile);
  while not EOF(SrceFile) do
  begin
    Readln(SrceFile,BufLine);
    Command := Copy(BufLine,1,4);
```

```
    if Command = 'MOVE' then
    begin
      Readln(NodeFile,NodeName);
      if Length(NodeName) > 0 then
      begin
        if not StartUp then
        begin
          Writeln(BufferFile1,'end;');
          Writeln(BufferFile1);
        end
        else
          StartUp := False;
        Writeln(BufferFile1,'procedure ',NodeName,'(var Percent : integer);');
        Writeln(BufferFile1,'begin');
        Writeln(BufferFile1,'  SetColor(',NodeName,'Color);');
        Writeln(BufferFile1,'  CurrentLevel := ',NodeName,'Level;');
        SearchParameter;
        X := XStr;
        Y := YStr;
      end;
    end;
    if Command = 'LINE' then
    begin
      SearchParameter;
      Parameter := 'DrawLevel(' + X + ',' + Y + ',' + XStr + ',' + YStr + ',Percent);';
      Writeln(BufferFile1,'  ',Parameter);
      X := XStr;
      Y := YStr;
      Writeln(BufferFile1,'  ',NodeName,'Level := CurrentLevel;');
    end;
  end;
  Writeln(BufferFile1,'end;');
end {ConvertDataFile};

procedure DataFileProcessing;
begin {DataFileProcessing}
  Assign(BufferFile1,'BUFFER1.$$$');
  Rewrite(BufferFile1);
  ConvertDataFile;
  Writeln(DestFile,'unit ',UnitName,';');
  Writeln(DestFile);
  Writeln(DestFile,'interface');
  Writeln(DestFile);
  Writeln(DestFile,'uses Grph7220;');
  Writeln(DestFile);
  Reset(NodeFile);
  while not EOF(NodeFile) do
  begin
    Readln(NodeFile,NodeName);
    Writeln(DestFile,'procedure ',NodeName,'(var Percent : integer);');
    Writeln(DestFile,'var ',NodeName,'Level : integer;');
    Writeln(DestFile,'    ',NodeName,'Percent : integer;');
    Writeln(DestFile,'    ',NodeName,'Color : byte;');
    Writeln(DestFile);
```

```
  end;
  Writeln(DestFile,'implementation');
  Writeln(DestFile);
  Reset(BufferFile1);
  while not EOF(BufferFile1) do
  begin
    Readln(BufferFile1,BufLine);
    Writeln(DestFile,BufLine);
  end;
  Writeln(DestFile);
  Writeln(DestFile,'begin');
  Reset(NodeFile);
  while not EOF(NodeFile) do
  begin
    Readln(NodeFile,NodeName);
    Writeln(DestFile,'  ',NodeName,'Level := 0;');
  end;
  Writeln(DestFile,'end.');
  Close(BufferFile1);
  Close(DestFile);
end {DataFileProcessing};

begin
  StartUp := True;
  NodeCount := 1;
  Writeln('CNVRTLEV Version 1.00A by Phakorn Hutasangkas. 26 June 1988');
  Writeln;
  if OpenReadFile('Enter Plot Filename',SrceFile,FileName,'.PLT')
    and OpenReadFile('Enter Node Filename',NodeFile,FileName,'.NDE') then
  begin
    if OpenWriteFile('Enter Destination Filename',DestFile,FileName,'.PAS') then
      DataFileProcessing;
    Close(SrceFile);
  end;
end.
```