โปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์

COMPUTER-AIDED PCB DESIGN

สุทธินันท์ ปรมาธิกุล

SUTTINUN PORAMATIKUL

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2531

โปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์

COMPUTER-AIDED PCB DESIGN

สุทธินันท์ ปรมาธิกุล

SUTTINUN PORAMATIKUL

อาจารย์ที่ปรึกษา

ผู้ช่วยศาสตราจารย์ ดร. กนก เจนจิระพงศ์เวช

ADVISOR

Asst.Prof. Dr. Kanok Janjirapongvej

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2531

สารบัญ

		หน้า
บทคัดย่อ		I
Abstract		II
บทที่ 1	บทนำ	1
บทที่ 2	ส่วนประกอบ และคำสั่งที่ใช้ในการสร้างแผ่นวงจรพิมพ์	4
2.1	ส่วนประกอบที่สำคัญของแผ่นวงจรพิมพ์	4
2.2	การใช้แป้นพิมพ์ควบคุมการทำงาน	8
2.3	คำสั่งที่ใช้ในการควบคุมการสร้างแผ่นวงจรพิมพ์	9
2.4	คำสั่งควบคุมการสร้างภาพ	25
2.5	คำสั่งและโปรแกรมควบคุม	26
บทที่ 3	ตัวอย่างการใช้งาน	29
3.1	การเริ่มต้นในการใช้งานโปรแกรม	29
3.2	การสร้างอุปกรณ์เก็บไว้ในไลบรารี	29
3.3	การสร้างแผ่นวงจรพิมพ์	31
บทที่ 4	การทำงานของโปรแกรม	47
4.1	โปรแกรมหลัก	47
4.2	โปรแกรมควบคุมการแสดงรายการต่างๆ	50
4.3	โปรแกรมควบคุมการทำงานทั้งหมด	51
4.4	โปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์	53
4.5	โปรแกรมควบคุมการสร้างอุปกรณ์	54
4.6	โปรแกรมควบคุมการพิมพ์	56
4.7	โปรแกรมกำหนดค่าสภาวะในการทำงาน	57
4.8	โปรแกรมควบคุมการสร้างภาพ	58

4.9 โปรแกรมควบคุมการสร้างลาย เส้น 60
4.10 โปรแกรมควบคุมการสร้างวงกลม 62
4.11 โปรแกรมควบคุมการย้ายข้อความ 64
4.12 โปรแกรมย้ายข้อความ 65
4.13 โปรแกรมสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์ 67
4.14 โปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์ 68
4.15 โปรแกรมควบคุมตำแหน่งในการสร้างภาพ 70
4.16 โปรแกรมใส่โดนัท 72
4.17 โปรแกรมแสดงภาพโดยการปรับตำแหน่งการวาดใหม่ 73
4.18 โปรแกรมควบคุมการย้าย เส้น 75
4.19 โปรแกรม เคลื่อนย้าย เส้น 76
4.20 โปรแกรมแสดงภาพโดยการวาดซ้ำ 78
4.21 โปรแกรมแสดงผลภาพแบบขยาย 80
4.22 โปรแกรมควบคุมที่ทำงานด้วยการกำหนดค่าตัวแปร 82
4.23 โปรแกรมใส่โดนัทและ เปลี่ยนด้านแผ่นวงจรพิมพ์ 83
4.24 โปรแกรมลบโดนัท 84
4.25 โปรแกรมลบวงกลม 84
4.26 โปรแกรมควบคุมการแสดงผลกริด 85
4.27 การแสดงผลภาพแบบกราฟฟิก 86
บทที่ 5 ผลลัพธ์การพิมพ์ 95
5.1 ผลลัพธ์การพิมพ์แบบต่าง ๆ 95
5.2 ตัวอย่างฟิลม์แบบที่ 1 95
5.3 ตัวอย่างฟิลม์แบบที่ 2 97
บทที่ 6 สรุปผลการวิจัย 100
กิตติกรรมประกาศ 104
เอกสารอ้างอิง 105
ภาคผนวก 1 การจัดวางข้อมูลแผ่นวงจรพิมพ์ 106
ภาพผนวก 2 การจัดวางข้อมูลอุปกรณ์ในไลบรารี 107
ภาคผนวก 3 การกำหนดขนาดและชนิดของโดนัท 110
ภาคผนวก 4 รายละ เอียดของโปรแกรม 112

หัวข้อวิทยานิพนธ์ โปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์
นักศึกษา นาย สุทธินันท์ ปรมาธิกุล
อาจารย์ที่ปรึกษา ผศ. ดร. กนก เจนจิระพงศ์เวช
ระดับการศึกษา วิศวกรรมศาสตรมหาบัณฑิตทางวิศวกรรมไฟฟ้า

บทคัดย่อ

การออกแบบวงจรอิเลคทรอนิคส์นั้นส่วนหนึ่งที่มีความสำคัญในการสร้างวงจรก็คือการออกแบบแผ่นวงจรพิมพ์ ซึ่งส่วนใหญ่ในปัจจุบันใช้เครื่องไมโครคอมพิวเตอร์ และโปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์สำหรับงานดังกล่าว แต่โปรแกรมที่ใช้ช่วยในการออกแบบส่วนใหญ่มักจะใช้งานไม่สะดวก และมีขีดความสามารถจำกัด โดยเฉพาะแผ่นวงจรพิมพ์ที่ใช้อุปกรณ์บางชนิดมีที่ระยะห่างของขาไม่เป็นมาตรฐานทั่ว ๆ ไป และอุปกรณ์ชนิดติดผิวแผ่นวงจรพิมพ์ (Surface Mounting Device) ซึ่งมีขนาดเล็ก และไม่ต้องการรูในการใส่ขาอุปกรณ์ ทำให้โปรแกรมช่วยในออกแบบที่มีใช้กันอยู่ทั่ว ๆ ไป อย่างเช่น smARTWORK และ PROTEL ไม่สามารถนำมาใช้งานแบบนี้ได้

วิทยานิพนธ์ฉบับนี้ขอเสนอโปรแกรมช่วยในออกแบบแผ่นวงจรพิมพ์ที่มีความสามารถสูง เนื่องจากพัฒนาขึ้นมาจากการศึกษาข้อดีและข้อเสียของโปรแกรมต่าง ๆ เหล่านั้น และเพิ่มฟังก์ชั่น (Function) การทำงานนอกเหนือจากที่มีใช้กันอยู่ทั่ว ๆ ไป เช่น สามารถสร้างไลบรารีของอุปกรณ์ (Library) ต่าง ๆ เปลี่ยนแปลงขนาดหรือชนิดของโคนัทได้ตามความเหมาะสม เคลื่อนย้ายเส้นแทนการลบและสร้างใหม่ อีกทั้งยังสามารถเลือกโหมด (Mode) การแสดงผลในแบบธรรมดา (Normal Mode) หรือ แบบความเร็วสูง (Fast Mode or Check Plot Mode) ได้ และอื่นๆ

Thesis Title	Computer-Aided PCB Design.
Name	Mr. Suttinun Poramatikul.
Thesis Advisor	Asst. Prof Dr.Kanok Janjirapongvej.
Level of Study	Master of Engineering in Electrical Engineering
Academic Year	1988

ABSTRACT

Printed circuit board design is an important procedure in electronics work. Most of them are worked on microcomputer with the aided of CAD program. For example the smARTWORK and PROTEL but there program do not satisfy for users due to the limitation of utility of the program. In case of special purposed design, such as the non standard devices or surface mounting device.

This thesis presents the method of PCB design with the aided of microcomputer. It is shown that the program proposed here make it possible to design any printed circuit with high efficiency and high performance. For example the library devices can be created by user, as well as changing sizes or shapes of PCB's donut or moving PCB's wire and selectable screen display mode in fast mode or normal mode etc.

บทที่ 1
บทนำ

การออกแบบวงจรอิเลคทรอนิคส์สิ่งหนึ่งที่จะมองข้ามไม่ได้ก็คือการออกแบบแผ่นวงจรพิมพ์ ซึ่งเปรียบเสมือนเป็นพื้นฐานของการสร้างงานทางด้านอิเลคทรอนิคส์ โดยจะทำหน้าที่เชื่อมต่อสัญญาณทางไฟฟ้าให้กับชิ้นส่วนต่าง ๆ ของวงจรอิเลคทรอนิคส์ ด้วยลายเส้นตัวนำซึ่งโดยมากแล้วเป็นโลหะจำพวกทองแดง ลายเส้นเหล่านี้จะถูกสร้างขึ้นจากแบบขยายที่มีขนาดเป็นเท่าตัวแล้วย่อขนาดลงเท่าขนาดจริง เพื่อให้ลายเส้นของวงจรมีความคมชัดสูงขึ้น

ในกรณีที่วงจรมีความซับซ้อนมาก การออกแบบลายเส้นวงจรเพื่อสร้างแผ่นวงจรพิมพ์ก็จะมีความซับซ้อนเพิ่มมากขึ้นเป็นเงาตามตัว ด้วยเหตุนี้เองทำให้การออกแบบ การแก้ไข และ การเพิ่มเติมลายเส้นทำได้ยากขึ้นด้วย ปัญหาต่าง ๆ เหล่านี้เริ่มลดน้อยลงเรื่อย ๆ เมื่อมีการนำเอาเครื่องไมโครคอมพิวเตอร์มาช่วยในการออกแบบแผ่นวงจรพิมพ์ เป็นผลให้ความสามารถในการสร้างลายวงจรมีประสิทธิภาพสูง และรวดเร็วขึ้น อีกทั้งการแก้ไขข้อผิดพลาด และ เพิ่มเติม ส่วนที่ขาดหายไป หรือ ไม่สมบูรณ์ของแผ่นวงจรพิมพ์ทำได้สะดวกมากขึ้น ด้วยเหตุนี้เอง เป็นเหตุให้มีการพัฒนาโปรแกรม เพื่อช่วยในการออกแบบแผ่นวงจรพิมพ์มากขึ้น และ มีการนำมาใช้งานกันอย่างกว้างขวาง แต่จากการศึกษาโดยการนำมาใช้งาน เพื่อพิจารณาถึงขีดความสามารถในการทำงานโปรแกรมต่าง ๆ นั้น พบว่าโปรแกรมแต่ละโปรแกรม มีข้อเสียที่น่าจะมีการพัฒนา และ แก้ไข วิทยานิพนธ์ฉบับนี้จึงใคร่ขอเสนอโปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์ ซึ่งได้พัฒนาขึ้นเพื่อแก้ไขข้อเสียต่าง ๆ ของโปรแกรมเหล่านั้น และ เพิ่มเติมความสามารถบางอย่าง เพื่อให้การใช้งานมีความสะดวกมากขึ้น อีกทั้งเป็นการรองรับการเปลี่ยนแปลงของเทคโนโลยีทางอิเลคทรอนิคส์ที่ค่อนข้างจะรวดเร็ว และนำไปสู่หนทางการพึ่งตัวเองในอนาคต เนื่องจากในปัจจุบันโปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์นี้ ต้องซื้อหากันจากต่างประเทศ ทำให้ต้องสูญเสียเงินตราออกนอกประเทศ และเป็นเรื่องที่ยุ่งยากมาก เมื่อต้องการแก้ไข หรือเพิ่มเติมความสามารถในการทำงานให้กับโปรแกรมบางส่วน ซึ่งนั่นอาจหมายความถึงการซื้อหาเพื่อนำมาใช้งานอีก

โปรแกรมที่พัฒนาขึ้นมานี้สามารถช่วยให้การออกแบบวงจรจรพิมพ์ ทำได้ทั้งชนิดแบบหน้าเดียว สองหน้า และชนิดติดผิว (Surface Mounted) นอกจากนี้ยัง สามารถกำหนดระยะห่างระหว่างขาอุปกรณ์ได้อย่างอิสระ เหมาะสำหรับอุปกรณ์บางชนิด ซึ่งมีระยะห่างระหว่างขาไม่ได้มาตรฐานทั่วไป อีกทั้งยังได้เพิ่มเติมคำสั่งพิเศษอื่น ๆ ขึ้นมา เพื่อช่วยให้

การออกแบบเส้นลายวงจรมีความสะดวก รวดเร็วขึ้น และ ให้ผลลัพธ์การพิมพ์ลายเส้นต่อพื้นที่สูง ทำให้มีความอิสระในการเดินลายเส้นวงจรมากขึ้น ส่วนการแสดงผลทางจอภาพนั้นให้ความเหมาะสมกับผู้ใช้งานมาก ส่วนหนึ่งที่สำคัญก็คือการลดปัญหาขนาดของภาพทางด้านกว้างและด้านยาวให้มีสัดส่วนตามความเป็นจริง ทำให้เกิดความรู้สึก และ ความเข้าใจต่อภาพดีขึ้น อีกทั้งให้ผลลัพธ์ภาพที่อัตราการขยายภาพสูงสุดสามารถทำการตรวจเช็คระยะห่างของเส้น และอุปกรณ์ต่าง ๆ ได้ใกล้เคียงกับผลลัพธ์การพิมพ์ ทำให้เกิดความสะดวกต่อการตัดสินใจก่อนพิมพ์ลายวงจรพิมพ์

การเปรียบเทียบความสามารถในการทำงานของโปรแกรมนี้ กับ โปรแกรมที่มีใช้กันแพร่หลายอยู่ในท้องตลาด ดังเช่นโปรแกรม smARTWORK และ PROTEL มีรายละเอียดสำคัญต่าง ๆ ดังนี้

(วิทยานิพนธ์นี้ขอใช้ชื่อโปรแกรมเป็น PCBCAD ในการเปรียบเทียบ)

ข้อแตกต่าง	PCBCAD	PROTEL	smARTWORK[5]
จำนวนโดนัท	12	21	4
ขนาดและชนิดโดนัท	ผู้ใช้กำหนดได้	แน่นอน	แน่นอน
ขนาดเส้น	4 ขนาด	4 ขนาด	2 ขนาด
เส้นเฉียง	อิสระทุกมุม	45 องศา	45 องศา
เส้นแสดงแนวการลาก	3 แบบ	1 แบบ	ไม่มี
ขนาดการขยายภาพ	6 ขนาด	4 ขนาด	2 ขนาด
ความหนาแน่นเส้น/พื้นที่	สูง	สูง	ต่ำ
จำนวนอุปกรณ์ในไลบรารี่	แล้วแต่จะสร้าง	แล้วแต่จะสร้าง	ไม่ได้
เคลื่อนย้ายเส้น	สูงสุด 40 เส้น	ไม่ได้	ไม่ได้

ข้อจำกัดของโปรแกรม smARTWORK และ PROTEL ในการสร้างแผ่นวงจรพิมพ์ที่ใช้กับอุปกร์ชนิดติดผิวก็เนื่องมาจากขนาดของโดนัทใหญ่กว่าขาของอุปกรณ์มาก (อุปกรณ์ชนิดนี้มีจำนวนขาอุปกรณ์มากและระยะระหว่างขาใกล้กัน ถ้าใช้โดนัทขนาดใหญ่จะทำให้ขาแต่ละขานั้นช็อต (short) กันได้) รวมทั้งความหนาแน่นของลายเส้นวงจรต่อหนึ่งหน่วยพื้นที่ต้องมี

ค่าสูง เนื่องจากขาแต่ละขาของอุปกรณ์ชนิดนี้อยู่ใกล้กันมาก ทำให้โปรแกรม smARTWORK ไม่สามารถทำได้ด้วย เหตุที่การสร้างลายวงจร เป็นแบบแพ็ท เทิร์น (Pattern) หรือใช้ กราฟฟิกฟอนท์ (Graphic Font) มาต่อ เรียงกัน เป็นลายวงจรทำให้ เกิดข้อจำกัดที่ขนาด ของแพ็ท เทิร์น

รายละ เอียดบางอย่างของข้อ เปรียบ เทียบจะ เห็นได้ชัด เจนต่อ เมื่อได้มีการนำมาใช้ งาน แต่อย่างไรก็ตามการพัฒนาโปรแกรมขึ้นมาใช้ เองนั้น เสมือน เป็นการก้าวของการพัฒนา ในขั้นต่อไป

สำหรับอุปกรณ์ที่ใช้ในวิทยานิพนธ์ฉบับนี้ประกอบด้วย เครื่องไมโครคอมพิว เตอร์ IBM PC/XT รวมทั้ง แผ่นวงจรและจอแสดงผลรายละ เอียดภาพสูง (EGA Card and Monitor) เครื่องพิมพ์แบบ เอกซ์วายพล็อท เตอร์ (X-Y Plotter)

ในบทที่ 2 กล่าวถึงส่วนประกอบและคำสั่งที่ใช้ในการสร้างแผ่นวงจรพิมพ์

ในบทที่ 3 ยกตัวอย่างการใช้งาน

ในบทที่ 4 อธิบายถึงการทำงานโปรแกรม

ในบทที่ 5 แสดงให้ เห็นถึงผลลัพธ์การพิมพ์ต่าง ๆ

ในบทที่ 6 สรุปผลการวิจัย

ภาคผนวก 1. เป็นรายละ เอียดการจัดวางข้อมูลของแผ่นวงจรพิมพ์

ภาคผนวก 2. เป็นรายละ เอียดการจัดวางข้อมูลอุปกรณ์

ภาคผนวก 3. รายละ เอียดการกำหนดขนาดและชนิดของโดนัท

ภาคผนวก 4. รายละ เอียดของโปรแกรมทั้งหมด

บทที่ 2
ส่วนประกอบและคำสั่งที่ใช้ในการสร้างแผ่นวงจรพิมพ์

แผ่นวงจรพิมพ์เป็นส่วนสำคัญของอุปกรณ์อิเลคทรอนิคส์ต่าง ๆ มีอยู่ด้วยกันหลายแบบ เช่น แบบมีทองแดงด้านเดียว สองด้าน และหลาย ๆ ด้านซึ่งซ้อนกันอยู่ ในแต่ละแบบนี้มีกรรมวิธีการสร้างที่แตกต่าง ๆ กันออกไป อีกทั้งชนิดขนาด รูปร่าง ของลายเส้น รวมทั้งระยะห่างขาของอุปกรณ์มีความแตกต่างกันออกไปตามการใช้งาน และการเปลี่ยนแปลงทางเทคโนโลยี อย่างเช่นอุปกรณ์ชนิดติดผิว และอุปกรณ์จำพวกคอนเน็คเตอร์ (Connector) ความแตกต่างกันเหล่านี้เป็นปัญหาสำหรับการสร้างแผ่นวงจรพิมพ์ ซึ่งขึ้นอยู่กับโปรแกรมที่จะนำมาใช้ช่วยในการออกแบบ แต่อย่างไรก็ตามส่วนประกอบที่สำคัญยังคงเหมือนเดิม

2.1 ส่วนประกอบที่สำคัญของแผ่นวงจรพิมพ์ สามารถแบ่งออกได้เป็นดังนี้

2.1.1 แผ่นลายวงจรทองแดง ทำหน้าที่เชื่อมต่อสัญญาณไฟฟ้าระหว่างชิ้นส่วนอิเลคทรอนิคส์ต่าง ๆ ซึ่งโดยทั่ว ๆ ไปการสร้างแผ่นลายวงจรทองแดงนั้นจำเป็นอย่างยิ่งที่จะต้องมีแบบอาร์ทเวิร์ค (ArtWork) [12] ก่อนที่จะทำกรรมวิธีทางเคมี และเจาะรูแผ่นวงจรพิมพ์ ด้วยเหตุนี้การสร้างแบบอาร์ทเวิร์คจึงเป็นจุดเริ่มต้นของการสร้างแผ่นวงจรพิมพ์ ซึ่งอาจประกอบด้วยอาร์ทเวิร์คที่ใช้สร้างแผ่นวงจรพิมพ์เพียงด้านเดียว (Single Side) สองด้าน (Double Side) หรือหลาย ๆ ด้าน (Multi - Layer) ก็ได้ขึ้นอยู่กับความจำเป็นและความต้องการของผู้ออกแบบใช้งาน โดยทั่ว ๆ ไปแล้วถ้าการใช้งานแผ่นวงจรพิมพ์นั้นมีจำนวนไม่มากการสร้างแผ่นวงจรแบบหลาย ๆ ด้านจะมีราคาแพงกว่ามากด้วยเหตุที่กรรมวิธีการสร้างที่มีความสลับซับซ้อน และต้องการความเที่ยงตรงสูง ดังนั้นการสร้างแบบด้านเดียว หรือสองด้านจึงมีความนิยม และใช้งานกว้างขวางในวงการอิเลคทรอนิกส์ขนาดเล็ก ด้วยเหตุนี้เอง การออกแบบสร้างอาร์ทเวิร์คของโปรแกรมในวิทยานิพนธ์ฉบับนี้จึงกำหนดการใช้งานไว้เพียงสองด้านจะเลือกใช้ด้านเดียว หรือสองด้านก็ได้ โดยสุดท้ายแล้วผลลัพธ์ที่ได้จะประกอบด้วยแบบอาร์ทเวิร์คแบบต่าง ๆ ดังนี้

- ด้านบัดกรี (Solder Side) เป็นด้านที่ใช้สำหรับบัดกรีขาของอุปกรณ์ต่าง ๆ ให้ติดกับแผ่นวงจรพิมพ์
- ด้านอุปกรณ์ (Component Side) เป็นด้านที่ใช้วางอุปกรณ์ต่าง ๆ ซึ่งอยู่ตรงข้ามด้านบัดกรี

2.1.2 ส่วนสกรีนแสดงตำแหน่งอุปกรณ์ (Component Screen) ทำหน้าที่แสดงตำแหน่งที่อยู่ของอุปกรณ์ต่าง ๆ เพื่อลดความผิดพลาดในการจัดวาง ได้มาจากการนำเอาแบบอาร์ทเวิร์คที่เป็นภาพของอุปกรณ์ต่าง ๆ ในตำแหน่งที่ถูกจัดวางไว้บนแผ่นวงจรพิมพ์มาสร้างเป็นแบบพิมพ์ผ้าไหม (Silk Screen) เพื่อพิมพ์สีที่ใช้ในการพิมพ์ (โดยทั่ว ๆ ไปแล้วใช้สีขาว) พิมพ์ลงบนแผ่นลายวงจรทองแดง

2.1.3 ส่วนสกรีนเคลือบลายเส้นทองแดง (Solder Resist Screen) ทำหน้าที่ป้องกันความสกปรกของลายเส้นทองแดง และช่วยให้การเกาะตัวของตะกั่วบัดกรีติดเฉพาะที่ตำแหน่งขาของอุปกรณ์เท่านั้นทำให้ไม่สิ้นเปลืองตะกั่วบัดกรีไปในส่วนที่ไม่จำเป็น อีกทั้งยังมีความสำคัญในการทำขั้นตอนทางเคมีในส่วนของการสร้างเพลททลูโฮล (Plate through hole) [13] ส่วนนี้สามารถสร้างได้ด้วยวิธีการเช่นเดียวกันกับการสร้างแบบพิมพ์ผ้าไหมดังในส่วนของการสกรีนแสดงตำแหน่งอุปกรณ์ซึ่งแตกต่างกันที่แบบของอาร์ทเวิร์คโดยลักษณะของอาร์ทเวิร์คเป็นขาของอุปกรณ์ หรือที่เรียกกันว่าโดนัท (Donut) [12]

2.1.4 ส่วนประกอบของลายวงจรที่ใช้สร้างแผ่นวงจรพิมพ์ โดยส่วนใหญ่แล้วประกอบด้วยอุปกรณ์หรือชิ้นส่วนต่าง ๆ ที่นำมาเชื่อมต่อกันเพื่อให้วงจรสามารถทำงานได้ มีส่วนประกอบที่สำคัญในการสร้างอาร์ทเวิร์คดังนี้

2.1.4.1 ลายเส้นวงจร ทำหน้าที่เชื่อมต่อสัญญาณทางไฟฟ้าจากอุปกรณ์หนึ่งไปยังอีกอุปกรณ์หนึ่ง ประกอบด้วยเส้นขนาดต่าง ๆ หลายขนาดให้ผู้ใช้งานสามารถเลือกใช้งานได้ตามความเหมาะสม ดังเช่น 12 16 20 และ50มิล ในแต่ละขนาดของการใช้งานขนาดลายเส้นนั้น มีความเกี่ยวข้องกับระยะทางในการเดินลายวงจร ความหนาแน่นของลายเส้น ความต้านทานที่เกิดขึ้นในลายวงจร ซึ่งอาจเกี่ยวข้องกับค่าพารามิเตอร์ (Parameter) ของการทำงานภายในวงจรของอุปกรณ์แต่ละอุปกรณ์ที่นำมาใช้งาน เช่นอุปกรณ์บางตัวต้องการสายสัญญานในการต่อเชื่อมระหว่างอุปกรณ์สิ้นที่สุด มีความต้านทานในลายวงจรน้อย โดยจะเป็นผลให้ผลการทำงานของวงจรมีเสถียรภาพมากที่สุด ซึ่งส่วนนี้เป็นข้อคิดของวิศวกรในการพิจารณาการออกแบบลายวงจร เลือกลายเส้นระยะห่างของแต่ละลายเส้นที่จะนำมาใช้งาน เพื่อปรับปรุงเสถียรภาพการทำงาน [12]

2.1.4.2 ขาของอุปกรณ์ หรือที่เรียกว่าโดนัท โดยมีหน้าที่โดยตรงในการเชื่อมต่อขาของอุปกรณ์กับลายเส้นวงจร แต่ก็ยังสามารถนำมาใช้ในการเดิน

ลายเส้นวงจรแบบสองหน้าสำหรับการเดินลายเส้นต่อเนื่องระหว่างด้าน รูปแบบของโดนัทในการนำมาใช้งานนั้นมีด้วยกันหลายแบบ โดยทั่วไปจะเป็นแบบวงกลม แบบสี่เหลี่ยม และแบบแถบ (เหมาะสำหรับการใช้เป็นจุดต่อสาย (Connector tongue) แบบใช้แผ่นวงจรพิมพ์เสียบต่อกับคอนเน็คเตอร์โดยตรง อุปกรณ์ชนิดติดผิว และอื่นๆ) ในแต่ละแบบของโดนัทมีขนาดต่าง ๆ ตามความเหมาะตามขนาดของขาอุปกรณ์ และลายเส้นที่ใช้งานร่วม

จากหัวข้อข้างบนที่กล่าวมานั้นเป็นส่วนประกอบที่สำคัญสำหรับผู้ใช้งานทั่วๆ ไป แต่สำหรับทางด้านอุตสาหกรรมแล้วสิ่งต่าง ๆ เหล่านี้ยังไม่เพียงพอกับการสร้างแผ่นวงจรพิมพ์สิ่งหนึ่งที่สำคัญ และน่าจะมีความจำเป็นต่อขั้นตอนการผลิตก็คือข้อมูลคำสั่งที่ใช้ควบคุมการทำงานให้กับเครื่องจักรอัตโนมัติ โดยการต่อเชื่อมเข้ากับเครื่องไมโครคอมพิวเตอร์ทำให้การสร้างแผ่นวงจรพิมพ์มีความสะดวก และรวดเร็วในการผลิต อีกทั้งยังสามารถลดข้อผิดพลาดในกระบวนการเจาะรูได้ สำหรับผู้ที่มีความสนใจในหัวข้อนี้สามารถศึกษาได้จากหนังสือการประชุมทางวิชาการวิศวกรรมไฟฟ้าครั้งที่ 10 เล่มที่ 1 ในหัวข้อการประยุกต์ใช้ไมโครคอมพิวเตอร์กับการออกแบบแผ่นวงจรพิมพ์ ตั้งแต่หน้าที่ 1-117 ถึง 1-128

2.2 แนวทางในการสร้างโปรแกรมช่วยในการสร้างแผ่นวงจรพิมพ์

แผ่นวงจรพิมพ์โดยทั่ว ๆ ไป ประกอบลายเส้น และโดนัทหรือขาอุปกรณ์ ดังนั้นการสร้างแผ่นวงจรพิมพ์โดยพื้นฐานนั่นก็คือการควบคุมตัวชี้ในการแสดงผลบนจอภาพแบบกราฟฟิกให้เคลื่อนที่ไปยังตำแหน่งที่ต้องการจะใส่โดนัทและลากเส้นเชื่อมต่อจนเป็นลายวงจร แต่ในการสร้างแบบพื้นฐานนี้ทำการใช้งานไม่ค่อยสะดวก ดังนั้นจึงมีการเพิ่มเติมฟังก์ชั่นการใช้งานขึ้นตามความเหมาะสมโดยแบ่งส่วนพิจารณาจากลายวงจรโดยทั่วไปดังนี้

2.2.1 การสร้างลายวงจร พิจารณาโดยส่วนใหญ่แล้วแผ่นวงจรพิมพ์มีความต้องการในขนาดของลายเส้นที่แตกต่างกันตามความต้องการทางด้านกระแสไฟฟ้า ของอุปกรณ์ ดังนั้นจึงควรมีขนาดของลายวงจรให้เลือก โดยทั่ว ๆ ไปแล้วมีขนาด 12 16 ,20 ,50 มิล [5] (ข้อมูลที่ได้จาก โปรแกรม smARTWORK และ Protel) และเพื่อให้การลากลายเส้นทำได้สะดวก สิ่งหนึ่งก็คือเส้นแนวการลากควรที่จะทำได้หลาย ๆ แบบ เพื่อทำให้การใช้งานมีความสะดวกมากขึ้น.

2.2.2 ระยะการเคลื่อนที่ตัวชี้ เป็นสิ่งหนึ่งที่ช่วยให้เกิดความสะดวกในการใช้งานนั่นก็คือสามารถเคลื่อนตัวชี้ได้รวดเร็วขึ้นในกรณีที่ต้องการสร้างลายวงจรในตำแหน่ง

ที่อยู่ไกลจากตำแหน่งที่ตัวชี้อยู่ในขณะนั้น และสามารถสร้างลายวงจรให้ลายเส้นมีความหนาแน่นต่อหนึ่งหน่วยพื้นที่สูง เมื่อระยะการเคลื่อนที่ของตัวชี้ทำได้ในระยะใกล้ โดยทั่วไปแล้วระยะการเคลื่อนที่ต่ำสุดเท่ากับ 1 มิล

2.2.3 การเปลี่ยนตำแหน่งภาพของแผ่นวงจรพิมพ์ โดยทั่ว ๆ ไปขนาดของแผ่นวงจรพิมพ์มีขนาดไม่ค่อยแน่นอนขึ้นอยู่กับความต้องการและจำนวนของอุปกรณ์บนแผ่นวงจรพิมพ์ ดังนั้นถ้าขนาดของแผ่นวงจรมีขนาดใหญ่แต่การแสดงผลบนจอภาพทำได้จำกัด จึงทำให้การสร้างลายวงจรทำได้ไม่สะดวก และเกิดความผิดพลาดขึ้นได้ง่ายถ้าไม่มีการปรับขนาดภาพบนจอ ดังนั้นการที่จะทำให้การสร้างแผ่นวงจรพิมพ์มีความสะดวกและสมบูรณ์ขึ้นจึงควรที่จะมีการขยายภาพในอัตราการขยายขนาดต่าง ๆ ได้ตามความเหมาะสมโดยสามารถแสดงรายละเอียดต่าง ๆ ได้เพียงพอ ซึ่งก็เป็นเหตุให้การแสดงผลกระทำได้ไม่ทั่วทั้งแผ่นวงจรพิมพ์ทำให้ภาพบางส่วนไม่สามารถแสดงออกมาได้ ดังนั้นเพื่อให้การสร้างแผ่นวงจรพิมพ์สามารถทำได้ทั่วทุกจุดจึงจำเป็นที่จะต้องมีการเลื่อนภาพเพื่อปรับให้ภาพในตำแหน่งที่ต้องการสร้างหรือแก้ไขได้มีการแสดงผล

2.2.4 การใช้งานอุปกรณ์ที่มีรูปร่างและขนาดแน่นอน อุปกรณ์ที่ใช้ในการสร้างแผ่นวงจรพิมพ์โดยทั่ว ๆ ไปมีรูปแบบที่ค่อนข้างจะแน่นนอนดังนั้นการสร้างอุปกรณ์ชนิดนี้เก็บไว้ในแผ่นดิสค์หรือหน่วยความจำ และนำมาใช้งานได้ใหม่เมื่อต้องการจะทำให้การสร้างแผ่นวงจรทำได้สะดวกกว่าการสร้างอุปกรณ์นั้นทีละตัว ด้วยสาเหตุนี้เองการนำอุปกรณ์มาใส่ลงบนแผ่นวงจรพิมพ์จึงจำเป็นต้องมีการควบคุมทิศทางการจัดวางการเคลื่อนย้าย เพื่อให้ได้อุปกรณ์ในตำแหน่งที่ต้องการ

2.2.5 การพิมพ์แบบอาร์ทเวิร์คหรือลายวงจร เดิมทีลายวงจรที่ผู้ใช้สร้างขึ้นนั้นเป็นเพียงข้อมูลทางคอมพิวเตอร์ยังไม่สามารถนำไปสร้างแผ่นวงจรพิมพ์ได้ ด้วยเหตุนี้จึงต้องมีการพิมพ์ผลลัพธ์ที่ผู้ใช้ต้องการออกมายังเครื่องพิมพ์ และให้ผู้ใช้สามารถเลือกแบบที่ผู้ใช้สร้างขึ้นพิมพ์ตามความต้องการในลักษณะต่าง ๆ เพื่อการนำไปสร้างเป็นส่วนประกอบในการสร้างแผ่นวงจรพิมพ์ (เครื่องพิมพ์ที่ใช้พิมพ์แบบอาร์ทเวิร์คมีด้วยกันหลายแบบ เช่น แบบดอทแม็ททริก (Dot-matrix printer) และแบบเอ็กซ์วายพล็อทเตอร์ (X-Y plotter) ในที่นี้จะพิจารณาการพิมพ์ผลลัพธ์ออกทางเครื่องพิมพ์แบบเอ็กซ์วายพล็อทเตอร์ เนื่องจากว่าเครื่องพิมพ์แบบดอทแม็ททริกนั้นมีโครงสร้างซึ่งถูกออกแบบมาสำหรับใช้ในการพิมพ์ข้อความหรือภาพแบบแพล็ทเทริน เท่านั้นดังนั้นการใช้งานด้านการสร้างลายเส้นจึงไม่สามารถทำได้ละเอียดเพียงพอเท่ากับเครื่องพิมพ์แบบเอ็กซ์วายพล็อทเตอร์ซึ่งถูกออกแบบมาสำหรับงานด้าน CAD,CAM โดยตรง)

คำสั่งและการใช้งาน

การสร้างแผ่นวงจรพิมพ์ด้วยเครื่องไมโครคอมพิวเตอร์โดยทั่วไปนั้นการแสดงผลภาพแบบกราฟฟิก และ การสั่งงานควบคุมจากแป้นพิมพ์มีความสำคัญ เสมือนเป็นหัวใจของการทำงานในการติดต่อกันระหว่างผู้ใช้กับเครื่อง ดังนั้นจึงจำเป็นอย่างที่จะต้องทราบถึงการใช้งาน เพื่อให้การควบคุมแป้นพิมพ์ และการตอบรับการทำงานของเครื่องไมโครคอมพิวเตอร์เป็นไปได้อย่างถูกต้อง

2.2 การใช้แป้นพิมพ์ควบคุมการทำงาน สามารถแบ่งการทำงานของแป้นพิมพ์ได้ดังนี้

2.2.1 แป้นพิมพ์อ่านค่าจากรายการคำสั่ง ทำหน้าที่รับคำสั่งจากภาพด้วยการเลื่อนตัวชี้ ไปยังข้อความแสดงคำสั่ง ประกอบด้วยแป้นพิมพ์ที่ใช้ควบคุมการทำงานดังนี้

- แป้นพิมพ์ <- ทำหน้าที่เลื่อนตัวชี้คำสั่งไปทางด้านซ้าย
- แป้นพิมพ์ -> ทำหน้าที่เลื่อนตัวชี้คำสั่งไปทางด้านขวา
- แป้นพิมพ์ Return ทำหน้าที่รับคำสั่งในตำแหน่งตัวชี้

2.2.2 แป้นพิมพ์คำสั่ง สามารถแบ่งได้เป็น 2 ประเภท

- แป้นพิมพ์แบบอักษร เช่น ตัวหนังสือ ตัวเลข และสัญญาลักษณ์ต่าง ๆ
- แป้นพิมพ์แบบฟังชั่น ประกอบด้วย F1 ถึง F10

2.2.3 แป้นพิมพ์ควบคุมทิศทาง และตำแหน่งตัวชี้ แบ่งได้เป็น 4 ทิศดังนี้

- แป้นพิมพ์ <- ทำหน้าที่เลื่อนตัวชี้ไปทางด้านซ้าย
- แป้นพิมพ์ -> ทำหน้าที่เลื่อนตัวชี้ไปทางด้านขวา
- แป้นพิมพ์ ทำหน้าที่เลื่อนตัวชี้ไปทางด้านบน
- แป้นพิมพ์ ทำหน้าที่เลื่อนตัวชี้ไปทางด้านล่าง

2.2.4 แป้นพิมพ์ควบคุมขนาดรัศมี และความกว้างของมุมในการสร้างส่วนโค้งของวงกลม แบ่งได้เป็นดังนี้

- แป้นพิมพ์ <- ทำหน้าที่เพิ่มค่ามุม
- แป้นพิมพ์ -> ทำหน้าที่ลดค่ามุม

2.3 คำสั่งที่ใช้ในการควบคุมการสร้างแผ่นวงจรพิมพ์ โดยใช้แป้นพิมพ์อ่านค่าจากรายการคำสั่ง สามารถแบ่งตามลักษณะการใช้งานหลัก ได้ดังนี้

2.3.1 PCBD ทำหน้าที่สร้างแผ่นวงจรพิมพ์ โดยการสร้างลายเส้น และนำอุปกรณ์ต่าง ๆ มาช่วยในการสร้าง ประกอบด้วยคำสั่งต่าง ๆ ดังต่อไปนี้

2.3.1.1 LOAD ทำหน้าที่อ่านข้อมูลแผ่นวงจรพิมพ์ จากแผ่นดิสค์ เข้าสู่เครื่องไมโครคอมพิวเตอร์

2.3.1.2 SAVE ทำหน้าที่เก็บข้อมูลแผ่นวงจรพิมพ์จากเครื่องไมโครคอมพิวเตอร์ลงสู่ดิสค์

2.3.1.3 EDIT ทำหน้าที่สร้างและแก้ไขแผ่นวงจรพิมพ์ ประกอบด้วยคำสั่งย่อยดังต่อไปนี้

2.3.1.3.1 CURW ทำหน้าที่เลือกชนิดเส้นแสดงแนวการลากซึ่งจะแสดงผลได้ก็ต่อเมื่อ มีการกำหนดจุดเริ่มต้นของการลากเส้น ดังรูปที่ 2.1 สามารถแบ่งออกได้เป็น 3 ชนิดด้วยกันดังนี้

- Angl เส้นตรงที่ลากจากจุดเริ่มต้นไปยังตำแหน่งตัวชี้
- Orth เส้นตรงที่ลากตามแนวตั้ง และ แนวนอน จากจุดเริ่มต้นถึงตำแหน่งตัวชี้
- 45Dg เส้นตรงที่ลากทำมุม 45องศาจากจุดเริ่มต้นถึงตำแหน่งตัวชี้

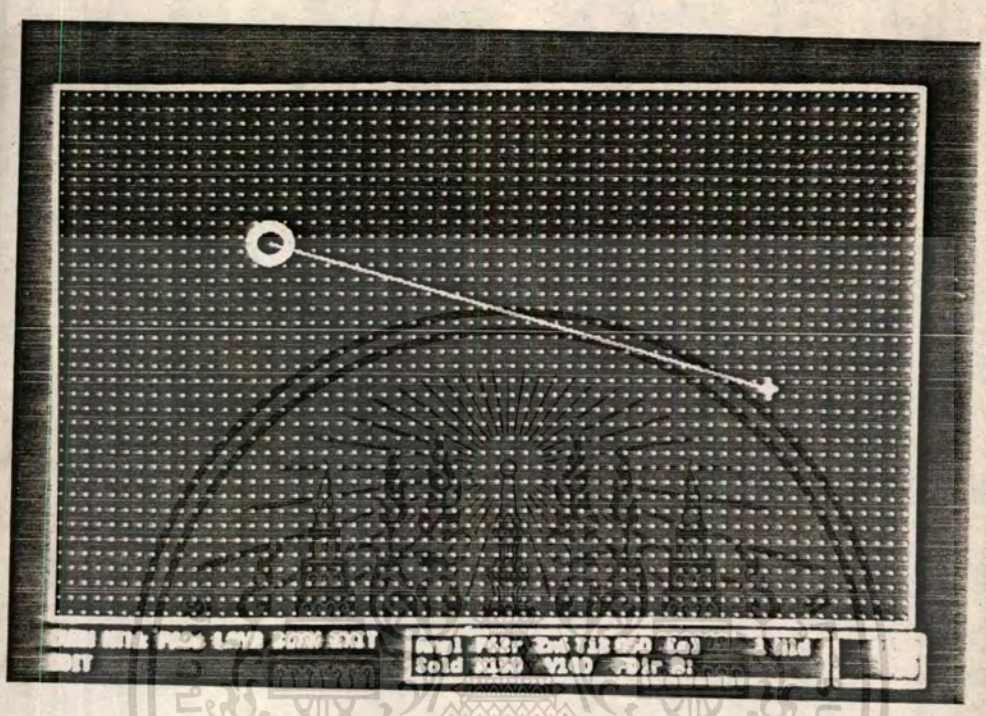
2.3.1.3.2 WThk ทำหน้าที่เลือกขนาดเส้นในการสร้างลายวงจร ดังรูปที่ 2.2 สามารถเลือกได้ 4 ขนาดดังนี้

- Th12 ขนาดความหนาของเส้นมีค่าเท่ากับ 12 มิล (mil)
- Th16 " " 16 มิล
- Th20 " " 20 มิล
- Th50 " " 50 มิล

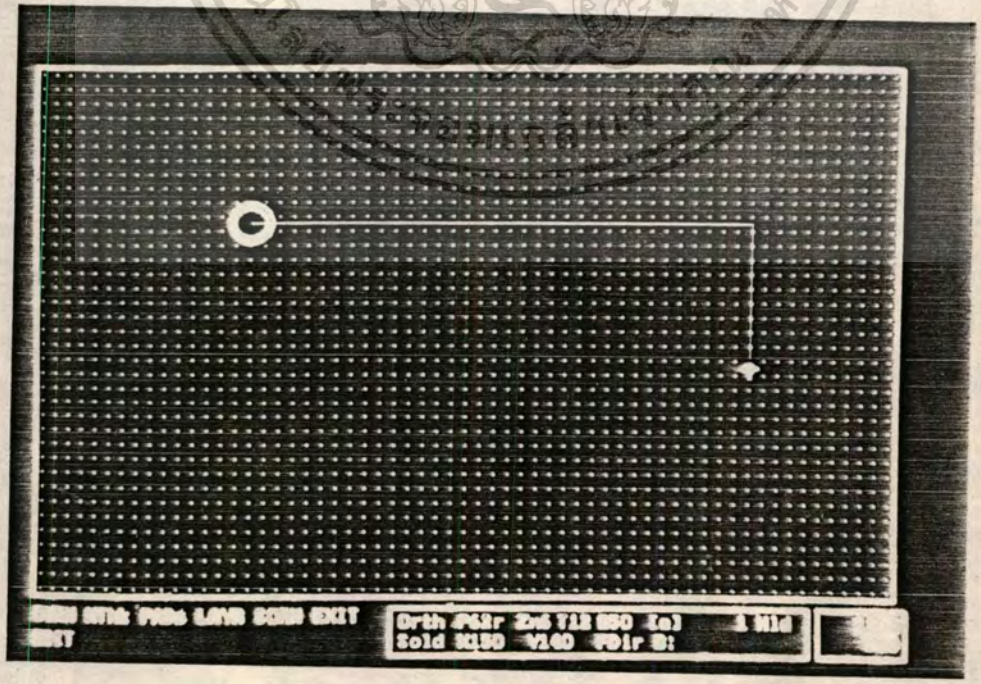
(1000 มิล เท่ากับ 1 นิ้ว)

2.3.1.3.3 PADs ทำหน้าที่เลือกขนาด และชนิดของขาของอุปกรณ์ โดยมีให้เลือกได้ 12 แบบ ดังรูปที่ 2.3 ในแต่ละแบบผู้ใช้สามารถกำหนดเองได้ แบ่งออกได้เป็น 3 ประเภทดังนี้

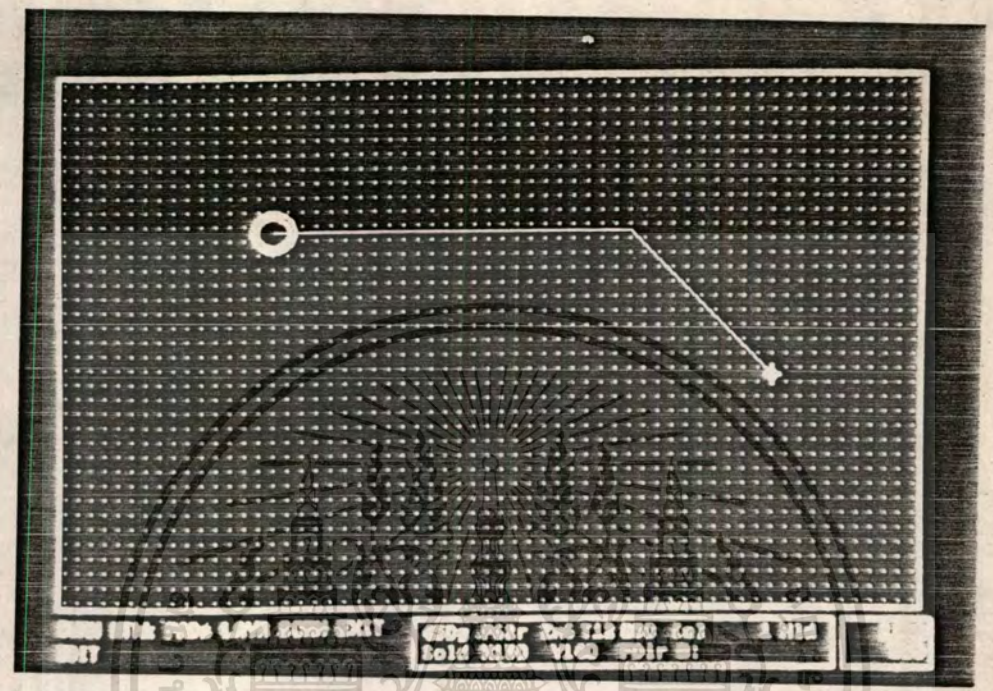
- แบบวงกลม กำหนดขนาดรูด้านในและด้านนอกได้
- แบบสี่เหลี่ยม กำหนดขนาดรูด้านในและสี่เหลี่ยมจัตุรัสด้านนอกได้



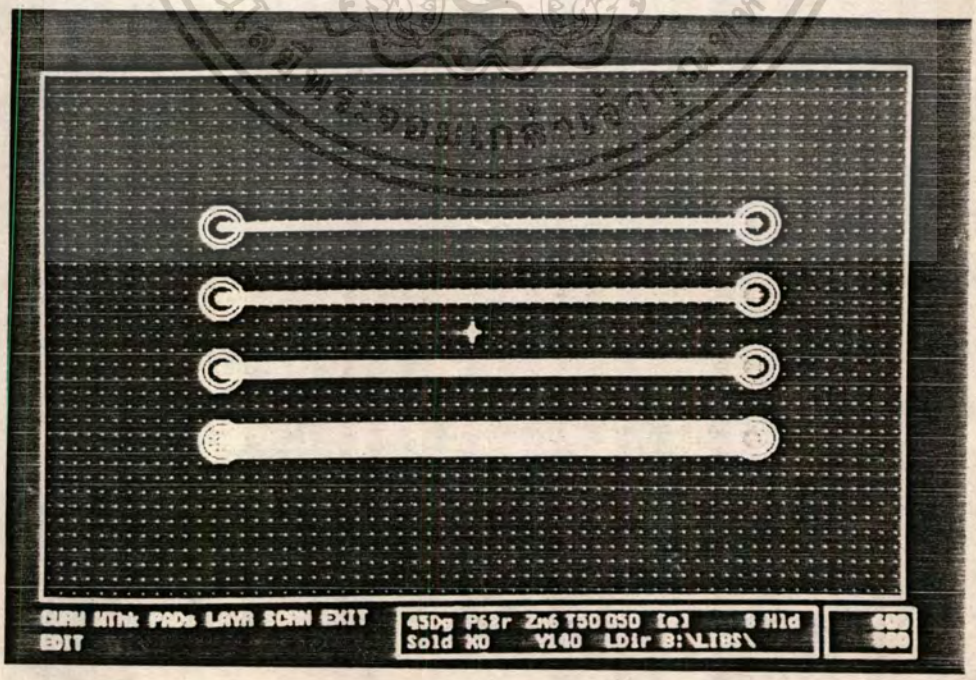
รูปที่ 2.1ก แสดงเส้นแสดงแนวการลาก แบบ Angl



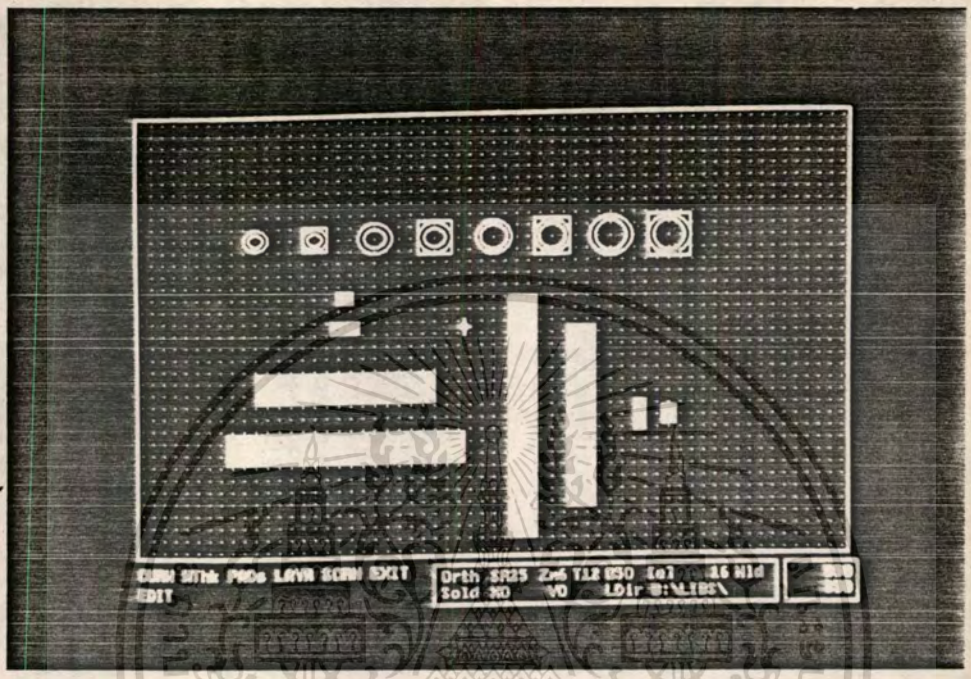
รูปที่ 2.1ข แสดงเส้นแสดงแนวการลาก แบบ Orth



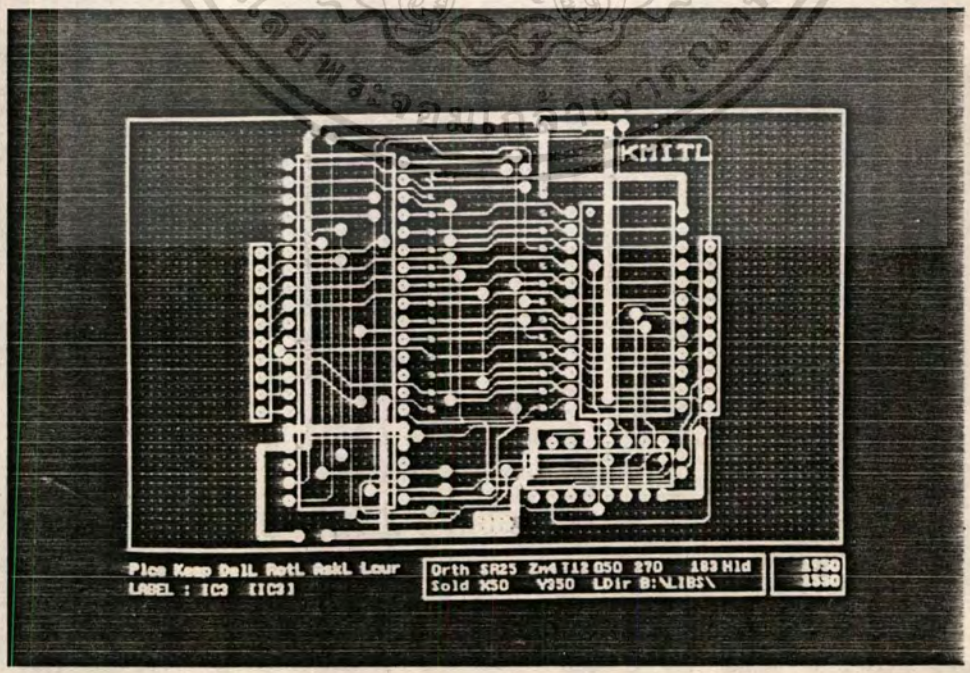
รูปที่ 2.1ค แสดงเส้นแสดงแนวการลาก แบบ 45Dg



รูปที่ 2.2 แสดงความหนาของเส้นขนาดต่าง ๆ



รูปที่ 2.3 แสดงโดนัทแบบต่าง ๆ



รูปที่ 2.4 การทำงานของคำสั่ง Move (ไอซีขนาด 28 ขา กำลังจะถูกเคลื่อนย้าย)

- แบบแถบ กำหนดขนาดด้านกว้าง และด้านยาวได้ นำไปใช้งานได้เฉพาะด้านของแผ่นวงจรพิมพ์เหมาะสำหรับขาของอุปกรณ์ชนิดติดผิว

2.3.1.3.4 **LAYR** ทำหน้าที่ เลือกด้านของแผ่นวงจรพิมพ์ เพื่อสร้างและแก้ไข มีอยู่ด้วยกัน 3 ด้านและมีหน้าที่การใช้งานดังนี้

- ด้าน **Sold** หรือ ด้านบัดกรี
- ด้าน **Comp** หรือ ด้านอุปกรณ์
- ด้าน **Silk** หรือ ด้านสกรีนแสดงตำแหน่งอุปกรณ์

2.3.1.3.5 **STEP** ทำหน้าที่ กำหนดระยะในการเคลื่อนที่ของตัวชี้ โดยมีช่วงระยะ 1, 5, 10, 25, 50 และ 100 มิล ต่อการเคลื่อนที่ 1 ครั้ง

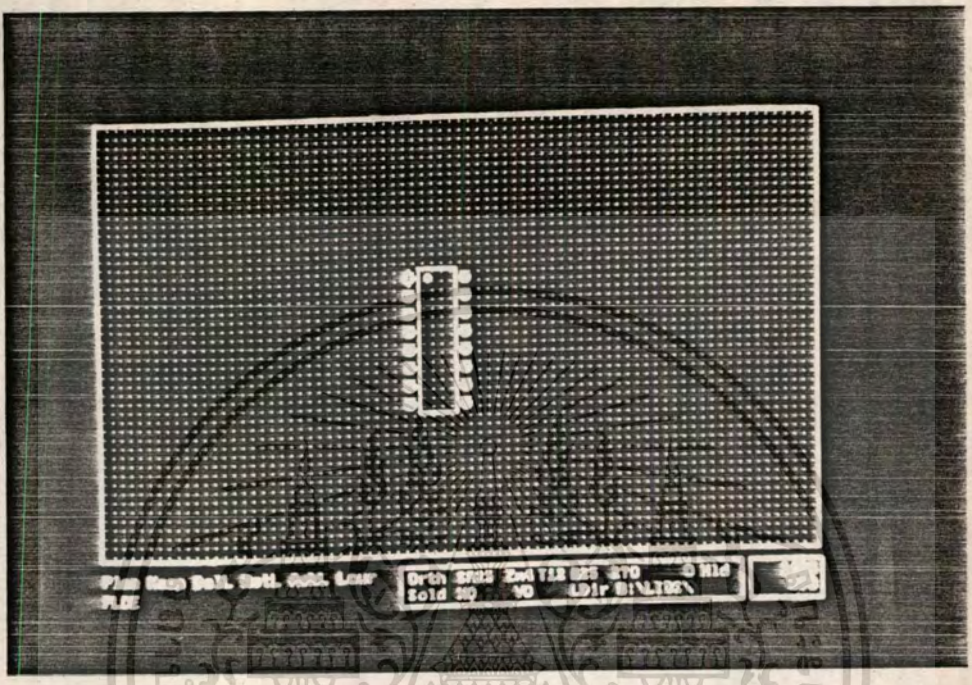
2.3.1.3.6 **EXIT** ทำหน้าที่ยกเลิกการใช้งานคำสั่ง EDIT

2.3.1.4 **LIBS** ทำหน้าที่ สร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์ ประกอบด้วยคำสั่งย่อยดังต่อไปนี้

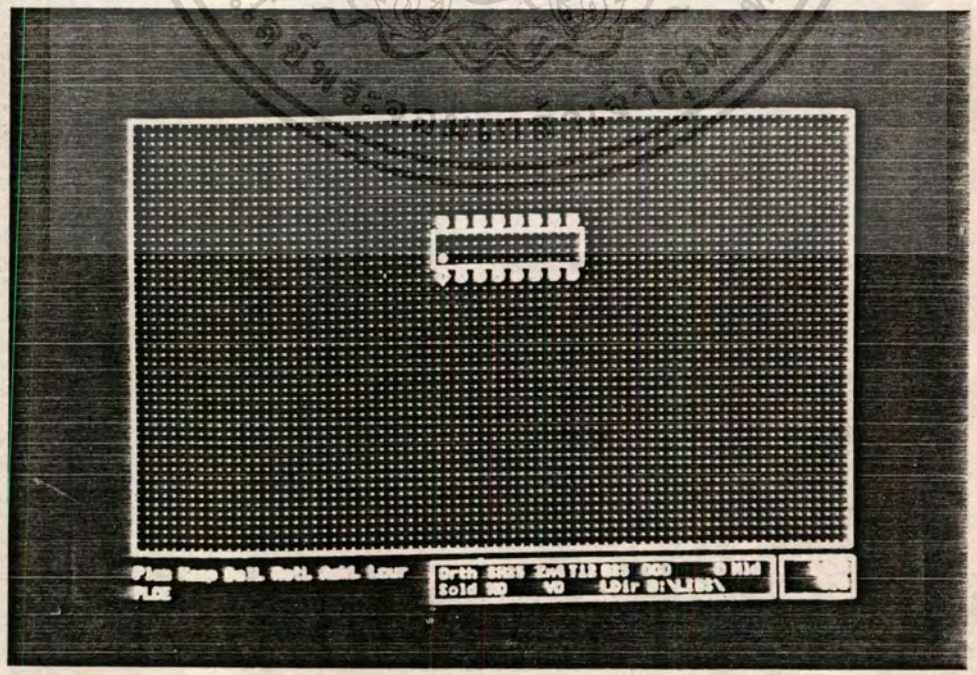
2.3.1.4.1 **PLCE** ทำหน้าที่ ควบคุมการจัดวางอุปกรณ์ ประกอบด้วยคำสั่งย่อยดังต่อไปนี้

- **Plce** ทำหน้าที่นำอุปกรณ์ที่ต้องการเคลื่อนย้ายหรือเพิ่มเติมใส่ลงบนแผ่นวงจรพิมพ์
- **Move** ทำหน้าที่ นำอุปกรณ์ที่ตรวจพบในตำแหน่งตัวชี้เคลื่อนย้ายไปในตำแหน่งที่ต้องการ ดังรูปที่ 2.4
- **DelL** ทำหน้าที่ลบอุปกรณ์ในตำแหน่งตัวชี้ออกจากแผ่นวงจรพิมพ์
- **RotL** ทำหน้าที่เปลี่ยนตำแหน่งการวางอุปกรณ์ในมุมต่าง ๆ ดังนี้ 0, 90, 180 และ 270 องศา ดังรูปที่ 2.5
- **AskL** ทำหน้าที่แสดงชื่ออุปกรณ์ ณ.ตำแหน่งตัวชี้
- **Lcur** ทำหน้าที่กำหนดค่าสภาวะของการแสดงรายละเอียดต่าง ๆ ของภาพอุปกรณ์ก่อนการนำอุปกรณ์ใส่ลงบนแผ่นวงจรพิมพ์ ดังรูปที่ 2.6

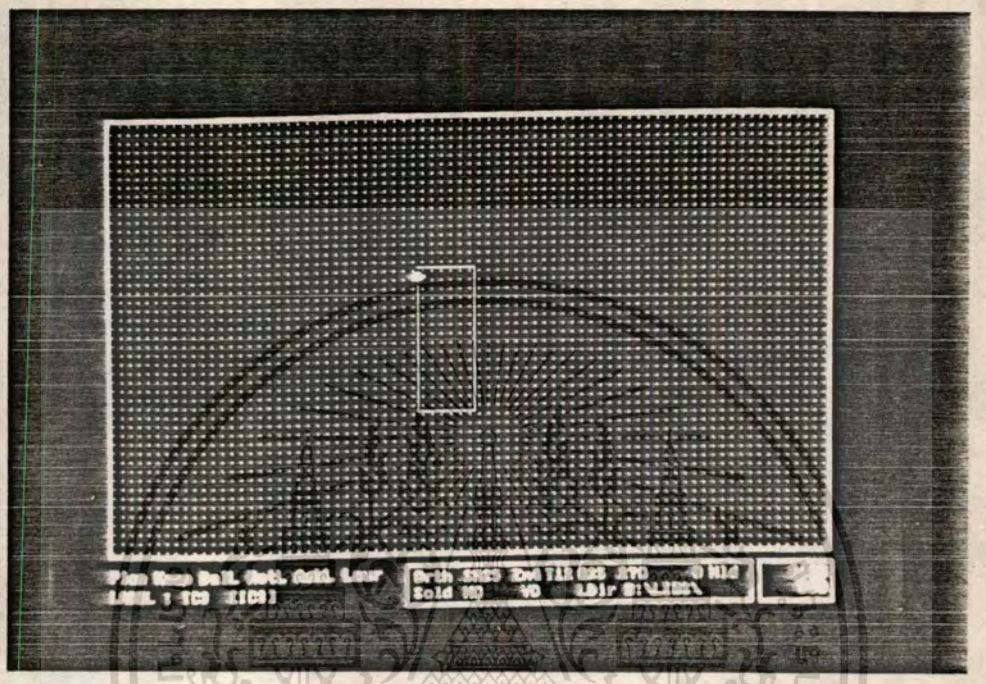
การยกเลิกใช้คำสั่ง PLCE สามารถทำได้โดยใช้แป้นพิมพ์อักษร Q, q, E หรือ e เนื่องจากการแสดงคำสั่งของเมนูมีขนาดจำกัด ดังนั้นคำสั่งนี้จึงไม่สามารถแสดงการใช้แป้นพิมพ์ได้



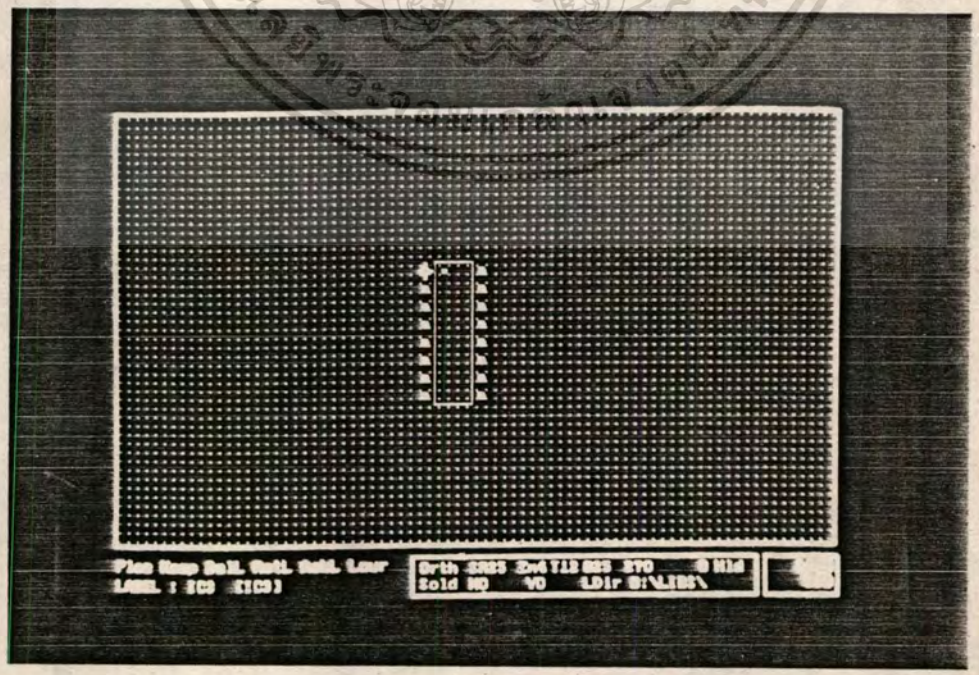
รูปที่ 2.5ก แสดงภาพอุปกรณ์ที่ต้องการหมุน



รูปที่ 2.5ข แสดงภาพอุปกรณ์ที่ถูกหมุนแล้ว



รูปที่ 2.6ก ภาพแสดงขอบเขตของอุปกรณ์



รูปที่ 2.6ข ภาพแสดงรายละเอียดอุปกรณ์ก่อนใส่ลงบนแผ่นวงจรพิมพ์

2.3.1.4.2 **LOAD** ทำหน้าที่ อ่านข้อมูลอุปกรณ์ที่ได้สร้าง เก็บไว้แล้วจากดิสค์ นำมาใช้ในการสร้างแผ่นวงจรพิมพ์

2.3.1.4.3 **LDIR** ทำหน้าที่กำหนดชื่อไดเร็คทอรี่ของไฟล์ข้อมูลอุปกรณ์ที่สร้าง เก็บไว้เป็นไลบรารี่

2.3.1.4.4 **EXIT** ทำหน้าที่ยกเลิกการใช้คำสั่ง LIBS

2.3.1.5 **CLRS** ทำหน้าที่ลบภาพบนจอภาพ และยกเลิกข้อมูลทั้งหมดของแผ่นวงจรพิมพ์ขณะนั้น

2.3.1.6 **EXIT** ทำหน้าที่ยกเลิกการใช้คำสั่ง PCBD

2.3.2 **DEVC** ทำหน้าที่สร้างอุปกรณ์ทางอิเลคทรอนิคส์ และเก็บไว้ เพื่อการนำมาใช้ในการสร้างแผ่นวงจรพิมพ์ ดังรูปที่ 2.7 ประกอบด้วยคำสั่งดังต่อไปนี้

2.3.2.1 **LDIR** ทำหน้าที่กำหนดชื่อไดเร็คทอรี่ของไฟล์ข้อมูลอุปกรณ์ที่สร้างเก็บไว้เป็นไลบรารี่

2.3.2.2 **SAVE** ทำหน้าที่นำข้อมูลอุปกรณ์ที่สร้าง เก็บลงสู่ดิสค์

2.3.2.3 **EDIT** ทำหน้าที่สร้างภาพอุปกรณ์ต่าง ๆ โดยใช้คำสั่งย่อยแบบเดียวกับคำสั่งในหัวข้อ 2.3.1.3

2.3.2.4 **CLRS** ทำหน้าที่ลบภาพบนจอภาพ และลบข้อมูลของอุปกรณ์ในหน่วยความจำ

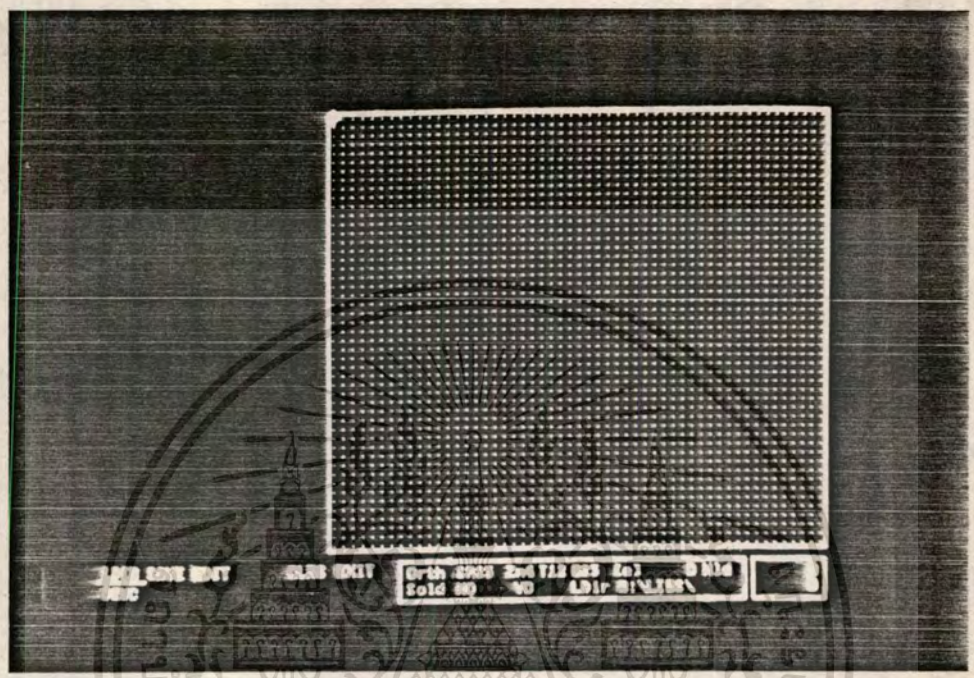
2.3.2.5 **EXIT** ทำหน้าที่ยกเลิกการใช้คำสั่ง DEVC

2.3.3 **DIRS** ทำหน้าที่กำหนดชื่อไดเร็คทอรี่ของข้อมูลแผ่นวงจรพิมพ์ เพื่อความสะดวกในการติดต่อกับดิสค์

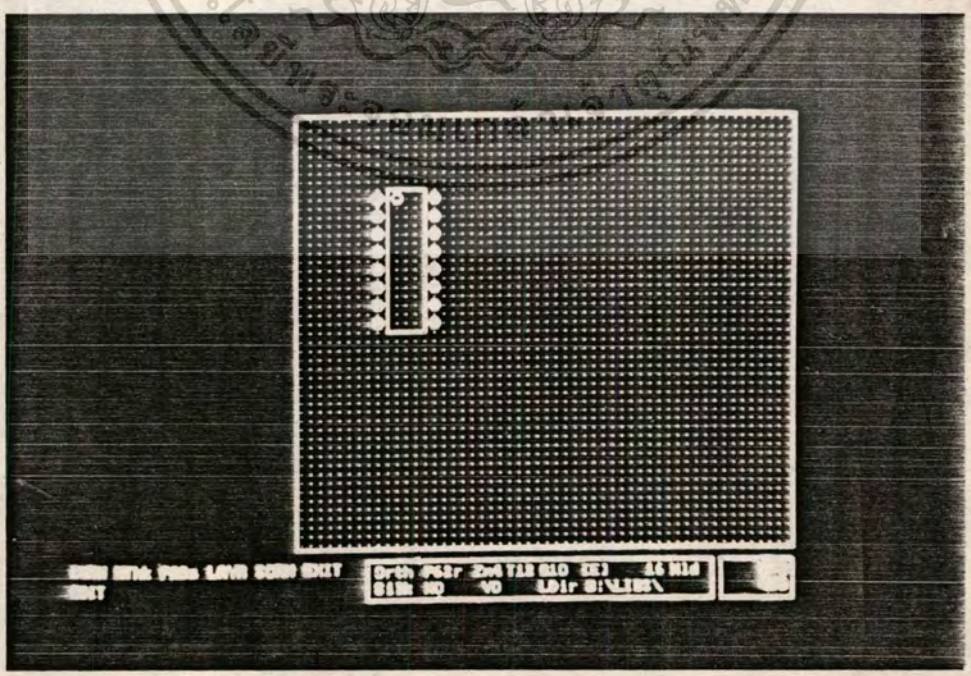
2.3.4 **PLOT** ทำหน้าที่พิมพ์ข้อมูลแผ่นวงจรพิมพ์เพื่อนำไปสร้าง สามารถกำหนดการพิมพ์ในส่วนต่าง ๆ ได้ดังรูปที่ 2.8

2.3.5 **STAT** ใช้ในการกำหนดค่าสภาวะต่าง ๆ ในการทำงาน ประกอบด้วยคำสั่งย่อยดังต่อไปนี้

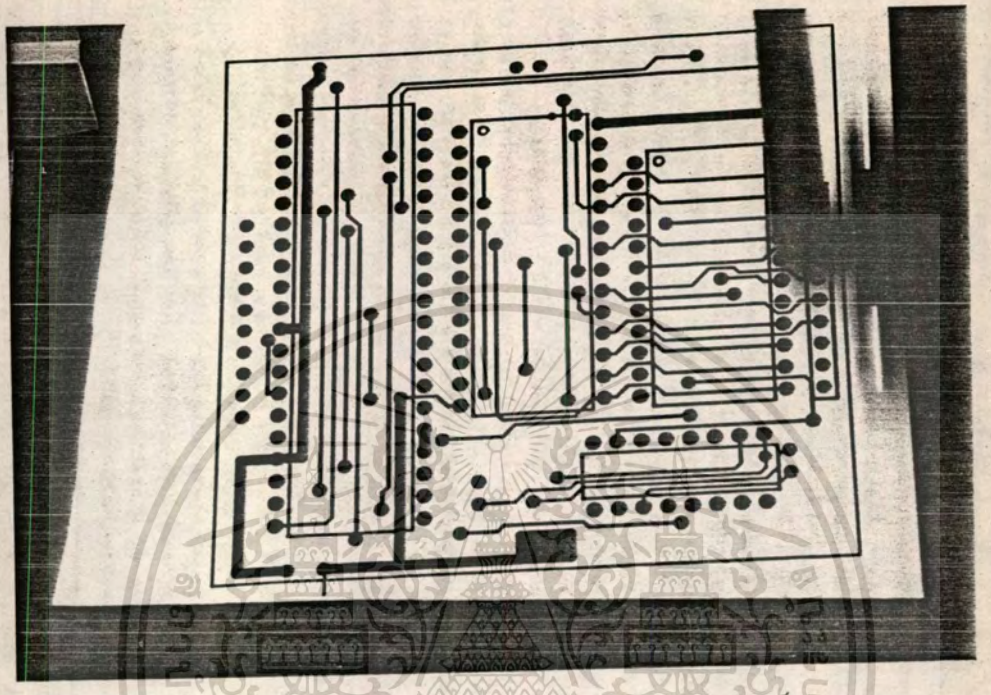
2.3.5.1 **LAYR** ทำหน้าที่ เลือกด้าน และ ข้อมูลของแผ่นวงจรพิมพ์ เพื่อกำหนดผลลัพท์การพิมพ์ และการแสดงผลทางจอภาพ ดังรูปที่ 2.8 และ 2.9



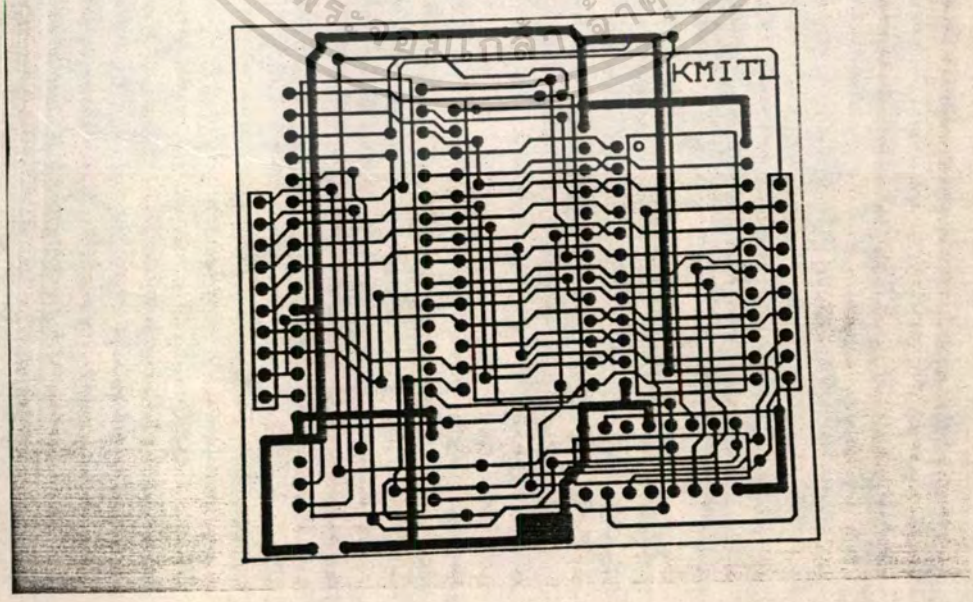
รูปที่ 2.7ก แสดงการแสดงผลบนจอภาพของคำสั่ง DEVC



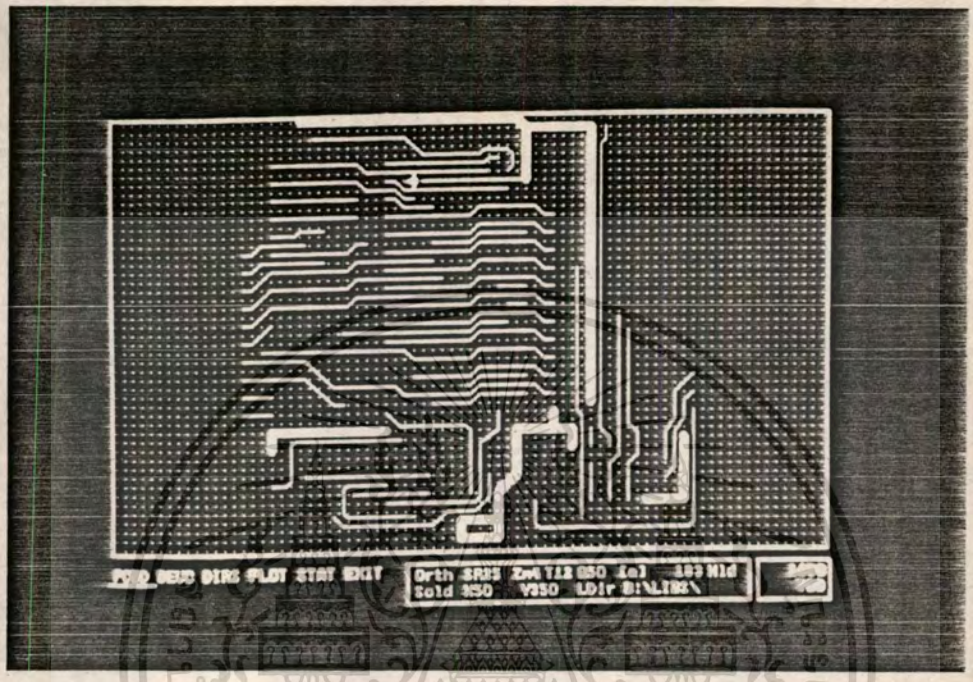
รูปที่ 2.7ข แสดงการแสดงผลบนจอภาพของคำสั่ง DEVC เมื่อมีการสร้างภาพ



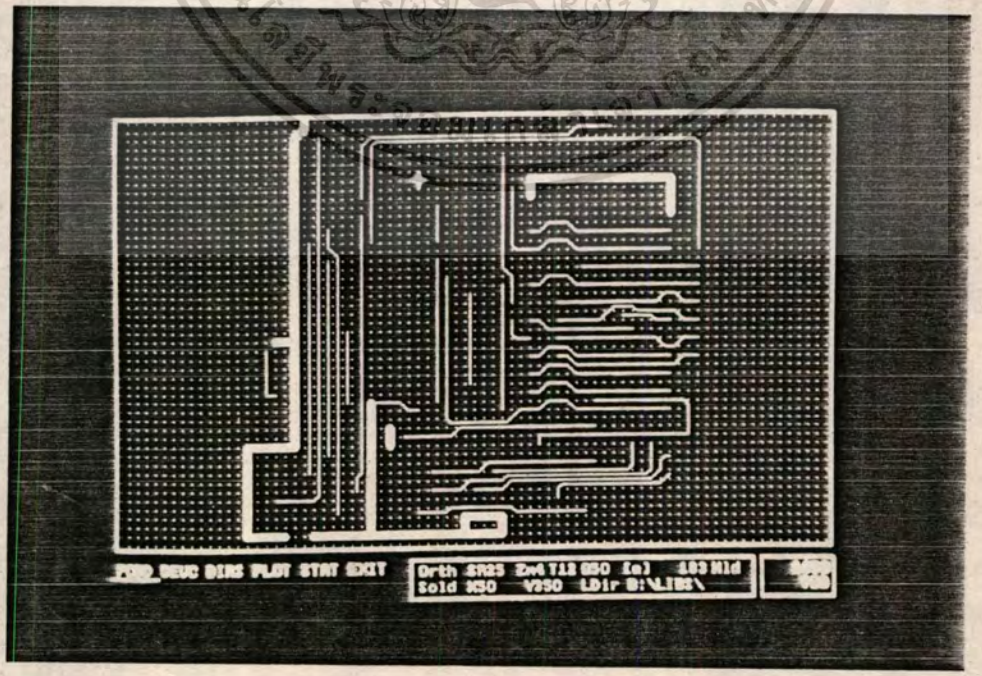
รูปที่ 2.8ก ภาพแสดงผลการพิมพ์ลาย เส้นด้านอุปกรณ์



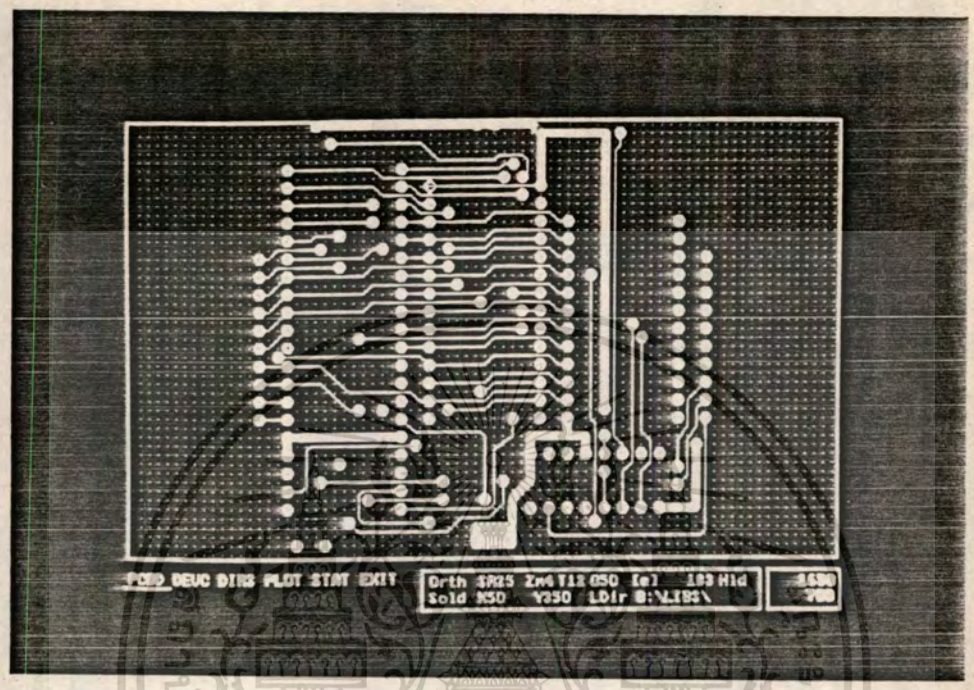
รูปที่ 2.8ข ภาพแสดงผลการพิมพ์แบบรวม



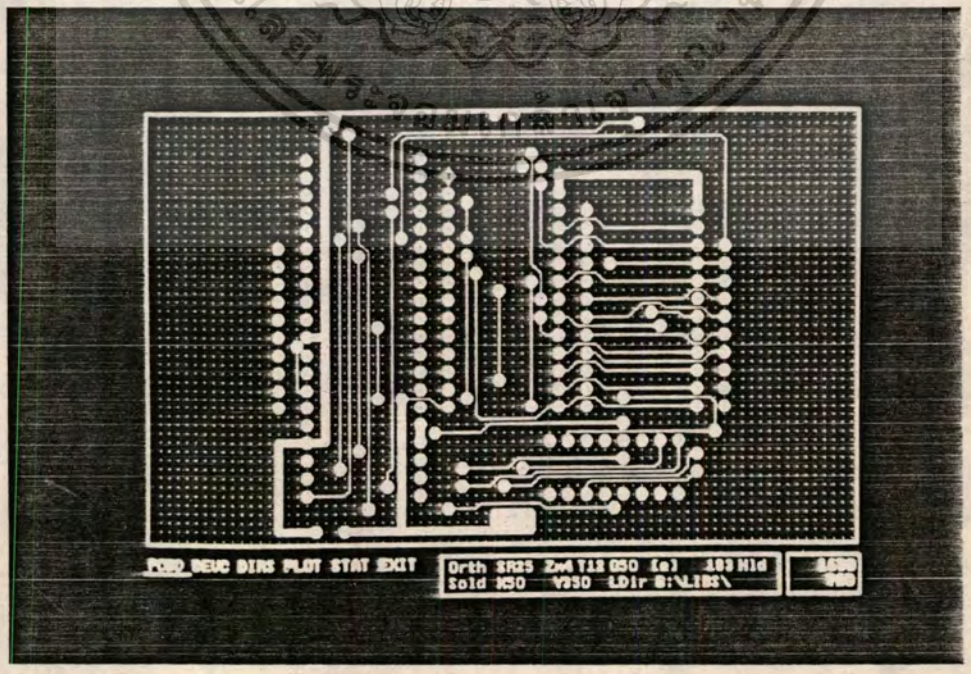
รูปที่ 2.9ก ภาพแสดงลายเส้นด้านอุปกรณ์



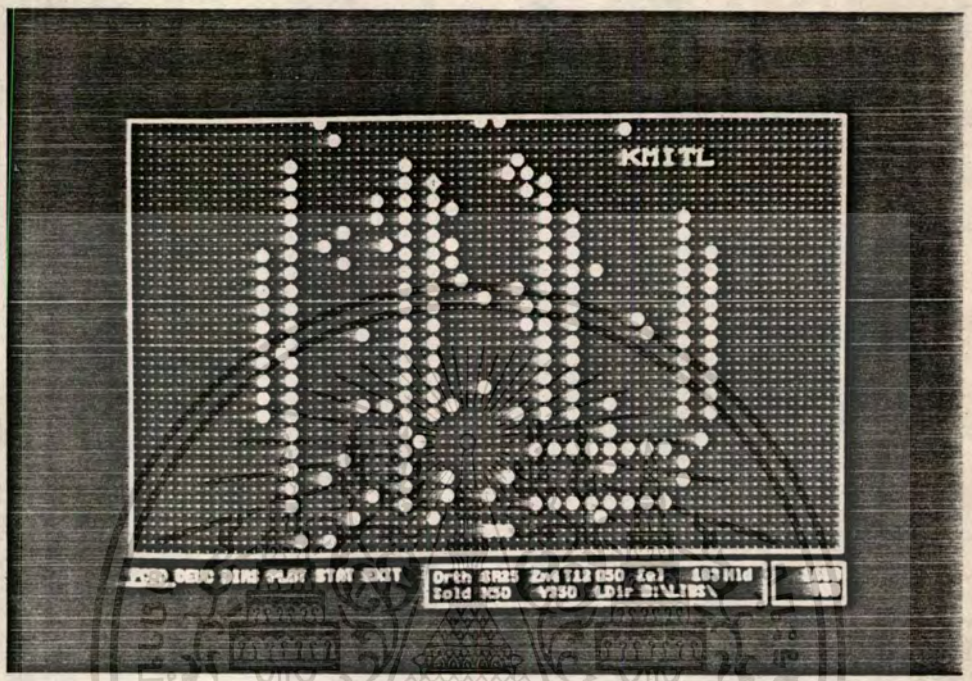
รูปที่ 2.9ข ภาพแสดงลายเส้นด้านบัดกรี



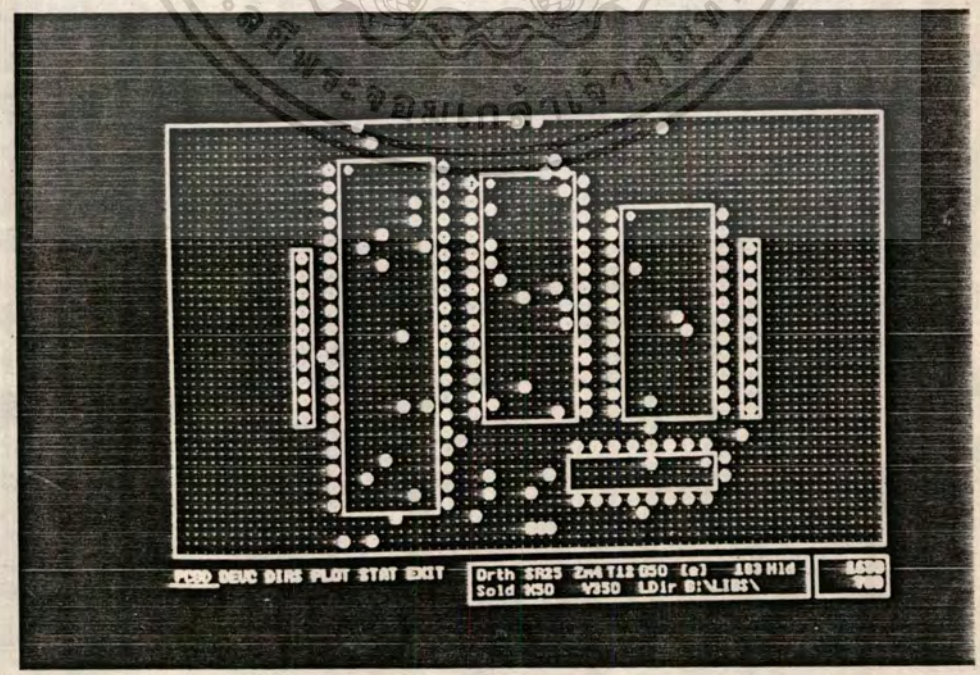
รูปที่ 2.9ค ภาพแสดงลายเส้นด้านอุปกรณ์ และโดนัท



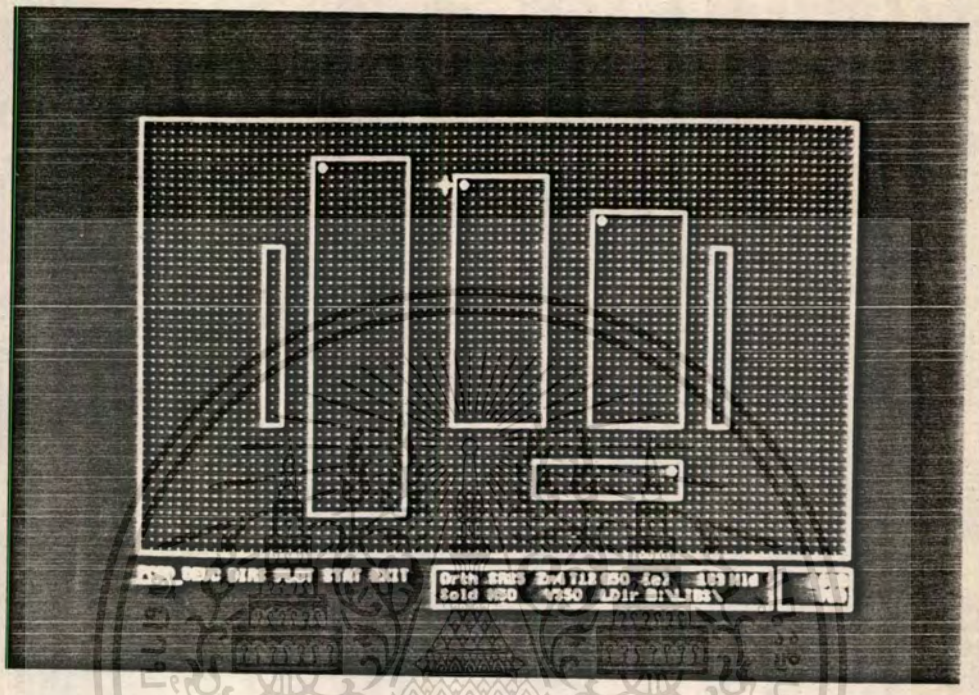
รูปที่ 2.9ง ภาพแสดงลายเส้นด้านบัดกรี และโดนัท



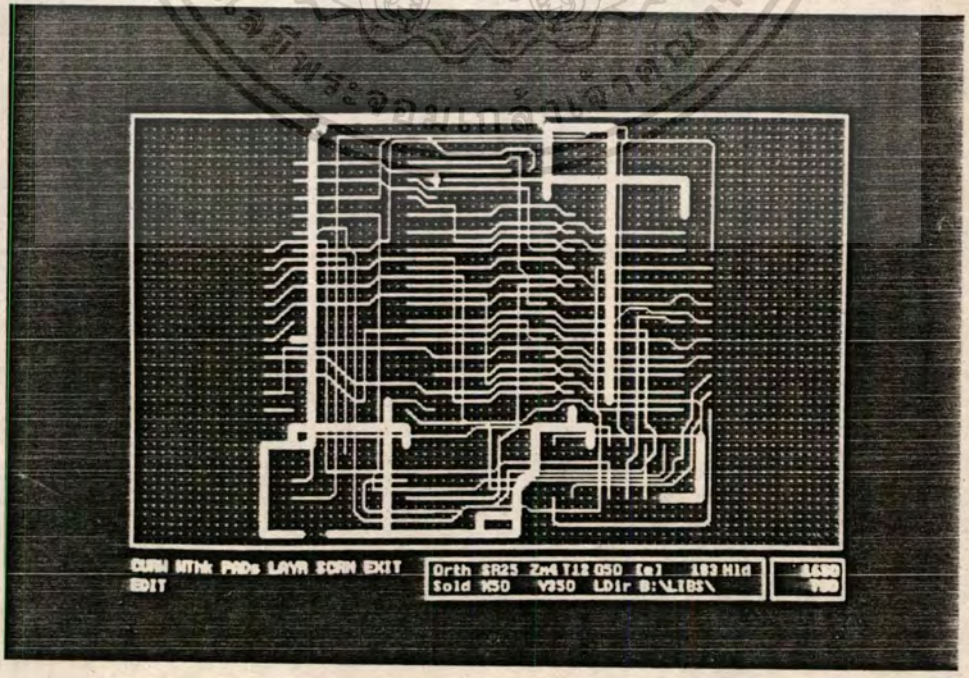
รูปที่ 2.9จ ภาพแสดงโคนัท และข้อความ



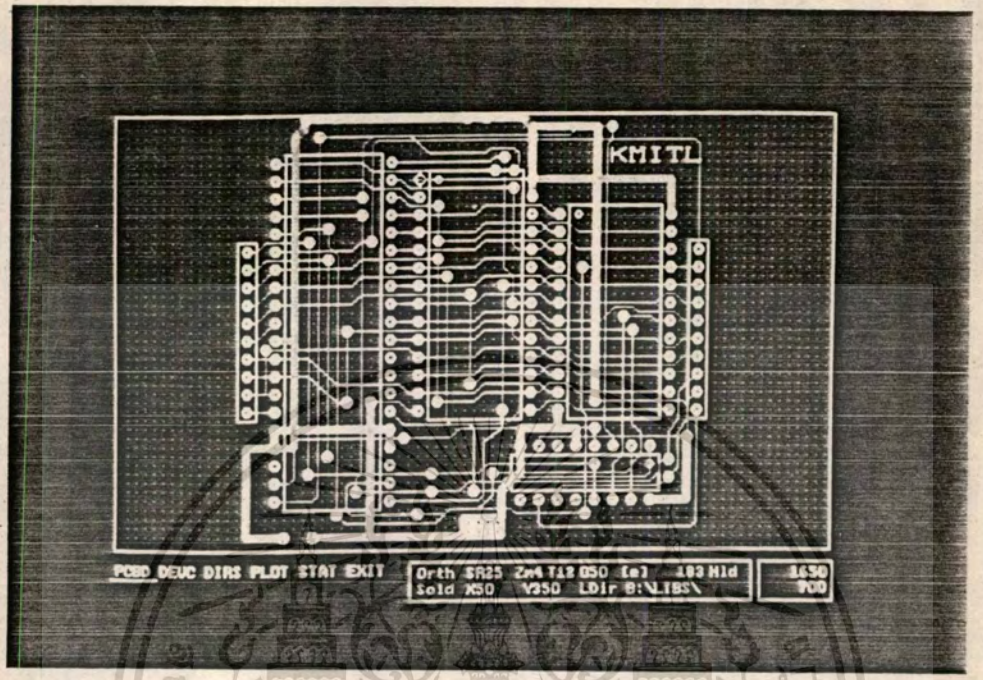
รูปที่ 2.9ฉ ภาพแสดงลายเส้นด้านสกรีน และโคนัท



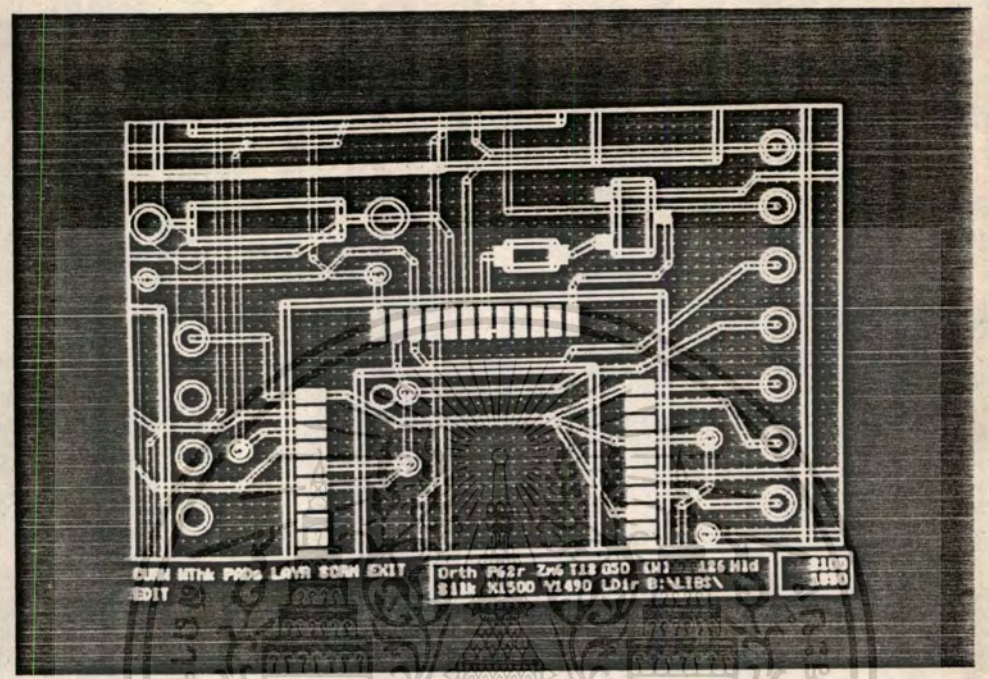
รูปที่ 2.9ช ภาพแสดงลายเส้นด้านสกรีน



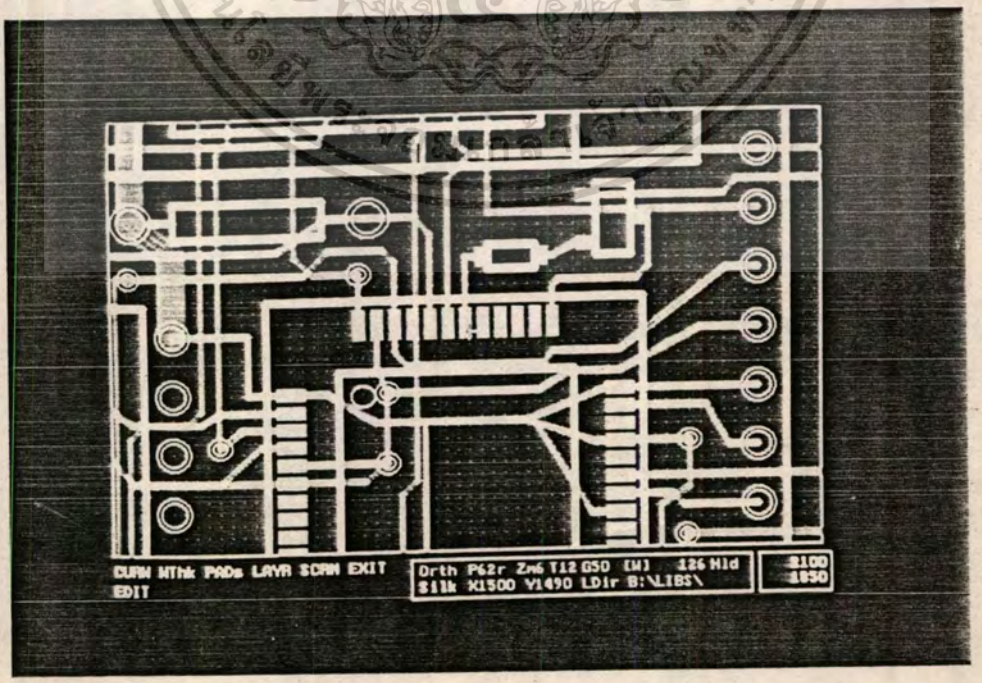
รูปที่ 2.9ช ภาพแสดงลายเส้นด้านบัดกรี และอุปกรณ์



รูปที่ 2.9ฌ ภาพแสดงลายเส้นทุกด้าน และโดนัท



รูปที่ 2.10ก ภาพแสดงรายละเอียดเส้นแบบเช็คพล็อท



รูปที่ 2.10ข ภาพแสดงรายละเอียดเส้นแบบทั่วไป

ตามลำดับ และข้อมูลต่าง ๆ ที่สามารถกำหนดได้ มีดังนี้

Sold : แผ่นลายวงจรด้านบัดกรี

Comp : แผ่นลายวงจรด้านอุปกรณ์

Silk : ภาพอุปกรณ์

Pad : ภาพโดนัท

Text : ข้อความ

Exit : แสดงผลภาพตามการกำหนด และยกเลิกคำสั่ง LAYR

2.3.5.2 **ORGC** : ทำหน้าที่เลือกการแสดงผลค่าตำแหน่งของตัวชี้ มี/ไม่มี

2.3.5.3 **ZmEn** : ทำหน้าที่เลือกการแสดงรายละเอียดเส้นของแผ่นวงจร-ดังรูปที่ 2.10 มีด้วยกัน 2 แบบคือ

- แสดงรายละเอียดเส้นแบบพอสังเขป หรือเช็คพล็อท (Check Plot) เพื่อลดเวลาการแสดงผลภาพ
- แสดงรายละเอียดเส้นแบบทั่วไป

2.3.5.4 **EXIT** : ยกเลิกการใช้คำสั่ง STAT และเข้าสู่คำสั่งหลัก

2.4 คำสั่งควบคุมการสร้างภาพ โดยใช้แป้นพิมพ์คำสั่ง สามารถแบ่งการทำงานคำสั่งได้ดังนี้

F1 : ลากเส้น (แบบต่อเนื่อง)

F2 : ยกเลิกการลากเส้น

F3 : ใส่โดนัท

F4 : ใส่โดนัท และเปลี่ยนด้านใช้งานแผ่นวงจรพิมพ์

F5 : สร้างวงกลม หรือ ส่วนโค้งวงกลม ประกอบด้วยคำสั่งต่อเนื่องดังนี้

Return ครั้งที่ 1 : กำหนดจุดศูนย์กลาง

Return ครั้งที่ 2 : กำหนดรัศมีวงกลม โดยอ่านค่าจากแป้นพิมพ์ควบคุมขนาดรัศมี

Return ครั้งที่ 3 : กำหนดความกว้างของส่วนโค้งเริ่มต้น โดยการอ่านค่าจาก แป้นพิมพ์ควบคุมความกว้างมุมของส่วนโค้ง

Return ครั้งที่ 4 : กำหนดความกว้างของส่วนโค้งปลาย โดยการ

อ่านค่าจาก แป้นพิมพ์ควบคุมความกว้างมุมของ ส่วนโค้ง

F6 : ลบโดนัท

F7 : เคลื่อนย้ายลายเส้น เพื่อแก้ไขลายเส้นของแผ่นวงจรพิมพ์ ประกอบด้วยคำสั่งต่อเนื่องดังนี้

F8 : ลบลายเส้น

F9 : แสดงภาพโดยการวาดซ้ำ

F10 : แสดงภาพโดยปรับตำแหน่งการวาดใหม่

ESC : ยกเลิกการเคลื่อนย้าย

Return : ใส่เส้นที่เคลื่อนย้ายลงในตำแหน่งตัวชี้

F8 : ลบวงกลม

F9 : แสดงภาพโดยการวาดซ้ำ

F10 : แสดงภาพโดยปรับตำแหน่งการวาดใหม่

C,c : เลือกชนิดของเส้นแสดงแนวการลาก

D,d : กำหนดทิศทางการจัดวางข้อความบนแผ่นวงจรพิมพ์

G,g : แสดงผลกริด (แสดง/ไม่แสดง)

L,l : เลือกด้านของแผ่นวงจรพิมพ์ เพื่อสร้างและแก้ไข

M,m : วาดข้อความลงบนแผ่นวงจรพิมพ์

P,p : เลือกขนาด และชนิดของโดนัท

S,s : กำหนดระยะในการเคลื่อนที่ตัวชี้

U,u : เคลื่อนย้ายข้อความ

W,w : กำหนดขนาดเส้นลายวงจร

X,x : กำหนดทิศทางการจัดวางโดนัทบนแผ่นวงจรพิมพ์

Z,z : แสดงภาพขนาดขยาย

E,e,Q,q : ยกเลิกการทำงาน

2.5 คำสั่งและโปรแกรมควบคุม

การสร้างแผ่นวงจรพิมพ์นั้นทำได้โดยอาศัยการควบคุมจากคำสั่งต่าง ๆ เพื่อสั่งงาน ซึ่งคำสั่งเหล่านี้จะถูกควบคุมด้วยโปรแกรม สามารถแบ่งได้เป็นดังนี้

PCBD โปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์

LOAD โปรแกรมอ่านข้อมูลแผ่นวงจรพิมพ์จากดิสค์

SAVE โปรแกรม เก็บข้อมูลแผ่นวงจรพิมพ์ลงสู่ดิสค์

EDIT โปรแกรมควบคุมการสร้างภาพ

F1 และ F2 โปรแกรมควบคุมการสร้างลาย เส้น

F3 โปรแกรมใส่โดนัท

F4 โปรแกรมใส่โดนัท และ เปลี่ยนด้านแผ่นวงจรพิมพ์

F5 โปรแกรมควบคุมการสร้างวงกลม

F6 โปรแกรมลบโดนัท

F7 โปรแกรมควบคุมการย้าย เส้น

F8 โปรแกรมลบวงกลม

F9 โปรแกรมแสดงภาพโดยการวาดซ้ำ

F10 โปรแกรมแสดงภาพโดยปรับตำแหน่งการวาดใหม่

C,c โปรแกรม เลือกชนิดของ เส้นแสดงแนวการลาก

D,d โปรแกรม เลือกทิศทางการจัดวางข้อความ

G,g โปรแกรมควบคุมการแสดงผลกริด

L,l โปรแกรม เลือกด้านของแผ่นวงจรพิมพ์

M,m โปรแกรมสร้างข้อความ

P,p โปรแกรม เลือกขนาด และชนิดของโดนัท

S,s โปรแกรม เลือกระยะการ เคลื่อนที่ของตัวชี้

U,u โปรแกรมย้ายข้อความ

W,w โปรแกรม เลือกขนาดของลาย เส้น

X,x โปรแกรม เลือกทิศทางการจัดวางโดนัท

Z,z โปรแกรมแสดงผลภาพขนาดขยาย

LIBS โปรแกรมสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์

PLCE โปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์

DEVC โปรแกรมควบคุมการสร้างอุปกรณ์

SAVE โปรแกรม เก็บข้อมูลอุปกรณ์ลงสู่ดิสค์

EDIT โปรแกรมการทำงาน เหมือนกับโปรแกรม ในส่วนของโปรแกรมควบคุม การสร้างแผ่นวงจรพิมพ์

PLOT โปรแกรมควบคุมการพิมพ์

STAT โปรแกรมกำหนดค่าสภาวะในการทำงาน

บทที่ 3
ตัวอย่างการใช้งาน

โปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์นี้ ประกอบด้วยคำสั่งควบคุมการทำงานดังรากคำสั่งที่ได้แสดงไว้ดังรูปที่ 3.1 และเมื่อใช้งานคำสั่ง EDIT แล้วผู้ใช้สามารถเลือกคำสั่งต่าง ๆ ได้อีกด้วยคำสั่งควบคุมจากแป้นพิมพ์ตัวอักษร ดังหัวข้อที่ 2.4 ในบทที่ 2

3.1 การเริ่มต้นในการใช้งานของโปรแกรม

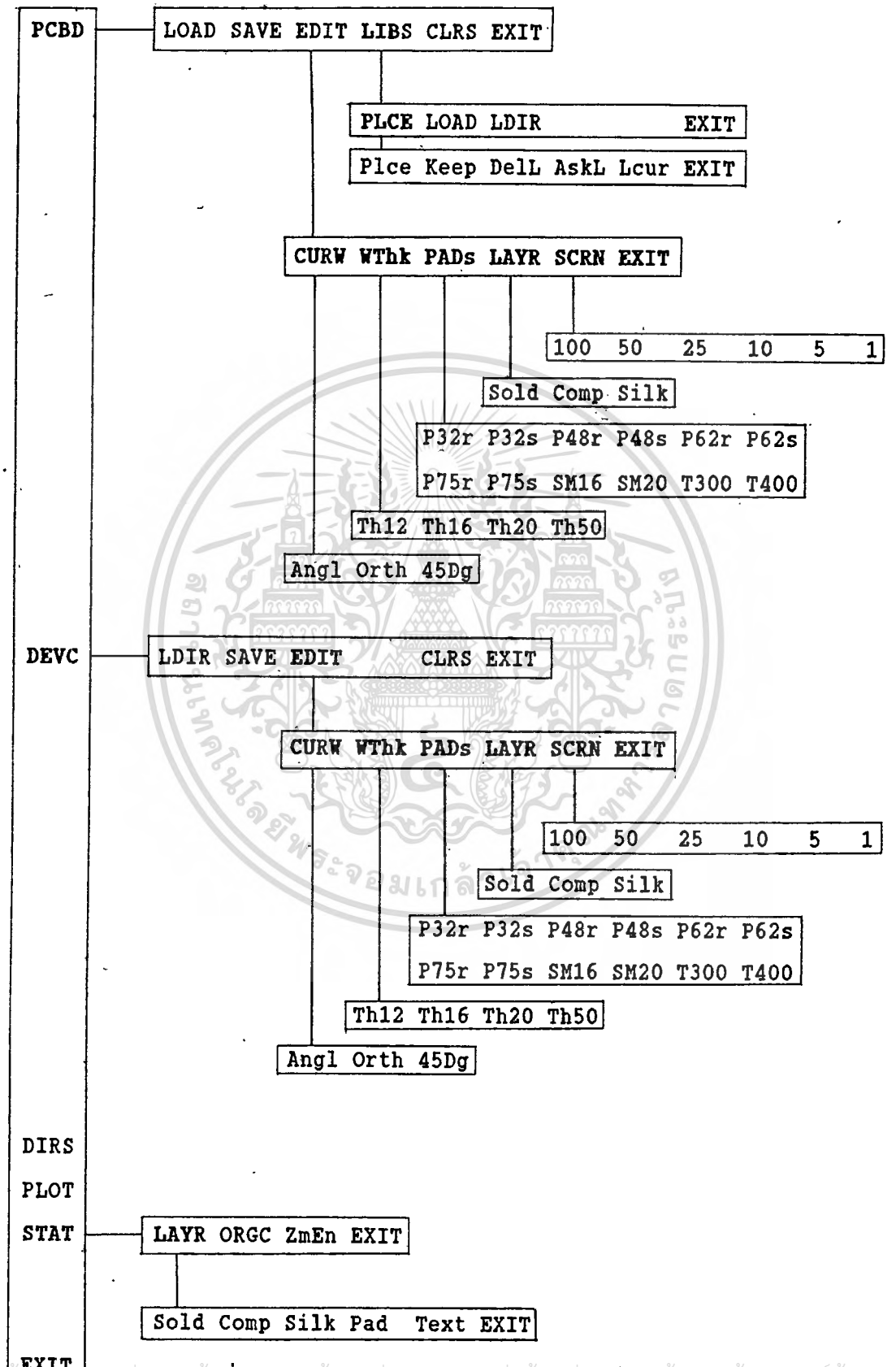
เมื่อเริ่มเปิดเครื่องไมโครคอมพิวเตอร์โปรแกรมควบคุมระบบจะทำการตรวจเช็คส่วนต่าง ๆ ของเครื่องและรอรับคำสั่งจากผู้ใช้ เมื่อผู้ใช้พิมพ์ชื่อโปรแกรมว่า PCBCAD ซึ่งเป็นการเริ่มต้นการทำงานของโปรแกรมช่วยในการออกแบบวงจรพิมพ์โดยจะแสดงข้อความและรายละเอียดต่าง ๆ ของผู้สร้าง เมื่อกดแป้นพิมพ์ใดแป้นพิมพ์หนึ่งการแสดงผลต่าง ๆ จะเปลี่ยนไปเป็นการแสดงผลเพื่อการสร้างภาพ ดังรูปที่ 3.2 และรอรับคำสั่งจากรายการคำสั่งโดยแป้นพิมพ์ควบคุมรายการคำสั่งต่อไป

3.2 การสร้างอุปกรณ์เก็บไว้ในไลบรารี

ตัวอย่างการสร้างทรานซิสเตอร์

เลือกคำสั่ง DEVC จากรายการคำสั่ง เพื่อเป็นการเข้าสู่การสร้างอุปกรณ์โดยการแสดงภาพจะเปลี่ยนไป ดังรูปที่ 2.7ก และแสดงรายการคำสั่งที่ผู้ใช้สามารถเลือกใช้ได้ เมื่อผู้ใช้เลือกคำสั่ง EDIT การทำงานจะเข้าสู่การสร้างภาพ

- เลื่อนตัวชี้ไปยังตำแหน่งที่เหมาะสม
- เปลี่ยนขนาดการขยายของภาพให้ได้ภาพขนาดโตสุด
- เลือกขนาดโดนัทโดยกดอักษร P แล้วเลือกโดนัทขนาด P62r จากรายการ
- ใส่โดนัทอันที่ 1 และเลื่อนตัวชี้ไปยังตำแหน่งต่อไป เพื่อใส่โดนัทอันที่ 2 และอันที่ 3 โดยมีระยะห่างตามขนาดที่ต้องการ ดังรูปที่ 3.3
- เปลี่ยนด้านแผ่นวงจรพิมพ์ โดยกดอักษร L แล้วเลือกด้าน Silk จากรายการ
- เลื่อนตัวชี้ไปยังตำแหน่งกึ่งกลางระหว่างโดนัททั้ง 3 และกดแป้นพิมพ์ F5 จากนั้นกดแป้นพิมพ์ Return เพื่อกำหนดจุดศูนย์กลางให้กับวงกลม ดังรูปที่ 3.4 เมื่อกำหนดเรียบร้อยแล้ว เตรียมปรับขนาดรัศมีโดยแป้นพิมพ์ควบคุมขนาดรัศมี เมื่อมี



รูปที่ 3.1 แสดงรากในการแสดงรายการคำสั่งต่าง ๆ

ขนาดเป็นที่ต้องการแล้วกดแป้นพิมพ์ Return โปรแกรมจะแสดงผล ดังรูปที่ 4.5 และจะรอรับการกำหนดมุมเริ่มต้น โดยแป้นพิมพ์ควบคุมความกว้างของมุมในการสร้างส่วนโค้ง เมื่อกดแป้นพิมพ์ Return จะได้มุมเริ่มต้น และรอรับการกำหนดค่ามุมสุดท้าย โดยแป้นพิมพ์ควบคุมความกว้างของมุมในการสร้างส่วนโค้ง ดังรูปที่ 3.6 เมื่อกดแป้นพิมพ์ Return ก็จะได้ส่วนโค้งวงกลมตามที่ต้องการดังรูปที่ 3.7

- กดแป้นพิมพ์อักษร Q เพื่อเป็นการยกเลิกการสร้างวงกลม
- กดแป้นพิมพ์ F8 เพื่อลบวงกลม หรือส่วนโค้งออก
- สร้างวงกลมใหม่โดยวิธีเดิมแต่ไม่ต้องปรับค่ามุมเริ่มต้นและมุมสุดท้ายจะได้วงกลมดังรูปที่ 3.8
- สร้างวงกลมที่ตำแหน่งโดนัททั้ง 3 โดยให้ขนาดรัศมีใหญ่กว่าโดนัท จะได้วงกลมดังรูปที่ 3.9 และสร้างวงกลมให้มีขนาดใหญ่กว่าวงกลมดังรูปที่ 3.10
- กดแป้นพิมพ์อักษร Q เพื่อยกเลิกการสร้างวงกลม และกดแป้นพิมพ์อักษร C เลือกเส้นแสดงแนวการลากแบบ 45Dg จากรายการคำสั่ง
- เลื่อนตัวชี้ไปยังขอบของวงกลมขนาดโตสุด แล้วกดแป้นพิมพ์ F1 ดังรูปที่ 3.10 และ 3.11 ก็จะได้รูปทรานซิสเตอร์ตามต้องการ
- เลื่อนตัวชี้ไปยังโดนัทอันที่ 1 เพื่อกำหนดให้เป็นจุดอ้างอิง
- กดแป้นพิมพ์อักษร Q หรือ E เพื่อยกเลิกการสร้าง
- เลือกคำสั่ง LDIR จากรายการเพื่อกำหนดไดเรคทรอรี โดยใส่ชื่อว่า B:
- เลือกคำสั่ง SAVE จากรายการเพื่อเก็บตัวทรานซิสเตอร์ โดยใส่ชื่อไฟล์ว่า TR1

ข้อมูลตัวทรานซิสเตอร์จะถูกเก็บไว้ในไดร์ฟ(Drive) B และมีชื่อไฟล์ว่า TR1.LIB สามารถนำไปใช้ในการสร้างแผ่นวงจรพิมพ์ต่อไป

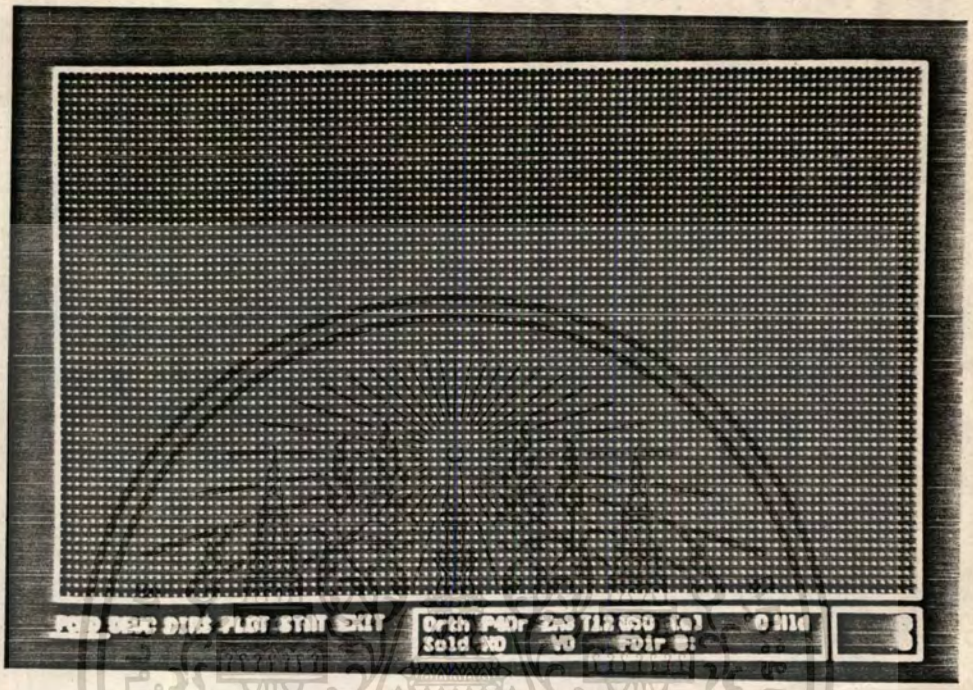
3.2 การสร้างแผ่นวงจรพิมพ์

ตัวอย่างการสร้างวงจรทรานซิสเตอร์

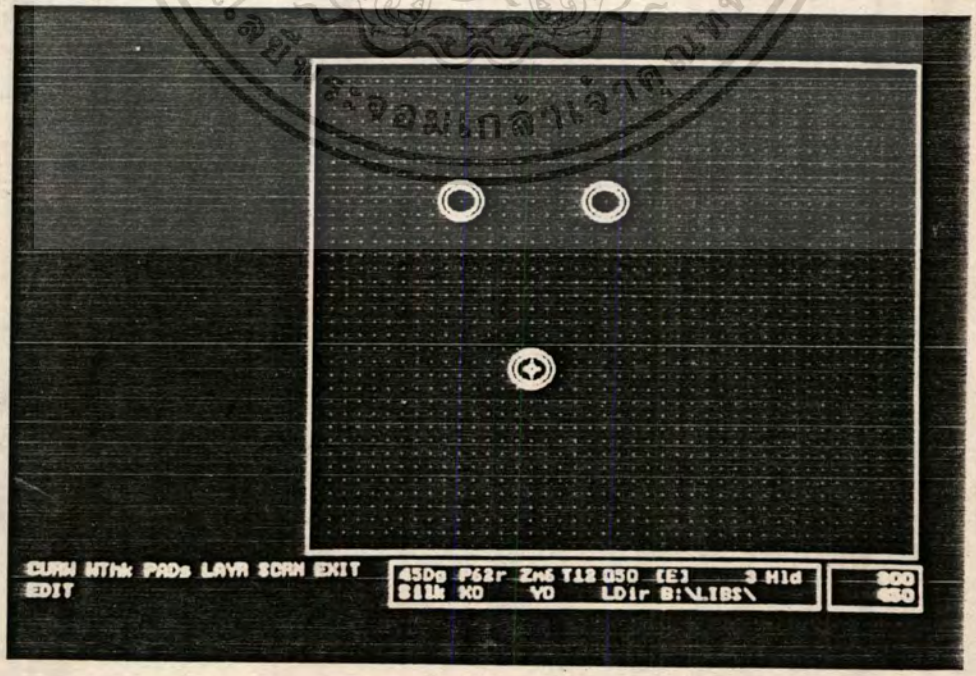
เลือกคำสั่ง PCBD จากรายการคำสั่ง เพื่อเป็นการเข้าสู่การสร้างแผ่นวงจรพิมพ์ โดยจะแสดงรายการคำสั่งที่ผู้ใช้สามารถเลือกใช้ได้ และเมื่อผู้ใช้เลือกคำสั่ง LIBS การทำงานจะเข้าสู่การนำเอาอุปกรณ์มาสร้างแผ่นวงจรพิมพ์

- เลือกคำสั่ง LDIR จากรายการเพื่อกำหนดไดเรคทรอรี ใส่ชื่อว่า B:
- เลือกคำสั่ง LOAD จากรายการแล้วใส่ชื่อไฟล์ว่า TR1 เพื่อนำข้อมูลอุปกรณ์มาสร้างแผ่นวงจรพิมพ์

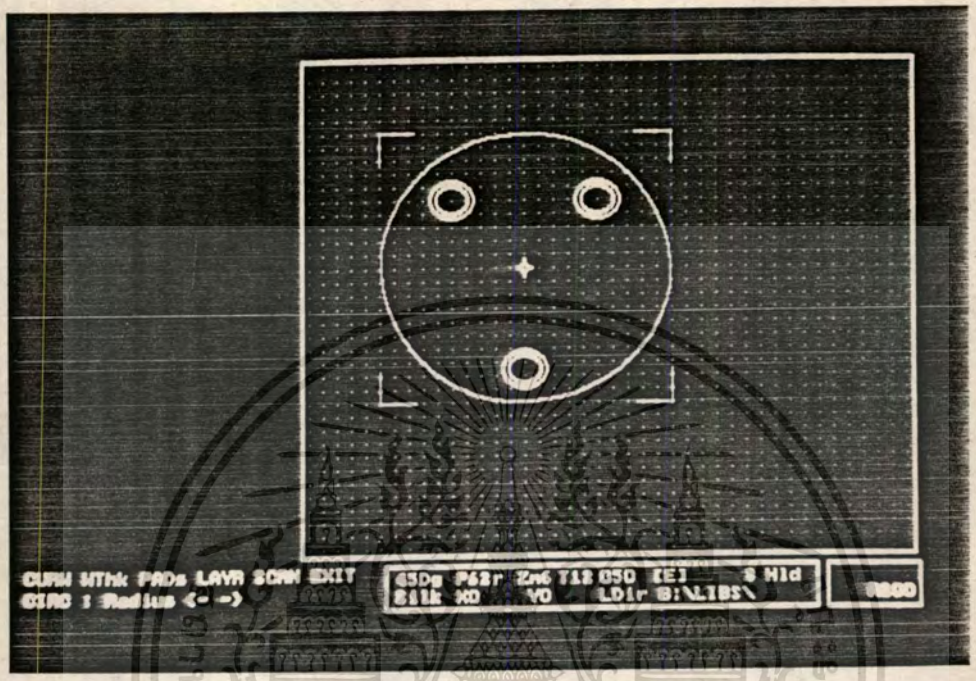
- เลือกคำสั่ง PLCE จากรายการเพื่อจัดวางอุปกรณ์ โดยจะมีการแสดงรายการคำสั่งชุดใหม่ให้เลือกใช้งาน
- เลื่อนตัวชี้ไปยังตำแหน่งที่ต้องการ ดังรูปที่ 3.12 รอเวลาสักครู่จะเกิดภาพดังรูปที่ 3.13 และกดแป้นพิมพ์ Z เลือกรายการหมายเลข 5 การแสดงผลจะเปลี่ยนไป ดังรูปที่ 3.14
- เลือกคำสั่ง Plce ใส่ชื่อหรือหมายเลขอุปกรณ์ว่า TR01 เมื่อมีการแสดงข้อความ LABEL และใส่รายละเอียดของอุปกรณ์ว่า 2SC301 เมื่อมีการแสดงข้อความ COMMENT ตัวทรานซีสเตอร์จะถูกใส่ลงในแผ่นวงจรพิมพ์ ดังรูปที่ 3.15
- เลื่อนตัวชี้ไปยังตำแหน่งในรูปที่ 3.16 เลือกคำสั่ง Plce อีกและใส่ชื่อว่า TR02 และใส่รายละเอียดของอุปกรณ์ว่า 2SC301 จะได้ตัวทรานซีสเตอร์ 2 ตัว ดังรูปที่ 3.16
- เลือกคำสั่ง Move จากรายการเพื่อเปลี่ยนตำแหน่ง หรือมุมในการจัดวาง โดยกดแป้นพิมพ์อักษร R และกดแป้นพิมพ์ [->] หรือ [<-] อุปกรณ์จะเปลี่ยนมุมในการจัดวางดังรูปที่ 3.17 3.18 และ 3.19 ตามลำดับ
- เลื่อนตัวชี้ไปทางซ้าย เพื่อให้อุปกรณ์ที่จะใส่ไม่ทับกัน ดังรูปที่ 3.20 โดยเลือกคำสั่ง Plce เพื่อใส่อุปกรณ์ จากนั้นกดแป้นพิมพ์อักษร W เลือกเส้นขนาด 50 มิล
- เลื่อนตัวชี้มายังโคนัทของอุปกรณ์ตัวแรกเพื่อลากลายเส้น โดยกดแป้นพิมพ์ F1 ดังรูปที่ 3.21 3.22 และ 3.23 ตามลำดับ จากนั้นกดแป้นพิมพ์อักษร W เปลี่ยนขนาดเส้นเป็น 20 มิล และลากเส้นต่อไปดังรูปที่ 3.23 และ3.24 จากนั้นเปลี่ยนด้านการทำงานโดยกดแป้นพิมพ์อักษร L เลือกคำสั่ง Comp จากรายการและสร้างลายเส้นต่อไปจะได้ภาพดังรูปที่ 3.25
- กดแป้นพิมพ์ F10 เลื่อนภาพไปในตำแหน่งใหม่ และกดแป้นพิมพ์อักษร D เลือกทิศทางข้อความการเขียนไปทางทิศตะวันออก หรือ [E]
- เลื่อนตัวชี้ไปยังตำแหน่งที่ตองการใส่ข้อความ จากนั้นกดแป้นพิมพ์อักษร M พิมพ์คำว่า KMITL แสดงได้ดังรูปที่ 3.26 เมื่อต้องการเคลื่อนย้ายข้อความกดแป้นพิมพ์อักษร U ข้อความนั้นจะถูกลบออกจากแผ่นวงจรพิมพ์ และพร้อมที่จะนำไปไว้ในตำแหน่งอื่น ดังรูปที่ 3.27
- เลื่อนตัวชี้ไปยังตำแหน่งที่ต้องการกดแป้นพิมพ์ Return เพื่อใส่ข้อความที่ต้องการเคลื่อนย้ายลงในตำแหน่งใหม่ ดังรูปที่ 3.28 ซึ่งรวมถึงการใส่ข้อความในทิศทางต่าง ๆ ด้วย



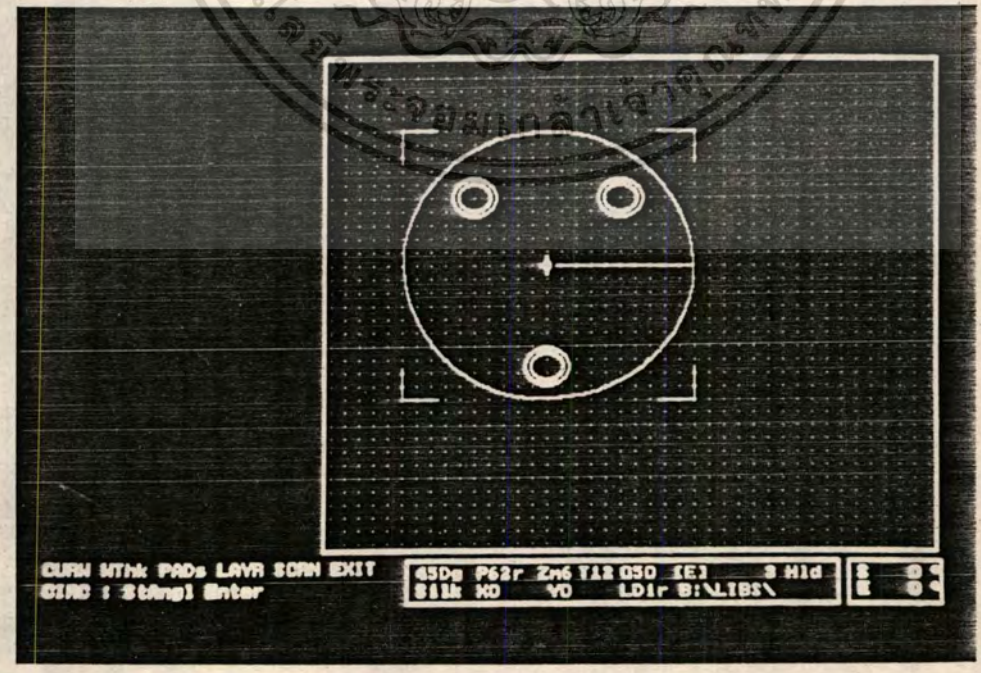
รูปที่ 3.2 แสดงจอภาพของโปรแกรมเมื่อเริ่มสร้างแผ่นวงจรพิมพ์



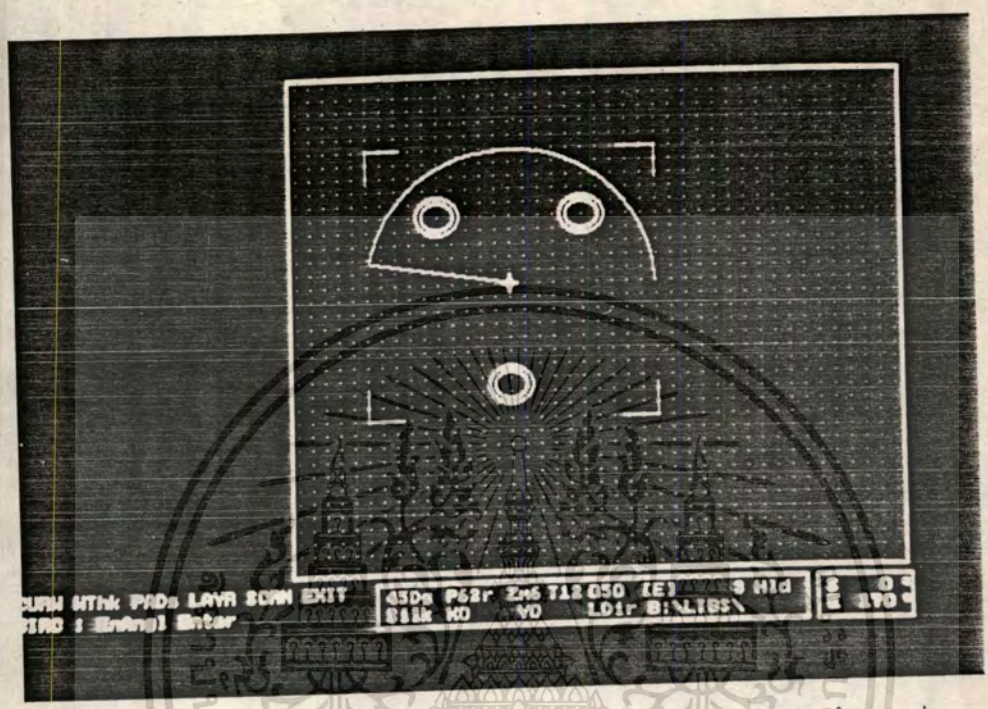
รูปที่ 3.3 แสดงการใส่ขาทรานซิสเตอร์หรือไดโอด 3 อัน



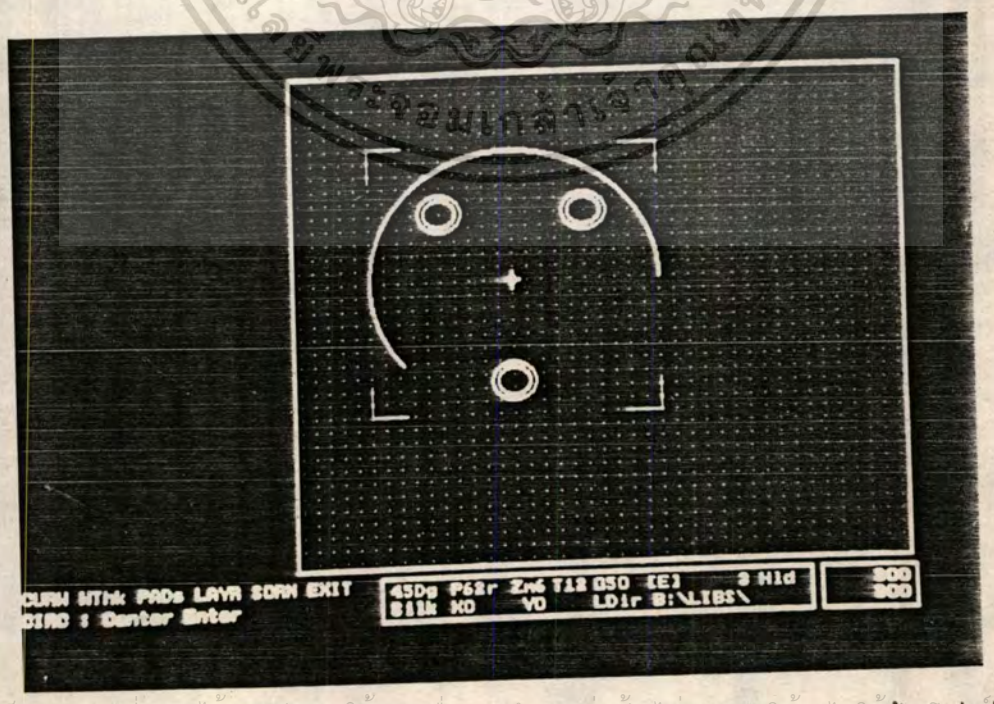
รูปที่ 3.4 แสดงการสร้างวงกลมล้อมรอบขาทรานซิสเตอร์ทั้ง 3 ขั้นตอนที่ 1



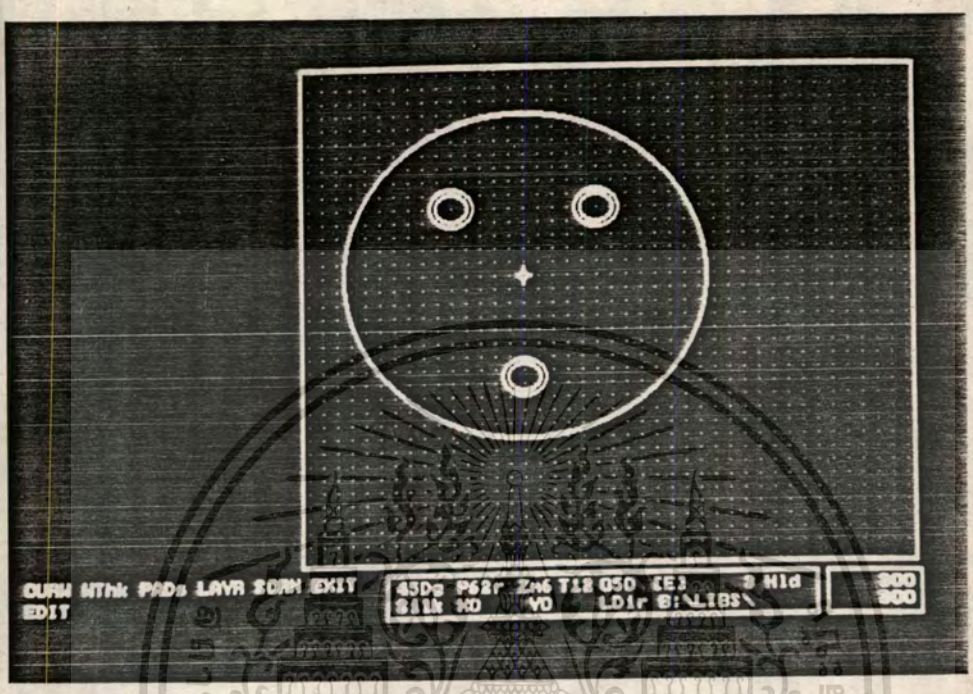
รูปที่ 3.5 แสดงการสร้างวงกลมล้อมรอบขาทรานซิสเตอร์ทั้ง 3 ขั้นตอนที่ 2



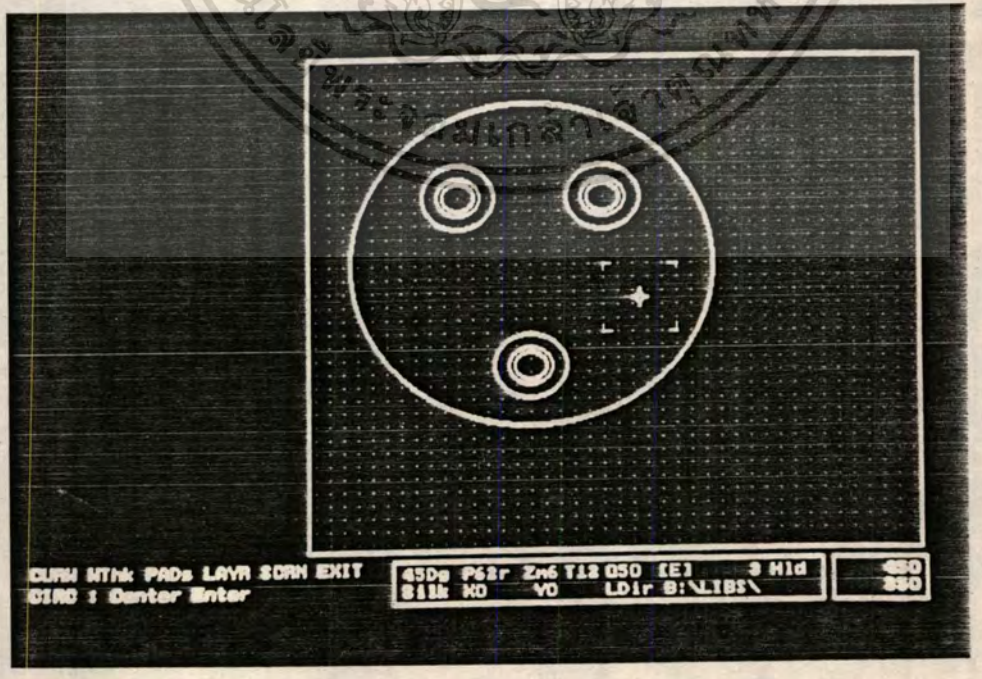
รูปที่ 3.6 แสดงการสร้างวงกลมล้อมลอบขาทรานซิสเตอร์ทั้ง 3 ขั้นตอนที่ 3



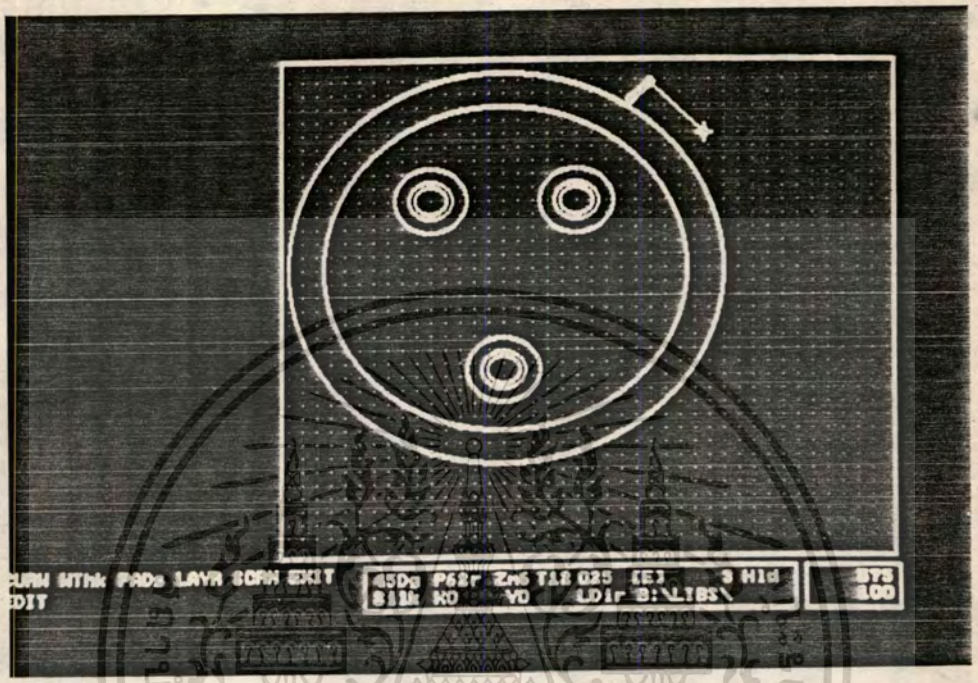
รูปที่ 3.7 แสดงการสร้างส่วนโค้งของวงกลม เมื่อกำหนดมุมเริ่มต้นและมุมสุดท้ายไม่เท่ากัน



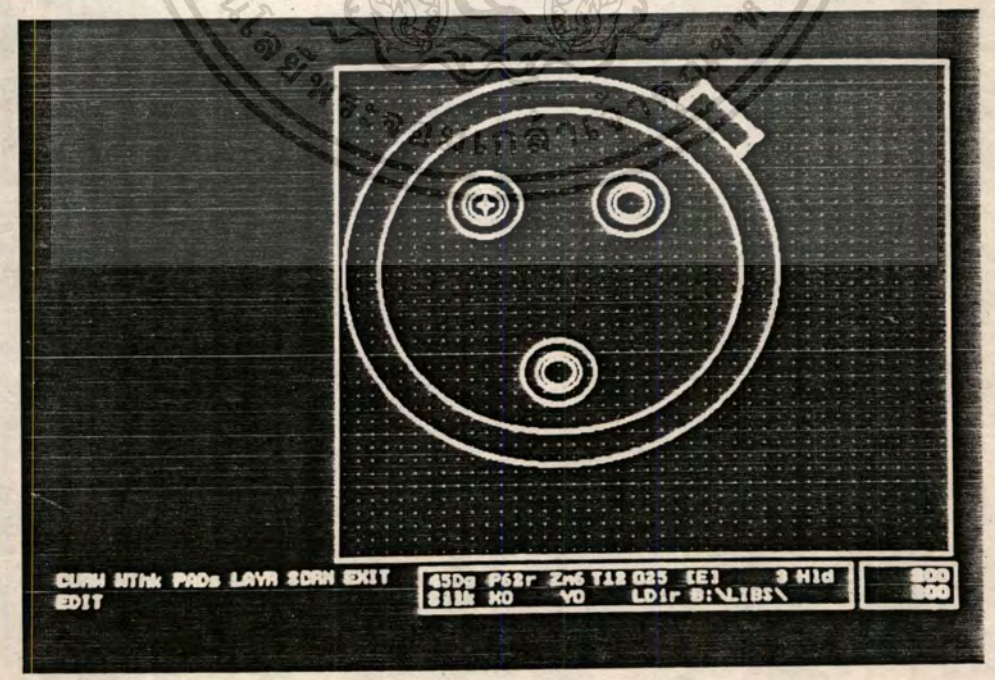
รูปที่ 3.8 แสดงการสร้างวงกลม เมื่อกำหนดมุม เริ่มต้นและมุมสุดท้าย เท่ากัน



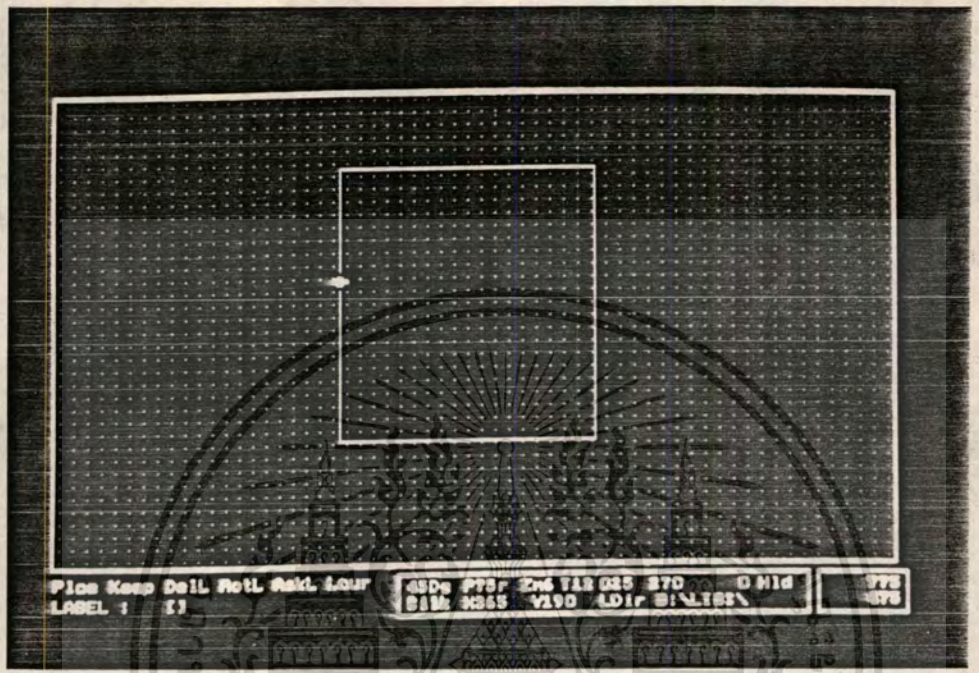
รูปที่ 3.9 แสดงขั้นตอนสร้างวงกลมล้อมรอบขาแต่ละขาของทรานซิสเตอร์



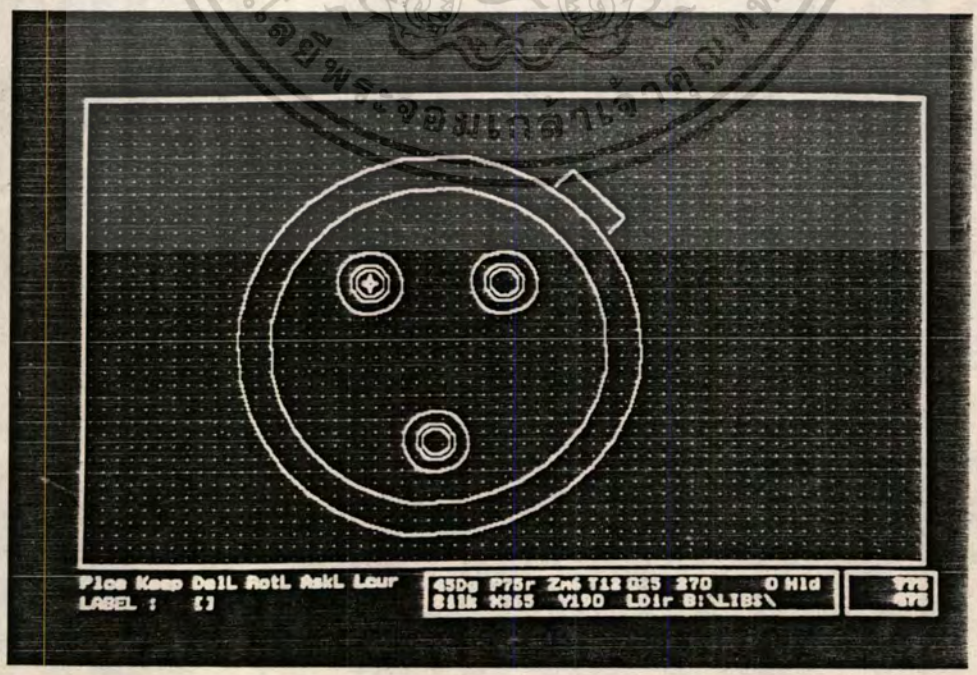
รูปที่ 3.10 แสดงการสร้างวงกลมใหญ่และลากเส้นแสดงตำแหน่งขาของทรานซิสเตอร์



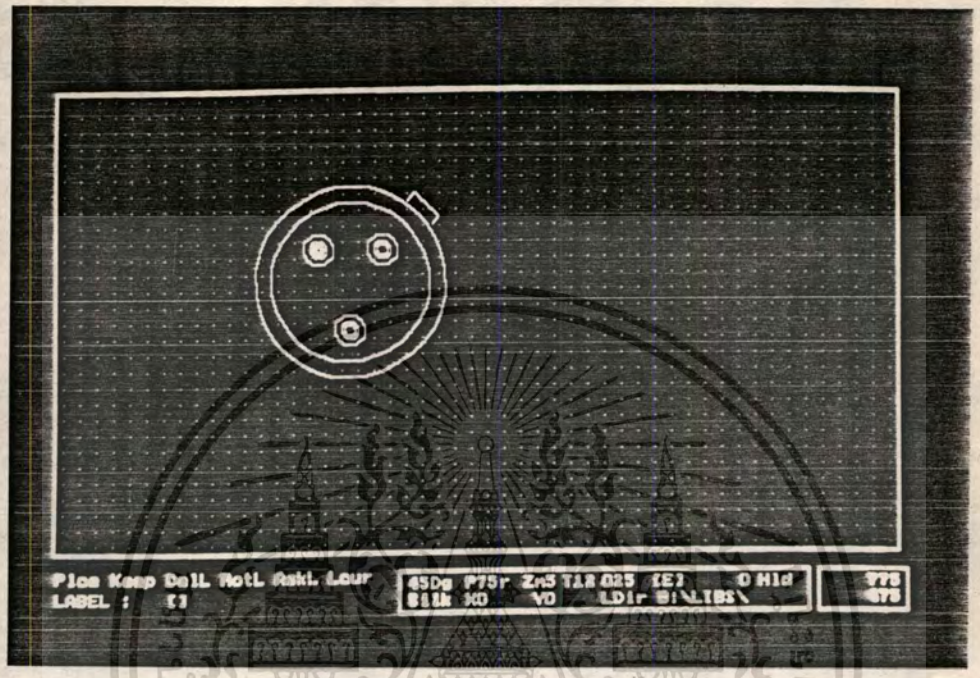
รูปที่ 3.11 แสดงทรานซิสเตอร์ที่พร้อมจะนำเก็บไว้ในไลบรารี



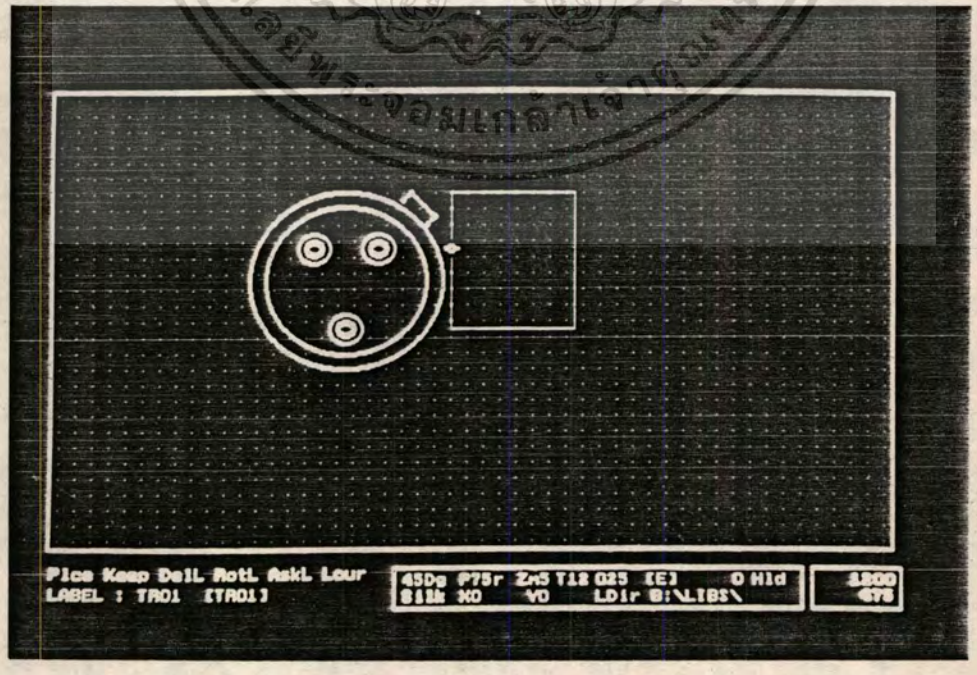
รูปที่ 3.12 แสดงการ Load ตัวทรานซิสเตอร์และใช้คำสั่ง PLCE สร้างลายวงจร



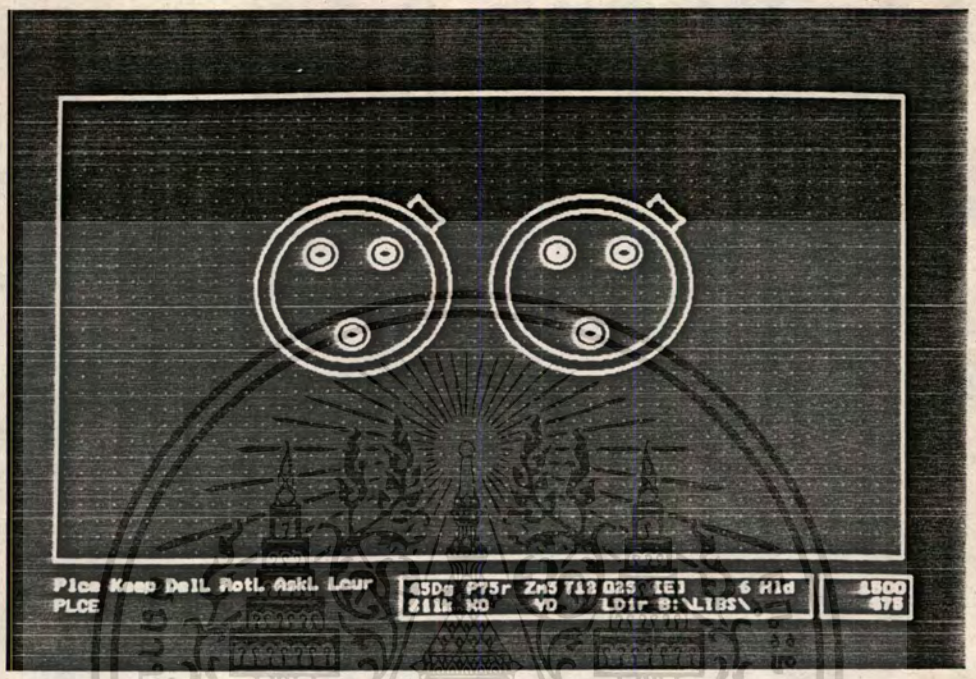
รูปที่ 3.13 แสดงภาพทรานซิสเตอร์ก่อนใส่ลงบนแผ่นวงจรพิมพ์



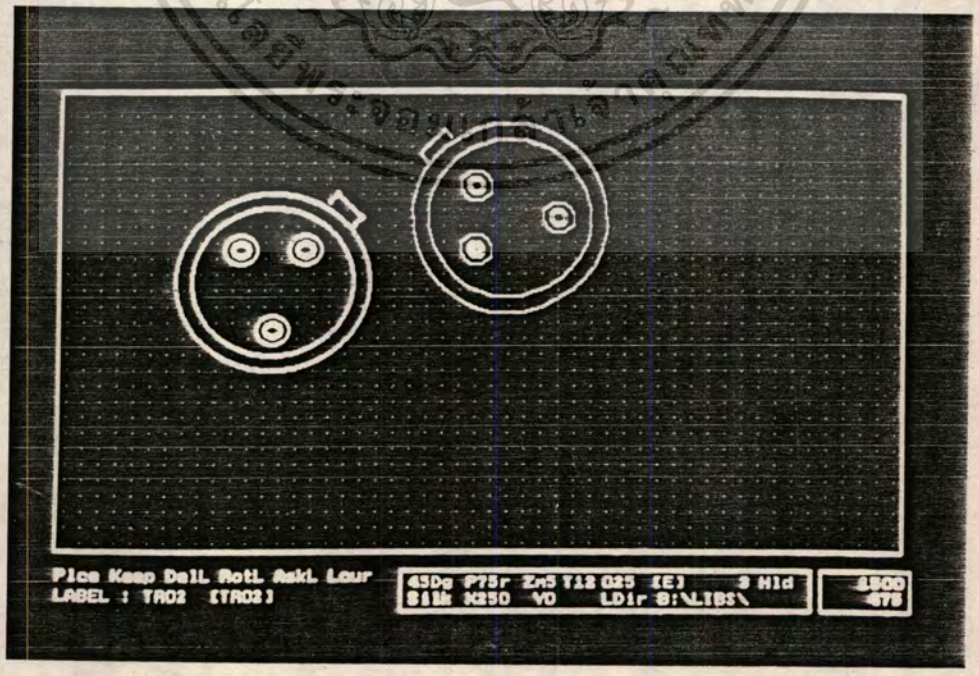
รูปที่ 3.14 แสดงภาพทรานซิสเตอร์เมื่อขนาดการขยายภาพเท่ากับ 5



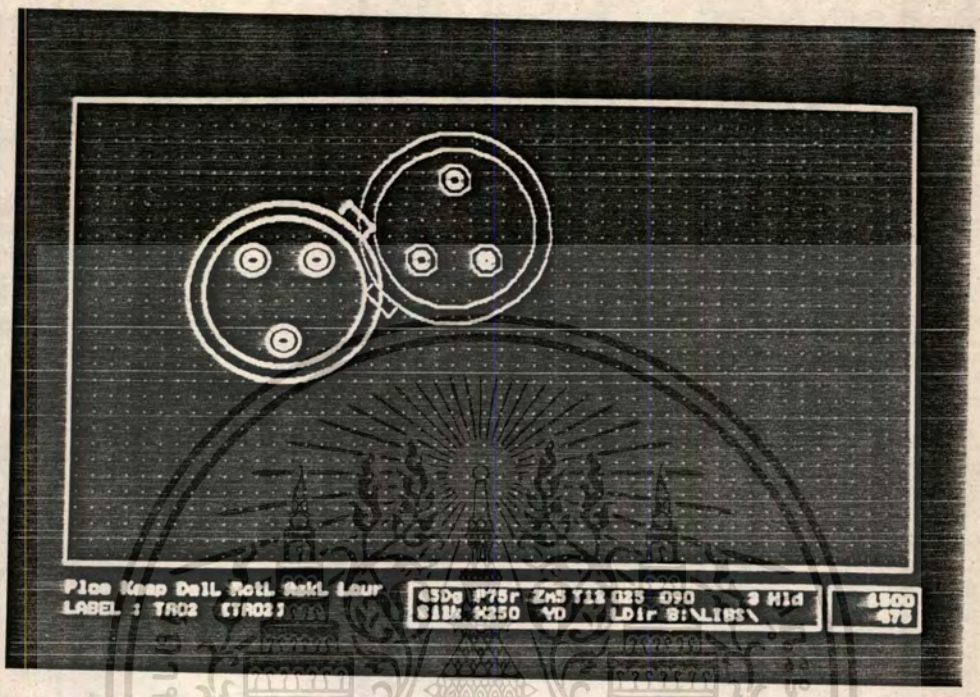
รูปที่ 3.15 แสดงการใส่ทรานซิสเตอร์ตัวที่ 1



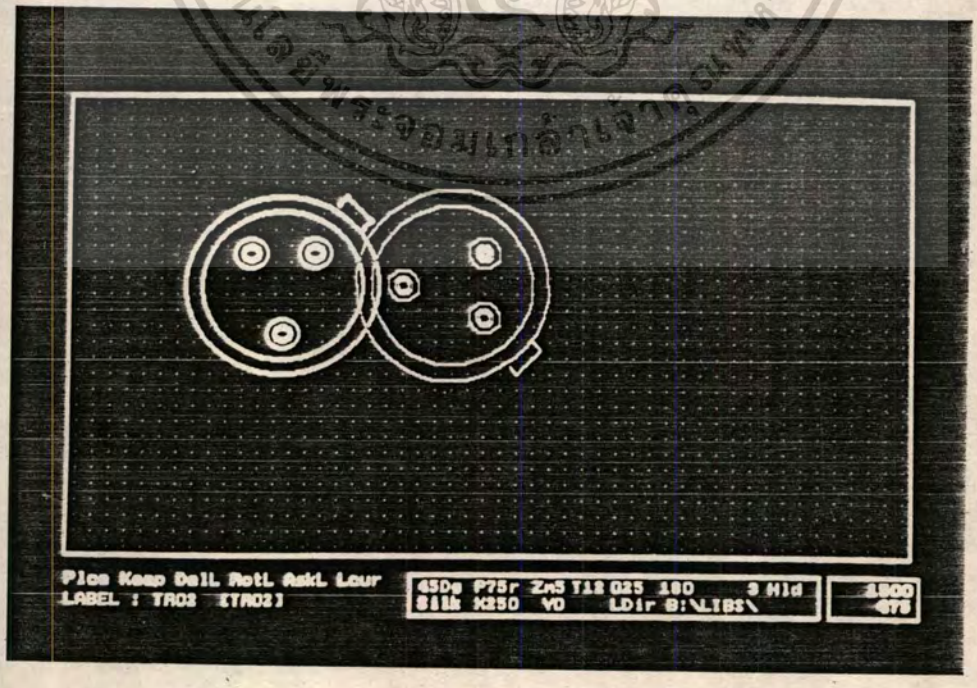
รูปที่ 3.16 แสดงการใส่ทรานซิสเตอร์ตัวที่ 2



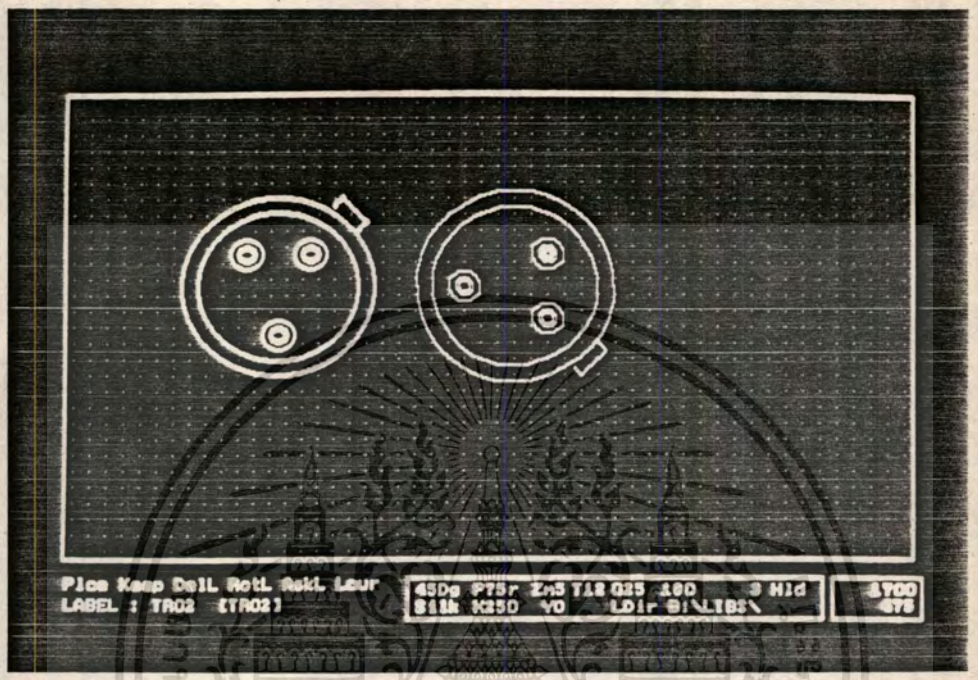
รูปที่ 3.17 แสดงการหมุนทรานซิสเตอร์ในทิศทางที่ 1



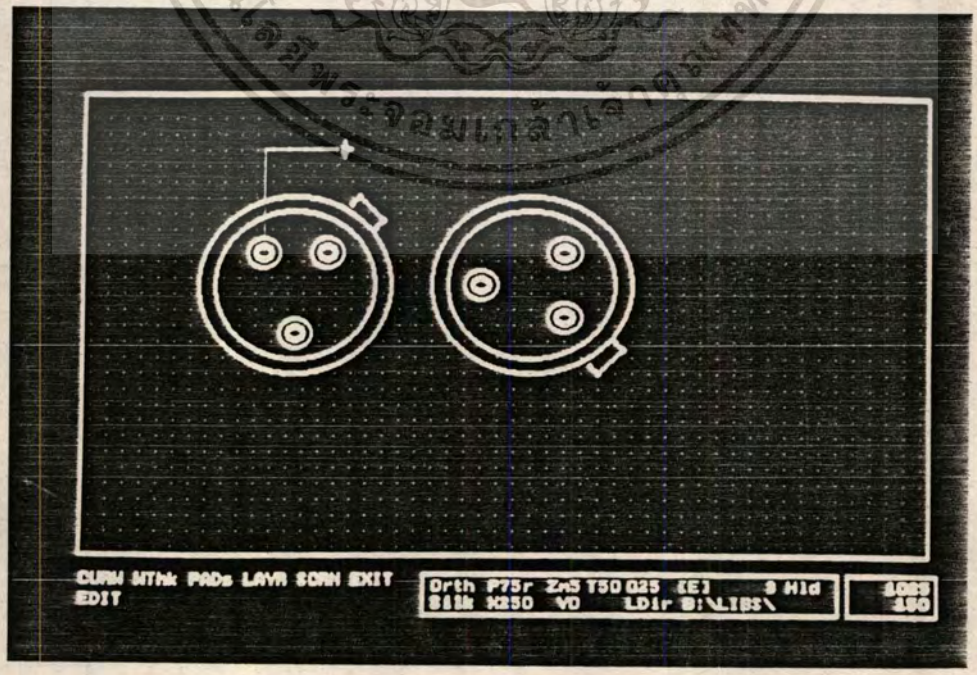
รูปที่ 3.18 แสดงการหมุนทรานซิสเตอร์ในทิศทางที่ 2



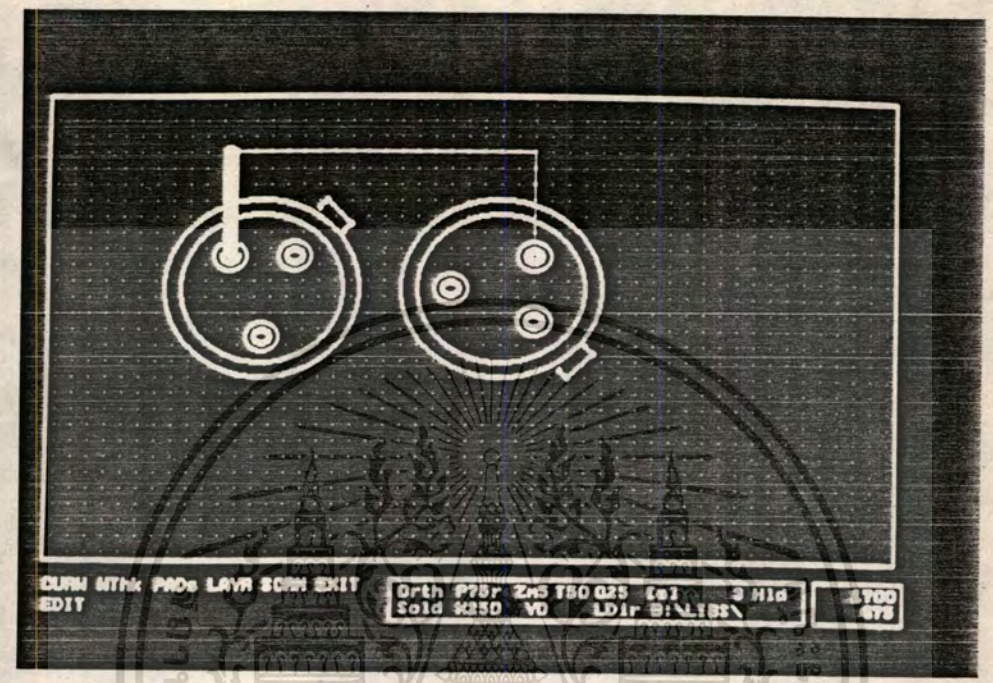
รูปที่ 3.19แสดงการหมุนทรานซิสเตอร์ในทิศทางที่ 3



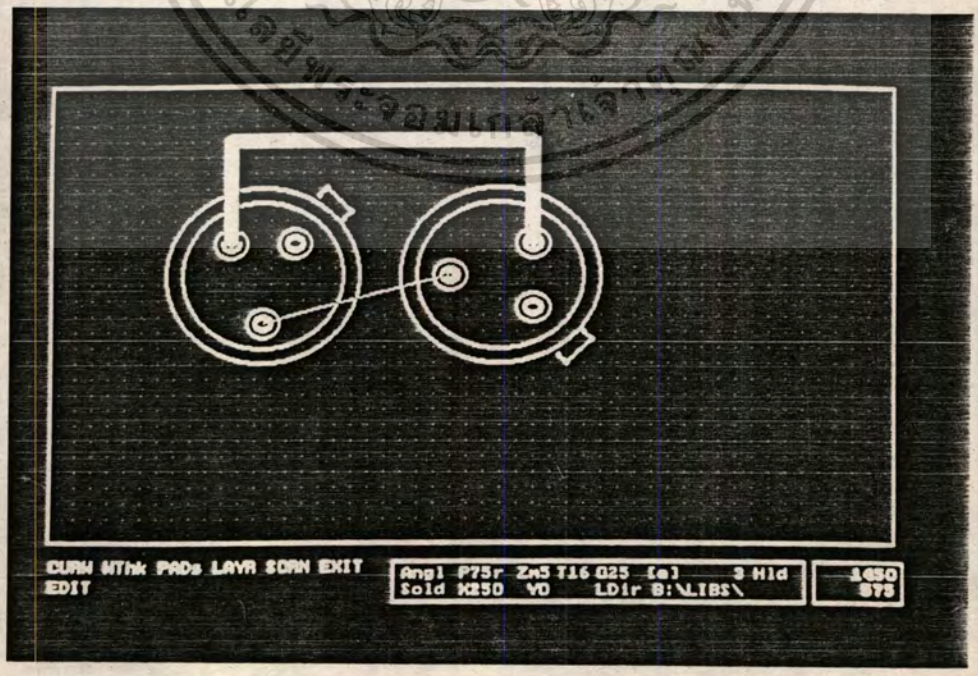
รูปที่ 3.20 แสดงการหมุนทรานซิสเตอร์ในทิศทางที่ 4



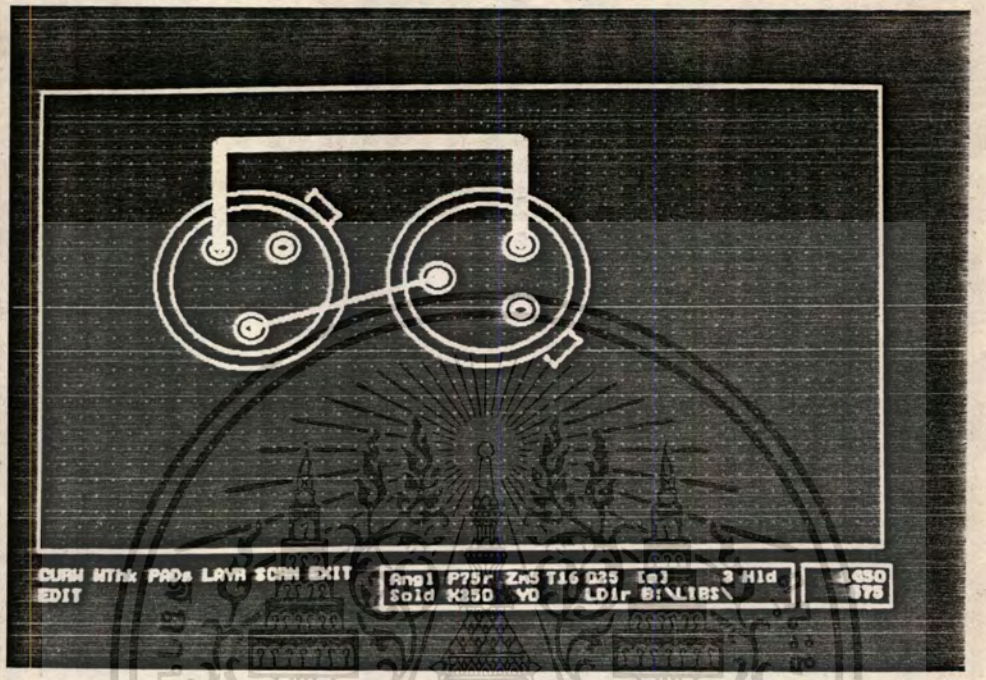
รูปที่ 3.21 แสดงขั้นตอนการสร้างลายเส้นวงจรลำดับที่ 1



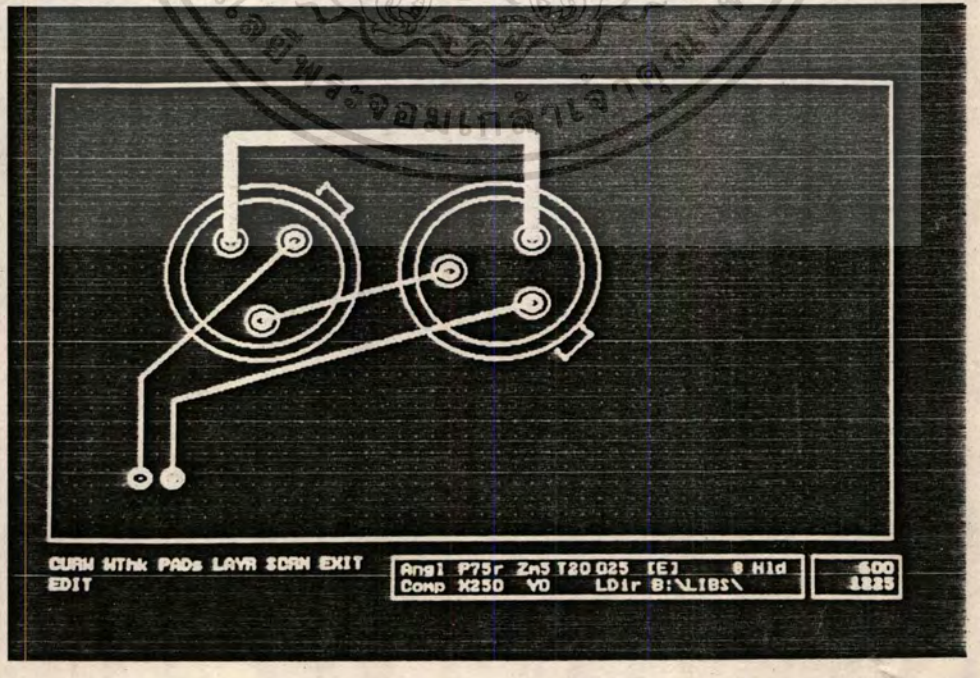
รูปที่ 3.22 แสดงขั้นตอนการสร้างลาย เส้นวงจรลำดับที่ 2



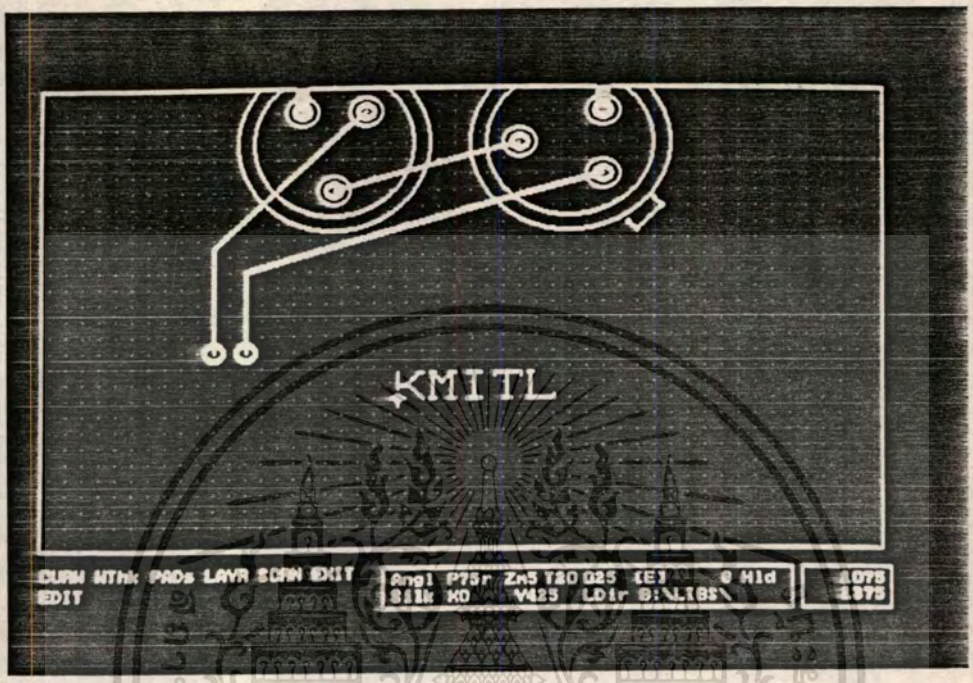
รูปที่ 3.23 แสดงขั้นตอนการสร้างลาย เส้นวงจรลำดับที่ 3



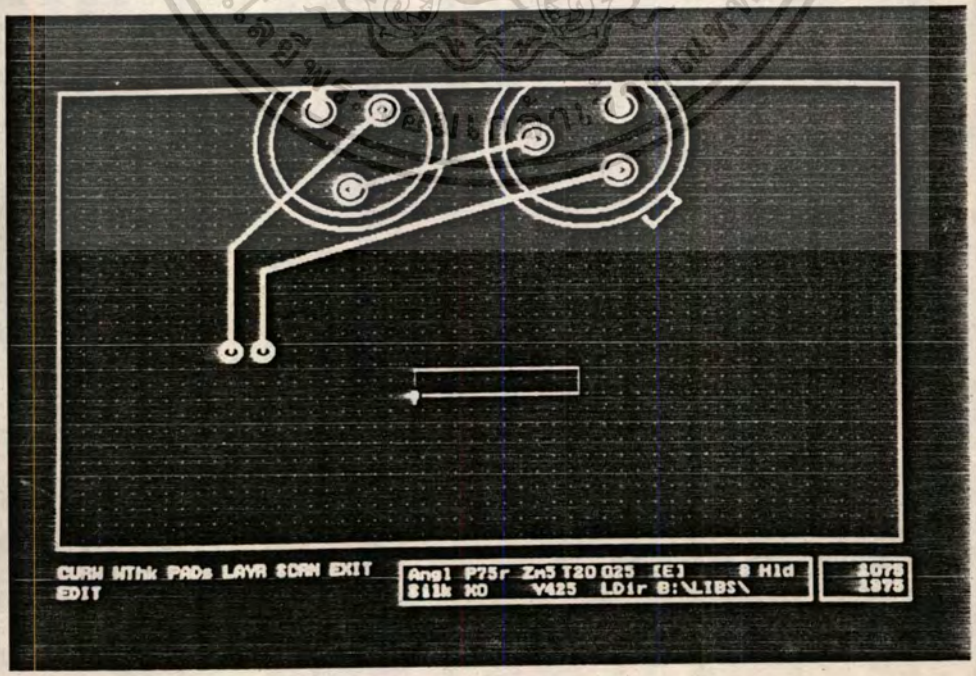
รูปที่ 3.24 แสดงขั้นตอนการสร้างลายเส้นวงจรลำดับที่ 4



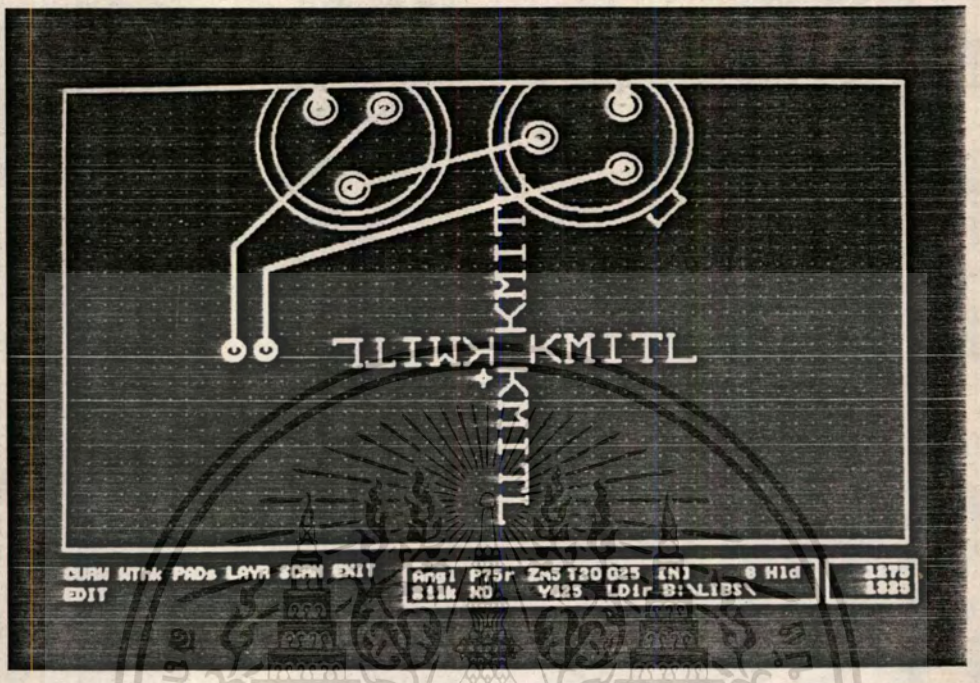
รูปที่ 3.25 แสดงลายเส้นวงจรเมื่อสร้างเสร็จแล้ว



รูปที่ 3.26 แสดงการใส่ข้อความลงบนแผ่นวงจรพิมพ์



รูปที่ 3.27 แสดงการเคลื่อนย้ายข้อความ



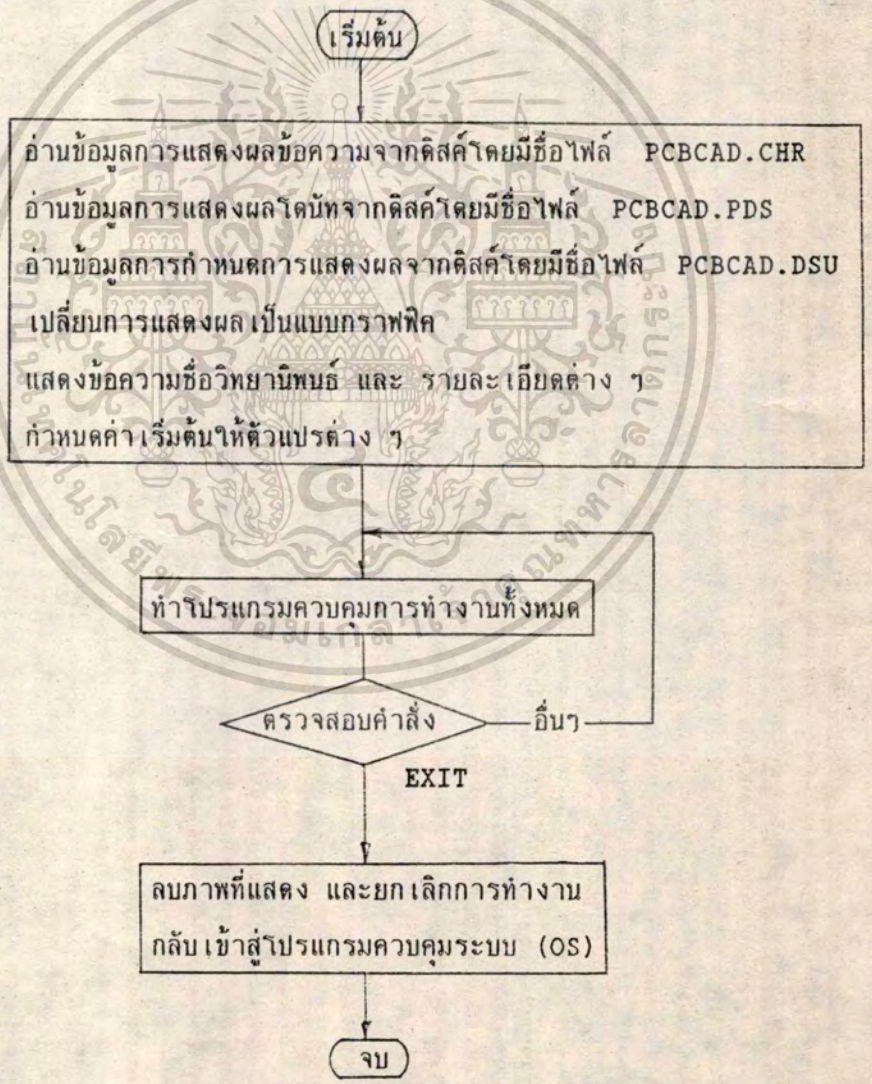
รูปที่ 3.28 แสดงข้อความในทิศทางต่าง ๆ

บทที่ 4

การทำงานของโปรแกรม

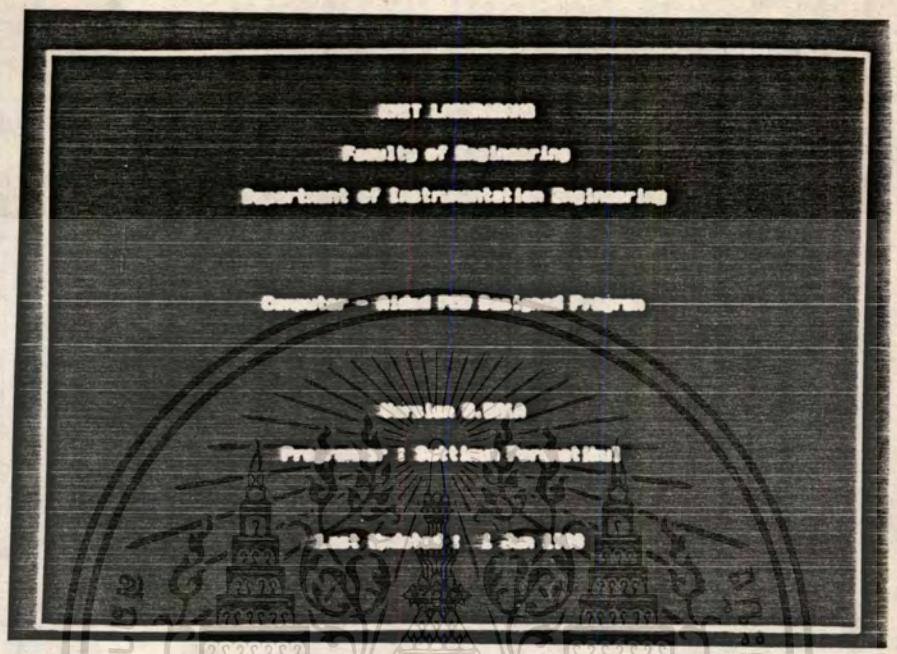
การทำงานของโปรแกรม PCBCAD สามารถแบ่งการทำงานได้เป็นส่วน ๆ ตามการใช้งานได้ดังนี้

4.1 โปรแกรมหลัก มีการทำงานของโปรแกรมตามโฟลว์ชาร์ท (Flow chart) ดังนี้

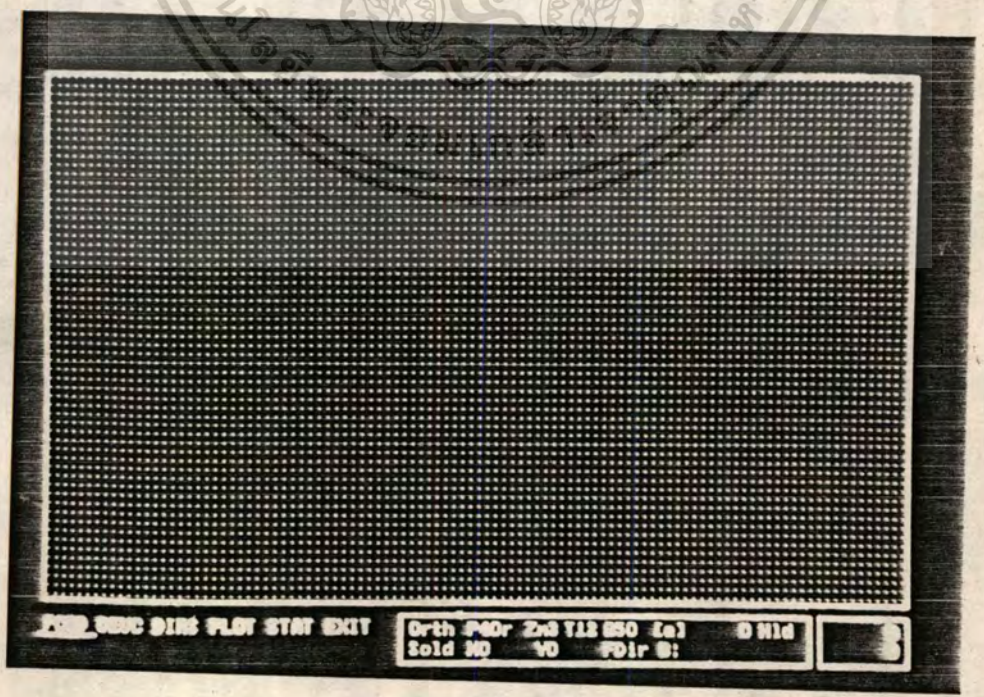


รูปที่ 4.1 แสดงการทำงานของโปรแกรมหลัก

- อ่านข้อมูลการแสดงผลข้อความจากดิสค์โดยมีชื่อไฟล์ PCBCAD.CHR ข้อมูลนี้เป็นข้อมูลตัวอักษรที่ถูกนำมาใช้ในการแสดงผล และการพิมพ์ ซึ่งเป็นคำสั่งสร้างเส้นตรงหลาย ๆ เส้นประกอบกันเป็นตัวอักษร โดยสามารถกำหนดการใช้งานได้ตามแป้นพิมพ์อักษรต่าง ๆ
- อ่านข้อมูลการแสดงผลโดนัทจากดิสค์ โดยมีชื่อไฟล์ PCBCAD.PDS ข้อมูลนี้เป็นข้อมูลที่ใช้สำหรับสร้างขาอุปกรณ์ จุดต่อลายเส้นและอื่น ๆ ซึ่งผู้ใช้สามารถกำหนดขนาด และ ชนิดในการใช้งานได้ ดังได้กล่าวไว้ในภาคผนวก 3.
- อ่านข้อมูลการกำหนดการแสดงผลจากดิสค์ โดยมีชื่อไฟล์ PCBCAD.DSU ข้อมูลนี้เป็นข้อมูลที่ใช้ในการกำหนดขอบเขต หรือ ขนาดของการแสดงผลบนจอภาพของแผ่นวงจรพิมพ์
- แสดงข้อความชื่อวิทยานิพนธ์ และรายละเอียดต่าง ๆ เช่น ชื่อสถาบัน ผู้สร้าง วัน เดือน ปีที่สร้าง ดังรูปที่ 4.1 และเข้าสู่การทำงานโปรแกรม ดังรูปที่ 4.2
- กำหนดค่าเริ่มต้นให้กับตัวแปรต่าง ๆ ในโปรแกรม
- ทำโปรแกรมควบคุมการทำงานทั้งหมด โดยส่วนนี้จะทำหน้าที่ควบคุมการใช้งานในแบบต่าง ๆ ที่ผู้ใช้ต้องการ
- ตรวจสอบคำสั่ง ถ้าการตรวจสอบพบว่าไม่ใช่คำสั่ง EXIT แล้ว ให้กลับไปทำงานโปรแกรมควบคุมการใช้งานคำสั่งหลักและถ้าตรวจสอบพบว่าเป็นคำสั่ง EXIT จะออกจากโปรแกรม PCBCAD กลับเข้าสู่โปรแกรมควบคุมระบบ (OS)



รูปที่ 4.2 ภาพแสดงรายละเอียดของวิทยานิพนธ์

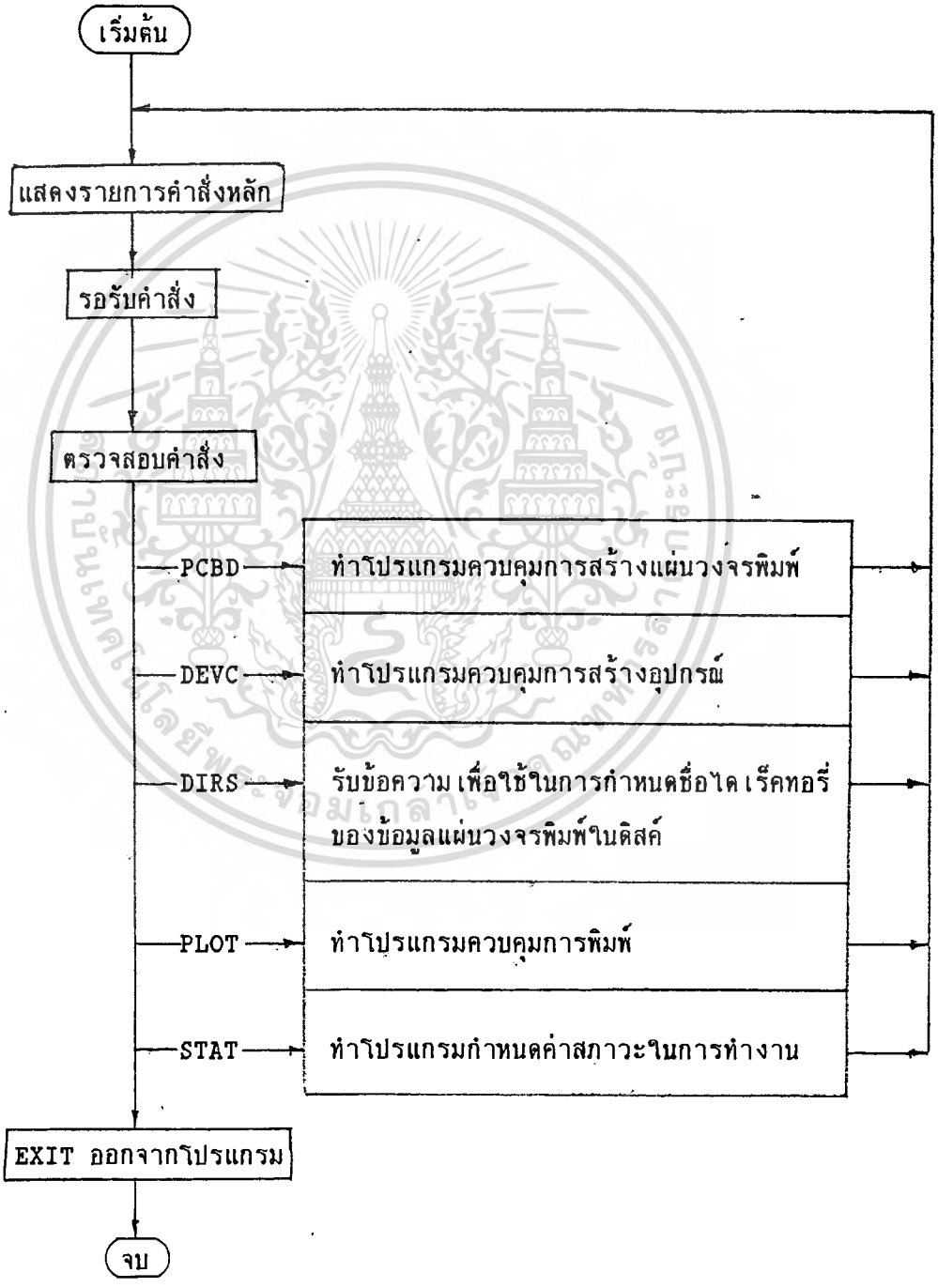


รูปที่ 4.3 ภาพแสดงการเริ่มต้นของการทำงานโปรแกรม

4.2 โปรแกรมควบคุมการแสดงรายการคำสั่ง ทำหน้าที่ในการแสดงรายการคำสั่งต่าง ๆ ที่ใช้ในการทำงาน ดังรูปที่ 3.1 ซึ่งผู้ใช้เป็นผู้เลือก มีด้วยกันหลายส่วน และมีการแสดงผลดังนี้

- ส่วนเลือกใช้คำสั่งหลัก	PCBD DEVC DIRS PLOT STAT EXIT
- ส่วนสร้างแผ่นวงจรพิมพ์	LOAD SAVE EDIT LIBS CLRS EXIT
- ส่วนสร้างอุปกรณ์	LDIR SAVE EDIT - CLRS EXIT
- ส่วนกำหนดค่าสภาวะในการทำงาน	- - LAYR ORGC ZmEn EXIT
- ส่วนสร้างภาพ	CURW WThk PADs LAYR SCRN EXIT
- ส่วนสร้างภาพด้วยอุปกรณ์	PLCE LOAD LDIR - - EXIT
- ส่วนควบคุมการใช้งานอุปกรณ์	Plce Move DelL AskL Lcur EXIT
- ส่วนเลือกชนิดเส้นแสดงแนวการลาก	Angl Orth 45Dg - - -
- ส่วนเลือกขนาดเส้น	Th12 Th16 Th20 Th50 - -
- ส่วนเลือกระยะการเคลื่อนตัวชี้	100 50 25 10 5 1
- ส่วนเลือกอัตราการขยายภาพ	1 2 3 4 5 6
- ส่วนเลือกด้านแผ่นวงจรพิมพ์	Sold Comp Silk - - -
- ส่วนการเลือกการแสดงผล	Sold Comp Silk Pad Text EXIT
- ส่วนแเลือกขนาด และชนิดโดนัท	P32r P32s P48r P48s P62r P62s
	P75r P75s SM16 SM20 T300 T400

4.3 โปรแกรมควบคุมการทำงานทั้งหมด ทำหน้าที่จัดการเกี่ยวกับการใช้งานคำสั่งหลักต่าง ๆ โดยมีลำดับขั้นการทำงานของโปรแกรมตามโฟลว์ชาร์ท ดังรูปที่ 4.4



รูปที่ 4.4 แสดงการทำงานโปรแกรมควบคุมการทำงานทั้งหมด

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการทำงานทั้งหมด อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ควบคุมการใช้งานคำสั่งหลักต่าง ๆ มีหลักการทำงานดังนี้

- แสดงรายการคำสั่งหลัก
- รอรับคำสั่ง
- ตรวจสอบคำสั่ง ถ้าตรวจพบว่า เป็นคำสั่งที่มีใช้และตรงกับรายการคำสั่งที่ได้แสดง จะทำงานโปรแกรมตามคำสั่งต่าง ๆ ดังนี้

  PCBD ทำโปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์

  DEVC ทำโปรแกรมควบคุมการสร้างอุปกรณ์

  DIRS รับข้อความเพื่อใช้ในการกำหนดชื่อไดเร็คทอรี่ของข้อมูลแผ่นวงจรพิมพ์ในดิสค์

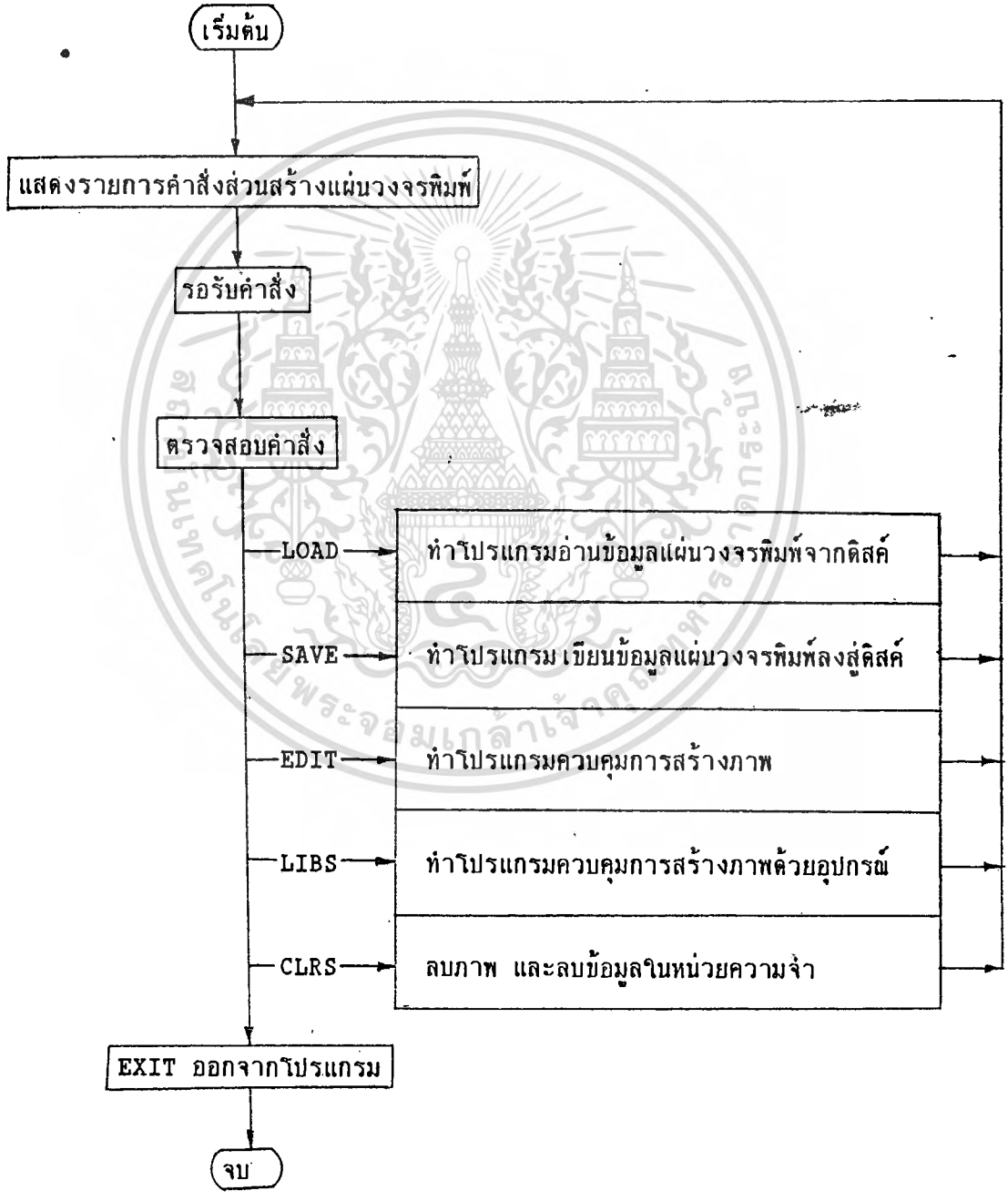
  PLOT ทำโปรแกรมควบคุมการพิมพ์

  STAT ทำโปรแกรมกำหนดค่าสภาวะในการทำงาน

  EXIT ยกเลิกการทำงานโปรแกรม PCBCAD

  ถ้าผลของการตรวจสอบพบว่าคำสั่งนั้นไม่ตรงกับคำสั่งที่มีให้ใช้ โปรแกรมจะกลับไปทำงานในตำแหน่งเริ่มต้นใหม่

4.4 โปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ ทำหน้าที่จัดการเกี่ยวกับการใช้งานคำสั่งต่าง ๆ ในการสร้างแผ่นวงจรพิมพ์ โดยมีลำดับขั้นการทำงานของโปรแกรมตามโฟลว์ชาร์ทดังรูปที่ 4.5



รูปที่ 4.5 แสดงการทำงานโปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์

4.5 โปรแกรมควบคุมการสร้างอุปกรณ์ ทำหน้าที่จัดการเกี่ยวกับการใช้งานคำสั่งต่าง ๆ ในการสร้างอุปกรณ์ เพื่อนำมาสร้างแผ่นวงจรพิมพ์โดยมีลำดับขั้นการทำงานของโปรแกรมตามโฟลว์ชาร์ทดังรูปที่ 4.6

รูปที่ 4.6 แสดงการทำงานโปรแกรมควบคุมการสร้างอุปกรณ์

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ อธิบายได้ดังนี้

- แสดงรายการคำสั่งส่วนสร้างแผ่นวงจรพิมพ์
- รอรับคำสั่งจากผู้ใช้
- ตรวจสอบคำสั่ง ถ้าตรวจพบว่าเป็นคำสั่งที่มีใช้และตรงกับรายการคำสั่งที่ได้แสดง ผลจะมีการทำงานโปรแกรมตามคำสั่งดังนี้

  LOAD : ทำโปรแกรมอ่านข้อมูลแผ่นวงจรพิมพ์จากดิสค์

  SAVE : ทำโปรแกรมเขียนข้อมูลแผ่นวงจรพิมพ์ลงสู่ดิสค์

  EDIT : ทำโปรแกรมควบคุมการสร้างภาพ

  LIBS : ทำโปรแกรมควบคุมการสร้างภาพด้วยอุปกรณ์จากไลบรารี่

  CLRS : ลบภาพ และลบข้อมูลมนหน่วยความจำ

  EXIT : ยกเลิกการทำงานโปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์

  ถ้าผลของการตรวจสอบพบว่าคำสั่งนั้นไม่ตรงกับคำสั่งที่มีให้ใช้ โปรแกรมจะกลับไปทำงานในตำแหน่งเริ่มต้นใหม่

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการสร้างอุปกรณ์ อธิบายได้ดังนี้

- แสดงรายการคำสั่งส่วนควบคุมการสร้างอุปกรณ์
- รอรับคำสั่ง
- ตรวจสอบคำสั่ง ถ้าตรวจพบว่าเป็นคำสั่งที่มีใช้และตรงกับรายการคำสั่งที่ได้แสดง ผลจะมีการทำงานโปรแกรมตามคำสั่งดังนี้

  LDIR : รับข้อความ เพื่อใช้ในการกำหนดชื่อไดเร็คทอรี่ของข้อมูลอุปกรณ์ในดิสค์

  SAVE : เขียนข้อมูลอุปกรณ์ลงสู่ดิสค์เพื่อเก็บไว้เป็นไลบรารี่

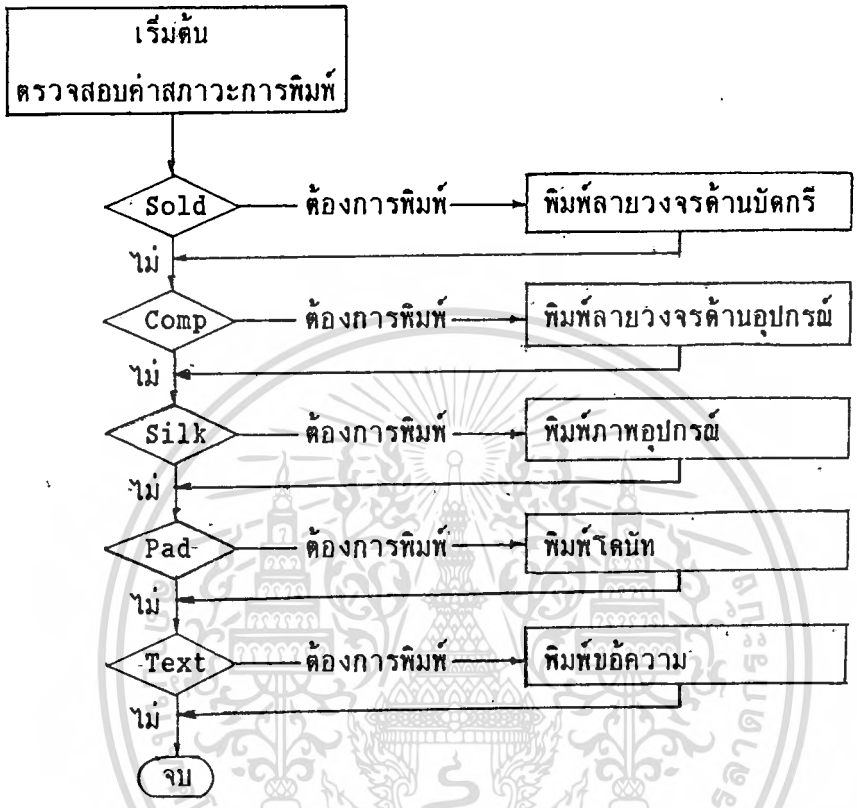
  EDIT : โปรแกรมควบคุมการสร้างภาพ

  CLRS : ยกเลิกข้อมูล และลบภาพอุปกรณ์

  EXIT : ยกเลิกการทำงานโปรแกรมควบคุมการสร้างอุปกรณ์

  ถ้าผลของการตรวจสอบพบว่าคำสั่งนั้นไม่ตรงกับคำสั่งที่มีให้ใช้ โปรแกรมจะกลับไปทำงานในตำแหน่งเริ่มต้นใหม่

4.6 โปรแกรมควบคุมการพิมพ์ มีลำดับขั้นการทำงาน ดังนี้



รูปที่ 4.7 แสดงการทำงานโปรแกรมควบคุมการสร้างอุปกรณ์

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการพิมพ์ อธิบายได้ดังนี้

- ตรวจสอบค่าสภาวะการพิมพ์ จากค่าสภาวะการทำงานที่ผู้ใช้กำหนดขึ้นโดยคำสั่ง LAYR มีค่าสภาวะต่าง ๆ ดังนี้

  Sold ค่าสภาวะการพิมพ์ลายวงจรด้านบัดกรี โปรแกรมจะทำการตรวจสอบและพิมพ์ภาพแผ่นวงจรพิมพ์ด้านบัดกรี เมื่อค่าสภาวะเป็นจริง

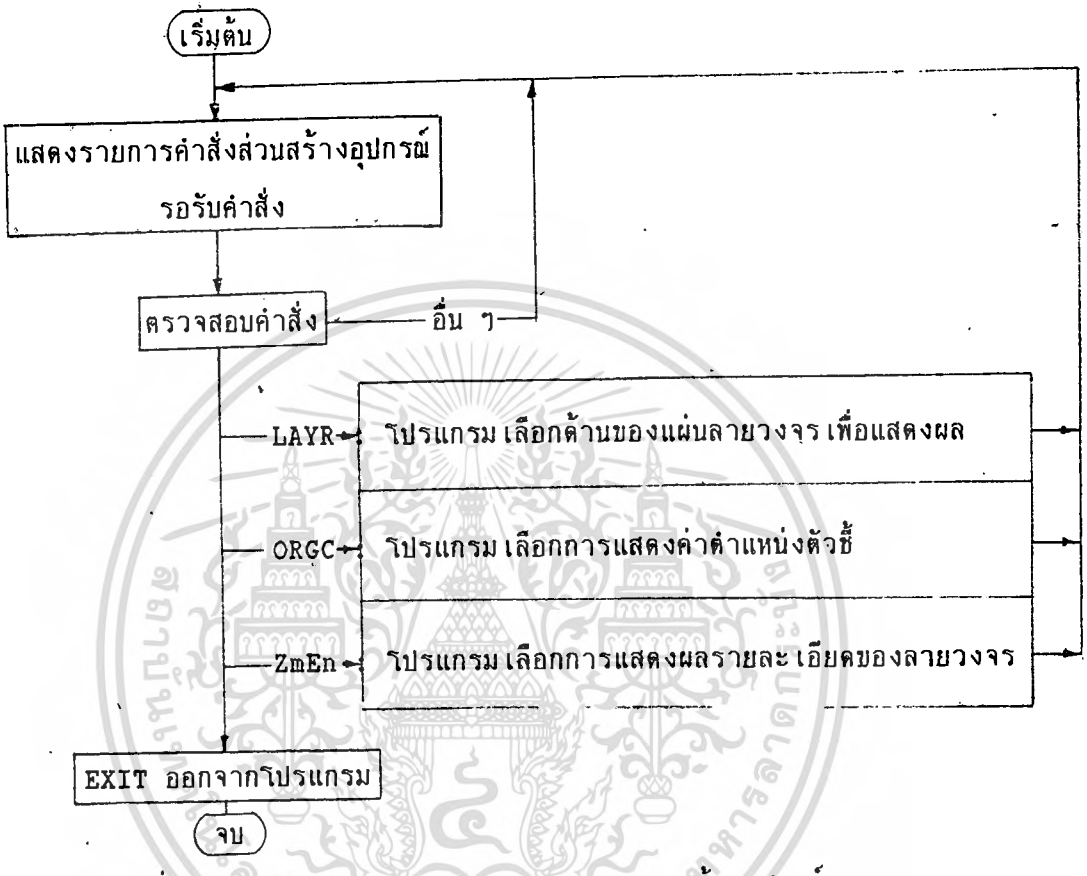
  Comp ค่าสภาวะการพิมพ์ลายวงจรด้านอุปกรณ์ โปรแกรมจะทำการตรวจสอบและพิมพ์ภาพแผ่นวงจรพิมพ์ด้านอุปกรณ์ เมื่อค่าสภาวะเป็นจริง

  Silk ค่าสภาวะการพิมพ์ภาพอุปกรณ์ โปรแกรมจะทำการตรวจสอบและพิมพ์ภาพอุปกรณ์ เมื่อค่าสภาวะเป็นจริง

  Pad ค่าสภาวะการพิมพ์โดนัท โปรแกรมจะทำการตรวจสอบ และพิมพ์ภาพโดนัท เมื่อค่าสภาวะเป็นจริง

  Text ค่าสภาวะการพิมพ์ข้อความ โปรแกรมจะทำการตรวจสอบ และพิมพ์ข้อความ เมื่อค่าสภาวะเป็นจริง

4.7 โปรแกรมกำหนดค่าสภาวะการทำงาน มีลำดับขั้นการทำงาน ดังนี้



รูปที่ 4.8 แสดงการทำงานโปรแกรมควบคุมการสร้างอุปกรณ์

โฟลว์ชาร์ทการทำงานของโปรแกรมกำหนดค่าสภาวะการทำงาน อธิบายได้ดังนี้

- แสดงรายการคำสั่งส่วนกำหนดค่าสภาวะการทำงาน
- รอรับคำสั่ง
- ตรวจสอบคำสั่ง ถ้าตรวจพบว่าเป็นคำสั่งที่มีใช้และตรงกับข้อความเมนูที่ได้แสดงผลจะมีการทำงานโปรแกรมตามคำสั่งดังนี้

  LAYR : โปรแกรมเลือกด้านของแผ่นวงจรพิมพ์ เพื่อแสดงผล

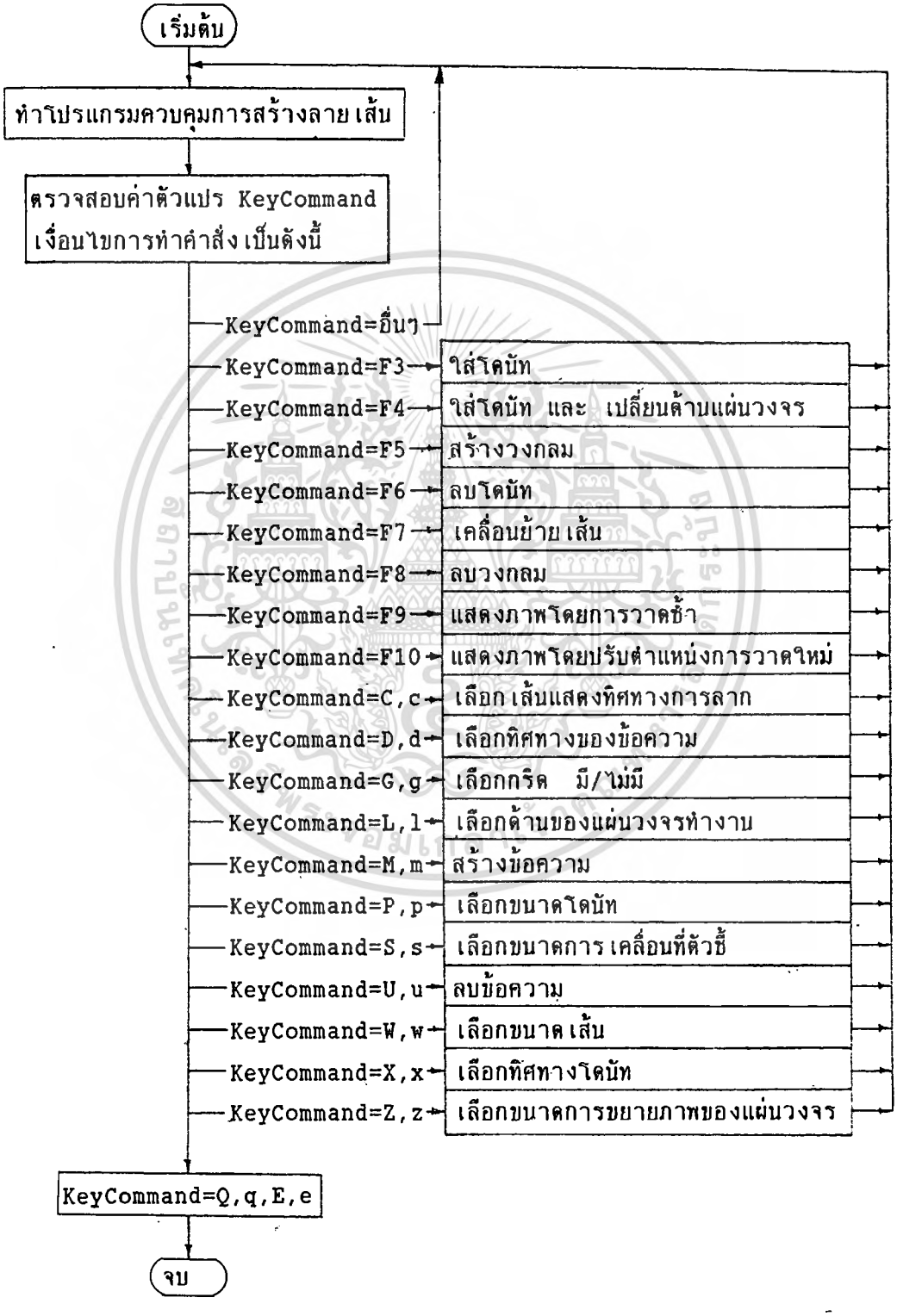
  ORGC : โปรแกรมเลือกการแสดงค่าตำแหน่งตัวชี้ (แสดง/ไม่แสดง)

  ZmEn : โปรแกรมเลือกการแสดงผลรายละเอียดของลายวงจร

  EXIT : ยกเลิกการทำงานของโปรแกรมกำหนดค่าสภาวะการทำงาน

  ถ้าผลของการตรวจสอบพบว่าคำสั่งนั้นไม่ตรงกับคำสั่งที่มีให้ใช้ โปรแกรมจะกลับไปทำงานในตำแหน่ง เริ่มต้นใหม่

4.8 โปรแกรมควบคุมการสร้างภาพ



รูปที่ 4.9 แสดงการทำงานของโปรแกรมควบคุมการสร้างภาพ

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการสร้างภาพ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่จัดการทางด้านการสร้างลายเส้นวงจร และควบคุมการทำงานต่าง ๆ ในการสร้างแผ่นวงจรพิมพ์ มีลำดับขั้นการทำงานดังนี้

- ทำโปรแกรมควบคุมการสร้างลายเส้น โดยจะทำหน้าที่ควบคุมทิศทางการเคลื่อนที่ของตัวชี้ และนำค่าคำสั่งต่าง ๆ ที่ได้รับมาตรวจสอบการทำงานต่อไป
- ส่วนตรวจสอบค่าตัวแปร KeyCommand ทำหน้าที่สั่งงานควบคุมตามคำสั่งต่าง ๆ ที่ได้รับ ดังนี้

F3 : ทำโปรแกรมใส่โดนัท

F4 : ทำโปรแกรมใส่โดนัท และเปลี่ยนด้านแผ่นวงจร

F5 : ทำโปรแกรมสร้างวงกลม

F6 : ทำโปรแกรมลบโดนัท

F7 : ทำโปรแกรมเคลื่อนย้ายเส้นเพื่อการแก้ไขลายวงจร

F8 : ทำโปรแกรมลบวงกลม

F9 : ทำโปรแกรมแสดงภาพโดยการวาดซ้ำ

F10 : ทำโปรแกรมแสดงภาพโดยปรับตำแหน่งการวาดใหม่

C,c : ทำโปรแกรมเลือกชนิดของเส้นแสดงแนวการลาก

D,d : ทำโปรแกรมเลือกทิศทางการจัดวางข้อความ

G,g : ทำโปรแกรมเลือกกริด เพื่อใช้ในการอ้างอิงกับตำแหน่งตัวชี้

L,l : ทำโปรแกรมเลือกด้านของแผ่นวงจรพิมพ์

M,m : ทำโปรแกรมสร้างข้อความ

P,p : ทำโปรแกรมเลือกขนาด และชนิดของโดนัท

S,s : ทำโปรแกรมเลือกระยะการเคลื่อนที่ของตัวชี้

U,u : ทำโปรแกรมลบ หรือเคลื่อนย้ายข้อความ

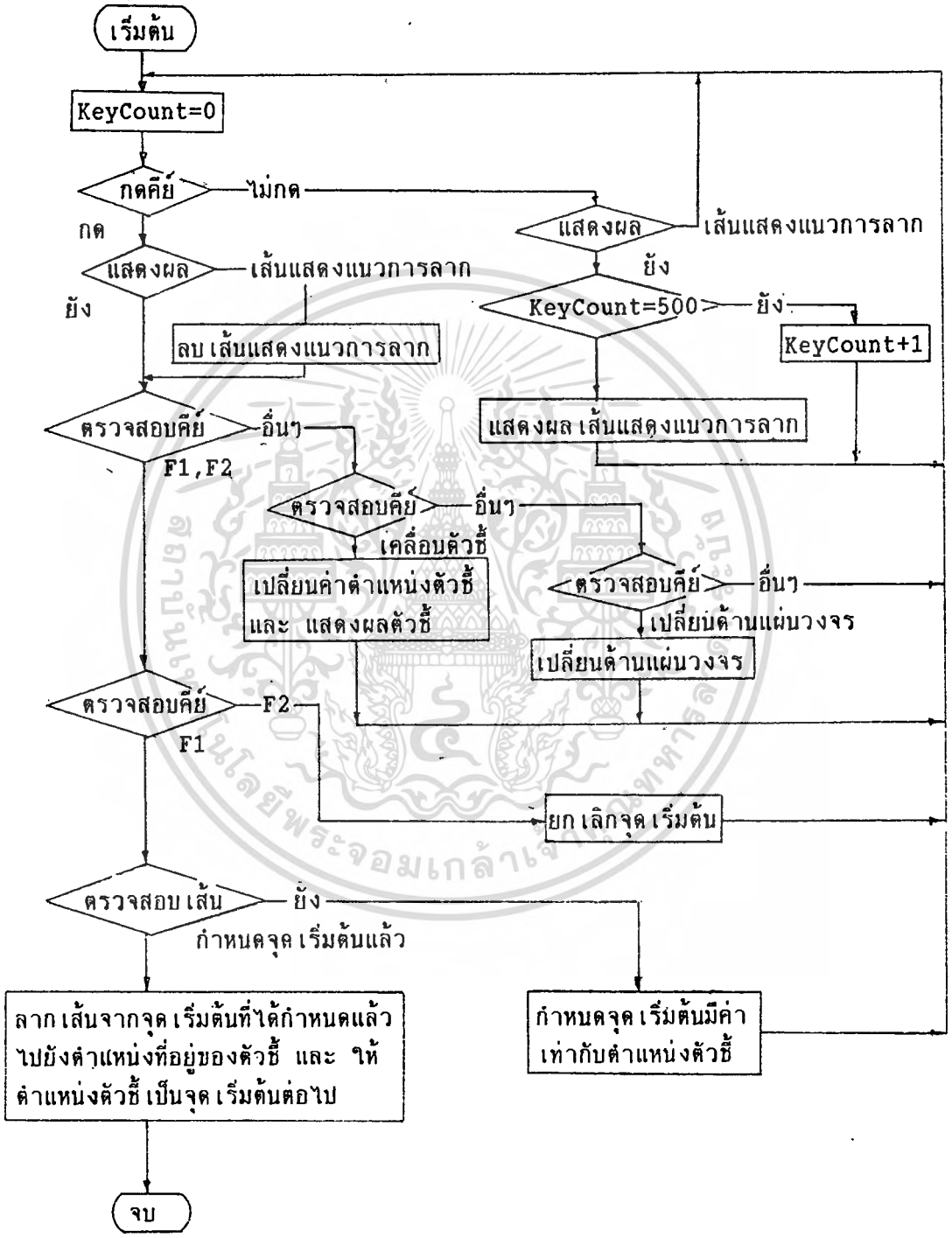
W,w : ทำโปรแกรมเลือกขนาดของลายเส้น

X,x : ทำโปรแกรมเลือกทิศทางการจัดวางโดนัท

Z,z : ทำโปรแกรมแสดงภาพขนาดขยาย

คำสั่งต่าง ๆ เหล่านี้ เมื่อสิ้นสุดการทำงานแล้วจะกลับไปทำส่วนควบคุมการสร้างภาพใหม่ และ สิ้นสุดการทำงานโปรแกรมนี้ เมื่อค่าตัวแปร KeyCommand มีค่าเท่ากับ Q, q, E หรือ e

4.9 โปรแกรมควบคุมการสร้างลายเส้น



รูปที่ 4.10 แสดงการทำงานของโปรแกรมควบคุมการสร้างลายเส้น

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการสร้างลายเส้น อธิบายได้ดังนี้

โปรแกรมส่วนนี้ทำหน้าที่ควบคุมตัวชี้ในการสร้างลายเส้น โดยมีลำดับขั้นในการทำงานโปรแกรมดังนี้

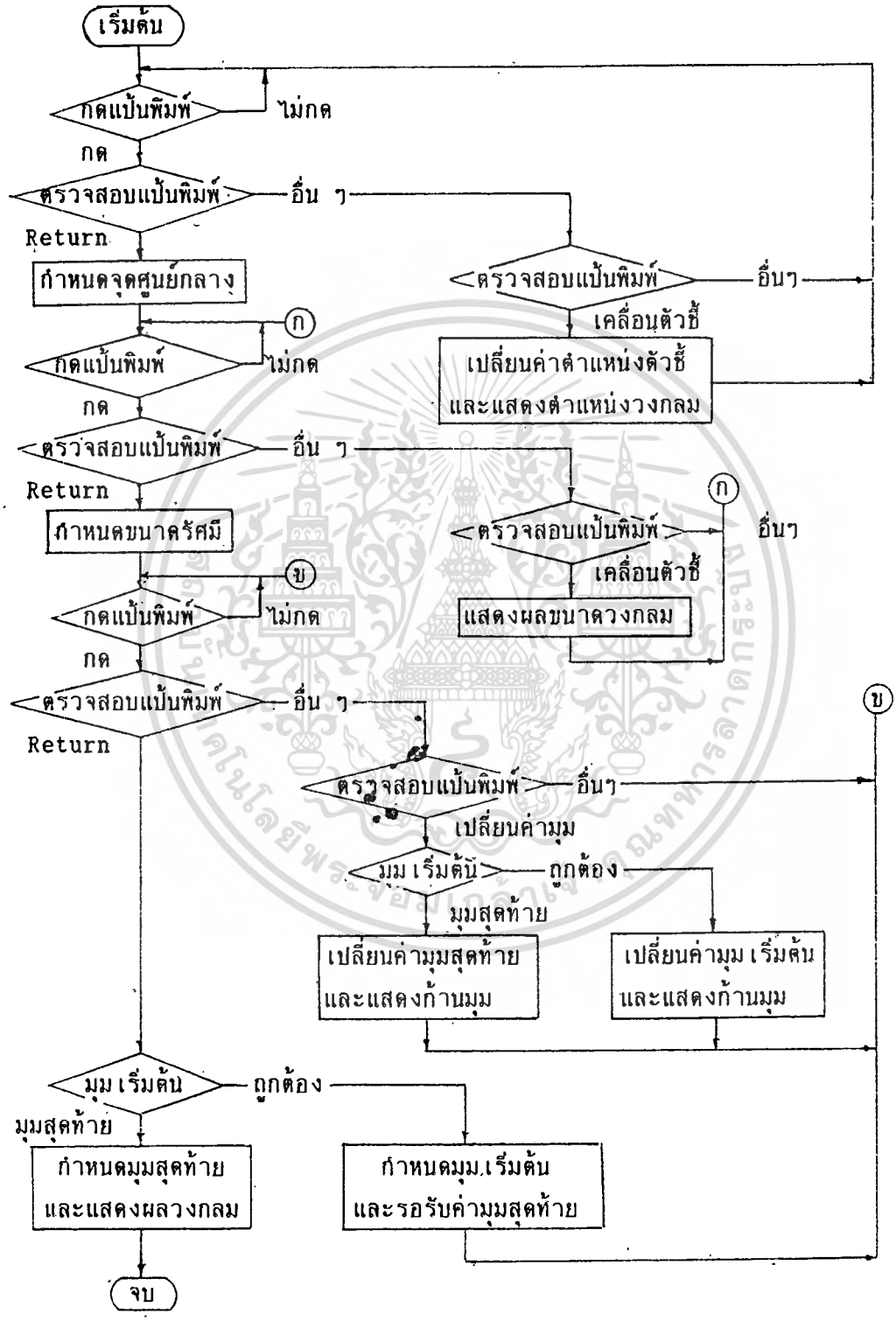
- กำหนดค่าตัวแปร KeyCount ที่ต้องการใช้เป็นตัวนับให้มีค่าเริ่มต้นเท่ากับศูนย์
- ตรวจสอบการกดแป้นคีย์

  ในกรณีที่ไม่มีการกดตัวแปร KeyCount จะถูกเพิ่มค่าทีละหนึ่ง และจะแสดงผลเส้นแสดงแนวการลาก เมื่อตัวแปรมีค่าเท่ากับ 500 ซึ่งเสมือนกับเป็นตัวควบคุมเวลาในการแสดงผล

  ส่วนถ้ามีการกดแป้นคีย์แล้ว ค่าตัวแปรที่ใช้ในการนับจะมีค่าเท่ากับศูนย์รวมทั้งลบเส้นแสดงแนวการลากออก ถ้ามีการแสดงผล และ นำคำสั่งที่ได้จากการกดแป้นคีย์เก็บไว้ในตัวแปร KeyCommand เพื่อการตรวจสอบในขั้นตอนต่อไป
- ตรวจสอบคำสั่ง โดยสามารถแบ่งการทำงานกำสั่งออกได้ดังนี้
  - แป้นคีย์ควบคุมทิศทาง และตำแหน่งตัวชี้ เคลื่อนที่ตัวชี้จากตำแหน่งที่อยู่ในปัจจุบันไปยังตำแหน่งใหม่ที่ต้องการ
  - แป้นพิมพ์ควบคุมการเปลี่ยนด้านแผ่นวงจร จะทำการเปลี่ยนด้านแผ่นวงจรพิมพ์เพื่อสร้าง และแก้ไขลายวงจร
  - คำสั่ง F1 จะทำหน้าที่ลากเส้น โดยตรวจสอบการกำหนดจุดเริ่มต้นถ้ายังไม่ได้กำหนดให้กำหนดจุดเริ่มต้น แต่ถ้ามีการกำหนดจุดเริ่มต้นแล้วจะลากเส้นจากจุดเริ่มต้นไปยังตำแหน่งตัวชี้
  - คำสั่ง F2 จะทำหน้าที่ยกเลิกการกำหนดจุดเริ่มต้น
  - คำสั่งอื่น ๆ จะถูกนำไปตรวจสอบต่อไป

เมื่อสิ้นสุดการทำงานโปรแกรมนี้ จะกลับเข้าสู่โปรแกรมควบคุมการสร้างภาพ เพื่อนำค่าคำสั่งที่ได้รับไปทำงาน

4.10 โปรแกรมควบคุมการสร้างวงกลม



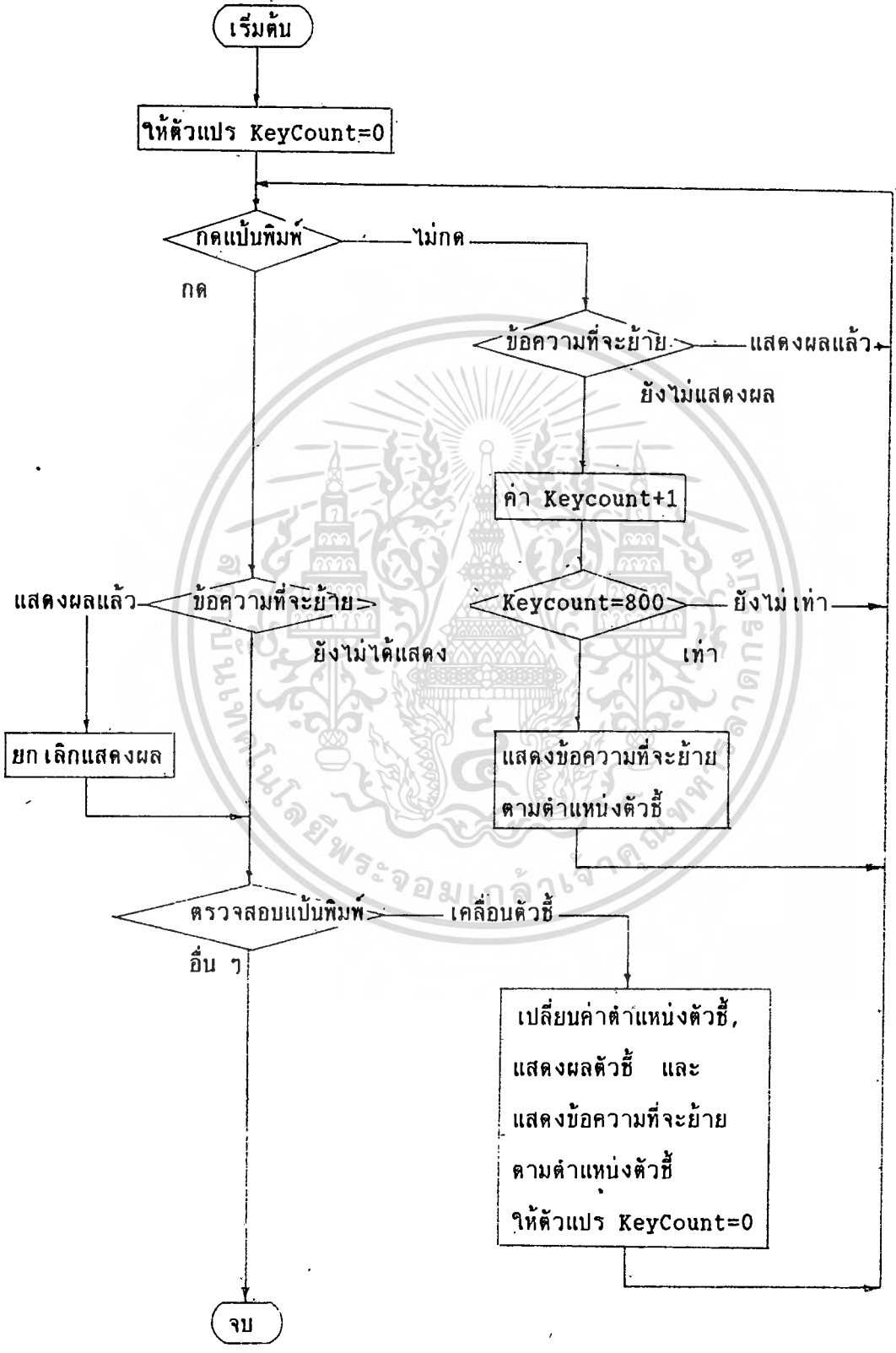
รูปที่ 4.11 แสดงการทำงานของโปรแกรมควบคุมการสร้างวงกลม

โฟลว์ชาร์ทการทำงานของโปรแกรมส่วนควบคุมการสร้างวงกลม อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ควบคุมการเคลื่อนที่ตัวชี้ให้ไปยังตำแหน่งที่ต้องการสร้างวงกลม โดยการกำหนดตำแหน่งจุดศูนย์กลาง ขนาดรัศมี รวมทั้งมุมเริ่มต้น และมุมสุดท้ายในการสร้างส่วนโค้งของวงกลม มีการทำงานดังนี้

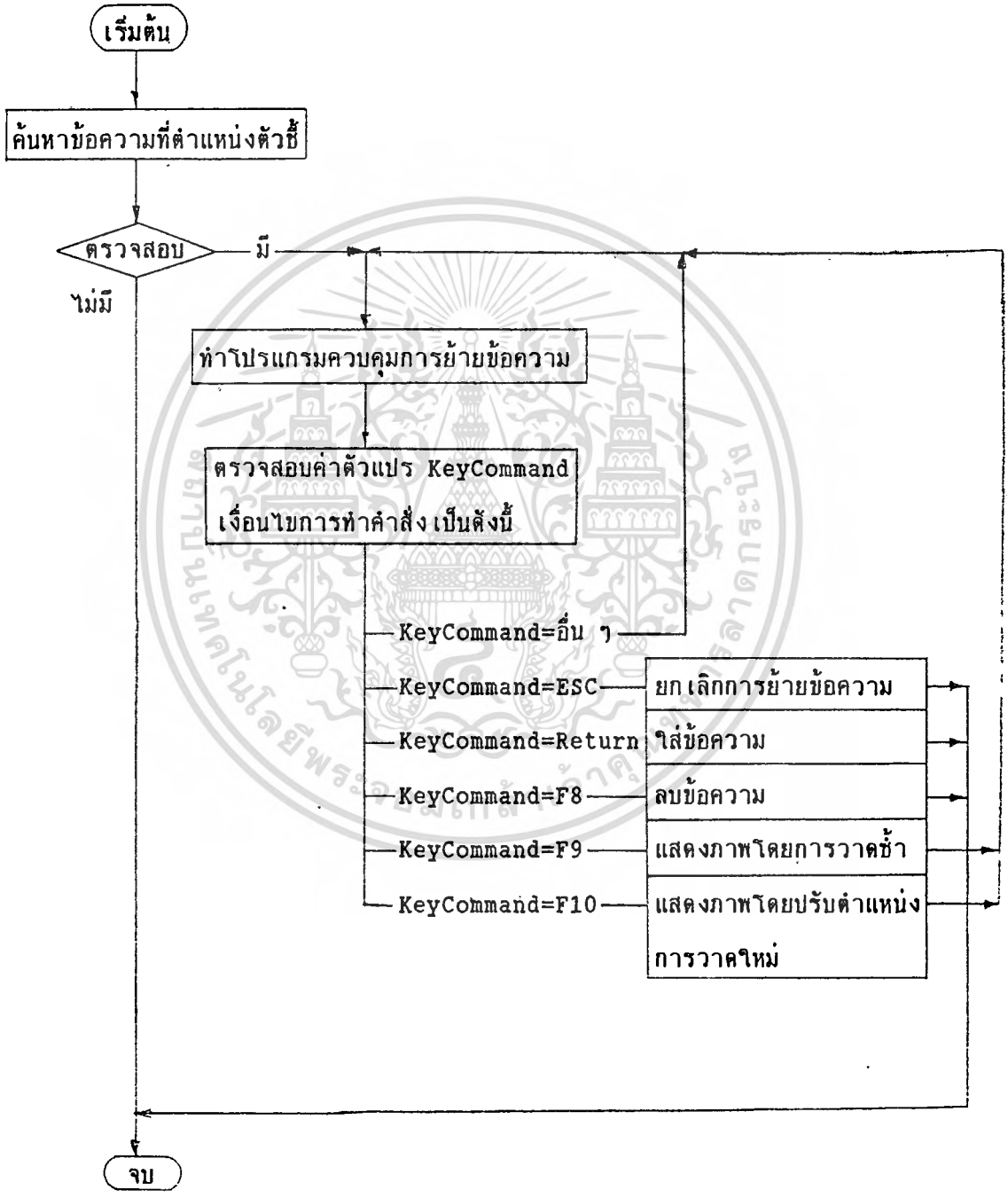
- ตรวจสอบการกดแป้นพิมพ์ ถ้าไม่มีการกดให้รอรับจนกว่าจะมีการกด
- ตรวจสอบค่าที่ได้จากแป้นพิมพ์ ถ้ามีค่าไม่เท่ากับแป้นพิมพ์ Return แล้วนำไปตรวจสอบค่ากับแป้นพิมพ์ควบคุมทิศทาง และตำแหน่งตัวชี้ ถ้าใช่เปลี่ยนค่าตำแหน่งของตัวชี้ พร้อมทั้งแสดงภาพของตัวชี้ในตำแหน่งใหม่ และกลับไปรอรับการกดแป้นพิมพ์ใหม่ จนกว่าแป้นพิมพ์ที่ได้มีค่าเท่ากับแป้นพิมพ์ Return จึงจะเป็นการกำหนดค่าตำแหน่งจุดศูนย์กลางของวงกลม และรอรับการกำหนดค่ารัศมีต่อไป
- ตรวจสอบการกดแป้นพิมพ์ ถ้าไม่มีการกดให้รอรับจนกว่าจะมีการกด
- ตรวจสอบค่าที่ได้จากแป้นพิมพ์ ถ้ามีค่าไม่เท่ากับแป้นพิมพ์ Return แล้วนำไปตรวจสอบค่ากับแป้นพิมพ์ควบคุมขนาดรัศมี ถ้าใช่เปลี่ยนค่ารัศมี และแสดงผลขนาดวงกลม และกลับไปรอรับการกดแป้นพิมพ์ใหม่ จนกว่าแป้นพิมพ์ที่ได้มีค่าเท่ากับแป้นพิมพ์ Return จึงจะทำการกำหนดขนาดของรัศมีของวงกลม และรอรับค่ามุมจากแป้นพิมพ์ต่อไป
- ตรวจสอบการกดแป้นพิมพ์ และรอรับการกดแป้นพิมพ์จนกระทั่งมีการกดเกิดขึ้น
- ตรวจสอบค่าที่ได้จากแป้นพิมพ์ ถ้าเป็นแป้นพิมพ์ควบคุมความกว้างของมุมในการสร้างส่วนโค้งจะนำค่าที่ได้นั้นเป็นจุดเริ่มต้นของส่วนโค้ง พร้อมทั้งแสดงผลก้านมุมให้กับผู้ใช้ได้เห็น และกลับไปรอรับแป้นพิมพ์ใหม่ ทำจนกระทั่งมุมที่ได้นั้นเป็นที่ต้องการ และแป้นพิมพ์ที่ได้เท่ากับ Return ก็จะรอรับการกำหนดค่ามุมสุดท้ายต่อไป ซึ่งมีวิธีการรับค่ามุมเหมือนกับการกำหนดค่ามุมเริ่มต้น แตกต่างกันที่การแสดงผลของก้านมุม ซึ่งจะวาดส่วนโค้งของวงกลมจากตำแหน่งมุมเริ่มต้นถึงมุมสุดท้ายที่ได้กำหนดด้วย ถ้ามุมเริ่มต้นมีค่าเท่ากับมุมสุดท้ายแล้วจะทำการสร้างวงกลมลงบนแผ่นวงจรพิมพ์ และถ้าไม่เท่ากันแล้วจะทำการสร้างส่วนโค้งของวงกลมโดยเริ่มตั้งแต่ตำแหน่งของมุมเริ่มต้นหมุนทวนเข็มนาฬิกาไปจนกระทั่งถึงตำแหน่งมุมสุดท้าย โดยมีขนาดรัศมีตามค่าที่ได้กำหนดไว้
- สิ้นสุดการสร้างวงกลม

4.11 โปรแกรมควบคุมการย้ายข้อความ



รูปที่ 4.12 การทำงานของโปรแกรมควบคุมการย้ายข้อความ

4.12 โปรแกรมย้ายข้อความ



รูปที่ 4.13 แสดงการทำงานของโปรแกรมย้ายข้อความ

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการย้ายข้อความ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ควบคุมการ เคลื่อนที่ตัวชี้ในการกำหนดตำแหน่งการจัดวางข้อความในแผ่นวงจรพิมพ์ มีการทำงานดังนี้

- กำหนดค่าตัวแปร KeyCount ที่ต้องการให้ เป็นตัวนับมีค่า เริ่มต้น เท่ากับศูนย์
- ตรวจสอบการกดแป้นพิมพ์ ในกรณีที่ไม่มีการกดแป้นพิมพ์ ค่าตัวแปรจะถูก เพิ่มค่าขึ้นทีละหนึ่ง และจะแสดงขอบ เขตข้อความ เมื่อตัวแปรมีค่า เท่ากับ 500 รวมทั้งยก เลิกการนับ เพื่อให้การแสดงผลกระทำได้ เพียงครั้ง เดียว และ เมื่อตรวจพบว่ามีการกดแป้นพิมพ์แล้ว จะลบการแสดงขอบ เขตข้อความ ถ้ามีการแสดงผล
- ตรวจสอบค่าที่ได้จากแป้นพิมพ์ ถ้ามีค่า เท่ากับแป้นพิมพ์ควบคุมทิศทาง และตำแหน่งตัวชี้จะ เปลี่ยนค่าตำแหน่งตัวชี้ และแสดงผลตัวชี้ในตำแหน่งใหม่ แต่ถ้าการตรวจสอบแป้นพิมพ์มีค่า เป็นคำสั่งอื่น ๆ แล้วจะกลับ เข้าสู่โปรแกรมย้ายข้อความ เพื่อนำคำสั่งที่ได้รับไปตรวจสอบ และทำงานต่อไป

โฟลว์ชาร์ทการทำงานของโปรแกรมย้ายข้อความ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ควบคุมการ เคลื่อนที่ตัวชี้ในการกำหนดตำแหน่งการจัดวางข้อความ โดยการตรวจสอบค้นหาข้อความบนแผ่นวงจรพิมพ์ก่อน มีการทำงานดังนี้

- ค้นหาข้อความในตำแหน่งตัวชี้ เพื่อที่จะนำข้อความ เคลื่อนย้ายไปสู่ตำแหน่งที่ต้องการ
- ตรวจสอบผลลัพท์การค้นหา กรณีที่ไม่พบข้อความจะกลับไปทำงานโปรแกรมควบคุมการสร้างภาพ และถ้าการตรวจสอบพบว่ามีข้อความแล้วจะทำงานโปรแกรมควบคุมการย้ายข้อความต่อไป รวมทั้งนำคำสั่งที่ได้จากโปรแกรมส่วนนี้มาตรวจสอบ และกระทำตามคำสั่งดังนี้

  F8 : ลบข้อความ

  F9 : ทำโปรแกรมแสดงภาพโดยการวาดซ้ำ

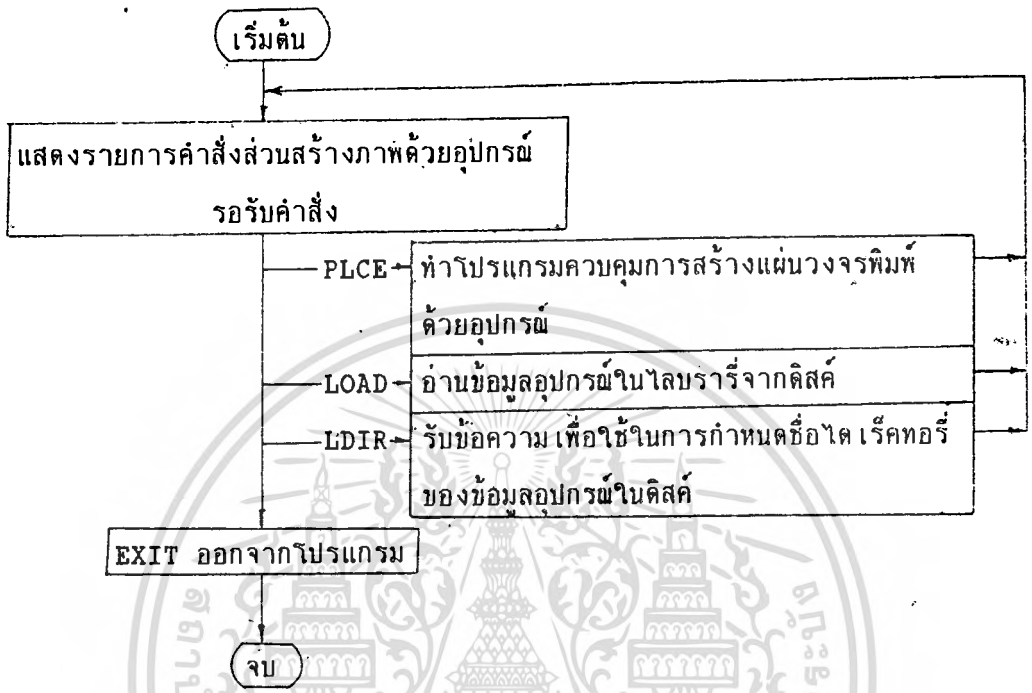
  F10 : ทำโปรแกรมแสดงภาพโดยปรับตำแหน่งการวาดใหม่

  ESC : ยกเลิกการย้ายข้อความ

  Return : ใส่ข้อความ

  เมื่อสิ้นสุดการทำคำสั่งต่าง ๆ จะกลับ เข้าสู่โปรแกรมควบคุมการสร้างภาพต่อไป

4.13 โปรแกรมสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์



รูปที่ 4.14 แสดงการทำงานโปรแกรมสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์

โฟลว์ชาร์ทการทำงานของโปรแกรมสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ควบคุมคำสั่งที่เกี่ยวกับการใช้งานอุปกรณ์ต่าง ๆ ที่ได้สร้างเก็บไว้ มีการทำงานตามลำดับขั้นตอนดังนี้

- แสดงรายการคำสั่งส่วนควบคุมการใช้งานอุปกรณ์
- รอรับคำสั่ง
- ตรวจสอบคำสั่ง ถ้าตรวจพบว่า เป็นคำสั่งที่มีใช้และตรงกับข้อความ เมนูที่ได้แสดงผลจะมีการทำงานโปรแกรมตามคำสั่งดังนี้

  PLCE : ทำโปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์

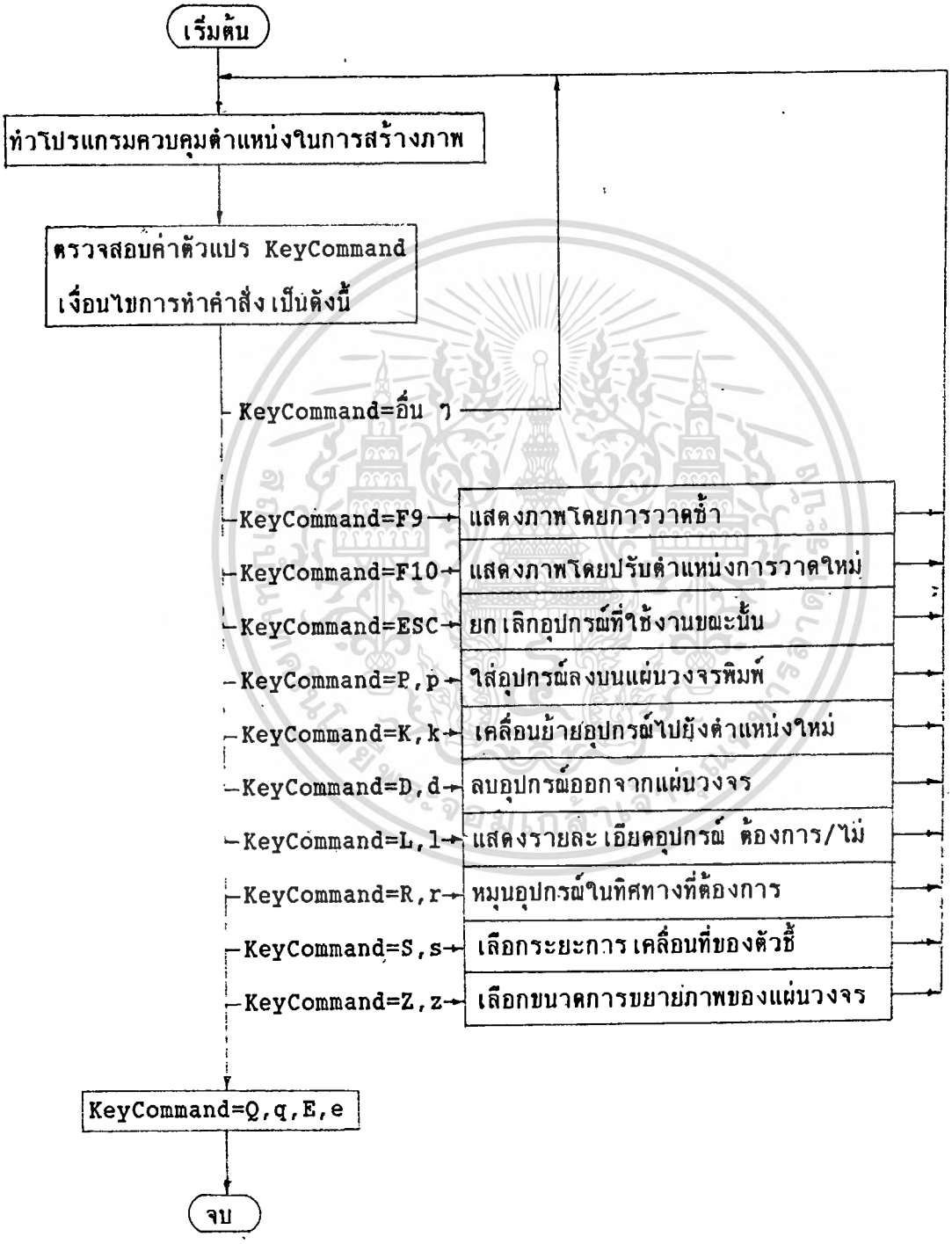
  LOAD : อ่านข้อมูลอุปกรณ์ในไลบรารี่จากดิสค์

  LDIR : รับข้อความ เพื่อใช้ในการกำหนดชื่อไดเร็คทอรี่ของข้อมูลอุปกรณ์ในดิสค์

  EXIT : ยกเลิกการทำงานของโปรแกรมสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์

  ถ้าผลของการตรวจสอบพบว่าคำสั่งนั้นไม่ตรงกับคำสั่งที่มีให้ใช้ โปรแกรมจะกลับไปทำงานในตำแหน่ง เริ่มต้นใหม่

4.14 โปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์



รูปที่ 4.15 แสดงการทำงานของโปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการสร้างแผ่นวงจรพิมพ์ด้วยอุปกรณ์ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ จะทำหน้าที่จัดการด้านคำสั่งที่ใช้ในการการควบคุมการสร้างแผ่นวงจรพิมพ์ มีการทำงานตามลำดับขั้นตอนดังนี้

- ทำงานโปรแกรมควบคุมตำแหน่งในการสร้างภาพ โดยการควบคุมทิศทางการเคลื่อนที่ของตัวชี้ และอุปกรณ์ต่าง ๆ บนแผ่นวงจรพิมพ์ รวมทั้งนำค่าคำสั่งต่าง ๆ ที่ได้รับมาให้กับส่วนตรวจสอบ เพื่อทำงานต่อไป ถ้าคำสั่งที่ส่งมาตรวจสอบพบว่าไม่เป็นคำสั่งที่มีให้ใช้ จะกลับไปทำงานส่วนนี้ซ้ำ
- ส่วนตรวจสอบคำสั่งจากค่าตัวแปร KeyCommand เพื่อสั่งงานควบคุมตามคำสั่ง ๆ ที่ได้รับ ดังนี้

  F9 : ทำโปรแกรมแสดงภาพโดยการวาดซ้ำ

  F10 : ทำโปรแกรมแสดงภาพโดยปรับตำแหน่งการวาดใหม่

  ESC : ยกเลิกอุปกรณ์ที่นำมาใช้งานจากดิสค์

  P,p : ใส่อุปกรณ์ลงบนแผ่นวงจรพร้อมทั้งใส่รายละเอียดชื่อของอุปกรณ์

  K,k : นำอุปกรณ์ที่ตรวจพบ ณ.ตำแหน่งตัวชี้ ออกจากแผ่นวงจรพิมพ์ เพื่อการเคลื่อนย้ายอุปกรณ์ไปยังตำแหน่งใหม่

  D,d : ลบอุปกรณ์ที่ตรวจพบ ณ.ตำแหน่งตัวชี้ออกจากแผ่นวงจรพิมพ์

  L,l : กำหนดค่าสภาวะในการแสดงรายละเอียดภาพอุปกรณ์ ก่อนใส่อุปกรณ์ลงบนแผ่นวงจรพิมพ์

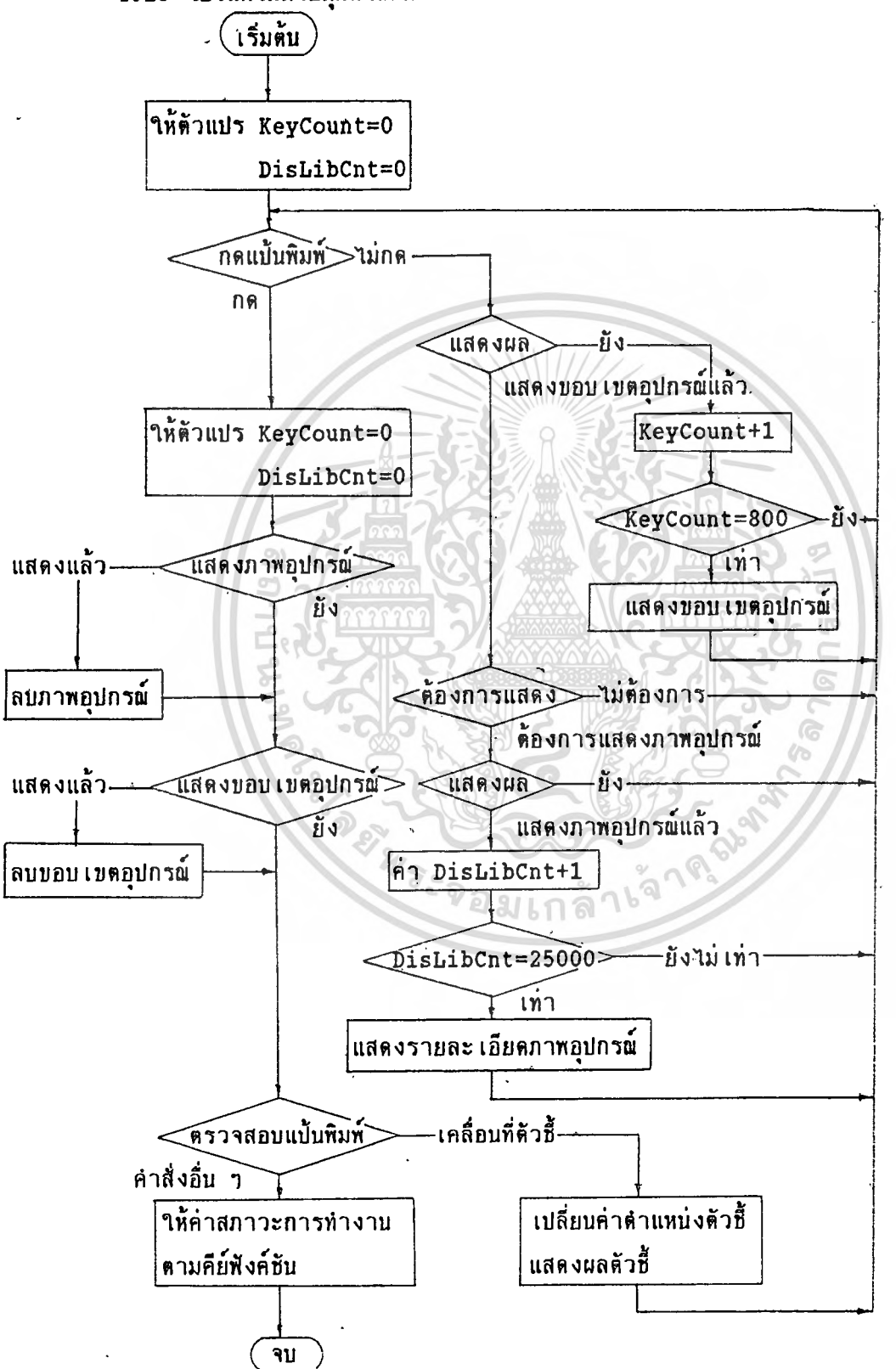
  R,r : หมุนอุปกรณ์ในทิศทางที่ต้องการ โดยมีจุดอ้างอิงของอุปกรณ์เป็นจุดหมุน มีทิศทางการหมุนดังนี้ 0, 90, 180 และ 270 องศา

  S,s : เลือกระยะในการเคลื่อนที่ของตัวชี้

  Z,z : ทำโปรแกรมแสดงภาพขนาดขยาย

คำสั่งต่าง ๆ เหล่านี้ เมื่อสิ้นสุดการทำงานแล้วจะกลับไปทำงานเริ่มต้นซ้ำ จนกว่า ค่าตัวแปร KeyCommand มีค่าเท่ากับ Q, q, E หรือ e ซึ่งหมายถึงการยกเลิกการทำงาน

4.15 โปรแกรมควบคุมตำแหน่งในการสร้างภาพ



รูปที่ 4.16 แสดงการทำงานของโปรแกรมควบคุมตำแหน่งในการสร้างภาพ

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมตำแหน่งในการสร้างภาพ อธิบายได้ดังนี้

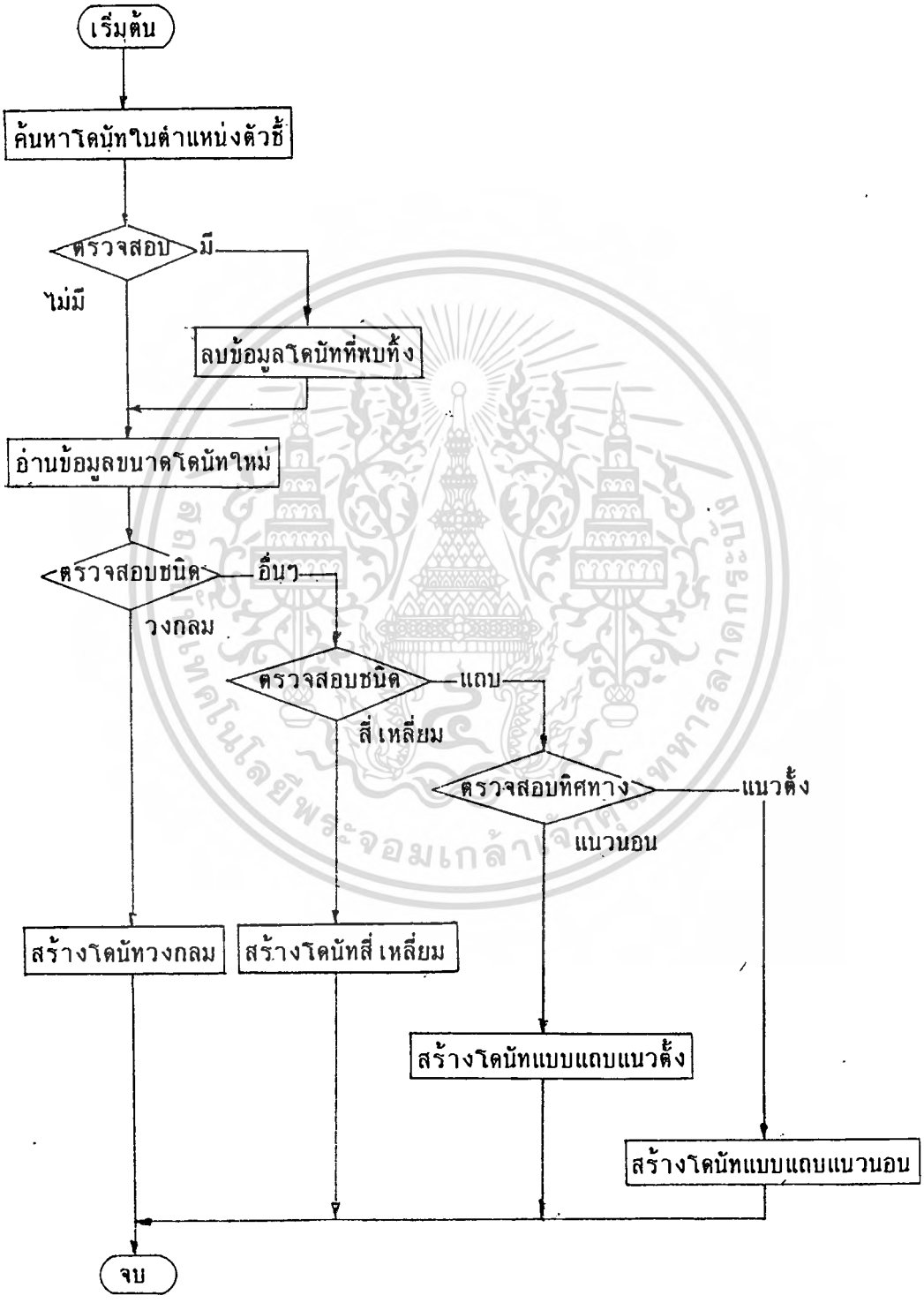
โปรแกรมส่วนนี้ ทำหน้าที่ ควบคุมตำแหน่งตัวชี้ และ ภาพอุปกรณ์ให้เคลื่อนที่ไปยังตำแหน่งที่ต้องการบนแผ่นวงจรพิมพ์ มีลำดับขั้นการทำงานดังนี้

- กำหนดค่าตัวแปร KeyCount และ DisLibCnt ที่ต้องการให้เป็นตัวนับมีค่าเริ่มต้นเท่ากับศูนย์
- ตรวจสอบการกดแป้นพิมพ์ ในกรณีที่ไม่มีการกดตัวแปร KeyCount จะถูกเพิ่มค่าที่ละหนึ่ง และจะแสดงผลขอบเขตของอุปกรณ์ เมื่อค่าตัวแปร KeyCount มีค่าเท่ากับ 800 ในระหว่างแสดงขอบเขตของอุปกรณ์เรียบร้อยแล้วถ้ายังตรวจพบว่าไม่มีการกดแป้นพิมพ์จนตัวแปร Disubcnt มีค่าเท่ากับ 25,000 และค่าสภาวะในการแสดงรายละเอียดภาพอุปกรณ์จากคำสั่ง LCur เป็นจริงแล้วภาพที่แสดงขอบเขตของอุปกรณ์จะถูกลบออก และจะแสดงภาพรายละเอียดอุปกรณ์เพื่อให้ผู้ใช้เห็นรายละเอียดภาพของอุปกรณ์ ก่อนการนำใส่ลงบนแผ่นวงจรพิมพ์

ถ้าตรวจพบว่ามีการกดแป้นพิมพ์ ตัวแปรต่าง ๆ ที่ใช้เป็นตัวนับจะถูกกำหนดค่าให้เท่ากับศูนย์ รวมทั้งลบภาพแสดงขอบเขตอุปกรณ์ หรือภาพรายละเอียดอุปกรณ์ถ้ามีการแสดงผล

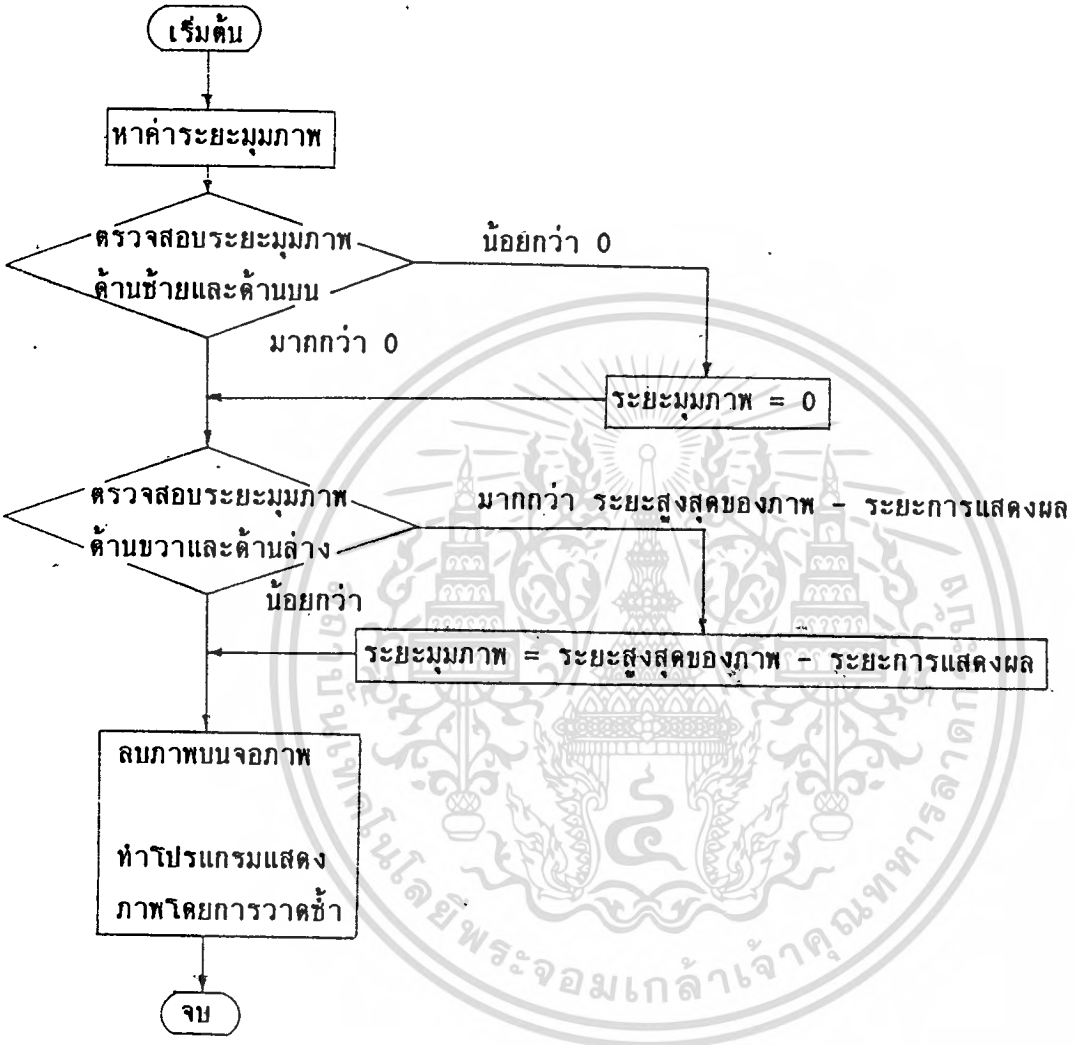
- ตรวจสอบแป้นพิมพ์ควบคุมทิศทางและตำแหน่งตัวชี้ กรณีที่ตรวจพบว่าใช่แล้วเปลี่ยนค่าตำแหน่งตัวชี้ และแสดงภาพตัวชี้ในตำแหน่งใหม่ รวมทั้งกลับไปเริ่มต้นการทำงานใหม่ แต่ถ้าการตรวจสอบแป้นพิมพ์พบว่าเป็นคำสั่งอื่น ๆ จะนำค่าคำสั่งเก็บไว้ในตัวแปร KeyCommand และออกจากโปรแกรมเพื่อนำค่าคำสั่งไปตรวจสอบการทำงานต่อไป

4.16 โปรแกรมใส่โคนัท



รูปที่ 4.17 แสดงการทำงานของโปรแกรมใส่โคนัท

4.17 โปรแกรมแสดงภาพโดยปรับตำแหน่งการวาดใหม่



รูปที่ 4.18 แสดงการทำงานของโปรแกรมแสดงภาพโดยการปรับตำแหน่งตัวชี้

หมายเหตุ

ระยะมุมภาพ : ตำแหน่งมุมจอภาพด้านซ้ายบน เทียบกับระยะมุมภาพของแผ่นลายวงจรด้านซ้ายบน
มีค่าเท่ากับ
ระยะทางของตำแหน่งตัวชี้ - (ระยะการแสดงผล / 2)

ระยะการแสดงผล : ระยะการแสดงผลบนจอภาพที่สามารถแสดงผลแผ่นลายวงจรได้

ระยะสูงสุดของภาพ : ระยะขอบเขตของภาพแผ่นลายวงจรสูงสุดที่สามารถสร้างได้

โฟลว์ชาร์ทการทำงานของโปรแกรมใส่โดนัท อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ควบคุมการสร้างโดนัทให้กับแผ่นวงจรพิมพ์โดยมีการนำค่าสภาวะในการกำหนดทิศทาง ชนิด และขนาดมาพิจารณาก่อน เพื่อให้เป็นไปตามจุดประสงค์ของผู้ใช้ มีการทำงานโปรแกรมดังนี้

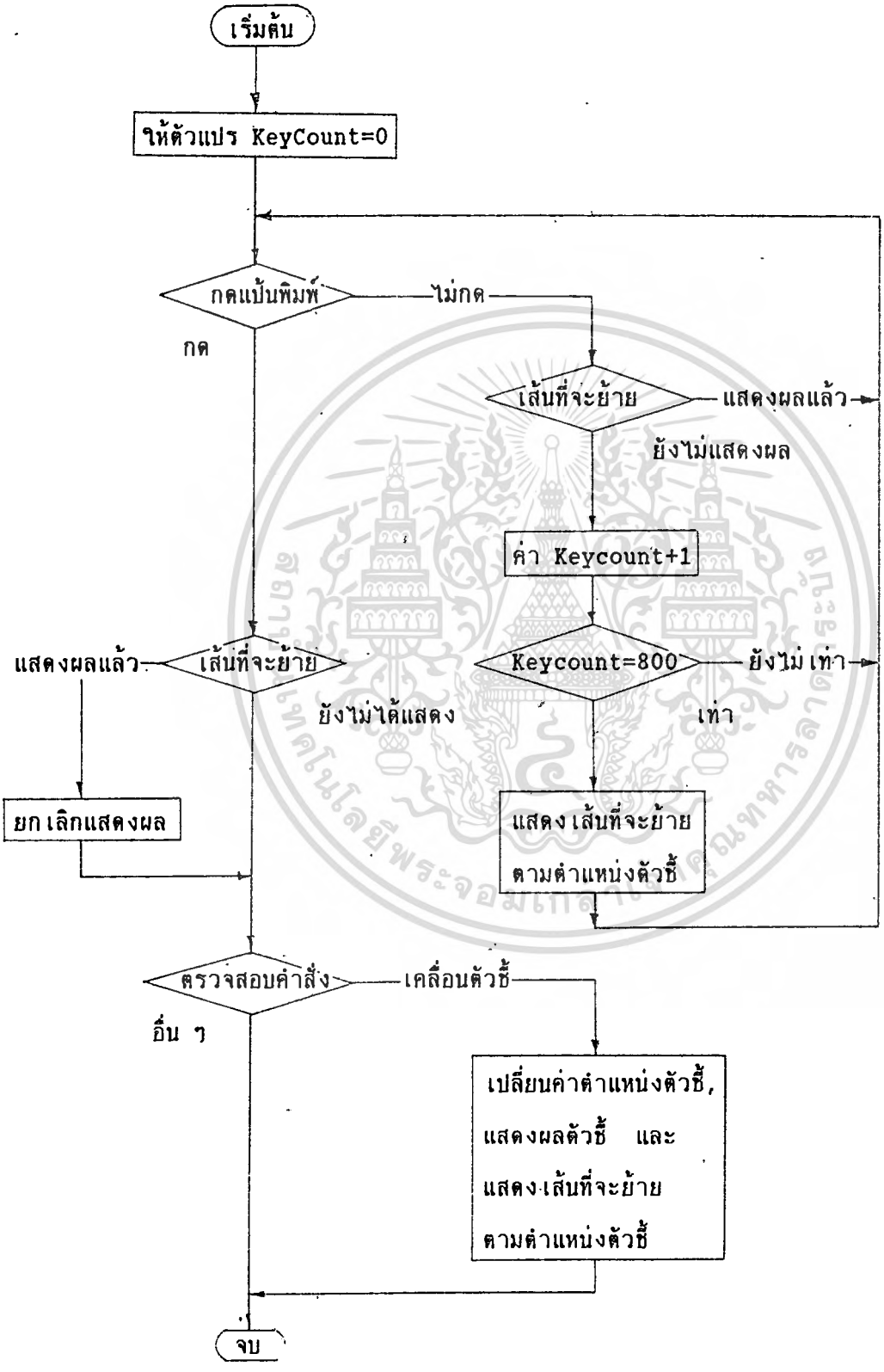
- ค้นหาโดนัทในตำแหน่งตัวชี้ เพื่อลดปัญหาการเกิดข้อมูลซ้ำ โดยจะตรวจสอบจากข้อมูลโดนัทของแผ่นวงจรพิมพ์ถ้าตรวจพบว่ามีโดนัทที่ตำแหน่งนั้นจะลบโดนัทนั้นออก
- อ่านข้อมูลขนาดโดนัทจากค่าสภาวะที่ผู้ใช้กำหนดไว้ และ นำมาตรวจสอบเพื่อแยกประเภทการทำงาน ถ้าการตรวจสอบพบว่าโดนัทเป็นแบบวงกลม หรือ สี่เหลี่ยมจัตุรัสก็ให้วาดโดนัทนั้นลงบนแผ่นวงจรพิมพ์ได้เลย ส่วนถ้าพบว่าโดนัทนั้นเป็นแบบแถบจะมีการตรวจสอบทิศทางของแถบซึ่งแบ่งได้เป็นแนวนอน และแนวตั้ง ถ้าตรงกับแบบใดก็สร้างโดนัทแบบนั้นขึ้นในด้านการทำงานของแผ่นวงจรพิมพ์นั้น

โฟลว์ชาร์ทการทำงานของโปรแกรมแสดงภาพโดยปรับตำแหน่งการวาดใหม่ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่นำข้อมูลของแผ่นวงจรพิมพ์มาสร้างภาพใหม่ โดยจะนำค่าตำแหน่งตัวชี้ที่ภาพเดิมมาสร้างเป็นภาพใหม่ ที่มีการปรับให้ภาพในตำแหน่งตัวชี้นั้นมาอยู่ในตำแหน่งกึ่งกลางของจอภาพ ทำให้สามารถเคลื่อนย้ายการแสดงผลภาพของแผ่นวงจรพิมพ์เพื่อการสร้างและแก้ไขแผ่นวงจรพิมพ์ในส่วนอื่น ๆ มีการทำงานโปรแกรมดังนี้

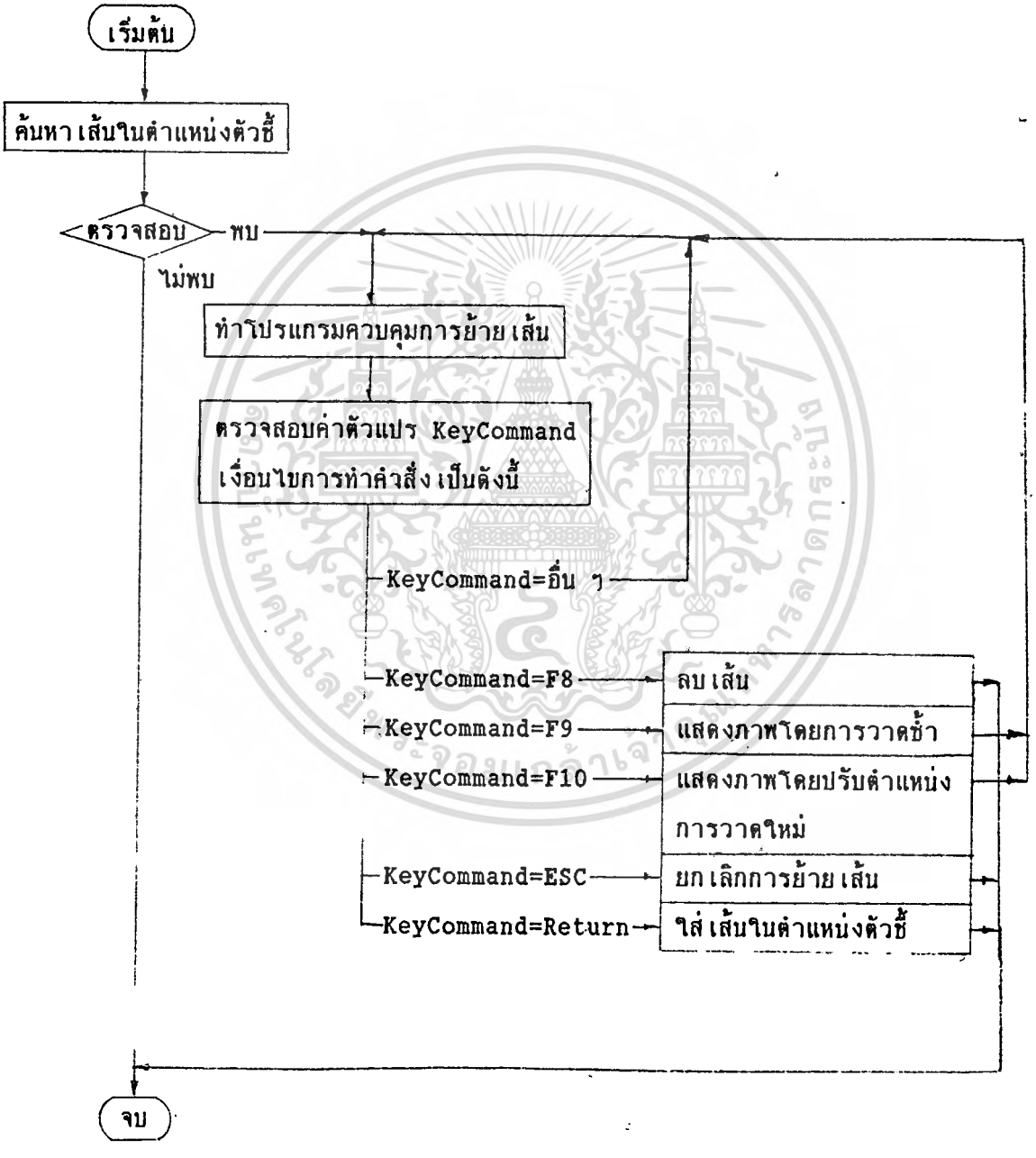
- หาค่าระยะมุมภาพ เพื่อให้ทราบถึงมุมภาพแสดงผลว่าอยู่ในตำแหน่งใดของแผ่นวงจรพิมพ์
- ตรวจสอบระยะมุมภาพ ถ้ามีค่ามากกว่า หรือน้อยกว่าขอบเขตสูงสุด และต่ำสุดของแผ่นวงจรพิมพ์แล้วปรับค่าระยะมุมภาพใหม่เหมาะสม
- ลบภาพบนจอภาพ
- วาดภาพใหม่ เพื่อให้ภาพในตำแหน่งนั้นปรากฏบนจอภาพ เพื่อรอการแก้ไข และเพิ่มเติมลายเส้นของแผ่นวงจรพิมพ์ต่อไป

4.18 โปรแกรมควบคุมการย้ายเส้น



รูปที่ 4.19 แสดงการทำงานของโปรแกรมควบคุมการย้ายเส้น

4.19 โปรแกรมเคลื่อนย้ายเส้น



รูปที่ 4.20 การทำงานของโปรแกรมเคลื่อนย้ายเส้น

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการย้ายเส้น อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ควบคุมการเคลื่อนที่ของตัวชี้ เพื่อกำหนดตำแหน่งในการจัดวางเส้นที่ต้องการเคลื่อนย้ายบนแผ่นวงจรพิมพ์ มีลำดับขั้นการทำงานดังนี้

- กำหนดค่าตัวแปร KeyCount ที่ต้องการให้เป็นตัวนับมีค่าเริ่มต้นเท่ากับศูนย์
- ตรวจสอบการกดแป้นพิมพ์ ในกรณีที่ไม่มีการกดใด ๆ ค่าตัวแปรจะถูกเพิ่มค่าขึ้นทีละหนึ่ง และจะแสดงเส้นที่ต้องการย้ายเมื่อตัวแปรมีค่าเท่ากับ 800 รวมทั้งยกเลิกการนับเพื่อให้การแสดงผลกระทำได้เพียงครั้งเดียว และเมื่อตรวจพบว่ามีการกดแป้นพิมพ์จะทำการลบการแสดงเส้นที่ต้องการย้ายออก ถ้ามีการแสดงผล
- ตรวจสอบค่าที่ได้จากแป้นพิมพ์ ถ้ามีค่าเท่ากับแป้นพิมพ์ควบคุมทิศทาง และตำแหน่งตัวชี้จะเปลี่ยนค่าตำแหน่งตัวชี้ และแสดงผลตัวชี้ในตำแหน่งใหม่ แต่ถ้าการตรวจสอบแป้นพิมพ์มีค่าเป็นคำสั่งอื่น ๆ แล้วจะกลับเข้าสู่โปรแกรมเคลื่อนย้ายเส้น เพื่อนำคำสั่งที่ได้รับไปตรวจสอบ และทำงานต่อไป

โฟลว์ชาร์ทการทำงานของโปรแกรมเคลื่อนย้ายเส้น อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่เคลื่อนย้ายเส้นในตำแหน่งตัวชี้ไปใส่ไว้ในตำแหน่งที่ต้องการ มีการทำงานโปรแกรมดังนี้

- ค้นหาเส้นในตำแหน่งตัวชี้ โดยตรวจสอบจากข้อมูลลายเส้นของแผ่นวงจร ถ้าการตรวจสอบพบว่าไม่มีเส้นในตำแหน่งนั้น จะออกจากโปรแกรมนี้ทันที ส่วนถ้าการตรวจสอบพบว่ามีเส้นในตำแหน่งนั้น จะทำงานโปรแกรมควบคุมการย้ายเส้นซึ่งจะนำเส้นที่ต้องการเคลื่อนย้ายไปยังตำแหน่งที่ต้องการ และให้ค่าคำสั่งจากผู้ใช้นำมาตรวจสอบการทำงานดังนี้

  F8 : ลบเส้น

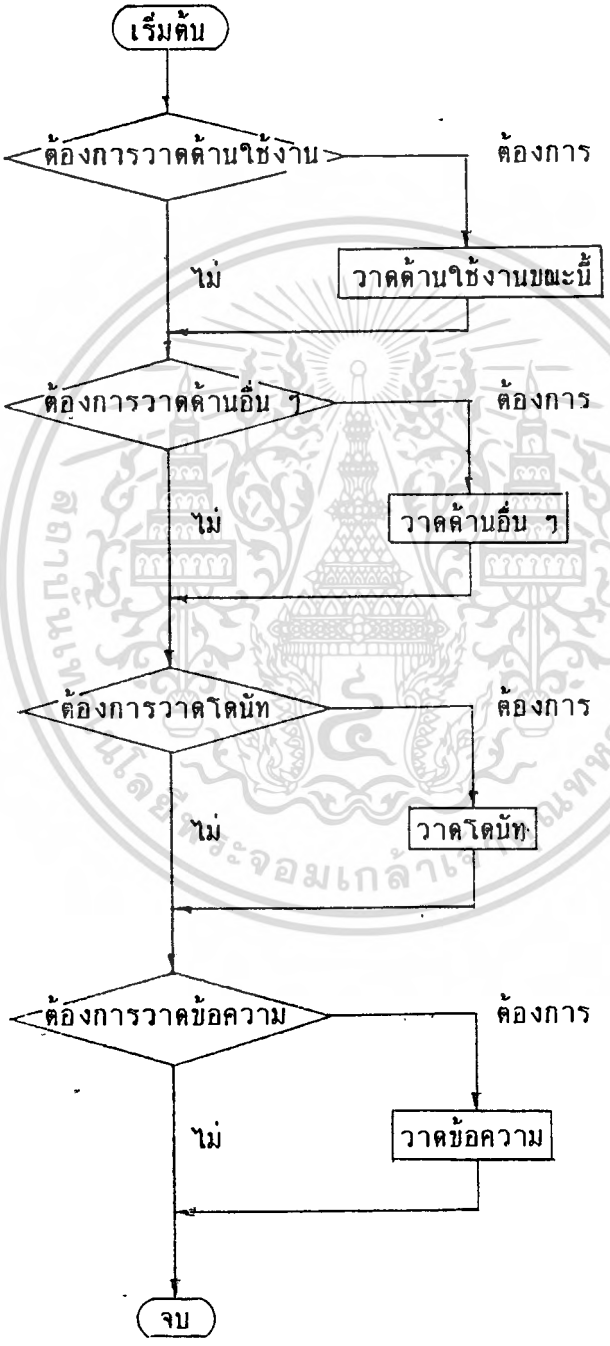
  F9 : ทำโปรแกรมแสดงภาพโดยการวาดซ้ำ

  F10 : ทำโปรแกรมแสดงภาพโดยปรับตำแหน่งการวาด

  ESC : ยกเลิกการย้ายเส้น

  Return : ใส่เส้นในตำแหน่งตัวชี้

4.20 โปรแกรมแสดงภาพโดยการวาดซ้ำ



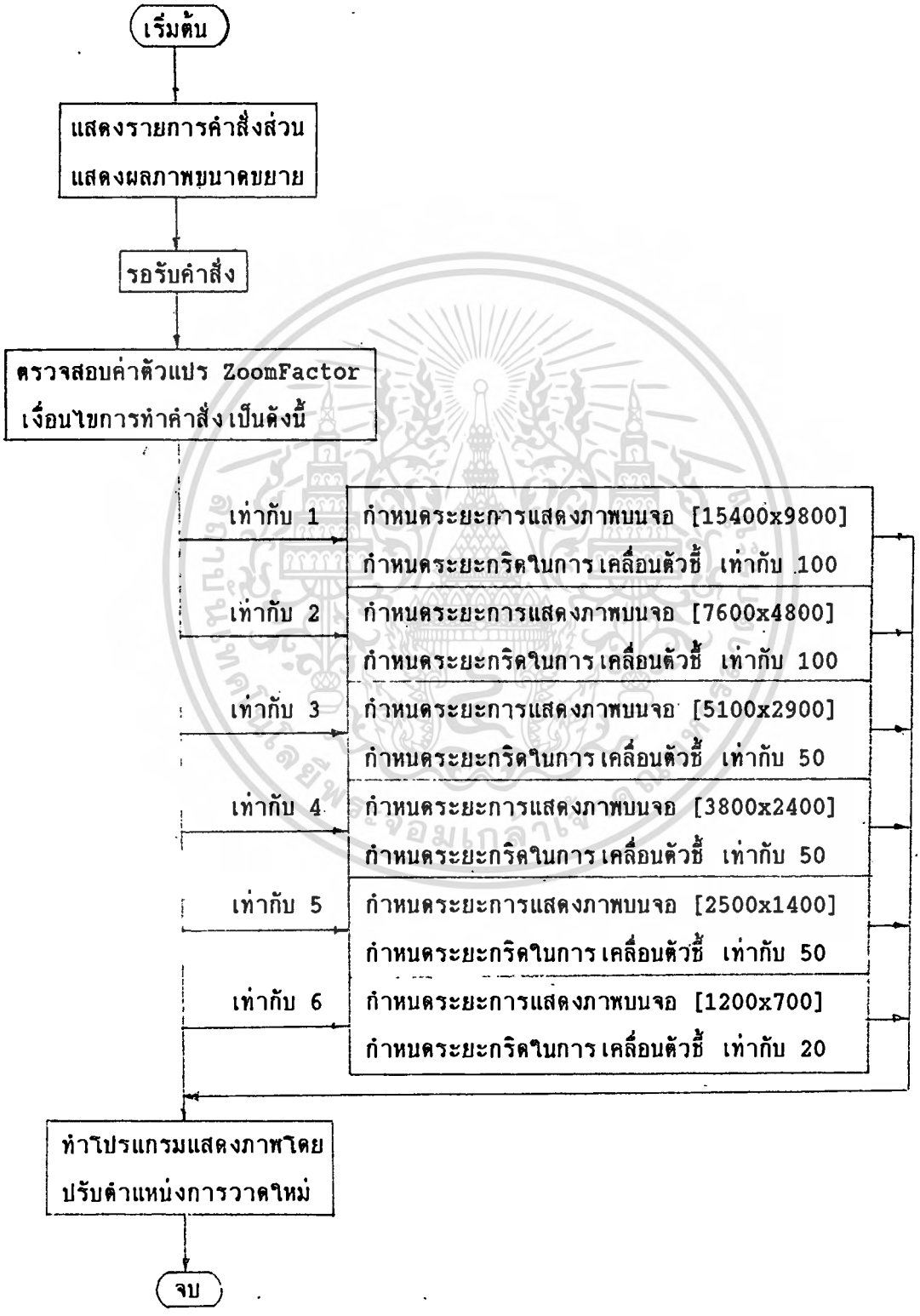
รูปที่ 4.21 แสดงการทำงานของโปรแกรมแสดงภาพโดยการวาดซ้ำ

โฟลว์ชาร์ทการทำงานของโปรแกรมแสดงภาพโดยการวาดซ้ำ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่วาดภาพที่สูญเสียรายละเอียดต่าง ๆ ณ.ตำแหน่งการแสดงผลเดิมใหม่ มีการทำงานโปรแกรมดังนี้

- ตรวจสอบด้านการใช้งานของข้อมูลลายเส้น และวาดภาพลายเส้นในส่วนของด้านที่ไม่ได้ใช้งานก่อน จากนั้นจึงวาดลายเส้นของด้านใช้งาน เพื่อให้ลายเส้นในด้านใช้งานนั้นไม่ถูกลายเส้นจากด้านอื่น ๆ วาดทับ ทำให้การแสดงผลของลายเส้นมีความสมบูรณ์ขึ้น ทั้งนี้ และทั้งนั้น การวาดแต่ละด้านจะมีการตรวจสอบก่อนว่าผู้ใช้ต้องการแสดงผลหรือไม่ โดยตรวจสอบจากค่าสภาวะการแสดงผลที่ถูกกำหนดค่าสภาวะด้วยคำสั่ง LAYR
- วาดภาพโดนัทตามการกำหนดค่าสภาวะการแสดงผลโดนัท
- วาดข้อความตามการกำหนดค่าสภาวะการแสดงผลข้อความ

4.21 โปรแกรมแสดงผลภาพขนาดขยาย



รูปที่ 4.22 แสดงการทำงานของโปรแกรมแสดงผลภาพขนาดขยาย

โฟลว์ชาร์ทการทำงานของโปรแกรมแสดงผลภาพขนาดขยาย อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่แสดงผลภาพขนาดขยาย ณ.ตำแหน่งที่ผู้ใช้ต้องการ เพื่อให้ภาพที่แสดงผลมีรายละเอียดเหมาะสมกับใช้งาน มีลำดับขั้นการทำงานดังนี้

- แสดงรายการคำสั่งส่วนเลือกอัตราการขยายภาพ
- รอรับคำสั่ง
- ตรวจสอบขนาดการขยาย เพื่อการหาค่าขนาดของการแสดงผลบนจอภาพ และค่าระยะของการเคลื่อนที่ตัวชี้ ต่อ หนึ่งช่วงของกริดบนจอภาพ ซึ่งจะนำไปเป็นข้อมูลการสร้างภาพให้โปรแกรมแสดงภาพโดยปรับตำแหน่งการวาด
- สร้างภาพโดยเคลื่อนย้ายภาพในตำแหน่งใหม่

4.22 โปรแกรมควบคุมที่ทำงานด้วยการกำหนดค่าตัวแปร ทำหน้าที่รับค่าที่ผู้ใช้ต้องการกำหนดให้กับโปรแกรม เพื่อทำให้การทำงานเป็นไปตามวัตถุประสงค์ โดยการนำค่าที่ได้รับส่งให้กับตัวแปรควบคุมส่วนต่าง ๆ ซึ่งประกอบด้วยโปรแกรมต่าง ๆ ดังนี้

4.22.1 โปรแกรมเลือกขนาดของลายเส้น

มีการทำงานดังนี้ - แสดงรายการคำสั่งส่วนเลือกขนาดเส้น
- รอรับคำสั่ง
- เก็บค่าขนาดเส้น ในตัวแปร Trace

4.22.2 โปรแกรมเลือกชนิดเส้นแสดงแนวการลาก

มีการทำงานดังนี้ - แสดงรายการคำสั่งส่วนเลือกชนิดเส้นแสดงแนวการลาก
- รอรับคำสั่ง
- เก็บค่าชนิดเส้นแสดงแนวการลาก ในตัวแปร RubberLineStyle

4.22.3 โปรแกรมเลือกด้านแผ่นวงจรพิมพ์

มีการทำงานดังนี้ - แสดงรายการคำสั่งส่วนเลือกด้านแผ่นวงจรพิมพ์
- รอรับคำสั่ง
- เก็บด้านที่ใช้งานแผ่นวงจรพิมพ์ ในตัวแปร Layer

4.22.4 โปรแกรมเลือกอัตราการขยายภาพ

มีการทำงานดังนี้ - แสดงรายการคำสั่งส่วนเลือกอัตราการขยายภาพ
- รอรับคำสั่ง
- เก็บค่าอัตราการขยาย ในตัวแปร ZoomFactor

4.22.5 โปรแกรมเลือกระยะการเคลื่อนที่ตัวชี้

มีการทำงานดังนี้ - แสดงรายการคำสั่งส่วนเลือกระยะการเคลื่อนที่
- รอรับคำสั่ง
- เก็บค่าระยะการเคลื่อนที่ ในตัวแปร GoStep

4.22.6 โปรแกรมเลือกขนาดและชนิดของโดนัท

มีการทำงานดังนี้ - แสดงรายการคำสั่งส่วนเลือกขนาดและชนิดของโดนัท
- รอรับคำสั่ง
- เก็บค่าขนาดและชนิดของโดนัท ในตัวแปร PadSize

4.22.7 โปรแกรมทิศทางการจัดวางข้อความ

มีการทำงานดังนี้ - แสดงทิศทางข้อความที่เป็นอยู่ขณะนั้น

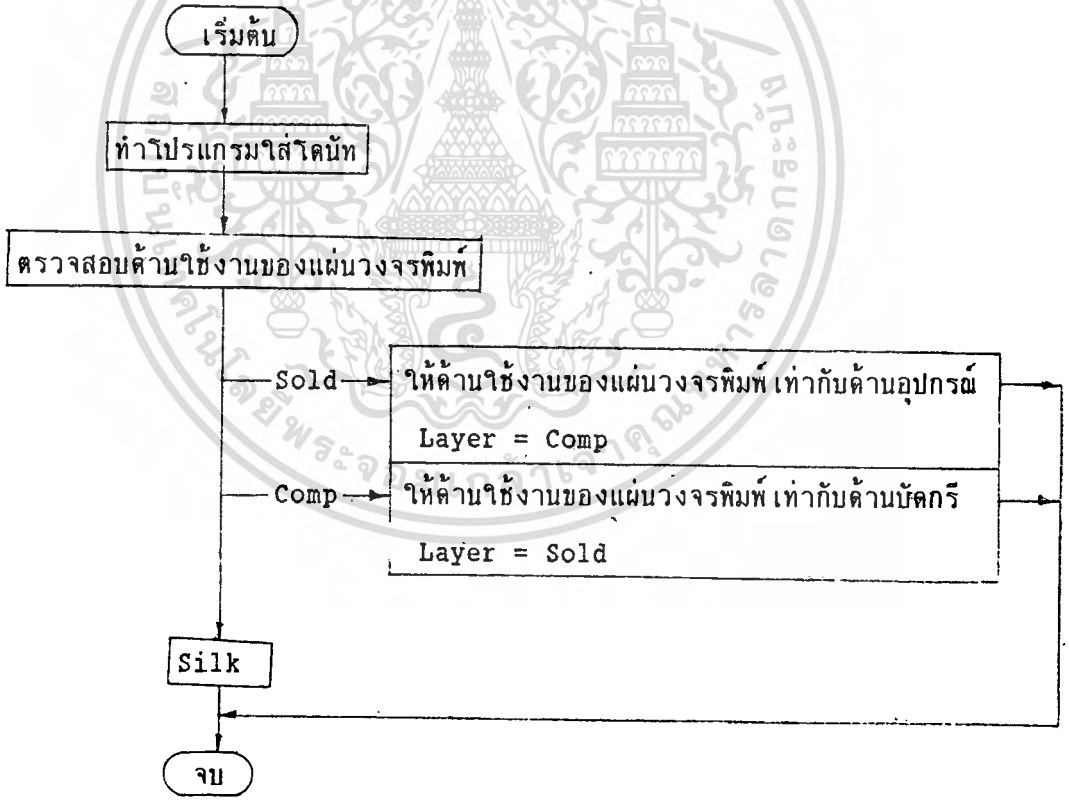
- รอรับคำสั่ง เพื่อเปลี่ยนทิศทางใหม่
- เก็บค่าทิศทางข้อความ ในตัวแปร TextDir

4.22.8 โปรแกรมทิศทางการจัดวางโดนัท

มีการทำงานดังนี้ - แสดงทิศทางโดนัทที่เป็นอยู่ขณะนั้น

- รอรับคำสั่ง เพื่อเปลี่ยนทิศทางใหม่
- เก็บค่าทิศทางของข้อความ ในตัวแปร PadDir

4.23 โปรแกรมใส่โดนัทและเปลี่ยนด้านแผ่นวงจรพิมพ์ มีลำดับการทำงานดังนี้

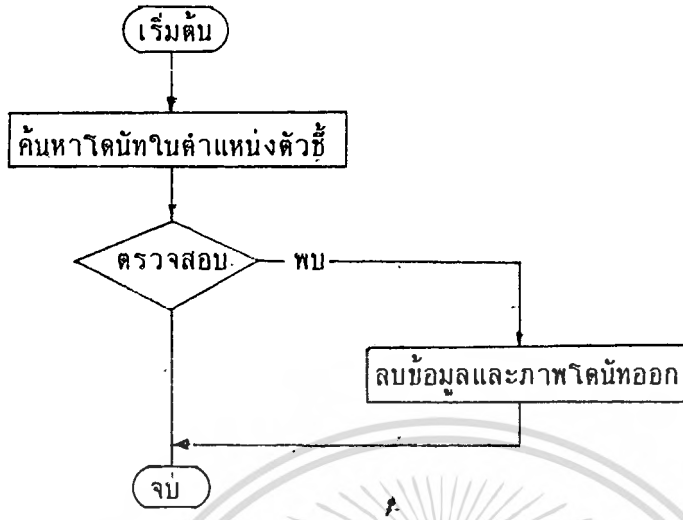


รูปที่ 4.23 แสดงการทำงานของโปรแกรมใส่โดนัทและเปลี่ยนด้านแผ่นวงจรพิมพ์

โฟลว์ชาร์ทการทำงานของโปรแกรมใส่โดนัทและเปลี่ยนด้านแผ่นวงจรพิมพ์ อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ใส่โดนัทให้กับแผ่นวงจรพิมพ์โดยจะทำการเปลี่ยนด้านของแผ่นวงจรพิมพ์ที่ใช้งานอยู่ขณะนั้นจากด้านอุปกรณ์เป็นด้านบัดกรี หรือ เปลี่ยนจากด้านบัดกรีเป็นด้านอุปกรณ์ ถ้าเป็นด้านอื่น ๆ จะไม่มีการเปลี่ยนแปลงใด ๆ

4.24 โปรแกรมลบโดนัท มีลำดับขั้นการทำงานดังนี้

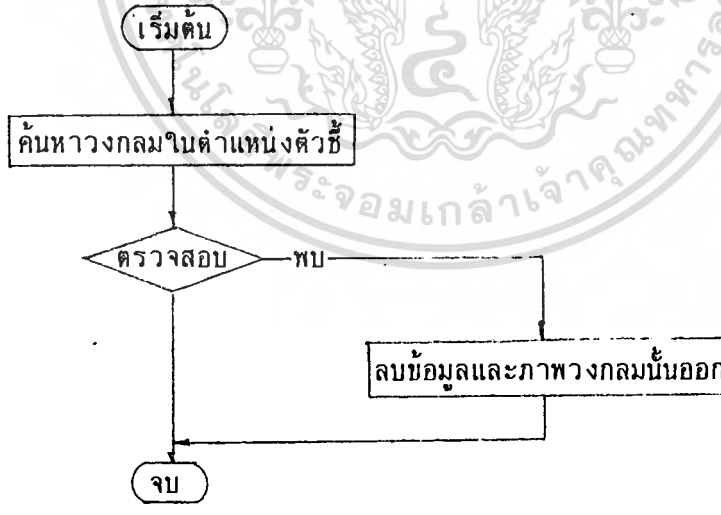


รูปที่ 3.24 แสดงการทำงานของโปรแกรมลบโดนัท

โฟลว์ชาร์ทการทำงานของโปรแกรมลบโดนัท อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ลบโดนัทที่ไม่ต้องการออกจากแผ่นวงจรพิมพ์โดยตรวจสอบข้อมูลโดนัทที่มีอยู่กับค่าตำแหน่งตัวชี้ ถ้าตรวจพบว่ามีจะลบข้อมูลและภาพโดนัทนั้นออก

4.25 โปรแกรมลบวงกลม มีลำดับขั้นการทำงานดังนี้

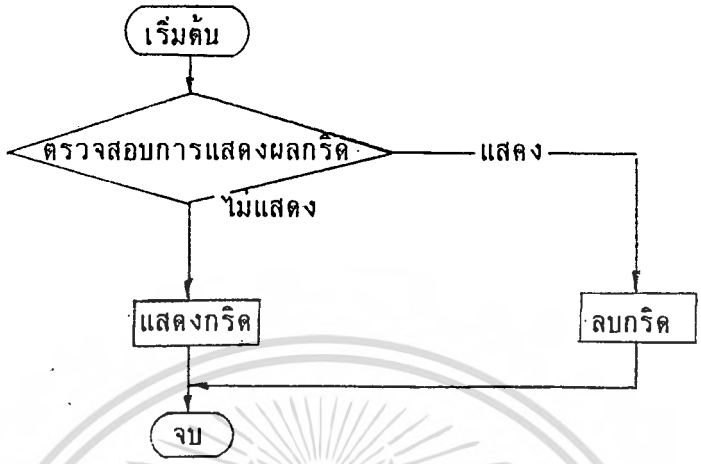


รูปที่ 3.25 แสดงการทำงานของโปรแกรมลบวงกลม

โฟลว์ชาร์ทการทำงานของโปรแกรมลบวงกลม อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่ลบวงกลมที่ไม่ต้องการออกจากแผ่นวงจรพิมพ์โดยตรวจสอบข้อมูลตำแหน่งจุดศูนย์กลางที่มีอยู่กับค่าตำแหน่งตัวชี้ ถ้าตรวจสอบพบว่ามีจะลบข้อมูลและภาพวงกลมนั้นออก

4.26 โปรแกรมควบคุมการแสดงผลกริด มีลำดับขั้นการทำงานดังนี้



รูปที่ 4.26 แสดงการทำงานของโปรแกรมควบคุมการแสดงผลกริด

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการแสดงผลกริด อธิบายได้ดังนี้

โปรแกรมส่วนนี้ ทำหน้าที่แสดงแนวบอกระยะทางบนจอภาพ และการเคลื่อนที่ของตัวชี้ เพื่อช่วยให้ผู้ใช้สามารถทราบถึงตำแหน่งต่าง ๆ ในการจัดวางอุปกรณ์ และสร้างลายเส้นบนจอภาพได้ถูกต้องมากขึ้น แต่ถ้าผู้ใช้ไม่ต้องการแสดงผลกริดก็สามารถทำได้ เนื่องจากการทำงานของคำสั่งนี้เป็นแบบท็อกเกิล (Toggle) คือถ้าเลือกใช้คำสั่งนี้โดยยังไม่มีการแสดงผลกริดโปรแกรมก็จะแสดงผลกริดนั้นออกมา แต่ถ้ามีการแสดงผลกริดอยู่จะลบกริดนั้นออก

4.27 การแสดงผลภาพแบบกราฟฟิก

การแสดงผลของโปรแกรมบนจอภาพโดยส่วนใหญ่แล้วมีการแสดงผลเป็นแบบกราฟฟิก ซึ่งสามารถสร้างภาพแบบต่าง ๆ โดยใช้คำสั่งสร้างภาพแบบกราฟฟิก ได้นอกเหนือจากการแสดงข้อความดังทั่ว ๆ ไป

4.27.1 คำสั่งสร้างภาพแบบกราฟฟิก โดยทั่วไปของการใช้คำสั่งส่วนใหญ่แล้วเป็นคำสั่งที่มีใช้อยู่ในตัวแปรภาษาปาสคาล [3] (Pascal) แต่คำสั่งที่เป็นคำสั่งกราฟฟิกหลักมีดังนี้

- คำสั่งสร้างเส้น เช่น เส้นตรง วงกลม และส่วนโค้ง
- คำสั่งแสดงข้อความ
- คำสั่งลบจอภาพ

นอกเหนือจากคำสั่งเหล่านี้ และคำสั่งที่มีในตัวแปรภาษาปาสคาล ยังมีคำสั่งที่ถูกสร้างขึ้นเพื่อให้การสร้างภาพแบบกราฟฟิกมีความสมบูรณ์ และทำให้การพัฒนาโปรแกรมนี้ทำได้สะดวกขึ้น ดังตัวอย่างเช่น

- คำสั่งพล็อทจุดภาพแบบ Xor
- คำสั่งสร้างเส้นแสดงแนวการลากแบบ Angl
- คำสั่งสร้างเส้นแสดงแนวการลากแบบ Orth
- คำสั่งสร้างเส้นแสดงแนวการลากแบบ 45Dg
- คำสั่งสร้างวงกลมแบบ Xor
- คำสั่งสร้างส่วนโค้งของวงกลมแบบ Xor
- คำสั่งสร้างเส้นแสดงขอบเขตของอุปกรณ์
- คำสั่งสร้างภาพแสดงรายละเอียดของอุปกรณ์

คำสั่งต่าง ๆ เหล่านี้มีประโยชน์มากในการสร้างเส้นที่ต้องการแสดงผลออกมาและสามารถลบออกจากจอภาพได้โดยไม่ทำให้ภาพที่มีอยู่เดิมนั้นถูกลบออกไปด้วยและไม่ต้องเก็บข้อมูลภาพเดิมก่อนการวาด วิธีการนี้ทำได้โดยการใช้คำสั่งพล็อทจุดภาพแบบ Xor เป็นคำสั่งพื้นฐานในการสร้างคำสั่งอื่น ๆ กล่าวได้ดังนี้

4.27.2 การสร้างคำสั่งพล็อทจุดภาพแบบ Xor มีโครงสร้างการทำงานดังนี้

```
procedure _XorDot (X ,Y : integer ;DotColor : byte);
var DotBuffer : byte
```

begin

- นำค่าตำแหน่งที่จะพล็อท X,Y มาคำนวนหาตำแหน่งแอสเดรส(Address) ของหน่วยความจำแสดงผล
- อ่านข้อมูลในหน่วยความจำที่ตำแหน่งนั้นมาเก็บไว้ที่ตัวแปร DotBuffer
- นำสีของภาพ (DotColor) ที่ต้องการจะพล็อทลงบนภาพมากระทำฟังค์ชันแบบบูลลีนด้วยการ Xor กับข้อมูลใน DotBuffer จะเป็นผลให้ข้อมูลมีค่าเปลี่ยนไป และนำเก็บไว้ในหน่วยความจำแสดงผลในตำแหน่งเดิม

end;

การพล็อทแบบ Xor สามารถอธิบายจุดประสงค์ของการทำงานได้ดังตัวอย่างดังนี้

ตัวอย่าง

```
_XorDot(100,200,7);
```

ตำแหน่งที่แอสเดรสจะถูกนำไปคำนวนด้วยค่า 100 และ 200 อ่านค่าหน่วยความจำนี้ (สมมุติ) มีค่าเท่ากับ 3 (0011B) นำค่านี้ Xor กับ สีที่ต้องการพล็อทคือ 7 (0111B) จะได้ผลลัพธ์เป็น 0011 Xor 0111 = 0100 หรือมีสีเป็นค่าเท่ากับ 4 จะเห็นได้ว่าสีของการพล็อทมีค่าเปลี่ยนไป ดังนั้นคำสั่ง _XorDot จึงไม่เหมาะกับการนำมาใช้สร้างภาพโดยตรง ข้อสังเกตุของการใช้คำสั่งนี้คือเมื่อใช้คำสั่งนี้ซ้ำอีกทีจะได้ค่าของสีดังนี้ 0100 Xor 0111 ผลลัพธ์มีค่าเท่ากับ 0011 หรือ 3 เหมือนเช่นเดิมก่อนการพล็อทนั่นก็คือสามารถนำคำสั่งนี้นำไปสร้างเส้นต่าง ๆ เพื่อแสดงแนวการลากโดยการวาดครั้งแรกเพื่อแสดงแนวเส้น วาดครั้งที่ 2 เป็นการลบเส้นแสดงแนวออก

4.27.3 คำสั่งสร้างเส้นแสดงแนวการลากแบบ Angl เป็นคำสั่งที่สร้างได้จากคำสั่ง _XorDot มีโครงสร้างโปรแกรมดังนี้

```
porcedure DrawAngLine (X1,Y1,X2,Y2:integer;Color:byte);
var X,Y,DeltaX,DeltaY,XStep,YStep,Direction : integer;
begin
     X := X1 ; Y := Y1;
     XStep := 1;YStep := 1;
     if X1 > X2 then XStep := -1;
```

```
if Y1 > Y2 then YStep := -1;
DeltaX := abs(X2-X1);
DeltaY := abs(Y2-Y1);
if DeltaX = 0 then Direction := -1
else Direction := 0;
while not ((X=X2) and (Y=Y2)) do
begin
     _XorDot (X,Y,Color);
     if Direction < 0 then
     begin
        Y := Y+YStep;
        Direction := Direction+DeltaX;
     end
     else
     begin
        X := X+XStep;
        Direction := Direction-DeltaY;
     end;
  end;
end;
```

4.27.4 คำสั่งสร้างเส้นแสดงแนวการลากแบบ Orth เป็นคำสั่งที่สร้างได้จากคำสั่ง _XorDot มีโครงสร้างโปรแกรมดังนี้

```
procedure DrawOrthLine(X1,Y1,X2,Y2:integer;Color:byte);
var  X,Y,DeltaX,DeltaY,XStep,YStep,Direction : integer;
     StepLengthX,StepLengthY : integer;
begin
     X := X1 ; Y := Y1;
     XStep := 1;YStep := 1;
     if X1 > X2 then XStep := -1;
```

```
        if Y1 > Y2 then YStep := -1;
        DeltaX := abs(X2-X1);
        DeltaY := abs(Y2-Y1);
        if StepLengthX >= StepLengthY then
        begin
             while not (X=X2) do
             begin
                  _XorDot (X,Y,Color);
                  X := X+XStep;
             end;
             while not (Y=Y2) do
             begin
                  _XorDot (X,Y,Color);
                  Y := Y+YStep;
             end;
        end;
        if StepLengthX < StepLengthY then
        begin
             while not (Y=Y2) do
             begin
                  _XorDot (X,Y,Color);
                  Y := Y+YStep;
             end;
             while not (X=X2) do
             begin
                  _XorDot (X,Y,Color);
                  X := X+XStep;
             end;
        end;
end;
```

4.27.5 คำสั่งสร้างเส้นแสดงแนวการลากแบบ 45Dg เป็นคำสั่งที่สร้างได้จากคำสั่ง DrawAngLine มีโครงสร้างโปรแกรมดังนี้

```
procedure Draw45dLine (X1,Y1,X2,Y2:integer;Color:byte);
var PosStepX,PosStepY,Mul,X,Y:integer;
begin
    PosStepX := X2-X1; PosStepY := Y2-Y1;
    if abs(PosStepX) >= abs(PosStepY) then
    begin
        if abs(PosStepX) <> abs(PosStepY) then
        begin
            if PosStepX < 0 then Mul := -1
            else Mul := 1;
            PosStepX := (abs(PosStepX)-abs(PosStepY))
                        * Mul;
            PosStepY := 0;
        end;
    end;
    else
    begin
        if  PosStepY < 0 then Mul := -1
        else  Mul := 1;
        PosStepY := (abs(PosStepY)-abs(PosStepX)
                    * Mul;
        PosStepX := 0;
    end;
    X := X1 + PosStepX;
    Y := Y1 + PosStepY;
    DrawAngLine(X1,Y1,X,Y,Color);
    DrawAngLine(X,Y,X2,Y2,Color);
end;
```

2.27.6 คำสั่งสร้างวงกลมแบบ Xor เป็นคำสั่งที่ใช้ในการแสดงภาพวงกลม เพื่อเป็นแนวก่อนการสร้างวงกลมลงในแผ่นวงจรพิมพ์ ด้วยเหตุผลที่วงกลมนี้ใช้เพียงเป็นแนวจึงไม่มีความจำเป็นที่จะต้องแสดงผลอย่างละเอียดอีกทั้งเพื่อให้การทำงานมีความเร็วสูงขึ้นจึงสร้างวงกลมที่ประกอบด้วยเส้นตรงจำนวน 72 เส้นล้อมรอบแต่ละเส้นทำมุมอยู่ 5 องศากับจุดศูนย์กลาง มีโครงสร้างการทำงานดังนี้

คำสั่งที่ใช้ในการคำนวนหาเส้นในการสร้างวงกลม

```
procedure CalCirArc(X,Y,Radius:integer);
begin
     for i := 0 to 71 do
     begin
          CirLineDataX1[i] := X + Radius*SinData[i];
          CirLineDataY1[i] := Y + Radius*CosData[i];
          j := i+1;
          if i := 71 then j := 0;
          CirLineDataX2[i] := X + Radius*SinData[j];
          CirLineDataY2[i] := Y + Radius*CosData[j];
     end;
end;
```

คำสั่งที่ใช้สร้างวงกลม

```
procedure DrawCirCle(X,Y,Radius:integer;Color:byte);
   begin
     CalCirArc(X,Y,Radius);
     for i := 0 to 71 do
     begin
       DrawAngLine(CirLineDataX1[i],CirLineDataY1[i],
                   CirLineDataX2[i],CirLineDataY2[i],Color);
     end;
```

2.27.7 คำสั่งสร้างส่วนโค้งของวงกลมแบบ Xor มีลักษณะการทำงานเช่นเดียวกับการสร้างวงกลมและใช้โปรแกรมส่วนของการคำนวณร่วมกับส่วนของวงกลม ด้วยคือ CalCirArc มีโครงสร้างซึ่งเป็นข้อแตกต่างกันดังนี้

```
procedure DrawArc(X,Y,R,StartAng,EndAng:integer;Color:byte);
begin
     CalCirArc(X,Y,R);
     StartAng := StartAng div 5  {Start 0 - 360 -> 0 - 71}
     if StartAng >= 72 then StartAng := StartAng mod 72;
     EndAng   := EndtAng div 5    {End 0 - 360 -> 0 - 71}
     if EndAng >= 72 then EndAng := EndAng mod 72;
     i := StartAng;
     while i <> EndAng do
     begin
          DrawAngLine(CirLineDataX1,CirLineDataY1,
                      CirLineDataX2,CirLineDataY2,Color);
          i := i+1;
          if i > 71 then i := 0;
     end;
```

4.27.8 คำสั่งสร้างเส้นแสดงขอบเขตของอุปกรณ์ นำไปใช้กับการแสดงขอบเขตของอุปกรณ์ในไลบรารี่ซึ่งข้อมูลนี้มีส่วนที่สำคัญคือค่าต่ำสุด สูงสุด และตำแหน่งจุดอ้างอิง มีโครงสร้างการทำงานดังนี้

```
procedure DrawLib (X,Y : integer;Color:byte);
var X0,X1,Y0,Y1:integer;
begin  {MinX := 0 MinY := 0}
     X0 := MinX + RefX; Y0 := MinY + RefY;
     X1 := MaxX + RefX; Y1 := MaxY + RefY;
     DrawAngLine(X+X0,Y+Y0,X+X1,Y+Y0,Color);
```

```
      DrawAngLine(X+X1,Y+Y0,X+X1,Y+Y1,Color);
      DrawAngLine(X+X1,Y+Y1,X+X0,Y+Y1,Color);
      DrawAngLine(X+X0,Y+Y1,X+X0,Y+Y0,Color);
end;
```

4.27.9 คำสั่งสร้างภาพแสดงรายละเอียดของอุปกรณ์ เป็นคำสั่งที่นำข้อมูลต่าง ๆ ของอุปกรณ์ที่จะมาใช้งานแสดงเป็นแนวทางก่อนการใส่ลงบนแผ่นวงจรพิมพ์ การทำส่วนนี้ประกอบด้วยโครงสร้างการทำงานหลักดังนี้

กำหนดให้ตัวแปรที่ใช้ในการเก็บข้อมูลอุปกรณ์ เป็นดังนี้

```
      DeviceType = Array [1..DevMax] of
                   record
                      Code : byte;
                      Data : Array [1..5] of integer;
                   end; {of record}
procedure  DrawDevice;
var  i : integer;
      Device : DeviceType;
begin
      for i := 1 to Num_Of_Command {จำนวนคำสั่งที่ใช้ในอุปกรณ์} Do
      begin
        Case Device[i].Code of  {ตรวจสอบชนิดของคำสั่ง}
        LineCode : begin {คำสั่งสร้างเส้น}
                      X1 := CursorX+Device[i].Data[1]+RefX;
                      Y1 := CursorY+Device[i].Data[2]+RefY;
                      X2 := CursorX+Device[i].Data[3]+RefX;
                      Y2 := CursorY+Device[i].Data[4]+RefX;
                      DrawAngLine(X1,Y1,X2,Y2,Color);
                   end;
        CircCode : begin {คำสั่งสร้างวงกลม}
                      X0 := CursorX+Device[i].Data[1]+RefX;
```

```
                    Y0 := CursorY+Device[i].Data[2]+RefY;
                    R  := Device[i].Data[3];
                    DrawCircle(X0,Y0,R);
                  end;
        ArcCode   : begin {คำสั่งสร้างส่วนโค้งของวงกลม}
                    X0 := CursorX+Device[i].Data[1]+RefX;
                    Y0 := CursorY+Device[i].Data[2]+RefY;
                    R  := Device[i].Data[3];
                    StartAng := Device[i].Data[4];
                    EndAng   := Device[i].Data[5];
                    DrawArc(X,Y,R,StartAng,EndAng);
                  end;
        PadCode   : begin {คำสั่งสร้างโดนัท}
                    X0 := CursorX+Device[i].Data[1]+RefX;
                    Y0 := CursorY+Device[i].Data[2]+RefY;
                    PadType := Device[i].Data[3];
                    PadDir  := Device[i].Data[4]
                    DrawPad(X0,Y0,PadType,PadDir);
                    {คำสั่งสร้างโดนัทนี้ประกอบด้วยคำสั่งสร้างวงกลม และ
                    สี่เหลี่ยมแบบต่างๆ ซึ่งขึ้นอยู่กับ PadType และPadDir
                    ลักษณะการทำงานโดยส่วนใหญ่แล้วเหมือนกับวิธีการดังที่
                    กล่าวมาแล้ว}
                  end;
      end;
    end;
end;
```

การทำงานของโปรแกรมส่วนนี้ จะวาดคำสั่งที่มีใช้ในอุปกรณ์ในลักษณะของ เส้นแสดงแนวการลาก เมื่อต้องการลบภาพของอุปกรณ์ก็สามารถทำได้โดยการวาดอุปกรณ์นี้ซ้ำ ภาพอุปกรณ์ก็จะหายไปโดยไม่ทำให้ภาพ เดิมก่อนการวาด เสียหายไปด้วยซึ่งก็ เป็นผลมาจากการสร้างภาพที่กระทำฟังค์ชั่น Xor

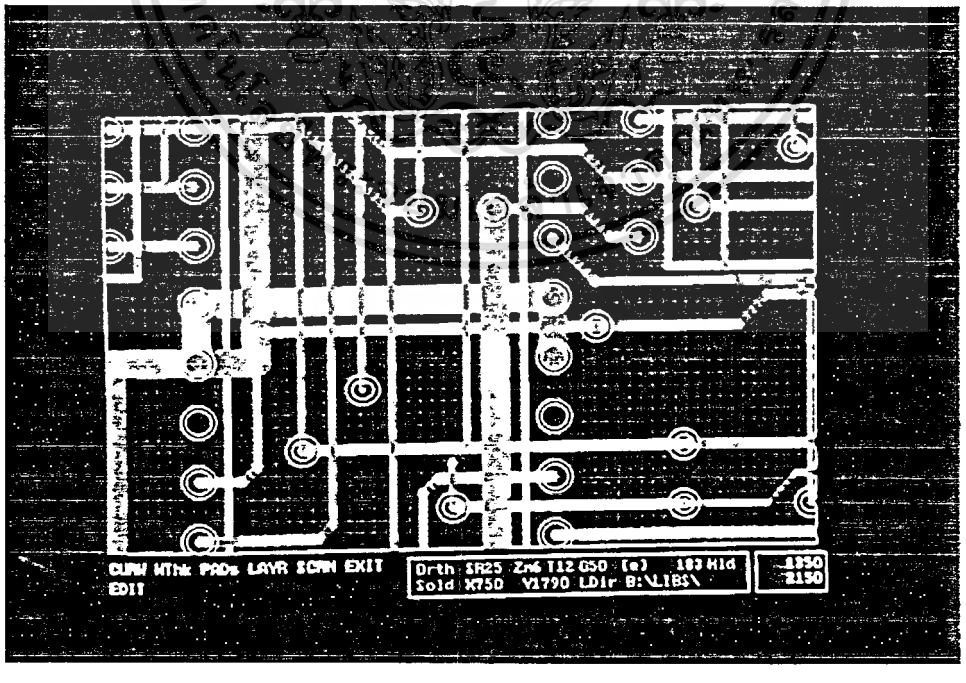
บทที่ 5
ผลลัพธ์การพิมพ์

การสร้างแผ่นวงจรพิมพ์ด้วยโปรแกรมสิ่งสำคัญสิ่งหนึ่งก็คือผลลัพธ์การพิมพ์แบบต่าง ๆ ที่เป็นส่วนประกอบของแผ่นวงจรพิมพ์ โดยผลลัพธ์การพิมพ์นี้มีขนาดขยายใหญ่กว่าขนาดจริงเป็นจำนวนเท่า ตัวอย่างเช่น 2 เท่า หรือ 4 เท่า เป็นต้น และนำไปสร้างเป็นฟิล์ม(Film) ที่มีขนาดขยายเท่ากับขนาดจริงของงาน เพื่อทำให้ลายเส้นทองแดงมีความคมชัด จากขั้นตอนนี้ผู้ใช้จะนำฟิล์มที่ได้ไปทำกระบวนการต่าง ๆ ซึ่งเป็นขั้นตอนการสร้างแผ่นวงจรพิมพ์ เช่น เจาะรู และทำวิธีการทางเคมี ต่อไป

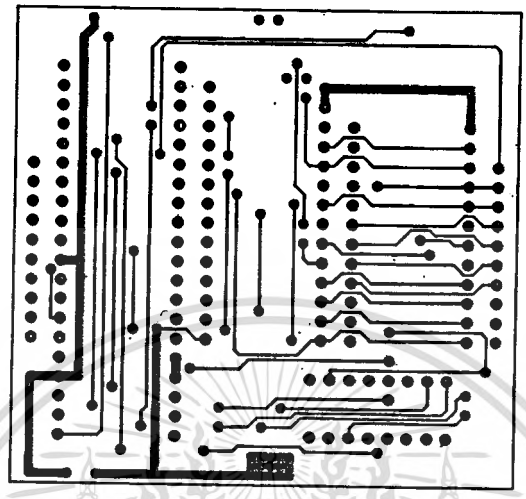
5.1 ผลลัพธ์การพิมพ์แบบต่าง ๆ ที่เป็นส่วนประกอบสำคัญของแผ่นวงจรพิมพ์แบ่งออกได้ดังนี้

- ลายวงจรด้านบัดกรี
- ลายวงจรด้านอุปกรณ์
- ลายภาพอุปกรณ์

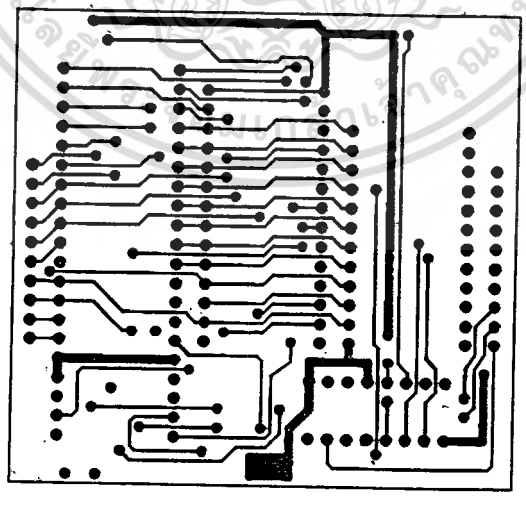
5.2 ตัวอย่างฟิล์มแบบที่ 1 ที่นำไปใช้สร้างแผ่นวงจรพิมพ์และภาพแสดงผลลายวงจรพิมพ์บนเครื่องไมโครคอมพิวเตอร์



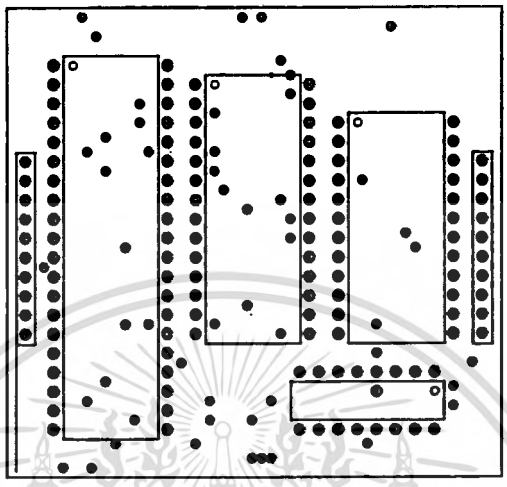
รูปที่ 5.1ก แสดงผลลายวงจรพิมพ์บนจอของเครื่องไมโครคอมพิวเตอร์



รูปที่ 5.1ข แสดงตัวอย่างฟิล์มลายวงจรด้านบัดกรี

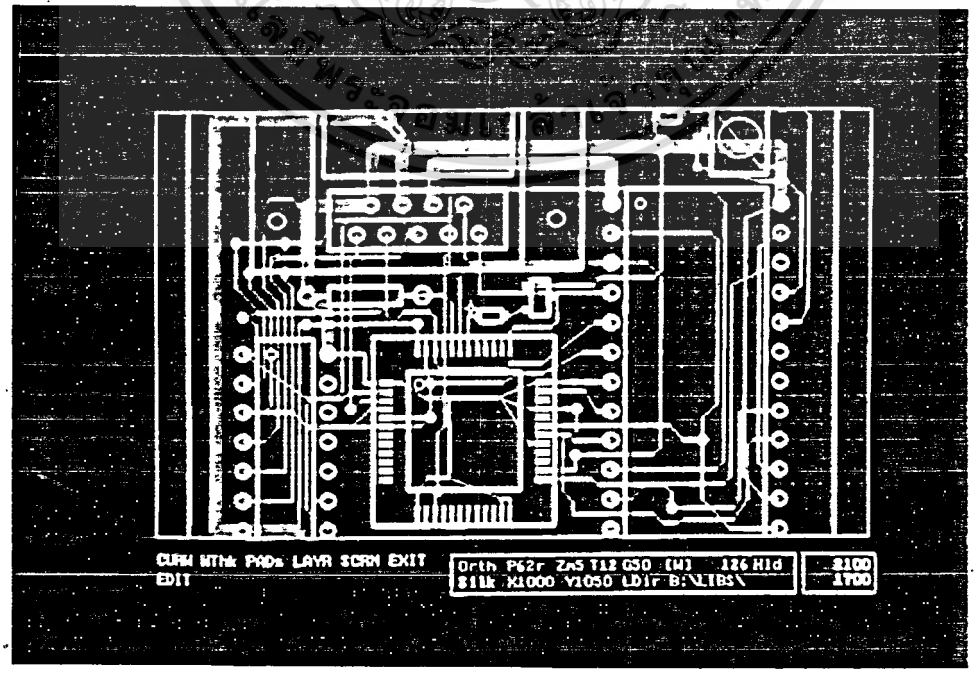


รูปที่ 5.1ค แสดงตัวอย่างฟิล์มลายวงจรด้านอุปกรณ์

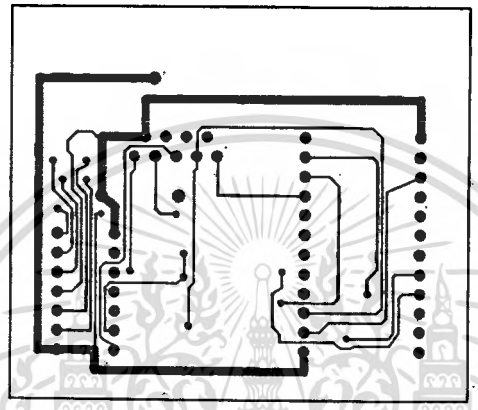


รูปที่ 5.1ง แสดงตัวอย่างฟิล์มด้านลายภาพอุปกรณ์ หรือสกรีนอุปกรณ์

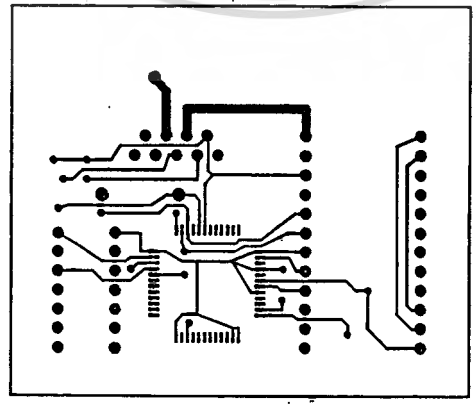
5.3 ตัวอย่างฟิล์มแบบที่ 2 ที่นำไปใช้สร้างแผ่นวงจรพิมพ์แบบใช้อุปกรณ์ชนิดติดผิว และภาพแสดงผลลายวงจรพิมพ์บนเครื่องไมโครคอมพิวเตอร์



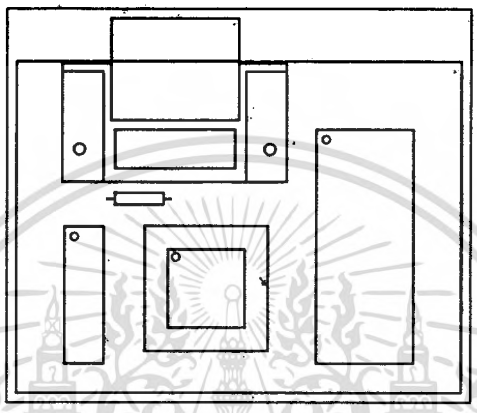
รูปที่ 5.2ก แสดงผลลายวงจรพิมพ์บนจอของเครื่องไมโครคอมพิวเตอร์



รูปที่ 5.2ข แสดงตัวอย่างฟิล์มลายวงจรด้านบัดกรี



รูปที่ 5.2ค แสดงตัวอย่างฟิล์มลายวงจรด้านอุปกรณ์



รูปที่ 5.2ง แสดงตัวอย่างฟิล์มด้านลายภาพอุปกรณ์ หรือสกรีนอุปกรณ์

บทที่ 6
สรุปผลการวิจัย

การออกแบบแผ่นวงจรพิมพ์ ด้วยโปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์ที่มีใช้กันอยู่ทั่ว ๆ ไปในปัจจุบันมีปัญหาบางอย่าง เกี่ยวกับขนาด และระยะห่างระหว่างขาอุปกรณ์ ทำให้ไม่สามารถสร้างแผ่นวงจรพิมพ์ที่ใช้อุปกรณ์นั้นได้ และบางโปรแกรมมีความยุ่งยากในการใช้งาน เนื่องมาจากมีคำสั่งควบคุมการทำงานให้ใช้น้อย ปัญหาเหล่านี้ได้ถูกหยิบยกขึ้นมาแก้ไขในวิทยานิพนธ์ฉบับนี้ซึ่งได้เสนอโปรแกรมช่วยในการออกแบบแผ่นวงจรพิมพ์ที่มีความคล่องตัว และมีประสิทธิภาพสูงโดยสามารถนำไปออกแบบแผ่นวงจรพิมพ์ที่ใช้กับอุปกรณ์แบบต่าง ๆ ได้อย่างเหมาะสม มีการเพิ่มคำสั่งและฟังก์ชั่นการทำงานพิเศษเพื่ออำนวยความสะดวกในการออกแบบอย่างมากมาย ถึงแม้ว่าจะไม่มีการลากเส้นโดยอัตโนมัติ (Automatic Routing) ก็ตาม เหตุผลที่ไม่พัฒนาฟังก์ชั่นนี้ ก็เนื่องจากการลากเส้นอัตโนมัตินั้นยังไม่มีความจำเป็นในการออกแบบเท่าใดนัก เพราะการลากเส้นอัตโนมัติจะใช้วิธีหาทางเดินของเส้นที่สั้นที่สุดโดยมิได้คำนึงถึงความสวยงาม และการเกิดสัญญาณรบกวนขึ้นในแผ่นวงจรพิมพ์ ดังนั้นการลากเส้นที่เหมาะสมจึงน่าจะขึ้นอยู่กับการตัดสินใจของผู้ใช้เองมากกว่า [12]

การออกแบบคำสั่งเพื่อช่วยให้การใช้งานสะดวกนั้นมีความสำคัญเป็นอย่างมาก ซึ่งจะช่วยลดเวลา และลดปัญหาในการสร้างแผ่นวงจรพิมพ์ทำให้การเดินลายวงจรสามารถทำได้อย่างมีประสิทธิภาพ ดังเช่น

- คำสั่งลากเส้นที่สามารถกำหนดเส้นแสดงแนวการลากในแบบต่าง ๆ ได้ทำให้การสร้างลายเส้นวงจรทำได้หลายแบบ
- คำสั่งใส่โดนัทที่สามารถกำหนดขนาด และชนิดของโดนัทได้ทำให้สามารถนำไปใช้งานกับอุปกรณ์อิเลคทรอนิคส์แบบต่าง ๆ ได้กว้างขึ้น
- คำสั่งใส่โดนัทและเปลี่ยนด้านแผ่นวงจรพิมพ์ช่วยลดคำสั่งในการเดินลายเส้นวงจรแบบต่อเนื่องโดยการใส่โดนัท และเปลี่ยนด้านแผ่นวงจรพิมพ์ไปพร้อม ๆ กัน
- คำสั่งควบคุมการแสดงผลแบบเช็คพล็อทซึ่งช่วยลดเวลาในการแสดงภาพแผ่นวงจรพิมพ์
- คำสั่งควบคุมการย้ายเส้น เป็นคำสั่งที่ช่วยให้การแก้ไขเส้นที่ผู้ใช้เดินผิดทำได้รวดเร็วขึ้นโดยไม่ต้องลบ และสร้างใหม่
- คำสั่งวาดภาพซ้ำ เพื่อช่วยให้รายละเอียดภาพบางส่วนที่ถูกลบไปแสดงผลดังเดิม

- คำสั่งขยายภาพในระดับการขยายต่าง ๆ มีให้เลือกได้ 6 ขนาด เพื่อปรับขนาดพื้นที่ในการแสดงผลของแผ่นวงจรพิมพ์บนจอภาพ โดยที่ระดับการขยายเท่ากับ 6 เป็นการขยายภาพในขนาดโตสุดซึ่งให้รายละเอียดเพียงพอในการตรวจสอบระยะห่างระหว่างลายเส้นวงจรที่ใกล้กัน
- คำสั่งแสดงภาพโดยปรับตำแหน่งการวาดใหม่ ให้ความสะดวกในการแก้ไขภาพในตำแหน่งใกล้เคียงที่ไม่ได้ปรากฏบนจอภาพ หรืออาจเรียกได้ว่าเป็นการเลื่อนภาพ
- คำสั่งเคลื่อนย้ายอุปกรณ์รวมทั้งปรับตำแหน่งและทิศทางการจัดวางอุปกรณ์ให้ความสะดวกในการแก้ไขกรณีที่มีการจัดวางอุปกรณ์ผิดพลาด
- คำสั่งกำหนดเส้นแสดงแนวการลากในแบบต่าง ๆ ทำให้การสร้างลายเส้นวงจรทำได้หลายแบบ เช่น เส้นแบบ Angl Orth และ 45Dg ในแต่ละแบบนี้ให้ผลแตกต่างกันที่แนวของเส้นที่จะลากจากจุดเริ่มต้นถึงตำแหน่งตัวชี้
- และคำสั่งอื่น ๆ ดังที่ได้กล่าวไว้ในบทต้น ๆ ซึ่งล้วนแต่มีประโยชน์ทั้งสิ้น

ขีดจำกัดของโปรแกรม

โปรแกรมที่มีใช้งานกันทั่ว ๆ ไปทุก ๆ โปรแกรมจะมีขีดจำกัดในการใช้งาน หรือความสามารถในการทำงานทั้งสิ้นซึ่งไม่สามารถหลีกเลี่ยงได้ด้วยข้อจำกัดต่าง ๆ เช่นขีดความสามารถของเครื่องไมโครคอมพิวเตอร์ ขนาดของหน่วยความจำ ความสามารถในการแสดงผลบนจอภาพ และอื่น ๆ ขีดจำกัดเหล่านี้ทำให้ขอบเขตของการใช้งานโปรแกรม PCBCAD มีดังนี้

- ขนาดของแผ่นวงจรพิมพ์โตสุด เท่ากับ 15.4 นิ้ว x 9.8 นิ้ว (กำหนดขึ้นตามความเหมาะสมของขนาดวงจรทั่ว ๆ ไป)
- จำนวนด้านของลายเส้นทองแดง มี 2 ด้านคือ ด้านบัดกรี และด้านอุปกรณ์ (กำหนดขึ้นตามเทคโนโลยีที่สามารถทำได้ภายในประเทศในปัจจุบัน)
- สามารถให้ผลลัพธ์ในการทำสกรีนเพื่อสกรีนภาพอุปกรณ์ได้ และทำสกรีนเคลือบลายวงจรทองแดง หรือโซล์เดอร์รีซิสท์สกรีนได้
- จำนวนคำสั่งต่าง ๆ ที่ใช้กับด้านบัดกรีใช้ได้ไม่เกิน 3000 คำสั่ง (มีขีดจำกัดเนื่องมาจากตัวแปรภาษา Turbo Pascal[3] ยอมรับการใช้ตัวแปรแบบอาเรย์ (Array) หนึ่งตัวมีขนาดสูงสุดได้ 64 กิโลไบท์)
- จำนวนคำสั่งต่าง ๆ ที่ใช้กับด้านอุปกรณ์ ใช้ได้ไม่เกิน 3000 คำสั่ง
- จำนวนคำสั่งต่าง ๆ ที่ใช้กับด้านสกรีนภาพอุปกรณ์ ใช้ได้ไม่เกิน 3000 คำสั่ง

- จำนวนโดนัทเลือกใช้ได้ 12 แบบ และใช้ได้ไม่เกิน 3000 คำสั่ง
- การกำหนดขนาดและชนิดของโดนัท มีข้อมูลเพียงพอเหมาะที่จะนำไปเชื่อมต่อกับเครื่องเจาะอัตโนมัติแบบหลาย ๆ หัวเจาะในโรงงานอุตสาหกรรม เพื่อสร้างเป็นระบบ CAM เนื่องจากข้อมูลนี้มีการกำหนดขนาดเส้นผ่าศูนย์กลางของดอกส่วานที่ใช้เจาะรูต่างๆ ด้วยทำให้ผู้ที่ต้องการใช้โปรแกรมนี้กับเครื่องจักรอัตโนมัติสามารถเขียนโปรแกรมแยกขนาดเส้นผ่าศูนย์กลางของดอกส่วาน และสั่งงานควบคุมให้เครื่องเจาะทำงานในตำแหน่งที่ต้องการเจาะรูในแต่ละขนาดได้
- จำนวนข้อความสูงสุดเท่ากับ 200 โดยมีความยาวแต่ละข้อความ 20 ตัวอักษร
- จำนวนอุปกรณ์ที่สร้างเก็บไว้ในไลบรารีไม่จำกัดรูปแบบ และจำนวน แต่เมื่อนำมารวมกันไม่ควรมีคำสั่งด้านบัดกรีเกิน 3000 คำสั่งรวมทั้งด้านอื่น ๆ และโดนัทในอุปกรณ์แต่ละตัวสามารถนำคำสั่งต่าง ๆ มาสร้างได้ 200 คำสั่ง (ได้จากการศึกษาข้อมูลรายละเอียดอุปกรณ์ [7,8,9] และนำมาประมาณค่าจำนวนการใช้งานสูงสุด)
- คำสั่งควบคุมการย้ายเส้น สามารถเคลื่อนย้ายเส้นได้สูงสุดจำนวน 40 เส้นไปพร้อม ๆ กันโดยมีการแสดงบนจอภาพด้วย
- เคลื่อนที่ตัวชี้ด้วยระยะไกลสุดเท่ากับ 1000 มิล และใกล้สุด 1 มิล
- ขยายภาพโตสุดเท่ากับ 1200 มิล x 700 มิล หรือ 1.2 นิ้ว x 0.7นิ้ว
- ย่อภาพได้เล็กสุดเท่ากับ 15.4 นิ้ว x 9.8 นิ้ว

การเก็บข้อมูลแผ่นวงจรพิมพ์ประกอบด้วยไฟล์ข้อมูลจำนวน 6 ไฟล์โดยมีชื่อไฟล์เดียวกันและมีชื่อหลังจุดที่แตกต่างกัน ดังนี้

ชื่อไฟล์.SLD เก็บคำสั่งด้านบัดกรี

ชื่อไฟล์.CMP เก็บคำสั่งด้านอุปกรณ์

ชื่อไฟล์.SLK เก็บคำสั่งด้านสกรีนภาพอุปกรณ์

ชื่อไฟล์.PAD เก็บคำสั่งโดนัท

ชื่อไฟล์.MSG เก็บคำสั่งสร้างข้อความ

ชื่อไฟล์.DEV เก็บข้อมูลอุปกรณ์ที่นำมาใช้สร้างแผ่นวงจรพิมพ์นี้

เหตุผลที่เก็บข้อมูลแผ่นวงจรพิมพ์เป็นแบบนี้ ก็เพื่อให้ผู้ที่จะนำโปรแกรมนี้ไปศึกษาต่อสามารถเข้าใจการทำงานได้ง่ายขึ้น เนื่องจากข้อมูลในแต่ละส่วนไม่เกี่ยวข้องกัน และทำให้การขยายขีดความสามารถในการทำงานที่เกี่ยวข้องกับข้อมูลในแต่ละส่วนทำได้ง่ายขึ้น โดยไม่สับสน แต่ก็มีข้อเสียในการอ่านโปรแกรมจากดิสค์ทำให้โปรแกรมทำงานช้าลง

ข้อแก้ไขและคำแนะนำ

โปรแกรมนี้เป็นโปรแกรมที่มีขีดความสามารถในการทำงานสูง แต่ก็ยังมีจุดอ่อนบางอย่างที่ต้องทำการแก้ไข อย่างเช่น โครงสร้างข้อมูลเนื่องจากการแบ่งข้อมูลต่าง ๆ ออกเป็น 6 ชุดหรือ 6 ไฟล์ ทำให้การเก็บคำสั่งต่าง ๆ สามารถทำได้มากที่สุดในแต่ละไฟล์มีขนาดเท่ากับ 64 กิโลไบท์ซึ่งก็ประมาณ 3000 คำสั่ง แต่ก็ให้ผลเสียที่ขนาดของโปรแกรมคือจะมีขนาดใหญ่ และมีความล่าช้าในการทำงานโปรแกรมบางส่วน ด้วยเหตุผลที่เกี่ยวข้องกันนี้ จะเห็นได้ว่าการจัดโครงสร้างข้อมูลนั้นมีความสำคัญมากต่อการทำงานและการขยายขีดความสามารถในการใช้งานต่าง ๆ อีกทั้งขนาดของตัวโปรแกรม จึงควรมีการศึกษากันเพื่อให้การสร้างโปรแกรมใช้งานในลักษณะแบบนี้มีความสมบูรณ์มากยิ่งขึ้น

สำหรับผู้ที่ต้องการจะศึกษาการทำงานของโปรแกรมทำได้โดยดูจาก ภาคผนวก 4. ซึ่งเป็นรายละเอียดของโปรแกรมทั้งหมด

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้ด้วยความเรียบร้อยดีทุกประการ ก็เพราะได้รับความช่วยเหลือจากผู้มีพระคุณหลายท่านซึ่งเอื้อเฟื้อและอำนวยความสะดวกต่าง ๆ ให้ จึงใคร่ขอแสดงความขอบพระคุณท่านที่เกี่ยวข้องและให้ความเอื้อเฟื้อต่าง ๆ ดังนี้

ขอกราบขอบพระคุณ คุณพ่อสุนทร ปรมาธิกุล และ คุณพี่ วราภรณ์ ปรมาธิกุล ที่ให้ความช่วยเหลือในทุก ๆ ด้านด้วยดีเสมอมาและให้กำลังใจมาโดยตลอด

ขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร. กนก เจนจิระพงศ์เวช อาจารย์ที่ปรึกษา และ รองศาสตราจารย์ กิตติ ตีรเศรษฐ ผู้ที่ให้ความอนุเคราะห์ทางด้านความรู้ แนวทาง เทคนิค และอุปกรณ์ที่ช่วยในการทำงานต่าง ๆ

ขอขอบคุณท่านอาจารย์ทุก ๆ ท่านในภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม ที่ให้ความช่วยเหลือเกื้อกูลมาตลอด

เอกสารอ้างอิง

1. สุทธิบันท์ ปรมาธิกุล , กนก เจนจิระพงศ์เวช , กิตติ ตีรเศรษฐ , " การประยุกต์ใช้ไมโครคอมพิวเตอร์กับการออกแบบแผ่นวงจรพิมพ์ " , การประชุมทางวิชาการวิศวกรรมไฟฟ้า สถาบันอุดมศึกษาแห่งประเทศไทยครั้งที่ 10 จุฬาลงกรณ์มหาวิทยาลัย หน้าที่ 1-117 ถึง 1-127 , เล่มที่ 1 , 24-25 พฤษจิกายน 2530.
2. Denis Jump, PROGRAMMER'S GUIDE TO MS-DOS , Reston Publishing Company, Inc., 1984.
3. " TURBO PASCAL " , Version 4.0 , Owner's Handbook , Borland International Inc. , 1987.
4. " X-Y Plotter DXY - 980 " , Operation manual , Roland DG Corporation.
5. " smARTWORK " , Reference manual , Wintex Inc.
6. " Techical Reference " , Personal computer hardware reference library , IBM Corporation. ,April 1983.
7. " Siemens Components Service " , Siemens Inc. , Preferred products 1986.
8. " Data Book " , Low power schottky TTL ICs , SGS Technology and Service , June 1985.
9. " Signatics logic - TTL Data manual " , Signatics Coropration , 1978.
10. " AutoCAD " , version 2.6 , Reference manual , AUTODESK Inc. , June 1986.
11. " MASTERING AutoCAD " , Version 2.6 , User's reference , AUTODESK Inc. , 1987.
12. Darryl Lindsey. " The design & drafting of printed circuits" Bishop Graphics,Inc. , 1984.
13. Howard H.Manko "Soldering handbook for printed circuits and surface mounting" ,Van nostrand Reinhold Company, 1986.

ภาคผนวก 1. การจัดวางข้อมูลของแผ่นวงจรพิมพ์ สามารถอธิบายได้ดังนี้

ตัวอย่างไฟล์ข้อมูล (.SLK .CMP .SLK)

```
123
1    0    1680  2040  1680  1540    0    1
3    0    1700  1520    30    45  180    1
1    1    1985  2320  1985  2125    0    1
1    0    2135  1450  2135  1650    0    2
2    0    1600  1710    20     0    0    1
1    0    1770  1155  3200  1155    0    4
-
-
```

ตัวเลข 123 ในบันทัดแรกของไฟล์หมายความถึงจำนวนคำสั่งต่าง ๆ ทั้งหมดของแผ่นวงจรพิมพ์ และบันทัดต่อมาคือคำสั่งโดยมีโครงสร้างข้อมูลดังนี้

```
1    0    1680  2040  1680  1540    0    1
|    |      |     |     |     |     |    |
|    |      |     |     |     |     |    ความหนาของเส้น
|    |      X0    Y0    X1    Y1    X2
|    หมายเลขอุปกรณ์ ถ้าเท่ากับ 0 ไม่ใช่ข้อมูลอุปกรณ์
|    เช่น 0 เป็นเส้นที่ลากใช้งานทั่ว ๆ ไป
|         3 เป็นเส้นของอุปกรณ์ตัวที่ 3
ชนิดคำสั่ง แบ่งออกได้เป็นดังนี้
```

1 เท่ากับเส้นตรง มีผลทำให้ข้อมูล X0,Y0 เท่ากับ จุดเริ่มต้นของเส้น
X1,Y1 เท่ากับ จุดปลายของเส้น

2 เท่ากับวงกลม มีผลทำให้ข้อมูล X0,Y0 เท่ากับ จุดศูนย์กลางของวงกลม
X1 เท่ากับ รัศมีของวงกลม

3 เท่ากับเส้นโค้ง มีผลทำให้ข้อมูล X0,Y0 เท่ากับ จุดเริ่มต้นของเส้น
X1 เท่ากับ รัศมีของวงกลม
Y1 เท่ากับ มุมส่วนโค้งเริ่มต้น
X2 เท่ากับ มุมส่วนโค้งปลาย

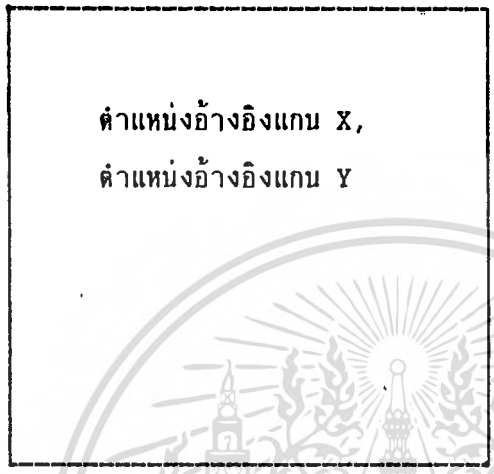
ภาคผนวก 2. การจัดข้อมูลไฟล์อุปกรณ์ในไลบรารี สามารถอธิบายได้ดังนี้

ตัวอย่างข้อมูลไฟล์ DIP14

```
0          ;ค่าขอบเขตของข้อมูลต่ำสุดแกน X
0          ;ค่าขอบเขตของข้อมูลต่ำสุดแกน Y
300        ;ค่าขอบเขตของข้อมูลสูงสุดแกน X
700        ;ค่าขอบเขตของข้อมูลสูงสุดแกน Y
0          ;ค่าตำแหน่งอ้างอิงแกน X
-50        ;ค่าตำแหน่งอ้างอิงแกน Y
14         ;จำนวนข้อมูลโคนัท
 7      0     50      0     ;คำสั่งสร้างโคนัท
 7      0    150      0
 7    300    550      0
 -
 -
12         ;จำนวนข้อมูลด้านบัดกรี
 1     50      0    50   700    0    1    ;คำสั่งสร้างลายเส้นวงจรด้านบัดกรี
 1     50    700   250   700    0    1
 -
 -
12         ;จำนวนข้อมูลด้านอุปกรณ์
 1     50      0    50   700    0    1    ;คำสั่งสร้างลายเส้นวงจรด้านอุปกรณ์
 1     50    700   250   700    0    1
 -
 -
12         ;จำนวนข้อมูลภาพอุปกรณ์
 1     50      0    50   700    0    1    ;คำสั่งสร้างภาพอุปกรณ์
 1     50    700   250   700    0    1
 -
 -
```

ขอบเขตภาพและจุดอ้างอิง

ค่าขอบเขตของข้อมูลต่ำสุดแกน X,
ค่าขอบเขตของข้อมูลต่ำสุดแกน Y



ค่าขอบเขตของข้อมูลสูงสุดแกน X,
ค่าขอบเขตของข้อมูลสูงสุดแกน Y

คำสั่งโดนัท ประกอบด้วยข้อมูลต่าง ๆ ดังนี้

7 0 50 0

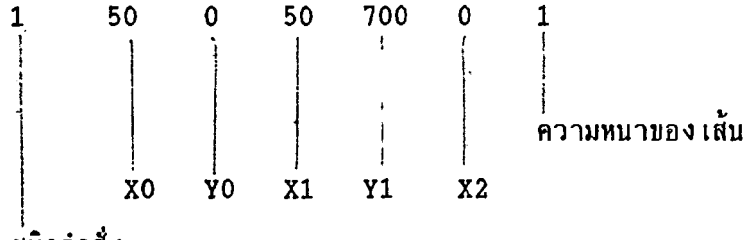
- 0 ทิศทางโดนัท (0 : แนวตั้ง, 1 : แนวนอน)
- 50 ตำแหน่ง Y
- 0 ตำแหน่ง X
- 7 หมายเลขโดนัทที่ใช้งาน (1 ถึง 12)

คำสั่งสร้างลายเส้นวงจรด้านบัดกรี ประกอบด้วยข้อมูลต่าง ๆ ดังนี้

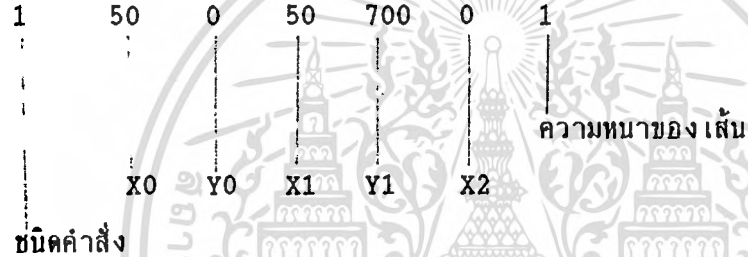
1 50 0 50 700 0 1

- 1 ความหนาของเส้น
- 0 X0
- 50 Y0
- 0 X1
- 700 Y1
- 50 X2
- 1 ชนิดคำสั่ง

คำสั่งสร้างลายเส้นวงจรด้านอุปกรณ์ ประกอบด้วยข้อมูลต่าง ๆ ดังนี้



คำสั่งสร้างภาพอุปกรณ์ ประกอบด้วยข้อมูลต่าง ๆ ดังนี้



ชนิดคำสั่ง

1 เท่ากับเส้นตรง มีข้อมูล X0,Y0 เท่ากับ จุดเริ่มต้นของเส้น
X1,Y1 เท่ากับ จุดปลายของเส้น

2 เท่ากับวงกลม มีข้อมูล X0,Y0 เท่ากับ จุดศูนย์กลางวงกลม
X1 เท่ากับ ขนาดรัศมี

3 เท่ากับเส้นโค้ง มีข้อมูล X0,Y0 เท่ากับ จุดศูนย์กลางวงกลม
X1 เท่ากับ ขนาดรัศมี
Y1 เท่ากับ มุมส่วนโค้งเริ่มต้น
X2 เท่ากับ มุมส่วนโค้งปลาย

<u>ความหนาของเส้น</u> แบ่งได้ดังนี้

0 : ความหนาขนาด 12 มิล

1 : ความหนาขนาด 16 มิล

2 : ความหนาขนาด 20 มิล

3 : ความหนาขนาด 50 มิล

ภาคผนวก 3. การกำหนดขนาดและชนิดของโดนัท สามารถอธิบายได้ดังนี้

ตัวอย่างไฟล์ข้อมูล (PCBCAD.PDS)

```
P40r
0     25    40    30
P40s
1     25    40    30
SM20
2     50    20    0
SR25
2     30    25    0
P58r
0     36    58    42
P58s
1     36    58    42
P62r
0     46    62    50
P62s
1     46    62    50
P75r
0     52    75    60
P75s
1     52    75    60
T300
2     300   50    0
T400
2     400   50    0
```

โดนัท 1 ตัวประกอบด้วยข้อมูลต่าง ๆ ดังนี้

NAME
TYPE INDM OUTD DRLL

ตัวอย่างเช่น

P40r
0 25 40 30

NAME หมายถึงชื่อที่ใช้ในการแสดงรายการโดนัทเพื่อให้ผู้ใช้สามารถเลือกขนาดและชนิดได้ถูกต้อง

TYPE หมายถึงชนิดของโดนัท แบ่งออกได้เป็น 3 ชนิดดังนี้

0 หมายถึง โดนัทแบบวงกลม และข้อมูลต่าง ๆ มีความหมายดังนี้

INDM หมายถึงขนาดเส้นผ่าศูนย์กลางวงในของโดนัท

OUTD หมายถึงขนาดเส้นผ่าศูนย์กลางวงนอกของโดนัท

DRLL หมายถึงขนาดเส้นผ่าศูนย์กลางของดอกสว่านที่ใช้เจาะรูโดนัท

1 หมายถึง โดนัทแบบสี่เหลี่ยม และข้อมูลต่าง ๆ มีความหมายดังนี้

INDM หมายถึงขนาดเส้นผ่าศูนย์กลางวงในของโดนัท

OUTD หมายถึงขนาดด้านของสี่เหลี่ยมด้านเท่า

DRLL หมายถึงขนาดเส้นผ่าศูนย์กลางของดอกสว่านที่ใช้เจาะรูโดนัท

2 หมายถึง โดนัทแบบแถบ และข้อมูลต่าง ๆ มีความหมายดังนี้

INDM หมายถึงขนาดด้านกว้างของสี่เหลี่ยม

OUTD หมายถึงขนาดด้านยาวของสี่เหลี่ยม

DRLL ไม่มีค่าในการนำมาใช้งาน

ภาคผนวก 4

รายละเอียดของโปรแกรม

```
Text File List Processing Program V4.00C Page : 1
File : A:DECLARE.PAS
Current Date : Sunday September 4, 1988
Current Time : 1:17 AM

Program : Declare Variable
Programmer : Suttinun Poramatikul


{
                +-----------------------------------+
                |                                   |
                | Unit  : Variable Declaration      |
                | Programmer : Suttinun Poramatikul.|
                | Version    : 1.00A                |
                | Last Updated :  3 Jun 1988        |
                |                                   |
                +-----------------------------------+
}
unit Declare; { Declaration Unit }

interface

Uses Graph;

const { Computing Section }
  Esc         = #27;
  On          = True;
  Off         = False;
  Left        = #75;
  Right       = #77;
  Home        = #71;
  UpK         = #72;
  PgUp        = #73;
  EndK        = #79;
  Down        = #80;
  PgDn        = #81;
  MaxProbe    = 22;
  GridOffs    = 0;
  OffS        = 2;     {Graphic Border Space Offset}
  FileLenght  = 40;
  FileLine    = 2;
  ORGX0       = 80;
  ORGY0       = 24;
  ORGLenght   = 8;
  Hidden      : FillPatternType = ($FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF);
  DelMask     : FillPatternType = ($00,$00,$00,$00,$00,$00,$00,$00);
  Angl        = 1;
  Orth        = 2;
  Dg45        = 3;

  P52r        = 1;
  P52s        = 2;
```

```
  P60r       = 3;
  P60s       = 4;
  P72r       = 5;
  P72s       = 6;
  P100r      = 7;
  P100s      = 8;
  P200r      = 9;
  P200s      = 10;
  THor       = 11;
  TVer       = 12;

  Sold       = 1;
  Comp       = 2;
  Silk       = 3;
  Drll       = 4;
  Hold1      = 1;
  Hold2      = 2;
  Hold3      = 3;
  Hold4      = 4;
  Thck1      = 1;
  Thck2      = 2;
  Thck3      = 3;
  Thck4      = 4;
  SolLayer   = LightRed;
  ComLayer   = LightGreen;
  SilLayer   = Yellow;
  DrlLayer   = White;
  NumDelStr  = 35;
  ComMax     = 500; {2000}
  DeviceMax  = 200;
  RootMax    = 10;
  GCPDMax    = 200; {200 Graphic Command per Device}
  MaxScrMono : integer = 17400;
  MaxScreenX : integer = 15400;
  MaxScreenY : integer = 9800;
  LineCode   = 01;
  CircCode   = 02;
  ArcCode    = 03;
  PadCode    = 04;
  notLib     = 0;

type
  PadAssType = array [1..12] of
               record
                 PadNme : String[4];
                 PadStl : integer;
                 InDia  : integer;
                 OutDia : integer;
                 DTool  : integer;
               end;
  FileType   = Text;
  StatFileT  = Text;
  PadAssFT   = Text;
  LineData   = array [0..72] of integer;
```

```
  SoldFtype      = ^SoldPntT;
  SoldPntT       = array [1..ComMax] of
                   record
                     CdeData : byte;
                     Libpnt  : byte;
                     PosData : array [1..4] of integer;
                     AssData : array [1..2] of integer;
                   end;
  CompFtype      = ^CompPntT;
  CompPntT       = array [1..ComMax] of
                   record
                     CdeData : byte;
                     Libpnt  : byte;
                     PosData : array [1..4] of integer;
                     AssData : array [1..2] of integer;
                   end;
  SilkFtype      = ^SilkPntT;
  SilkPntT       = array [1..ComMax] of
                   record
                     CdeData : byte;
                     Libpnt  : byte;
                     PosData : array [1..4] of integer;
                     AssData : array [1..2] of integer;
                   end;
  PadsFtype      = ^PadsPntT;
  PadsPntT       = array [1..ComMax] of
                   record
                     CdeData : byte;
                     Libpnt  : byte;
                     PosData : array [1..2] of integer;
                     AssData : array [1..2] of integer;
                   end;
  DevPntT        = array [1..GCPDMax] of
                   record
                     CdeData : byte;
                     Libpnt  : byte;
                     PosData : array [1..4] of integer;
                     AssData : array [1..2] of integer;
                   end;
  DevTemT        = array [1..GCPDMax] of
                   record
                     CdeData : byte;
                     Libpnt  : byte;
                     PosData : array [1..4] of integer;
                     AssData : array [1..2] of integer;
                   end;
  LibFtype       = ^LibPntT;
  LibPntT        = array [1..DeviceMax] of
                   record
                     LibLabel     : record
                                      XLbel,YLbel : integer;
                                      LabelName   : String[12];
                                    end;
                     LibCommt     : record
```

```
                                XCmmt,YCmmt : integer;
                                CommtName   : String[12];
                              end;
                  LibText     : array [1..32] of record
                                                   TxtLibX : integer;
                                                   TxtLibY : integer;
                                                   Lname   : string[2];
                                                 end;
                  XlibMin,YLibMin : integer;
                  XLibMax,YLibMax : integer;
                  XLibRef,YLibRef : integer;
                  XLibScn,YLibScn : integer;
                  LibStLn,LibEnLn : integer;
                end;

  TextFtype     = ^TextPntT;
  TextPntT      = array [1..200] of
                  record
                    TxtDir  : byte;
                    TxtLyr  : byte;
                    PosData : array [1..2] of integer;
                    Message : string[20];
                  end;
  TextComT      = array [1..26] of
                  record
                    TComNum : byte;
                    KeyChar : char;
                    TxtCom  : array [1..20] of record
                                                 DataX0 : integer;
                                                 DataY0 : integer;
                                                 DataX1 : integer;
                                                 DataY1 : integer;
                                               end;
                  end;
  RootLineType  = array [1..RootMax] of integer;
  POnCurType    = array [1..RootMax] of byte;
  LX0T          = array [1..RootMax] of integer;
  LY0T          = array [1..RootMax] of integer;
  LX1T          = array [1..RootMax] of integer;
  LY1T          = array [1..RootMax] of integer;
var
  SoldVar       : SoldFType;
  CompVar       : CompFType;
  SilkVar       : SilkFType;
  PadsVar       : PadsFType;
  LibVar        : LibFtype;
  DevVar        : DevPntT;
  TemVar        : DevTemT;
  TextVar       : TextFType;
  TxtComData    : TextComT;
  SoldFile      : FileType;
  CompFile      : FileType;
  SilkFile      : FileType;
  PadsFile      : FileType;
```

```
  LibsFile         : FileType;
  TextDtaFile      : FileType;
  TextComFile      : FileType;
  DataFile         : FileType;
  StatFile         : StatFileT;
  PadAssFile       : PadAssFT;
  PadAssData       : PadAssType;
  RootLine         : RootLineType;
  POnCur           : POnCurType;
  LX0              : LX0T;
  LY0              : LY0T;
  LX1              : LX1T;
  LY1              : LY1T;
  SimStep          : word;
  Step             : word;
  FileName         : string;
  ActiveDir        : string;
  LineDataXo       : LineData;
  LineDataYo       : LineData;
  LineDataXi       : LineData;
  LineDataYi       : LineData;
  GraphDriver      : integer;  { The Graphics device driver }
  GraphMode        : integer;  { The Graphics mode value }
  MaxStepOnScreen  : byte;
  StepGridOnScreenX: byte;
  StepGridOnScreenY: byte;
  GraphTextWidth   : byte;
  GraphTextHeight  : byte;
  AxisScreenX0     : byte;
  AxisScreenY0     : byte;
  AxisScreenX1     : byte;
  AxisScreenY1     : byte;
  TxtNum           : byte;
  MaxXGraph        : word;
  MaxYGraph        : word;
  GraphScreenWidth : word;
  GraphScreenHigth : word;
  GraphCornerX0    : word;
  GraphCornerX1    : word;
  GraphCornerY0    : word;
  GraphCornerY1    : word;
  GraphScreenX0    : word;
  GraphScreenY0    : word;
  GraphScreenX1    : word;
  GraphScreenY1    : word;
  MenuOffsX0       : word;
  MenuOffsY0       : word;
  MenuOffsX1       : word;
  MenuOffsY1       : word;
  LibOffsX0        : word;
  LibOffsY0        : word;
  LibOffsX1        : word;
  LibOffsY1        : word;
  TempOffsX0       : word;
```

```
  TempOffsY0        : word;
  TempOffsX1        : word;
  TempOffsY1        : word;
  LibWindowX0       : word;
  LibWindowY0       : word;
  LibWindowX1       : word;
  LibWindowY1       : word;
  AxisX0,AxisY0     : word;
  AxisX1,AxisY1     : word;
  FileAxisX0        : word;
  FileAxisY0        : word;
  FileAxisX1        : word;
  FileAxisY1        : word;
  MaxX,MaxY         : word;
  WindowX0          : word;
  WindowY0          : word;
  WindowX1          : word;
  WindowY1          : word;
  ORGWindowX0       : word;
  ORGWindowY0       : word;
  ORGWindowX1       : word;
  ORGWindowY1       : word;
  StatWindowX0      : word;
  StatWindowY0      : word;
  StatWindowX1      : word;
  StatWindowY1      : word;
  TextAxisX0        : word;
  TextAxisY0        : word;
  TextAxisX1        : word;
  TextAxisY1        : word;
  GridSizeX         : integer;   {Grid Space on X-Axis}
  GridSizeY         : Integer;   {Grid Space on Y-Axis}
  LIBTempX          : word;
  LIBTempY          : word;
  CIRTempX          : word;
  CIRTempY          : word;
  CIRPntX           : word;
  CIRPntY           : word;
  PntX ,PntY        : integer;
  TpX,TpY           : Longint;
  PntX1,PntY1       : integer;
  PntX2,PntY2       : integer;
  PntStepX,PntStepY: Longint;
  TempX,TempY       : integer;
  StartLineX        : Longint;
  StartLineY        : Longint;
  PenColor          : integer;
  LayerColor        : integer;
  SysColor          : integer;
  LWStat            : boolean;
  GridStat          : boolean;
  ORGStat           : boolean;
  OrgLine           : boolean;
  StartOrg          : boolean;
```

```
  ExtCommand       : boolean;
  CurStat          : boolean;
  GridOn           : boolean;
  LibFuncOn        : boolean;
  KeyPass          : boolean;
  GoStepStat       : boolean;
  LibCursorOn      : boolean;
  PadAssError      : boolean;
  MoreInFo         : boolean;
  LineErr          : boolean;
  LineOk           : boolean;
  SoldLayerEn      : boolean;
  CompLayerEn      : boolean;
  SilkLayerEn      : boolean;
  PadsLayerEn      : boolean;
  TextLayerEn      : boolean;
  ORGWindow        : pointer;
  StatWindow       : pointer;
  TextWindow       : pointer;
  CurBuffer        : pointer;
  KeyCommand       : char;
  TMCColor         : byte;
  TSCColor         : byte;
  TECColor         : byte;
  DMCColor         : byte;
  ErrColor         : byte;
  PadColor         : byte;
  DrllColor        : byte;
  RubberLineStyle  : byte;
  Tool             : byte;
  Layer            : byte;
  NextR            : byte;
  ScreenLayer      : byte;
  PadSize          : byte;
  Padnum           : byte;
  Trace            : byte;
  TextDir          : byte;
  DeviceDir        : byte;
  TotalHold        : word;
  CdeC             : integer;
  LIBX0,LIBY0      : integer;
  LIBX1,LIBY1      : integer;
  RefX,RefY        : integer;
  MinDataX         : integer;
  MinDataY         : integer;
  MaxDataX         : integer;
  MaxDataY         : integer;
  RefTemX,RefTemY  : integer;
  MinTemDataX      : integer;
  MinTemDataY      : integer;
  MaxTemDataX      : integer;
  MaxTemDataY      : integer;
  CIRX0,CIRY0      : integer;
  CIRX1,CIRY1      : integer;
```

```
  CIRCenX,CIRCenY  : Longint;
  CIRRadius        : Longint;
  OldCIRRadius     : Longint;
  StepLength       : integer;
  StartPosStepX    : integer;
  StartPosStepY    : integer;
  PosStepX         : integer;
  PosStepY         : integer;
  AreaScreenX0     : integer;
  AreaScreenY0     : integer;
  AreaScreenX1     : integer;
  AreaScreenY1     : integer;
  ZoomFactor       : integer;
  MovingStep       : integer;
  GoStep           : integer;
  StAngl           : integer;
  EnAngl           : integer;
  DataX0,DataY0    : Longint;
  DataX1,DataY1    : Longint;
  DataCenX,DataCenY: Longint;
  DataRadius       : Longint;
  PanOffsX0        : Longint;
  PanOffsY0        : Longint;
  ScreenCenX       : integer;
  ScreenCenY       : integer;
  ScreenLength     : integer;
  ScreenHeight     : integer;
  PositionCurX     : Longint;
  PositionCurY     : Longint;
  SoldPointer      : integer;
  CompPointer      : integer;
  SilkPointer      : integer;
  TextPointer      : integer;
  PadsPointer      : integer;
  LibPointer       : integer;
  DevPointer       : integer;
  TemPointer       : integer;
  RootLcnt         : integer;

implementation

procedure Initialize;
begin { Initialize }
end;

begin { Unit Declaration }
  Initialize;
end { Unit Declaration }.
```

```
Program : Main Program
Programmer : Suttinun Poramatikul

Program PCBCAD;

uses PCBUnit3;

begin { PCBCAD }
  PCBMain;
end. { PCBCAD }
```

```
Text File List Processing Program V4.00C Page : 10
File : A:PCBUNIT1.PAS
Current Date : Sunday September 4, 1988
Current Time : 1:19 AM

Program : PCB Part I
Programmer : Suttinun Poramatikul

{
              +-----------------------------------+
              |                                   |
              | Unit  : PCB-CAD Unit I            |
              | Programmer : Suttinun Poramatikul |
              | Version    : 1.00A                |
              | Last Updated : 24 Jun 1988        |
              |                                   |
              +-----------------------------------+

}

unit PCBUnit1;

interface

uses Printer,Dos,Crt,Graph,Declare,PCBUnit2,PCBUnit4,PCBunit5;

const
  Cursor : FillPatternType = ($08,$08,$08,$77,$08,$08,$08,$00);
  XCharOffset = 10; { x,y lines offset from upper left corner }
  YCharOffset = 2;
  PaperOffX   = 400;
  PaperOffY   = 400;
  T12         = 1;
  T16         = 2;
  T20         = 3;
  T50         = 4;
  RDGL        = 1;
  DXY         = 2;
type
  ThickT     = array [1..4,1..4] of byte;
  LoopTk     = array [1..4] of byte;
  CirLOn     = array [1..4,1..4] of byte;
var
  Thick      : ThickT;
  LoopT      : LoopTk;
  CirOn      : CirLOn;
  MaxColor   : word;     { The maximum color value available }
  ViewPort   : ViewPortType;
  TextFile   : Text;
  Command    : string;
  DataExist  : boolean;
  IOErr      : boolean;
  MenuSelect : byte;
```

```
  PntMode    : byte;
  ThkZoom    : integer;
  AdjLS      : integer;
  PaperLX    : integer;
  PaperLY    : integer;

procedure InitialGraphics;

procedure SelectMenu;

procedure MenuBlock (MenuBlockOn : boolean);

procedure PadMenuBlock (Y,Sel:integer;MenuBlockOn : boolean);

procedure ReadTextXY (x,y : word; var Str : string);

procedure OpenWindow (X0,Y0,X1,Y1 : byte);

procedure CloseWindow;

procedure ClearWindow;

procedure LibWindow;

procedure CursorBeep;

procedure EndBeep;

procedure DrawPlotter;

implementation

var
  ErrorCode   : integer;  { Reports any graphics errors }
  Ch : char;
  Size : word;
  Number : word;

procedure InitialGraphics;
begin { InitialGraphics }
{ Initialize graphics and report any errors that may occur }
{ when using Crt and graphics, turn off Crt's memory-mapped writes }
  GraphDriver := Detect;                { use autodetection }
  DetectGraph (GraphDriver,GraphMode);
  case GraphDriver of
    EGA : begin
            DirectVideo := True;
            GraphMode := EGAHi;
            TMCColor  := LightCyan;
            TSCColor  := LightGreen;
            TECColor  := White;
            DMCColor  := Black;
            PadColor  := Yellow;
            DrllColor := Blue;
```

```
            ErrColor  := LightRed;
          end;
    CGA : begin
            DirectVideo := False;
            GraphMode := CGAHi;
            TMCColor  := Yellow;
            TSCColor  := Green;
            TECColor  := White;
            DMCColor  := Black;
          end;
    HercMono : begin
            DirectVideo := False;
            GraphMode := HercMonoHi;
            TMCColor  := White;
            TSCColor  := White;
            TECColor  := White;
            DMCColor  := Black;
            PadColor  := White;
            DrllColor := White;
            ErrColor  := White;
          end;

  end { case };
  InitGraph(GraphDriver, GraphMode, ''); { activate graphics }
  ErrorCode := GraphResult;              { error? }
  if ErrorCode <> grOk then
  begin
    Writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  MaxColor := GetMaxColor; { Get the maximum allowable drawing color }
  MaxX := GetMaxX;         { Get screen resolution values }
  MaxY := GetMaxY;
  SetColor(MaxColor);
  SetLineStyle(SolidLn, 0, NormWidth);
  SetViewPort(0,0,MaxX,MaxY,ClipOn);
  GetViewSettings(ViewPort);
  case GraphDriver of
    EGA,HercMono : begin
            GraphTextHeight := MaxY div 25;
          end;
    CGA : begin
            GraphTextHeight := TextHeight ('M');
          end;
  end { case };
  GraphTextWidth := MaxX div 80;
end { InitialGraphics };

procedure ReadTextXY (x,y : word; var Str : string);
var
  OldColor : word;
begin { ReadTextXY }
  OldColor := GetColor;
  Str := '';
```

```
  Dec(x);
  Dec(y);
  x := x * GraphTextWidth;
  y := y * GraphTextHeight;
  SetColor(White);
  OutTextXY(x,y,'_');
  Ch := #0;
  repeat
    if not KeyPressed then
    begin
      { User Task }
    end
    else
    begin
      Ch := ReadKey;
      Ch := UpCase(Ch);
      if (Ch = ^H) and (Length(Str) > 0) then
      begin
        Delete(Str,Length(Str),1);
        Dec(x,GraphTextWidth);
        if GraphMode <> HercMonoHi then
           SetFillPattern(Hidden,DMCColor)
        else
           SetFillPattern(DelMask,DMCColor);
        Bar(x,y,x+GraphTextWidth,y+GraphTextHeight);
        SetColor(Black);
        OutTextXY(x+GraphTextWidth,y,'_');
        SetColor(White);
        OutTextXY(x,y,'_');
      end
      else
      begin
        if Ch <> #0 then
        begin
          if (Ch in [#32..#125]) and (Length(str) < 25) then
          begin
            Str := Str + Ch;
            SetColor(Black);
            OutTextXY(x,y,'_');
            SetColor(OldColor);
            OutTextXY(x,y,Ch);
            SetColor(White);
            OutTextXY(x+GraphTextWidth,y,'_');
            Inc(x,GraphTextWidth);
          end
          else Write(^G);
        end
        else Ch := ReadKey;
      end;
    end;
  until Ch = #$0D;
  SetColor(Black);
  OutTextXY(x+GraphTextWidth,y,'_');
  SetColor(OldColor);
```

```
end { ReadTextXY };

procedure ErrorBeep;
begin { ErrorBeep }
  Sound(2000);
  Delay(1000);
  NoSound;
end { ErrorBeep };

procedure CursorBeep;
begin { CursorBeep }
  Sound (5000);
  Delay (100);
  NoSound;
end { CursorBeep };

procedure EndBeep;
begin { CursorBeep }
  Sound (2000);
  Delay (100);
  NoSound;
end { CursorBeep };

procedure SelectMenuBeep;
begin { SelectMenuBeep }
  Sound (3000);
  Delay (100);
  NoSound;
end { SelectMenuBeep };

procedure Title;
var
  Count : byte;
  PassCode : boolean;

procedure GraphWritelnCenter (Y : byte; Str : string);
begin { GraphWritelnCenter }
  OutTextXY (MaxX div 2,Y * GraphTextHeight,Str);
end { GraphWritelnCenter };

begin { Title }
  with ViewPort do
    Rectangle(0, 0, x2-x1, y2-y1);
  SetTextJustify(CenterText,CenterText);
  SetColor(White);
  GraphWritelnCenter (3,'KMIT LADKRABANG');
  GraphWritelnCenter (5,'Faculty of Engineering');
  GraphWritelnCenter (7,'Department of Instrumentation Engineering');
  GraphWritelnCenter (12,'Computer - Aided PCB Designed Program');
  GraphWritelnCenter (17,'Version 0.001A');
  GraphWritelnCenter (19,'Programmer : Suttinun Poramatikul');
  GraphWritelnCenter (23,'Last Updated :  1 Jun 1988');
  PassCode := true;
  for Count := 1 to 6 do
```

```
  begin
    Ch := ReadKey;
    case Count of
      1 : if Upcase(Ch) <> 'P' then PassCode := False;
      2 : if Upcase(Ch) <> 'C' then PassCode := False;
      3 : if Upcase(Ch) <> 'B' then PassCode := False;
      4 : if Upcase(Ch) <> 'C' then PassCode := False;
      5 : if Upcase(Ch) <> 'A' then PassCode := False;
      6 : if Upcase(Ch) <> 'D' then PassCode := False;
    end;
  end;
  if PassCode = False then
  begin
     CloseGraph;
     Halt;
  end;
end { Title };

procedure OpenWindow (X0,Y0,X1,Y1 : byte);
begin { OpenWindow }
  Dec(X0);
  Dec(Y0);
  WindowX0 := X0 * GraphTextWidth;
  WindowY0 := Y0 * GraphTextHeight;
  WindowX1 := X1 * GraphTextWidth;
  WindowY1 := Y1 * GraphTextHeight;
  SetViewPort(WindowX0,WindowY0,WindowX1,WindowY1,ClipOn);
  GetImage(WindowX0,WindowY0,WindowX1,WindowY1,TextWindow^);
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar(WindowX0,WindowY0,WindowX1,WindowY1);
  SetColor(LightRed);
  Rectangle(WindowX0,WindowY0,WindowX1,WindowY1);
end { OpenWindow };

procedure CloseWindow;
begin { CloseWindow }
  PutImage(WindowX0,WindowY0,TextWindow^,NormalPut);
end { CloseWindow };

procedure ClearWindow;
begin { ClearWindow }
  SetFillPattern(Hidden,Blue);
  FloodFill(WindowX0+1,WindowY0+1,LightRed);
end { ClearWindow };

procedure LibWindow;
var i : integer;
begin
  DispLibWindow;
  SetColor(White);
  Rectangle(0,0,LibWindowX1-LibWindowX0,LibWindowY1-LibWindowY0);
```

```
  SetColor(TECColor);
  WriteTextXY(2,2,'LWindow');
  SetColor(White);
  for i := 1 to LibPointer do
    WriteTextXY(2,3+i,LibVar^[i].LibLabel.Name);
end;

procedure MenuBlock (MenuBlockOn : boolean);
begin { MenuBlock }
  SetLineStyle(SolidLn,$FFFF,NormWidth);
  if MenuBlockOn then
    SetColor(White)
  else
    SetColor(Black);
  Line((MenuSelect - 1) * 40 + 2 ,GraphTextHeight-4,
       (MenuSelect - 1) * 40 + 44,GraphTextHeight-4);
  SetColor(TMCColor);
end { MenuBlock };

procedure PadMenuBlock (Y,Sel:integer;MenuBlockOn : boolean);
begin { MenuBlock }
  SetLineStyle(SolidLn,$FFFF,NormWidth);
  if MenuBlockOn then
    SetColor(White)
  else
    SetColor(Black);
  if Y = 0 then
  Line(Sel * 41 + 2 ,GraphTextHeight-4,
       Sel * 41 + 44,GraphTextHeight-4);
  if Y = 1 then
  Line(Sel * 41 + 2 ,GraphTextHeight+8,
       Sel * 41 + 44,GraphTextHeight+8);
  SetColor(TMCColor);
end { MenuBlock };

procedure SelectMenu;
begin { SelectMenu }
  repeat
    Ch := ReadKey;
    if Ch = #0 then
    begin
      Ch := ReadKey;
      if Ch in [Left,Right] then
        MenuBlock(Off);
      case Ch of
        Left : begin
                 if MenuSelect > 1 then
                   Dec(MenuSelect)
                 else
                   MenuSelect := 6;
               end;
        Right : begin
                  if MenuSelect < 6 then
                    Inc(MenuSelect)
```

```
                else
                  MenuSelect := 1;
              end;
      end { case };
      CursorBeep;
      MenuBlock(On);
    end;
  until Ch in [#$0D,' '];
  SelectMenuBeep;
end { SelectMenu };

procedure CGALoInitial;
begin
  if GraphDriver = CGA then
  begin
    GraphMode := CGAC2;
    InitGraph(GraphDriver, GraphMode, '');  { activate graphics }
    ErrorCode := GraphResult;               { error? }
    if ErrorCode <> grOk then
    begin
      Writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
      Halt(1);
    end;
    MaxX := GetMaxX;          { Get screen resolution values }
    MaxY := GetMaxY;
    GraphTextWidth := MaxX div 40;
    GraphTextHeight := TextHeight ('M');
  end;
end;

procedure ReadPadAssignData;
var ok : boolean;
    Str : String;
    i,j : integer;
begin
  assign(PadAssFile,'PCBCAD.PDS');
  {$i-} reset(PadAssFile); {$i+}
  Ok := (IOresult = 0);
  if Ok then
  begin
    Readln(PadAssFile,Str);
    for i := 1 to 12 do
    begin
      Readln(PadAssFile,PadAssData[i].PadNme);
      Read  (PadAssFile,PadAssData[i].PadStl);
      Read  (PadAssFile,PadAssData[i].InDia);
      Read  (PadAssFile,PadAssData[i].OutDia);
      Readln(PadAssFile,PadAssData[i].DTool);
    end;
    PadAssError := false;
  end
  else
  begin
    WriteTextXY(2,1,'Load PCBCAD.PDS Error!!!');
```

```
    ErrorBeep;
    PadAssError := true;
  end;
end;

procedure ReadSetUpFile;
var ok : boolean;
begin
  assign(StatFile,'PCBCAD.DSU');
  {$i-} reset(StatFile); {$i+}
  Ok := (IOresult = 0);
  if Ok then
  begin
    Readln(StatFile,MenuOffsX0,MenuOffsY0,
                    MenuOffsX1,MenuOffsY1);
    Readln(StatFile,LibOffsX0,LibOffsY0,
                    LibOffsX1,LibOffsY1);
    Readln(StatFile,LibWindowX0,LibWindowY0,
                    LibWindowX1,LibWindowY1);
    Readln(StatFile,ZoomFactor);
  end
  else
  begin
    WriteTextXY(2,1,'Load PCBCAD.DSU Error!!!');
    ErrorBeep;
    delay(500);
    MenuOffsX0 := 5;
    MenuOffsY0 := 10;
    MenuOffsX1 := 10;
    MenuOffsY1 := 0;
    LibOffsX0  := 5;
    LibOffsY0  := 10;
    LibOffsX1  := 10;
    LibOffsY1  := 0;
    LibWindowX0  := 0;
    LibWindowY0  := 0;
    LibWindowX1  := 90;
    LibWindowY1  := 300;
    ZoomFactor := 2;
  end;
  Close(StatFile);
end;

{-------------------------}

function NoPntError : boolean;
begin
  NoPntError := (IOResult = 0);
  Delay(50);
end;

function Scale (X : integer) : integer;
begin
    case PntMode of
```

```
    RDGL : Scale := trunc(X * 2);
    DXY  : Scale := trunc(X * 0.508);
    end;
end;

procedure LinePenUpToXY(X0,Y0 : integer);
begin
{$i-}
  X0 := PaperOffX + Scale(X0);
  Y0 := PaperOffY + Scale(Y0);
  repeat
    case PntMode of
    RDGL : writeln(TextFile,'PU',X0,',',Y0,';');
    DXY  : writeln(lst,'M',X0,',',Y0);
    end;
  until NoPntError;
{$i+}
end;

procedure PlotLine(X0,Y0,X1,Y1 : integer);
begin
{$i-}
  X0 := PaperOffX + Scale(X0);
  Y0 := PaperOffY + Scale(Y0);
  X1 := PaperOffX + Scale(X1);
  Y1 := PaperOffY + Scale(Y1);
  repeat
    case PntMode of
    RDGL : writeln(TextFile,'PD',X0,',',Y0,',',X1,',',Y1,';');
    DXY  : writeln(lst, 'D',X0,',',Y0,',',X1,',',Y1);
    end;
  until NoPntError;
{$i+}
end;

procedure PlotLineCir(X0,Y0,R : integer);
begin
{$i-}
  X0 := PaperOffX + Scale(X0);
  Y0 := PaperOffY + Scale(Y0);
  R  := Scale(R);
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'PD',X0,',',Y0,';');
             writeln(TextFile,'CI',R,';');
           end;
    DXY  : begin
             writeln(lst,'^PD',X0,',',Y0,';');
             writeln(lst,'^CI',R,';');
           end;
    end;
  until NoPntError;
{$i+}
```

```
end;

procedure PlotCircle(X0,Y0,R : integer);
begin
{$i-}
  X0 := PaperOffX + Scale(X0);
  Y0 := PaperOffY + Scale(Y0);
  R  := Scale(R);
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'PU',X0,',',Y0,';');
             writeln(TextFile,'CI',R,';');
           end;
    DXY  : begin
             writeln(lst,'A',X0,',',Y0);
             writeln(lst,'G',R,',0,360');
           end;
    end;
  until NoPntError;
{$i+}
end;

procedure PlotArc(X0,Y0,St,En,R : integer);
begin
{$i-}
  X0 := PaperOffX + Scale(X0);
  Y0 := PaperOffY + Scale(Y0);
  R  := Scale(R);
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'PU',X0,',',Y0,';');
             writeln(TextFile,'CI',R,';');
           end;
    DXY  : begin
             writeln(lst,'A',X0,',',Y0);
             writeln(lst,'G',R,',',St,',',En);
           end;
    end;
  until NoPntError;
{$i+}
end;

procedure PlotCirPad(X0,Y0,ID,OD : integer);
begin
{$i-}
  X0 := PaperOffX + Scale(X0);
  Y0 := PaperOffY + Scale(Y0);
  ID := 3; {Scale(ID div 2);}
  OD := Scale(OD div 2);
  repeat
    case PntMode of
    RDGL : writeln(TextFile,'PU',X0,',',Y0,';');
```

```
    DXY  : writeln(lst,'^PU',X0,',',Y0,';');
    end;
  until NoPntError;
  repeat
    repeat
      case PntMode of
      RDGL : writeln(TextFile,'CI',ID,';');
      DXY  : writeln(lst,'^CI',ID,';');
      end;
    until NoPntError;
    ID := ID + ThkZoom;
  until ID > OD - AdjLS;
  repeat
    case PntMode of
    RDGL : writeln(TextFile,'PU',X0,',',Y0,';');
    DXY  : writeln(lst,'M',X0,',',Y0);
    end;
  until NoPntError;
{$i+}
end;

procedure PlotSqr(X0,Y0,R : integer);
begin
{$i-}
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'PU',X0-R,',',Y0-R,';');
             writeln(TextFile,'PD',X0+R,',',Y0-R,',',
                              X0+R,',',Y0+R,',',
                              X0-R,',',Y0+R,',',
                              X0-R,',',Y0-R,';');
           end;
    DXY  : begin
             writeln(lst,'M',X0-R,',',Y0-R);
             writeln(lst,'D',X0+R,',',Y0-R,',',
                             X0+R,',',Y0+R,',',
                             X0-R,',',Y0+R,',',
                             X0-R,',',Y0-R);
           end;
    end;
  until NoPntError;
{$i+}
end;

procedure PlotSqrPad(X0,Y0,ID,OD : integer);
begin
{$i-}
  X0 := PaperOffX + Scale(X0);
  Y0 := PaperOffY + Scale(Y0);
  ID := 3; {Scale(ID div 2);}
  OD := Scale(OD div 2);
  repeat
    repeat
```

```
    case PntMode of
    RDGL : begin
             writeln(TextFile,'PU',X0,',',Y0,';');
             writeln(TextFile,'CI',ID,';');
           end;
    DXY  : begin
             writeln(lst,'A',X0,',',Y0);
             writeln(lst,'G',ID,'0,360');
           end;
    end;
  until NoPntError;
  PlotSqr(X0,Y0,ID);
  ID := ID + ThkZoom;
until ID > OD - AdjLS;
repeat
  case PntMode of
  RDGL : writeln(TextFile,'PU',X0,',',Y0,';');
  DXY  : writeln(lst, 'M',X0,',',Y0);
  end;
until NoPntError;
{$i+}
end;

procedure PlotTeeth(X0,Y0,TeethLenght,TeethWidth : integer);
var TL,TW,TM,TC : integer;
begin
{$i-}
    X0 := PaperOffX + Scale (X0);
    Y0 := PaperOffY + Scale (Y0);
    TL := Scale (TeethLenght div 2);
    TW := Scale (TeethWidth div 2);
    TC := 3;
    if TL > TW then
    begin
      repeat
        case PntMode of
        RDGL : writeln(TextFile,'PU',X0-TL,',',Y0-TC,';');
        DXY  : writeln(lst, 'M',X0-TL,',',Y0-TC);
        end;
      until NoPntError;
      repeat
        repeat
          case PntMode of
          RDGL : writeln(TextFile,'PD',X0-TL,',',Y0-TC,',',
                                       X0+TL,',',Y0-TC,',',
                                       X0+TL,',',Y0+TC,',',
                                       X0-TL,',',Y0+TC,',',
                                       X0-TL,',',Y0-TC,';');
          DXY  : writeln(lst, 'D',X0-TL,',',Y0-TC,',',
                                       X0+TL,',',Y0-TC,',',
                                       X0+TL,',',Y0+TC,',',
                                       X0-TL,',',Y0+TC,',',
                                       X0-TL,',',Y0-TC);
          end;
```

```
        until NoPntError;
        TC := TC + ThkZoom;
      until TW < TC - AdjLS;
    end
    else
    begin
      repeat
        case PntMode of
        RDGL : writeln(TextFile,'PU',X0-TC,',',Y0-TW,';');
        DXY  : writeln(lst, 'M',X0-TC,',',Y0-TW);
        end;
      until NoPntError;
      repeat
        repeat
          case PntMode of
          RDGL : writeln(TextFile,'PD',X0-TC,',',Y0-TW,',',
                                       X0+TC,',',Y0-TW,',',
                                       X0+TC,',',Y0+TW,',',
                                       X0-TC,',',Y0+TW,',',
                                       X0-TC,',',Y0-TW,';');
          DXY  : writeln(lst, 'D',X0-TC,',',Y0-TW,',',
                                       X0+TC,',',Y0-TW,',',
                                       X0+TC,',',Y0+TW,',',
                                       X0-TC,',',Y0+TW,',',
                                       X0-TC,',',Y0-TW);
          end;
        until NoPntError;
        TC := TC + ThkZoom;
      until TL < TC - AdjLS;
    end;
    repeat
      case PntMode of
      RDGL : writeln(TextFile,'PU',X0,',',Y0,';');
      DXY  : writeln(lst, 'M',X0,',',Y0);
      end;
    until NoPntError;
{$i+}
end;

procedure CalDeltaXY(var DeX,DeY : integer; TX,TY,Trce : integer);
var Theta : real;
begin
  Theta :=  Arctan (-TY/TX);
  DeX   := trunc((Trce) * Sin (Theta));
  DeY   := trunc((Trce) * Cos (Theta));
end;

procedure PlotPCBWire(X0,Y0,X1,Y1,Trce : integer);
var DeX,DeY,Thk,PX,PY    : integer;
    Theta,ThTrce,ThX,ThY : integer;
begin
  LinePenUpToXY(X0,Y0);
  DeX := X1-X0;
  DeY := Y1-Y0;
```

```
    if DeX = 0 then
    begin
      ThY := 0;
      ThX := 0;
      for i := 1 to LoopT[Trce] do
      begin
        ThX := ThX + Thick[Trce,i];
        if CirOn[Trce,i] = 1 then
          PlotLineCir(X0,Y0,ThX);
        PlotLine(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
        if CirOn[Trce,i] = 1 then
          PlotLineCir(X1,Y1,ThX);
        PlotLine(X1-ThX,Y1-ThY,X0-ThX,Y0-ThY);
      end;
    end;
    if DeY = 0 then
    begin
      ThX := 0;
      ThY := 0;
      for i := 1 to LoopT[Trce] do
      begin
        ThY := ThY + Thick[Trce,i];
        if CirOn[Trce,i] = 1 then
          PlotLineCir(X0,Y0,ThY);
        PlotLine(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
        if CirOn[Trce,i] = 1 then
          PlotLineCir(X1,Y1,ThY);
        PlotLine(X1-ThX,Y1-ThY,X0-ThX,Y0-ThY);
      end;
    end;
    if (DeX <> 0) and (DeY <> 0) then
    begin
      Thk := 0;
      ThX := DeX; ThY := DeY;
      for i := 1 to LoopT[Trce] do
      begin
        Thk := Thk + Thick[Trce,i];
        CalDelTaXY(PX,PY,ThX,ThY,Thk);
        if CirOn[Trce,i] = 1 then
          PlotLineCir(X0,Y0,Thk);
        PlotLine(X0+PX,Y0+PY,X1+PX,Y1+PY);
        if CirOn[Trce,i] = 1 then
          PlotLineCir(X1,Y1,Thk);
        PlotLine(X1-PX,Y1-PY,X0-PX,Y0-PY);
      end;
    end;
    LinePenUpToXY(X1,Y1);
end;

procedure SetLoopData;
begin

  Thick[T12,1]:=1; Thick[T12,2]:=0; Thick[T12,3]:=0; Thick[T12,4]:=0;
  Thick[T16,1]:=2; Thick[T16,2]:=0; Thick[T16,3]:=0; Thick[T16,4]:=0;
```

```
  Thick[T20,1]:=2; Thick[T20,2]:=2; Thick[T20,3]:=0; Thick[T20,4]:=0;
  Thick[T50,1]:=6; Thick[T50,2]:=6; Thick[T50,3]:=4; Thick[T50,4]:=2;

  CirOn[T12,1]:=0; CirOn[T12,2]:=0; CirOn[T12,3]:=0; CirOn[T12,4]:=0;
  CirOn[T16,1]:=0; CirOn[T16,2]:=0; CirOn[T16,3]:=0; CirOn[T16,4]:=0;
  CirOn[T20,1]:=0; CirOn[T20,2]:=0; CirOn[T20,3]:=0; CirOn[T20,4]:=0;
  CirOn[T50,1]:=0; CirOn[T50,2]:=1; CirOn[T50,3]:=0; CirOn[T50,4]:=1;

  LoopT[T12] := 1;
  LoopT[T16] := 1;
  LoopT[T20] := 2;
  LoopT[T50] := 4;

end;

procedure PlotPCBBorder;
var X0,Y0,X1,Y1 : integer;
begin
{$i-}
   X0 := PaperOffX - Scale(50);
   Y0 := PaperOffY - Scale(50);
   X1 := PaperOffX + Scale (PaperLX + 50);
   Y1 := PaperOffY + Scale (PaperLY + 50);
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'PU',X0,',',Y0,';');
             writeln(TextFile,'PD',X1,',',Y0,',',
                              X1,',',Y1,',',
                              X0,',',Y1,',',
                              X0,',',Y0,';');
             writeln(TextFile,'PD',X1,',',Y0,',',
                              X1,',',Y1,',',
                              X0,',',Y1,',',
                              X0,',',Y0,';');
             writeln(TextFile,'PU',X0,',',Y0,';');
           end;
    DXY  : begin
             writeln(lst, 'M',X0,',',Y0);
             writeln(lst, 'D',X1,',',Y0,',',
                              X1,',',Y1,',',
                              X0,',',Y1,',',
                              X0,',',Y0);
             writeln(lst, 'D',X1,',',Y0,',',
                              X1,',',Y1,',',
                              X0,',',Y1,',',
                              X0,',',Y0);
             writeln(lst, 'M',X0,',',Y0);
           end;
    end;
  until NoPntError;
{$i+}
end;
```

```
procedure PlotTextXY(TxtDir,Lyr : byte;PX,PY : integer;Name : string);
var ch    : char;
    buf   : string[1];
    s,i,l : integer;
    TX,TY : integer;
    PCurX,PCurY : integer;
    X0,Y0,X1,Y1 : integer;

procedure TextLine(X0,Y0,X1,Y1 : integer);
begin
{$i-}
  X0 := PCurX + X0 - MinDataX;
  Y0 := PCurY + Y0 - MinDataY;
  X1 := PCurX + X1 - MinDataX;
  Y1 := PCurY + Y1 - MinDataY;
  X0 := PaperOffX + Scale (X0);
  Y0 := PaperOffY + Scale (PaperLY - Y0);
  X1 := PaperOffX + Scale (X1);
  Y1 := PaperOffY + Scale (PaperLY - Y1);
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'PU',X0,',',Y0,';');
             writeln(TextFile,'PD',X1,',',Y1,';');
             writeln(TextFile,'PD',X0,',',Y0,';');
             writeln(TextFile,'PD',X1,',',Y1,';');
             writeln(TextFile,'PU;');
           end;
    DXY  : begin
             writeln(lst,'M',X0,',',Y0);
             writeln(lst,'D',X1,',',Y1);
             writeln(lst,'D',X0,',',Y0);
             writeln(lst,'D',X1,',',Y1);
             writeln(lst,'M',X1,',',Y1);
           end;
    end;
  until NoPntError;
{$i+}
end;

procedure PlotTextInEast;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataX0;
    Y0 := TxtComData[s].TxtCom[l].DataY0;
    X1 := TxtComData[s].TxtCom[l].DataX1;
    Y1 := TxtComData[s].TxtCom[l].DataY1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX + 100;
end;
```

```
procedure PlotTextInEastInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataX0;
    Y0 := TxtComData[s].TxtCom[l].DataY0 * -1;
    X1 := TxtComData[s].TxtCom[l].DataX1;
    Y1 := TxtComData[s].TxtCom[l].DataY1 * -1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX + 100;
end;

procedure PlotTextInNorth;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataY0;
    Y0 := TxtComData[s].TxtCom[l].DataX0 * -1;
    X1 := TxtComData[s].TxtCom[l].DataY1;
    Y1 := TxtComData[s].TxtCom[l].DataX1 * -1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY - 100;
end;

procedure PlotTextInNorthInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataY0 * -1;
    Y0 := TxtComData[s].TxtCom[l].DataX0 * -1;
    X1 := TxtComData[s].TxtCom[l].DataY1 * -1;
    Y1 := TxtComData[s].TxtCom[l].DataX1 * -1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY - 100;
end;

procedure PlotTextInWestInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataX0 * -1;
    Y0 := TxtComData[s].TxtCom[l].DataY0;
    X1 := TxtComData[s].TxtCom[l].DataX1 * -1;
    Y1 := TxtComData[s].TxtCom[l].DataY1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX - 100;
```

```
end;

procedure PlotTextInWest;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataX0 * -1;
    Y0 := TxtComData[s].TxtCom[l].DataY0 * -1;
    X1 := TxtComData[s].TxtCom[l].DataX1 * -1;
    Y1 := TxtComData[s].TxtCom[l].DataY1 * -1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX - 100;
end;

procedure PlotTextInSouthInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataY0;
    Y0 := TxtComData[s].TxtCom[l].DataX0;
    X1 := TxtComData[s].TxtCom[l].DataY1;
    Y1 := TxtComData[s].TxtCom[l].DataX1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY + 100;
end;

procedure PlotTextInSouth;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := TxtComData[s].TxtCom[l].DataY0 * -1;
    Y0 := TxtComData[s].TxtCom[l].DataX0;
    X1 := TxtComData[s].TxtCom[l].DataY1 * -1;
    Y1 := TxtComData[s].TxtCom[l].DataX1;
    TextLine(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY + 100;
end;

begin   {PlotTextXY}
  case TxtDir of
  1 : begin             {North}
        PCurX := PX - 80;
        PCurY := PY;
      end;
  2 : begin             {South}
        PCurX := PX + 80;
        PCurY := PY;
      end;
```

```
  3 : begin              {East}
        PCurX := PX;
        PCurY := PY - 80;
      end;
  4 : begin              {West}
        PCurX := PX;
        PCurY := PY + 80;
      end;
  5 : begin              {NorthInv}
        PCurX := PX + 80;
        PCurY := PY;
      end;
  6 : begin              {SouthInv}
        PCurX := PX - 80;
        PCurY := PY;
      end;
  7 : begin              {EastInv}
        PCurX := PX;
        PCurY := PY + 80;
      end;
  8 : begin              {WestInv}
        PCurX := PX;
        PCurY := PY - 80;
      end;
  end;
  SetLayerColor(Lyr);
  for i := 1 to Length(Name) do
  begin
    buf := copy(Name,i,1);
    if buf = ' ' then
    begin
      case TxtDir of
      1 : PCurY := PCurY - 100;
      2 : PCurY := PCurY + 100;
      3 : PCurX := PCurX + 100;
      4 : PCurX := PCurX - 100;
      5 : PCurY := PCurY - 100;
      6 : PCurY := PCurY + 100;
      7 : PCurX := PCurX + 100;
      8 : PCurX := PCurX - 100;
      end;
    end
    else
    begin
      for s := 1 to TxtNum do
      begin
        if TxtComData[s].KeyChar = buf then
        begin
          case TxtDir of
          1 : PlotTextInNorth;
          2 : PlotTextInSouth;
          3 : PlotTextInEast;
          4 : PlotTextInWest;
          5 : PlotTextInNorthInv;
```

```
          6 : PlotTextInSouthInv;
          7 : PlotTextInEastInv;
          8 : PlotTextInWestInv;
          end;
        end;
      end;
    end;
  end;
end;

{-----------------------------}

procedure SetPlotter;
begin
{$i-}
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'IN;');
             writeln(TextFile,'SP1;');
             writeln(TextFile,'VS2;');
           end;
    DXY  : begin
             writeln(Lst,'^IN;');
             writeln(Lst,'^SP1;');
             writeln(Lst,'^VS2;');
           end;
    end;
  until NoPntError;
{$i+}
end;

procedure ReturnPlot;
begin
{$i-}
  repeat
    case PntMode of
    RDGL : begin
             writeln(TextFile,'SP0;');
             writeln(TextFile,'IN;');
           end;
    DXY  : begin
             writeln(Lst,'^SP0;');
             writeln(Lst,'^IN;');
           end;
    end;
  until NoPntError;
{$i+}
end;

{-----------------------Plot}

procedure DrawPlotter;
var X0,Y0,X1,Y1,R : Longint;
```

```
    i,St,En : integer;
    SoldColor : integer;
    CompColor : integer;
    SilkColor : integer;
    TempLayer : integer;

procedure SoldPlot;
var i,Tr : integer;
begin
  SetLayerColor(Sold);
  for i := 1 to SoldPointer do
  begin
    case SoldVar^[i].CdeData of
    LineCode : begin
                 X0 := SoldVar^[i].PosData[1] - MinDataX;
                 Y0 := SoldVar^[i].PosData[2] - MinDataY;
                 Y0 := PaperLY - Y0;
                 X1 := SoldVar^[i].PosData[3] - MinDataX;
                 Y1 := SoldVar^[i].PosData[4] - MinDataY;
                 Y1 := PaperLY - Y1;
                 Tr := SoldVar^[i].AssData[1];
                 PlotPCBWire(X0,Y0,X1,Y1,Tr);
               end;
    CircCode : begin
                 X0 := SoldVar^[i].PosData[1] - MinDataX;
                 Y0 := SoldVar^[i].PosData[2] - MinDataY;
                 Y0 := PaperLY - Y0;
                 R  := SoldVar^[i].PosData[3];
                 PlotCircle(X0,Y0,R);
               end;
    ArcCode  : begin
                 X0 := SoldVar^[i].PosData[1] - MinDataX;
                 Y0 := SoldVar^[i].PosData[2] - MinDataY;
                 Y0 := PaperLY - Y0;
                 R  := SoldVar^[i].PosData[3];
                 St := SoldVar^[i].PosData[4];
                 En := SoldVar^[i].AssData[1];
                 PlotArc(X0,Y0,St,En,R);
               end;
    end;{case}
  end;{for Loop}
end;
procedure CompPlot;
var i,Tr : integer;
begin
  SetLayerColor(Comp);
  for i := 1 to CompPointer do
  begin
    case CompVar^[i].CdeData of
    LineCode : begin
                 X0 := CompVar^[i].PosData[1] - MinDataX;
                 Y0 := CompVar^[i].PosData[2] - MinDataY;
                 Y0 := PaperLY - Y0;
                 X1 := CompVar^[i].PosData[3] - MinDataX;
```

```
                Y1 := CompVar^[i].PosData[4] - MinDataY;
                Y1 := PaperLY - Y1;
                Tr := CompVar^[i].AssData[1];
                PlotPCBWire(X0,Y0,X1,Y1,Tr);
              end;
    CircCode : begin
                X0 := CompVar^[i].PosData[1] - MinDataX;
                Y0 := CompVar^[i].PosData[2] - MinDataY;
                Y0 := PaperLY - Y0;
                R  := CompVar^[i].PosData[3];
                PlotCircle(X0,Y0,R);
              end;
    ArcCode  : begin
                X0 := CompVar^[i].PosData[1] - MinDataX;
                Y0 := CompVar^[i].PosData[2] - MinDataY;
                Y0 := PaperLY - Y0;
                R  := CompVar^[i].PosData[3];
                St := CompVar^[i].PosData[4];
                En := CompVar^[i].AssData[1];
                PlotArc(X0,Y0,St,En,R);
              end;
    end;{case}
  end;{for Loop}
end;
procedure SilkPlot;
var i,Tr : integer;
    LayerI : byte;
begin
  SetLayerColor(Silk);
  for i := 1 to SilkPointer do
  begin
    case SilkVar^[i].CdeData of
    LineCode : begin
                X0 := SilkVar^[i].PosData[1] - MinDataX;
                Y0 := SilkVar^[i].PosData[2] - MinDataY;
                Y0 := PaperLY - Y0;
                X1 := SilkVar^[i].PosData[3] - MinDataX;
                Y1 := SilkVar^[i].PosData[4] - MinDataY;
                Y1 := PaperLY - Y1;
                Tr := SilkVar^[i].AssData[1];
                PlotPCBWire(X0,Y0,X1,Y1,Tr);
              end;
    CircCode : begin
                X0 := SilkVar^[i].PosData[1] - MinDataX;
                Y0 := SilkVar^[i].PosData[2] - MinDataY;
                Y0 := PaperLY - Y0;
                R  := SilkVar^[i].PosData[3];
                PlotCircle(X0,Y0,R);
              end;
    ArcCode  : begin
                X0 := SilkVar^[i].PosData[1] - MinDataX;
                Y0 := SilkVar^[i].PosData[2] - MinDataY;
                Y0 := PaperLY - Y0;
                R  := SilkVar^[i].PosData[3];
```

```
                St := SilkVar^[i].PosData[4];
                En := SilkVar^[i].AssData[1];
                PlotArc(X0,Y0,St,En,R);
              end;
    end;{case}
  end;{for Loop}
end;

procedure TextPlot;
var X0,Y0,i    : integer;
    TDir,Lyr   : byte;
    Str        : String[20];
begin
  for i := 1 to TextPointer do
  begin
    TDir := TextVar^[i].TxtDir;
    Lyr  := TextVar^[i].TxtLyr;
    X0   := TextVar^[i].PosData[1];
    Y0   := TextVar^[i].PosData[2];
    Str  := TextVar^[i].Message;
    PlotTextXY(TDir,Lyr,X0,Y0,Str);
  end;
end;

procedure PadsPlot;
var i,X0,Y0,T,InD,OtD : integer;
    PS,PD,Pc,Pl,RdLyr : byte;
begin
  for i := 1 to PadsPointer do
  begin
    X0 := PadsVar^[i].PosData[1] - MinDataX;
    Y0 := PadsVar^[i].PosData[2] - MinDataY;
    Y0 := PaperLY - Y0;
    PS := PadsVar^[i].AssData[1];
    PD := PadsVar^[i].AssData[2];
    AskPadDiameter(PS,T,InD,OtD);
    case T of
    0 : PlotCirPad(X0,Y0,InD,OtD);
    1 : PlotSqrPad(X0,Y0,InD,OtD);
    end;
  end;
end;

procedure UserPadsPlot(PLayer : byte);
var i,X0,Y0,T,InD,OtD : integer;
    PS,PD,Pc,Pl,RdLyr : byte;
begin
  for i := 1 to PadsPointer do
  begin
    X0 := PadsVar^[i].PosData[1] - MinDataX;
    Y0 := PadsVar^[i].PosData[2] - MinDataY;
    Y0 := PaperLY - Y0;
    PS := PadsVar^[i].AssData[1];
    PD := PadsVar^[i].AssData[2];
```

```
    AskPadDiameter(PS,T,InD,OtD);
    if T =  2 then
    begin
      DecodePadLyr(PD,Pc,Pl);
      if Pl = PLayer then
      begin
        case Pc of
          THor : PlotTeeth(X0,Y0,InD,OtD);
          TVer : PlotTeeth(X0,Y0,OtD,InD);
        end;
      end;
    end;
  end;{for Loop}
end;

procedure SetDataPlot;
begin
  ThkZoom :=  2 {6 : PCB*4};
  AdjLS   :=  2 {0 : PCB*4};
  PaperLX := MaxDataX - MinDataX;
  PaperLY := MaxDataY - MinDataY;
  PntMode := DXY;
end;

begin
  CheckMinData;
  CheckMaxData;
  SetDataPlot;
  SetLoopData;
  SetPlotter;
  PlotPCBBorder;
  if SoldLayerEn then
  begin
    SoldPlot;
    UserPadsPlot(Sold);
  end;
  if CompLayerEn then
  begin
    CompPlot;
    UserPadsPlot(Comp);
  end;
  if SilkLayerEn then
  begin
    SilkPlot;
    UserPadsPlot(Silk);
  end;
  if PadsLayerEn then
    PadsPlot;
  if TextLayerEn then
    TextPlot;
  ReturnPlot;
end;

{----------------------End Plot}
```

```
begin
  New(SoldVar);
  New(CompVar);
  New(SilkVar);
  New(TextVar);
  New(PadsVar);
  New(LibVar);
  SoldPointer := 0;
  CompPointer := 0;
  SilkPointer := 0;
  PadsPointer := 0;
  LibPointer  := 0;
  TextPointer := 0;

  InitialGraphics;
  Title;
  ReadTextCommandFile;
  ReadSetUpFile;
  MenuSelect := 1;
  DataExist  := False;
  CGALoInitial;
  SetParaZoom(ZoomFactor);
  PanOffsX0 := 0;
  PanOffsY0 := 0;
  SetCoordinateData;
  SetGraphMode(GraphMode);
  initPosValue;
  InitCursor;

  GetMem(TextWindow,10000);
  GetMem(ORGWindow,1000);
  GetMem(StatWindow,3000);
  GetPicCursor;
  GetMem(CurBuffer,100);

  DispGraph;
  PositionCurX := 0;
  PositionCurY := 0;
  GridStat     := On;
  Grid (GridStat);

  ORGStat    := True;
  OpenWindowORG(81,25,89,26);
  PntStepX   := 0;
  PntStepY   := 0;
  MovingStep := 10;
  GoStep     := 50;
  WriteORG(2,1);

  ReadPadAssignData;
  RubberLineStyle := Orth;
  Layer      := Sold;
  Tool       := Hold1;
```

```
  PadSize   := 1 {P72r};
  PadDir    := THor;
  Trace     := Thck1;
  TotalHold := 0;
  TextDir   := 7; {East Dir}
  DeviceDir := 1;
  GetDir(0,FileActDir);
  FileActDir := FileActDir+'\';
  LibsActDir := FileActDir+'\';
  OpenWindowStat(38,25,79,26);

  PntStepX  := 0;
  PntStepY  := 0;
  CIRRadius := 2;
  ReadDataCIR;

  LibFuncOn  := false;
  CurStat    := false;
  GoStepStat := true;
  OrgLine    := false;
  MoreInFo   := false;
  MinDataX   := 0;
  MinDataY   := 0;
  MaxDataX   := 0;
  MaxDataY   := 0;
  RefX       := 0;
  RefY       := 0;
  SoldLayerEn := true;
  CompLayerEn := true;
  SilkLayerEn := true;
  PadsLayerEn := true;
  TextLayerEn := true;
  ZoomEn      := false;
  StartLineX := 0;
  StartLineY := 0;
  LWStat     := Off;

end { unit Simunit1 }.
```

File : A:PCBUNIT2.PAS
Current Date : Sunday September 4, 1988
Current Time : 1:31 AM

Program : PCB Part II
Programmer : Suttinun Poramatikul

```
{
              +-----------------------------------+
              |                                   |
              | Unit  : PCB-CAD Unit II           |
              | Programmer : Suttinun Poramatikul |
              | Version    : 1.00A                |
              | Last Updated : 23 Jun 1988        |
              |                                   |
              +-----------------------------------+
}

Unit  PCBUnit2;

interface

Uses   Crt,Graph,Declare,PCBUnit4,PCBUnit5,PCBUnit6;

var
       LayerTemp : byte;
       PadLyr    : byte;
       LattPntY,LattPntX : integer;
       LattTempY,LattTempX : integer;
       ExtLattCommand : boolean;
       LineCurStartOK : boolean;
       FPosCur        : boolean;
       EndStat        : boolean;
       SinData : array [0..71] of Real;
       CosData : array [0..71] of Real;

procedure InitCursor;

procedure InitPosValue;

procedure TextCursor(X0,Y0,LX1,LY1 : integer);

procedure MoveCursor(var KeyCommand : Char);

Procedure MoveLIBCursor(var KeyCommand : char);

Procedure MoveCIRCursor(var KeyCommand : char);

procedure GetPicCursor;

procedure FillPadStyle(X,Y:integer;PadType,PadDir:byte);
```

```
procedure UnDispPads(PNum : integer);

procedure UnDispCirSold(i : integer);

procedure UnDispCirComp(i : integer);

procedure UnDispCirSilk(i : integer);

procedure UnDispArcSold(i : integer);

procedure UnDispArcComp(i : integer);

procedure UnDispArcSilk(i : integer);

procedure CursorOff;

procedure CursorOn;

procedure LibCurOff;

procedure LibCurOn;

Procedure MoveLineCursor(var KeyCommand : char);

Procedure MoveTextCursor(var KeyCommand : char; LX,LY:integer);

procedure CursorControl(var KeyCommand : char);

procedure ReadDataCIR;

procedure Redraw;

procedure DispLibDevice(DrawMode : boolean);

procedure DispCurDevice;

implementation

Const  HalfCurSize  = 5;
       CurStep      = 1;
       StepFast     = 10;
       ErrorKey     = 'E';

Var    Ch           : Char;
       CurBuff      : Pointer;
       CurPattern   : Pointer;
       GraphMode    : Integer;
       Size         : word;
       OldY         : word;

procedure CursorOff;
begin
   PutImage(TempX,TempY,CurPattern^,XorPut);
end;
```

```
procedure CursorOn;
begin
    PntX  := (((PositionCurX-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    PntX1 := PntX - HalfCurSize;
    PntY  := (((PositionCurY-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    PntY1 := PntY - HalfCurSize;
    PutImage(PntX1,PntY1,CurPattern^,XorPut);
    TempX := PntX1; TempY := PntY1;
end;

procedure Cursor(var CurX,CurY : Longint ;Direction : char );
begin
  case Direction of
    Home :  begin
              CurX := CurX - (CurStep*GoStep);
              CurY := CurY - (CurStep*GoStep);
            end;{ Home  + 10 }
    UpK  :  begin
              CurY := CurY - (CurStep*GoStep);
            end;{  UP   + 10 }
    PgUp :  begin
              CurX := CurX + (CurStep*GoStep);
              CurY := CurY - (CurStep*GoStep);
            end;{ PgUp  + 10 }
    Left :  begin
              CurX := CurX - (CurStep*GoStep);
            end;{ Left  + 10 }
    Right:  begin
              CurX := CurX + (CurStep*GoStep);
            end;{ Right + 10 }
    EndK :  begin
              CurX := CurX - (CurStep*GoStep);
              CurY := CurY + (CurStep*GoStep);
            end;{ End   + 10 }
    Down :  begin
              CurY := CurY + (CurStep*GoStep);
            end;{ DOWN  + 10 }
    PgDn :  begin
              CurX := CurX + (CurStep*GoStep);
              CurY := CurY + (CurStep*GoStep);
            end;{ PgDn  + 10 }
  end;{  Case }
  if (CurX > PanOffsX0 + ScreenLength) then
     CurX := PanOffsX0 + ScreenLength
  else if (CurX < PanOffsX0) then CurX := PanOffsX0;
  if (CurY > PanOffsY0 + ScreenHeight) then
     CurY := PanOffsY0 + ScreenHeight
  else if (CurY < PanOffsY0) then CurY := PanOffsY0;
end;  { Cursor }

Procedure CursorPattern(CenX,CenY,Color : Integer);
var  PntX1,PntX2,PntY1,PntY2 : Integer;
begin
```

```
  PntY1  := CenY - HalfCurSize;
  PntX1  := CenX - HalfCurSize;
  PntY2  := CenY + HalfCurSize;
  PntX2  := CenX + HalfCurSize;
  SetColor(Color);
  Circle (CenX,CenY,3);
  Line(CenX,PntY1,CenX,CenY-2);
  Line(CenX,CenY+2,CenX,PntY2);
  Line(PntX1,CenY,CenX-2,CenY);
  Line(CenX+2,CenY,PntX2,CenY);
end;

procedure GetPicCursor;
begin
  CursorPattern(PntX+10,PntY+10,White);
  GetImage(PntX1+10,PntY1+10,PntX2+10,PntY2+10,CurPattern^);
  PutImage(PntX1+10,PntY1+10,CurPattern^,XorPut);
end;

procedure FillCircle(X,Y,R: integer);
var X0,Y0,X1,Y1,K : integer;
begin
  R := ((R div 2) * GridSizeX) div StepLength;
  Circle(X,Y,R);
end;

procedure FillPadCir(X,Y,InD,OutD : integer);
begin
  InD := ((Ind div 2) * GridSizeX) div StepLength;
  OutD:= ((OutD div 2) * GridSizeX) div StepLength;
  Circle(X,Y,InD);
  Circle(X,Y,OutD);
end;

procedure DrawTeeth(X,Y,TeethLenght,TeethWidth : integer);
var Cx,Cy,i,Dy : integer;
begin
  TeethLenght := TeethLenght div 2;
  TeethWidth  := TeethWidth  div 2;
  Cx := (TeethLenght * GridSizeX) div StepLength;
  Cy := (TeethWidth  * GridSizeY) div StepLength;
  if Cx < 1 then Cx := 0;
  Dy := Cy * 2;
  if Cy < 1 then Dy := 0;
  for i := 0 to Dy do
    Line(X-Cx,Y-Cy+i,X+Cx,Y-Cy+i);
end;

procedure FillBox(X,Y,R: integer);
var XC,YC,X0,Y0,X1,Y1,RX,RY : integer;
begin
  R   := R div 2;
  RX  := (R * GridSizeX) div StepLength;
  RY  := (R * GridSizeY) div StepLength;
```

```
  X0 := X - RX;
  Y0 := Y - RY;
  X1 := X + RX;
  Y1 := Y + RY;
  Rectangle(X0,Y0,X1,Y1);
end;

procedure FillPadBox(X,Y,InD,OutD : integer);
var XC,YC,X0,Y0,X1,Y1,RX,RY,DX,DY : integer;
begin
  InD := InD div 2;
  OutD:= OutD div 2;
  DX  := (OutD * GridSizeX) div StepLength;
  DY  := (OutD * GridSizeY) div StepLength;
  RX  := (InD * GridSizeX) div StepLength;
  RY  := (InD * GridSizeY) div StepLength;
  X0 := X - DX;
  Y0 := Y - DY;
  X1 := X + DX;
  Y1 := Y + DY;
  Rectangle(X0,Y0,X1,Y1);
end;

procedure DrawPadTypeR(X,Y,InD,OutD,Dtl : integer);
begin
  SetColor(DrllColor);
  FillCircle(X,Y,Dtl);
  SetColor(PadColor);
  FillPadCir(X,Y,InD,OutD);
end;

procedure DrawPadTypeS(X,Y,InD,OutD,Dtl : integer);
begin
  SetColor(DrllColor);
  FillCircle(X,Y,Dtl);
  SetColor(PadColor);
  FillPadCir(X,Y,InD,OutD);
  FillPadBox(X,Y,InD,OutD);
end;

procedure DrawPadTypeT(X,Y,InD,OutD,Dtl : integer);
begin
  DrawTeeth(X,Y,InD,OutD);
end;

procedure FillPadStyle(X,Y:integer;PadType,PadDir:byte);
var PD : byte;
begin
  X := GraphScreenX0 + (((X-PanOffsX0)*GridSizeX) div StepLength);
  Y := GraphScreenY0 + (((Y-PanOffsY0)*GridSizeY) div StepLength);
  if PadAssError then
  begin
     case PadType of
     1 : DrawPadTypeR(X,Y,16,32,22);
```

```
    2 : DrawPadTypeS(X,Y,16,32,22);
    3 : DrawPadTypeR(X,Y,20,52,30);
    4 : DrawPadTypeS(X,Y,20,52,30);
    5 : DrawPadTypeR(X,Y,26,62,40);
    6 : DrawPadTypeS(X,Y,26,62,40);
    7 : DrawPadTypeR(X,Y,30,75,50);
    8 : DrawPadTypeS(X,Y,30,75,50);
    9 : DrawPadTypeR(X,Y,40,100,60);
   10 : DrawPadTypeS(X,Y,40,100,60);
11,12 : begin
          DecodePadLyr(PadDir,PD,PadLyr);{*****Err***}
          SetLayerColor(PadLyr);
          case PD of
          THor : DrawPadTypeT(X,Y,300,50,0);
          TVer : DrawPadTypeT(X,Y,50,300,0);
          end;
        end;
    end;
 end
 else
 begin
   case PadAssData[PadType].PadStl of
   0 {R}    : DrawPadTypeR (X,Y,
                            PadAssData[PadType].InDia,
                            PadAssData[PadType].OutDia,
                            PadAssData[PadType].DTool);
   1 {S}    : DrawPadTypeS (X,Y,
                            PadAssData[PadType].InDia,
                            PadAssData[PadType].OutDia,
                            PadAssData[PadType].DTool);
   2 {user} : begin
                DecodePadLyr(PadDir,PD,PadLyr);{*****Err***}
                SetLayerColor(PadLyr);
                case PD of
                THor : DrawPadTypeT (X,Y,
                                     PadAssData[PadType].InDia,
                                     PadAssData[PadType].OutDia,
                                     PadAssData[PadType].DTool);
                TVer : DrawPadTypeT (X,Y,
                                     PadAssData[PadType].OutDia,
                                     PadAssData[PadType].InDia,
                                     PadAssData[PadType].DTool);
                end;
              end;
    end;
  end;
end;

procedure UnDispPads(PNum : integer);
var PadColTemp : byte;
    DrlColTemp : byte;
    X0,Y0      : integer;
    PS,PD,Pc,Pl: byte;
begin
```

```
  DispGraph;
  CursorOff;
  PadColTemp := PadColor;
  DrlColTemp := DrllColor;
  PadColor   := Black;
  DrllColor  := Black;
  X0 := PadsVar^[PNum].PosData[1];
  Y0 := PadsVar^[PNum].PosData[2];
  PS := PadsVar^[PNum].AssData[1];
  PD := PadsVar^[PNum].AssData[2];
  DeCodePadLyr(PD,Pc,Pl);
  Pl := Black;
  EnCodePadLyr(PD,Pc,Pl);
  FillPadStyle(X0,Y0,PS,PD);
  PadColor  := PadColTemp;
  DrllColor := DrlColTemp;
  CursorOn;
end;

procedure TextCursor(X0,Y0,LX1,LY1 : integer);
var X1,Y1 : integer;
begin
  X1 := X0+LX1;
  Y1 := Y0+LY1;
  DrawLine(X0,Y0,X1,Y0);
  DrawLine(X1,Y0,X1,Y1);
  DrawLine(X1,Y1,X0,Y1);
  DrawLine(X0,Y1,X0,Y0);
end;

procedure DrawLIB(X,Y:integer);
var X0,Y0,X1,Y1 : longint;
begin
  X0 :=  RefX;
  Y0 :=  RefY;
  X1 :=  RefX + MaxDataX;
  Y1 :=  RefY + MaxDataY;
  DrawLine(X+X0,Y+Y0,X+X1,Y+Y0);
  DrawLine(X+X1,Y+Y0,X+X1,Y+Y1);
  DrawLine(X+X1,Y+Y1,X+X0,Y+Y1);
  DrawLine(X+X0,Y+Y1,X+X0,Y+Y0);
end;

procedure LibCurOff;
begin
  DrawLIB(PositionCurX,PositionCurY);
end;

procedure LibCurOn;
begin
  DrawLIB(PositionCurX,PositionCurY);
end;

procedure CircleOff(X0,Y0,R: integer);
```

```
begin
  SetColor(Black);
  Circle(X0,Y0,R);
  SetLayerColor(Layer);
end;

procedure ArcOff(X0,Y0,St,En,R : integer);
begin
  SetColor(Black);
  Arc(X0,Y0,St,En,R);
  SetLayerColor(Layer);
end;

procedure UnDispCirSold(i : integer);
var X0,Y0,R : integer;
begin
  X0 := ((SoldVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength + GraphScreenX0;
  Y0 := ((SoldVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength + GraphScreenY0;
  R  := (SoldVar^[i].PosData[3] * GridSizeX) div StepLength;
  CircleOff(X0,Y0,R);
end;

procedure UnDispCirComp(i : integer);
var X0,Y0,R : integer;
begin
  X0 := ((CompVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength + GraphScreenX0;
  Y0 := ((CompVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength + GraphScreenY0;
  R  := (CompVar^[i].PosData[3] * GridSizeX) div StepLength;
  CircleOff(X0,Y0,R);
end;

procedure UnDispCirSilk(i : integer);
var X0,Y0,R : integer;
begin
  X0 := ((SilkVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength + GraphScreenX0;
  Y0 := ((SilkVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength + GraphScreenY0;
  R  := (SilkVar^[i].PosData[3] * GridSizeX) div StepLength;
  CircleOff(X0,Y0,R);
end;

procedure UnDispArcSold(i : integer);
var X0,Y0,R,St,En : integer;
begin
  X0 := ((SoldVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength + GraphScreenX0;
  Y0 := ((SoldVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength + GraphScreenY0;
  R  := (SoldVar^[i].PosData[3] * GridSizeX) div StepLength;
  St :=  SoldVar^[i].PosData[4];
  En :=  SoldVar^[i].AssData[1];
  ArcOff(X0,Y0,St,En,R);
end;

procedure UnDispArcComp(i : integer);
var X0,Y0,R,St,En : integer;
begin
```

```
  X0 := ((CompVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength + GraphScreenX0;
  Y0 := ((CompVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength + GraphScreenY0;
  R  := (CompVar^[i].PosData[3] * GridSizeX) div StepLength;
  St :=  CompVar^[i].PosData[4];
  En :=  CompVar^[i].AssData[1];
  ArcOff(X0,Y0,St,En,R);
end;

procedure UnDispArcSilk(i : integer);
var X0,Y0,R,St,En : integer;
begin
  X0 := ((SilkVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength + GraphScreenX0;
  Y0 := ((SilkVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength + GraphScreenY0;
  R  := (SilkVar^[i].PosData[3] * GridSizeX) div StepLength;
  St :=  SilkVar^[i].PosData[4];
  En :=  SilkVar^[i].AssData[1];
  ArcOff(X0,Y0,St,En,R);
end;

{-----------------------------------------}

Procedure MoveLIBCursor(var KeyCommand : char);
var LineLIBStat : boolean;
    KeyPass,ok  : boolean;
    DisLibStat  : boolean;
    DisLibCnt   : integer;
    KeyCount    : integer;
    DispOn      : boolean;

procedure DispCurNotKeyPress;
begin
   if KeyCount = 800 then
   begin
     DrawLIB(PositionCurX,PositionCurY);
     KeyPass := true;
   end;
   KeyCount := KeyCount + 1;
end;

procedure DispLibNotKeyPress;
begin
  if LibCursorOn then
  begin
    if not DisLibStat then
    begin
      if DisLibCnt = 25000 then
      begin
        DisLibStat := true;
        DispCurDevice;
        DrawLIB(PositionCurX,PositionCurY);
        DispOn := true;
        KeyPass := false;
      end;
    end;
```

```
    DisLibCnt := DisLibCnt + 1;
  end;
end;

procedure DispOffCur;
begin
  if DisLibStat then
  begin
    DispCurDevice;
    ExtCommand := true;
    KeyCommand := ' '; {Return Routine again}
  end;
end;

begin
  SetColor(LightGray);
  KeyCount   := 0;
  DisLibCnt  := 0;
  KeyPass    := false;
  DisLibStat := false;
  ExtCommand := false;
  FuncKeysOn := false;
  DispOn     := false;
  repeat
   if Keypressed then
   begin
     KeyCommand := ReadKey;
     if KeyPass then LibCurOff;
     DispOffCur;
     KeyCount   := 0;
     DisLibCnt  := 0;
     DispOn     := false;
     KeyPass    := false;
     DisLibStat := false;
     if KeyCommand = #0 then
     begin
       KeyCommand := ReadKey;
       CursorOff;
       Cursor(PositionCurX,PositionCurY,KeyCommand);
       CursorOn;
       DispORG;    {Display Cursor Position}
       DispGraph;
       case KeyCommand of
       #67 : begin  {Function Key F9  [Redraw Screen]
               KeyCommand := 'B';
               ExtCommand := true;
               FuncKeysOn := true;
             end;
       #68 : begin  {Function Key F10 [Screen PAN]}
               KeyCommand := 'C';
               ExtCommand := true;
               FuncKeysOn := true;
             end;
       end; {case}
```

```
      end
      else
      begin
        ExtCommand := true;
      end;
    end
    else
    begin
      if not DispOn then
      begin
        if not KeyPass then
          DispCurNotKeyPress
        else
          DispLibNotKeyPress;
      end;
    end;
  Until ExtCommand;
end;

Procedure MoveLineCursor(var KeyCommand : char);
var LineLIBStat : boolean;
    KeyPass,ok  : boolean;
    KeyCount    : integer;
    DispOn      : boolean;

procedure DispCurNotKeyPress;
begin
   if KeyCount = 800 then
   begin
     MoveDelLineOn(PositionCurX,PositionCurY);
     KeyPass := true;
   end;
   KeyCount := KeyCount + 1;
end;

begin
  SetColor(White);
  KeyCount   := 0;
  KeyPass    := false;
  ExtCommand := false;
  FuncKeysOn := false;
  DispOn     := false;
  repeat
   if Keypressed then
   begin
     KeyCommand := ReadKey;
     if KeyPass then
        MoveDelLineOn(PositionCurX,PositionCurY);
     KeyCount   := 0;
     DispOn     := false;
     KeyPass    := false;
     if KeyCommand = #0 then
     begin
       KeyCommand := ReadKey;
```

```
      CursorOff;
      Cursor(PositionCurX,PositionCurY,KeyCommand);
      CursorOn;
      DispORG;    {Display Cursor Position}
      DispGraph;
      case KeyCommand of
      #66 : begin  {Function Key F8  [Delete Line]}
              KeyCommand := 'A';
              ExtCommand := true;
              FuncKeysOn := true;
            end;
      #67 : begin  {Function Key F9  [Redraw Screen]}
              KeyCommand := 'B';
              ExtCommand := true;
              FuncKeysOn := true;
            end;
      #68 : begin  {Function Key F10 [Screen PAN]}
              KeyCommand := 'C';
              ExtCommand := true;
              FuncKeysOn := true;
            end;
      end; {case}
    end
    else
    begin
      ExtCommand := true;
    end;
  end
  else
  begin
    if not DispOn then
    begin
      if not KeyPass then DispCurNotKeyPress;
    end;
  end;
 Until ExtCommand;
end;

{-------------------------------------------------}

Procedure MoveTextCursor(var KeyCommand : char; LX,LY:integer);
var LineLIBStat : boolean;
    KeyPass,ok  : boolean;
    KeyCount    : integer;
    DispOn      : boolean;

procedure DispTextCurNotKeyPress;
begin
  if KeyCount = 800 then
  begin
    TextCursor(PositionCurX,PositionCurY,LX,LY);
    KeyPass := true;
  end;
  KeyCount := KeyCount + 1;
```

```
end;

begin
  SetColor(White);
  KeyCount   := 0;
  KeyPass    := false;
  ExtCommand := false;
  FuncKeysOn := false;
  DispOn     := false;
  repeat
   if Keypressed then
   begin
     KeyCommand := ReadKey;
     if KeyPass then
        TextCursor(PositionCurX,PositionCurY,LX,LY);
     KeyCount   := 0;
     DispOn     := false;
     KeyPass    := false;
     if KeyCommand = #0 then
     begin
       KeyCommand := ReadKey;
       CursorOff;
       Cursor(PositionCurX,PositionCurY,KeyCommand);
       CursorOn;
       DispORG;    {Display Cursor Position}
       DispGraph;
       case KeyCommand of
       #66 : begin  {Function Key F8  [Delete Line]}
               KeyCommand := 'A';
               ExtCommand := true;
               FuncKeysOn := true;
             end;
       #67 : begin  {Function Key F9  [Redraw Screen]}
               KeyCommand := 'B';
               ExtCommand := true;
               FuncKeysOn := true;
             end;
       #68 : begin  {Function Key F10 [Screen PAN]}
               KeyCommand := 'C';
               ExtCommand := true;
               FuncKeysOn := true;
             end;
       end; {case}
     end
     else
     begin
       ExtCommand := true;
     end;
   end
   else
   begin
     if not DispOn then
     begin
       if not KeyPass then DispTextCurNotKeyPress;
```

```
    end;
   end;
  Until ExtCommand;
end;

{-------------------------------------------}

procedure CursorControl(var KeyCommand : char);
var ok,StartOk : boolean;
begin
  SetColor(LightGray);
  repeat
   ExtCommand := false;
   FuncKeysOn := false;
   if Keypressed then
   begin
     KeyCommand := ReadKey;
     if KeyCommand = #0 then
     begin
       KeyCommand := ReadKey;
       case KeyCommand of
       #67 : begin  {Function Key F9  [Redraw Screen]}
                KeyCommand := 'B';
                ExtCommand := true;
                FuncKeysOn := true;
             end;
        #68 : begin  {Function Key F10 [Screen PAN]}
                KeyCommand := 'C';
                ExtCommand := true;
                FuncKeysOn := true;
             end;
       end;
       CursorOff;
       Cursor(PositionCurX,PositionCurY,KeyCommand);
       DispORG;    {Display Cursor Position}
       DispGraph;
       CursorOn;
     end
     else
     begin
       ExtCommand := true;
     end;
   end;
  Until ExtCommand;
end;

{-------------------------------------------}

procedure ReadDataCIR;
var Datafilevar : file of Real;
    StepAng : Real;
    i       : integer;
begin
    assign(Datafilevar,'PCBCAD.DTA');
```

```
    reset(Datafilevar);
    StepAng := 2*Pi/72;
    for i := 0 to 71 do
    begin
      read(Datafilevar,SinData[i]);
    end;
    for i := 0 to 71 do
    begin
      read(Datafilevar,CosData[i]);
    end;
    close(Datafilevar);
end;

procedure CalPadCurLine(Radius:integer);
var Xs,Xe,Ys,Ye : integer;
    i,j         : integer;
begin
   i := 0;
   repeat
      LineDataXo[i] := trunc(((GridSizeX * Radius) div StepLength) * SinData[i]);
      LineDataYo[i] := trunc(((GridSizeY * Radius) div StepLength) * CosData[i]);
      j := i + 2;
      if i = 70 then j:=0;
      LineDataXi[i] := trunc(((GridSizeX * Radius) div StepLength) * SinData[j]);
      LineDataYi[i] := trunc(((GridSizeY * Radius) div StepLength) * CosData[j]);
      i := i + 2;
   Until i > 70;
end;

procedure PadDrawCir(X,Y,R : integer);
var X0,Y0,X1,Y1 : longint;
begin
  R := R Div 2;
  CalPadCurLine(R);
  i := 0;
  repeat
    X0 := GraphCornerX0 + LineDataXo[i] + ((X-PanOffsX0)*GridSizeX) div StepLength;
    Y0 := GraphCornerY0 + LineDataYo[i] + ((Y-PanOffsY0)*GridSizeY) div StepLength;
    X1 := GraphCornerX0 + LineDataXi[i] + ((X-PanOffsX0)*GridSizeX) div StepLength;
    Y1 := GraphCornerY0 + LineDataYi[i] + ((Y-PanOffsY0)*GridSizeY) div StepLength;
    DrawAngLine(X0,Y0,X1,Y1);
    i := i + 2;
  until i > 70;
end;

procedure PadDrawBox(X,Y,R : integer);
var RX,RY : integer;
    X0,Y0,X1,Y1 : longint;
begin
  R   := R div 2;
  X   := GraphCornerX0 + ((X-PanOffsX0)*GridSizeX) div StepLength;
  Y   := GraphCornerY0 + ((Y-PanOffsY0)*GridSizeY) div StepLength;
  RX  := (R * GridSizeX) div StepLength;
  RY  := (R * GridSizeY) div StepLength;
```

```
  X0 := X - RX;
  Y0 := Y - RY;
  X1 := X + RX;
  Y1 := Y + RY;
  DrawAngLine(X0,Y0,X1,Y0);
  DrawAngLine(X1,Y0,X1,Y1);
  DrawAngLine(X1,Y1,X0,Y1);
  DrawAngLine(X0,Y1,X0,Y0);
end;

procedure DrawPadTeeth(X,Y,TeethLenght,TeethWidth : integer);
var Cx,Cy,i : integer;
    X0,Y0,X1,Y1 : longint;
begin
  TeethLenght := TeethLenght div 2;
  TeethWidth  := TeethWidth  div 2;
  X   := GraphCornerX0 + ((X-PanOffsX0)*GridSizeX) div StepLength;
  Y   := GraphCornerY0 + ((Y-PanOffsY0)*GridSizeY) div StepLength;
  Cx := (TeethLenght * GridSizeX) div StepLength;
  Cy := (TeethWidth  * GridSizeY) div StepLength;
  X0 :=  X - Cx;
  Y0 :=  Y - Cy;
  X1 :=  X + Cx;
  Y1 :=  Y + Cy;
  DrawAngLine(X0,Y0,X1,Y0);
  DrawAngLine(X1,Y0,X1,Y1);
  DrawAngLine(X1,Y1,X0,Y1);
  DrawAngLine(X0,Y1,X0,Y0);
end;

procedure DrawPadCur(X,Y,Ps,Pd: integer);
var i,InD,OuD,T : integer;
    R : integer;
    X0,Y0,X1,Y1 : longint;
    Pt : byte;
begin
  AskPadDiameter(Ps,T,InD,OuD);
  case T of
  0 : begin
        PadDrawCir(X,Y,InD);
        PadDrawCir(X,Y,OuD);
      end;
  1 : begin
        PadDrawCir(X,Y,InD);
        PadDrawBox(X,Y,OuD);
      end;
  2 : begin
        DecodePadLyr(Pd,Pt,PadLyr);{*****Err***}
        Pd := Pt;
        case Pd of
       THor : DrawPadTeeth(X,Y,InD,OuD);
        TVer : DrawPadTeeth(X,Y,OuD,InD);
        end;
      end;
```

```
  end;
end;

procedure CalCIRArc(StepX,StepY,Radius:integer);
var Xs,Xe,Ys,Ye : integer;
    i,j         : integer;
begin
   for i := 0 to 71 do
   begin
      LineDataXo[i] := GraphCornerX0 + (((StepX-PanOffsX0)*GridSizeX) div StepLength)
                     + trunc(((GridSizeX * Radius) div StepLength) * SinData[i]);
      LineDataYo[i] := GraphCornerY0 + (((StepY-PanOffsY0)*GridSizeY) div StepLength)
                     + trunc(((GridSizeY * Radius) div StepLength) * CosData[i]);
      j := i + 1;
      if i = 71 then j:=0;
      LineDataXi[i] := GraphCornerX0 + (((StepX-PanOffsX0)*GridSizeX) div StepLength)
                     + trunc(((GridSizeX * Radius) div StepLength) * SinData[j]);
      LineDataYi[i] := GraphCornerY0 + (((StepY-PanOffsY0)*GridSizeY) div StepLength)
                     + trunc(((GridSizeY * Radius) div StepLength) * CosData[j]);
   end;
end;
procedure DrawArcCur(PosF,PosE:integer);
var i : integer;
   ch : char;
begin
     PosF := PosF div 5;
     PosF := PosF + 18;
     if PosF >= 72 then PosF := PosF - 72;
     PosE := PosE div 5;
     PosE := PosE + 18;
     if PosE >= 72 then PosE := PosE - 72;
     i := PosF;
     while i <> PosE do
     begin
      DrawAngLine(LineDataXo[i],LineDataYo[i],LineDataXi[i],LineDataYi[i]);
      i := i + 1;
      if i > 71 then i := 0;
      If KeyPressed then ch := ReadKey;
     end;
end;

procedure DrawCircle;
var i : integer;
begin
  for i := 0 to 71 do
    DrawAngLine(LineDataXo[i],LineDataYo[i],LineDataXi[i],LineDataYi[i]);
end;

procedure DrawLineCIR(StepX,StepY,Radius:integer);
var Xs,Xe,Ys,Ye : integer;
    i,j         : integer;
begin
   for i := 0 to 71 do
   begin
```

```
      LineDataXo[i] := GraphCornerX0 + (((StepX-PanOffsX0)*GridSizeX) div StepLength)
                     + trunc(((GridSizeX * Radius) div StepLength) * SinData[i]);
      LineDataYo[i] := GraphCornerY0 + (((StepY-PanOffsY0)*GridSizeY) div StepLength)
                     + trunc(((GridSizeY * Radius) div StepLength) * CosData[i]);
      j := i + 1;
      if i = 71 then j:=0;
      LineDataXi[i] := GraphCornerX0 + (((StepX-PanOffsX0)*GridSizeX) div StepLength)
                     + trunc(((GridSizeX * Radius) div StepLength) * SinData[j]);
      LineDataYi[i] := GraphCornerY0 + (((StepY-PanOffsY0)*GridSizeY) div StepLength)
                     + trunc(((GridSizeY * Radius) div StepLength) * CosData[j]);
      DrawAngLine(LineDataXo[i],LineDataYo[i],LineDataXi[i],LineDataYi[i]);
    end;
end;

procedure DrawCIR(X,Y,Radius:integer);
var RadiusArm : integer;
begin
  RadiusArm := Radius div 4;
  DrawStepLine(X-Radius,Y-Radius,X-Radius+RadiusArm,Y-Radius);
  DrawStepLine(X+Radius-RadiusArm,Y-Radius,X+Radius,Y-Radius);
  DrawStepLine(X+Radius,Y-Radius,X+Radius,Y-Radius+RadiusArm);
  DrawStepLine(X+Radius,Y+Radius-RadiusArm,X+Radius,Y+Radius);
  DrawStepLine(X+Radius,Y+Radius,X+Radius-RadiusArm,Y+Radius);
  DrawStepLine(X-Radius+RadiusArm,Y+Radius,X-Radius,Y+Radius);
  DrawStepLine(X-Radius,Y+Radius,X-Radius,Y+Radius-RadiusArm);
  DrawStepLine(X-Radius,Y-Radius+RadiusArm,X-Radius,Y-Radius);
end;

procedure CalRadius;
begin
  if (PositionCurX <= CIRCenX) then
  begin
    CIRRadius := StepLength div 2;
  end
  else CIRRadius := (PositionCurX-CIRCenX);
  OldCIRRadius := CIRRadius;
end;

procedure DrawScaleCIRoff;
var DelayCount : integer;
    Pass : boolean;
begin
  DrawCIR(CIRCenX,CIRCenY,OldCIRRadius);
end;

procedure DrawScaleCIROn;
var DelayCount : integer;
    Pass : boolean;
begin
  DrawCIR(CIRCenX,CIRCenY,CIRRadius);
  DelayCount := 0;
  pass := false;
  repeat
   if not pass then
```

```
   begin
     if DelayCount = 3000 then
     begin
       DrawLineCIR(CIRCenX,CIRCenY,CIRRadius);
       pass := true;
     end;
     DelayCount := DelayCount+1;
   end;
  until Keypressed;
  if pass then
    DrawLineCIR(CIRCenX,CIRCenY,CIRRadius);
end;

procedure CalAngular(PosF : integer);
var X,Y : integer;
begin
  X := GraphCornerX0 + (((CIRCenX-PanOffsX0)*GridSizeX) div StepLength);
  Y := GraphCornerY0 + (((CIRCenY-PanOffsY0)*GridSizeY) div StepLength);
  DrawAngLine(X,Y,LineDataXo[PosF],LineDataYo[PosF]);
end;

procedure EndAnglOff(PosF,PosE : integer);
var i,X,Y : integer;
begin
  X := GraphCornerX0 + (((CIRCenX-PanOffsX0)*GridSizeX) div StepLength);
  Y := GraphCornerY0 + (((CIRCenY-PanOffsY0)*GridSizeY) div StepLength);
  DrawAngLine(X,Y,LineDataXo[PosE],LineDataYo[PosE]);
  if EndStat then
  begin
     i := PosF;
     while i <> PosE do
     begin
      DrawAngLine(LineDataXo[i],LineDataYo[i],LineDataXi[i],LineDataYi[i]);
      i := i + 1;
      if i > 71 then i := 0;
     end;
  end;
end;

procedure EndAnglOn(PosF,PosE : integer);
var i,j,EndCount,X,Y : integer;
    pass : boolean;
begin
  EndStat := false;
  X := GraphCornerX0 + (((CIRCenX-PanOffsX0)*GridSizeX) div StepLength);
  Y := GraphCornerY0 + (((CIRCenY-PanOffsY0)*GridSizeY) div StepLength);
  DrawAngLine(X,Y,LineDataXo[PosE],LineDataYo[PosE]);
  EndCount := 0;
  pass := false;
  repeat
   if not pass then
   begin
    EndCount := EndCount+1;
    if EndCount = 3000 then
```

```
    begin
     i := PosF;
     while i <> PosE do
     begin
      DrawAngLine(LineDataXo[i],LineDataYo[i],LineDataXi[i],LineDataYi[i]);
      i := i + 1;
      if i > 71 then i := 0;
      pass := true;
     end;
     EndStat := true;
    end;
   end;
  until keypressed;
end;

procedure KeyControlCur(var PosAng:integer;KeyCom:Char);
begin
   Case KeyCom of
   Left  : begin
             PosAng := PosAng - 1;
             if PosAng < 0 then PosAng := 71;
           end;{ Left  - 1 }
   Right : begin
             PosAng := PosAng + 1;
             if PosAng > 71 then PosAng := 0;
           end;{ Right + 1 }
   end;
end;

procedure CircleText(num:byte);
begin
  DispText;
  GetLayerColor(SysColor);
  SetColor(SysColor);
  DeleteLine(2);
  case num of
  1 : WriteTextXY(2,2,'CIRC : Center Enter');
  2 : WriteTextXY(2,2,'CIRC : Radius <- ->');
  3 : WriteTextXY(2,2,'CIRC : StAngl Enter');
  4 : WriteTextXY(2,2,'CIRC : EnAngl Enter');
  end;
  DispGraph;
end;

Procedure StartAnglCurSor(var KeyCommand : char);
var StartCir : boolean;
    MarkCir  : boolean;
    PosX,PosY: integer;
    PosF,PosE  : integer;
    KeyFunc    : boolean;

procedure GetAngl(Angl : integer);
begin
   Angl := Angl - 18; {90}
```

```
   if Angl < 0 then Angl := 72 - abs(Angl);
   if not MarkCir then
   begin StAngl := (Angl * 5);
         EnAngl := StAngl;
   end
   else  EnAngl := (Angl * 5);
end;

procedure CircleCommand;
var X,Y,R    : Longint;
begin
  GetLayerColor(LayerColor);
  SetColor(LayerColor);
  StAngl := 0;
  EnAngl := 0;
  DataCenX := CIRCenX;
  DataCenY := CIRCenY;
  DataRadius := CIRRadius;
  if PosF = PosE then
  begin
    CalAngular(PosE);
    X := GraphScreenX0 + (((CIRCenX-PanOffsX0)*GridSizeX) div StepLength);
    Y := GraphScreenY0 + (((CIRCenY-PanOffsY0)*GridSizeY) div StepLength);
    R := (CIRRadius*GridSizeX) div StepLength;
    CirCle(X,Y,R);
    StoreCircCommand; {Save Circle Command}
  end
  else
  begin
   EndAnglOff(PosF,PosE);
   X := GraphScreenX0 + (((CIRCenX-PanOffsX0)*GridSizeX) div StepLength);
   Y := GraphScreenY0 + (((CIRCenY-PanOffsY0)*GridSizeY) div StepLength);
   R := (CIRRadius*GridSizeX) div StepLength;
   StAngl := (PosF * 5) - 90;
   EnAngl := (PosE * 5) - 90;
   Arc(X,Y,StAngl,EnAngl,R);
   StoreArcCommand; {Save Arc Command}
  end;
end;

begin  {StartAnglCursor}
  StartCir := false;
  MarkCir  := false;
  PosX := 18; {Set angl = 0}
  PosE := 18;
  PosF := 18;
  DrawLineCir(CIRCenX,CIRCenY,CIRRadius);
  CalAngular(PosX);
  GetAngl(PosF);
  DispAng;
  DispGraph;
  repeat
    ExtCommand:= False;
    if Keypressed then
```

```
    begin
      KeyCommand := ReadKey;
      if KeyCommand = #0 then
      begin
        KeyCommand := ReadKey;
        if KeyCommand in [Left,Right] then
        begin
          if MarkCir then
            EndAnglOff(PosF,PosE)
          else
            CalAngular(PosE);
          KeyControlCur(PosX,KeyCommand);
          GetAngl(PosX);
          DispAng;
          DispGraph;
          if MarkCir then
            EndAnglOn(PosF,PosX)
          else
            CalAngular(PosX);
          PosE := PosX;
        end;
      end
      else
      begin
        if not MarkCir then
        begin
          if KeyCommand = #13 then
          begin
            CircleText(5);
            CircleText(4);
            CalAngular(PosE); {Clear Angl Line}
            DrawCircle;   {Clear Circle}
            MarkCir := true;
            PosF := PosX;
            PosE := PosF;
            EndAnglOn(PosF,PosX);
            GetAngl(PosE);
            DispAng;
            DispGraph;
          end
          else Write(^G);
        end
        else
        if KeyCommand = #13 then
        begin
           CircleCommand;
           ExtCommand := true;
        end;
      end;
    end;
  Until ExtCommand;
end;

procedure ArcLine;
```

```
begin
   CircleText(3);
   StartAnglCurSor(KeyCommand);
   CircleText(5);
end;

Procedure MoveCIRCursor(var KeyCommand : char);
var LineCIRStat : boolean;
    CIRCenter   : boolean;
    CenStatOn   : boolean;
    CIRStart    : boolean;
    ok          : boolean;

procedure CircleCursorOff;
begin
  if CIRCenter then
  begin
    if CenStatOn then
      DrawScaleCIROff
  end
  else
  if LineCIRStat then
  begin
    CursorOff;
    DrawCIR(CIRCenX,CIRCenY,OLDCIRRadius);
    LineCIRStat := false;
  end;
end;
procedure CircleCursorOn;
begin
  if CIRCenter then
  begin
    CalRadius;
    DispRad;
    DispGraph; {Display Circle Radius}
    DrawScaleCIROn;
  end
  else
  begin
    CIRCenX := PositionCurX;
    CIRCenY := PositionCurY;
    DrawCIR(CIRCenX,CIRCenY,CIRRadius);
    CursorOn;
    DispORG;    {Display Cursor Position}
    DispGraph;
    LineCIRStat := true;
  end;
  CenStatOn := On;
end;
procedure OffCIRCursor;
begin
  DrawCIR(CIRCenX,CIRCenY,OldCIRRadius);
  CursorOff;
end;
```

```
procedure OnCIRCenter;
var ch : char;
begin
  if CIRCenter then
  begin
    ArcLine;
    OffCIRCursor;
    DispORG;    {Display Cursor Position}
    DispGraph;
    ExtCommand := true;
  end
  else
  begin
    CIRCenter := true;
    CircleText(2);
    DispRad;
    DispGraph; {Display Circle Radius}
  end;
end;
procedure FirstOnCur;
begin
  if not ok then
  begin
    CursorOff;
    DrawCIR(CIRCenX,CIRCenY,CIRRadius);
    ok := true;
  end;
end;
procedure ActiveCur;
begin
  if not ok then
  begin
    CursorOff;
    DrawCIR(CIRCenX,CIRCenY,CIRRadius);
    ok := true;
    DrawCIR(CIRCenX,CIRCenY,CIRRadius);
    CursorOn;
    LineCIRStat := true;
    CenStatOn := On;
  end;
end;

begin
  SetColor(LightGray);
  ExtCommand  := false;
  LineCIRStat := false;
  CIRCenter   := false;
  CenStatOn   := false;
  CIRStart    := false;
  CIRCenX     := PositionCurX;
  CIRCenY     := PositionCurY;
  OldCIRRadius := CIRRadius;
  ok := false;
  DrawCIR(CIRCenX,CIRCenY,CIRRadius);
```

```
  CircleText(1);
  repeat
   ExtCommand:= false;
   if Keypressed then
   begin
     KeyCommand := ReadKey;
     if KeyCommand = #0 then
     begin
       KeyCommand := ReadKey;
       FirstOnCur;
       CircleCursorOff;
       Cursor(PositionCurX,PositionCurY,KeyCommand);
       CircleCursorOn;
     end { Function Key }
     else { Normal Key }
     begin
       if KeyCommand = #13 then
       begin
         ActiveCur;
         OnCIRCenter;
       end;
       if keyCommand in ['Q','q'] then
       begin
         OffCIRCursor;
         DispORG;    {Display Cursor Position}
         DispGraph;
         ExtCommand := true;
       end;
     end;
   end;
  Until ExtCommand;
  PositionCurX := CIRCenX;
  PositionCurY := CIRCenY;
  CursorOn;
end;

procedure InitPosValue;
begin
  PntX  := GraphCornerX0; PntY := GraphCornerY0;
  PntY1 := PntY - HalfCurSize; PntX1 := PntX - HalfCurSize;
  PntY2 := PntY + HalfCurSize; PntX2 := PntX + HalfCurSize;
  LattPntY := GraphScreenY1-10;
  TempX := PntX1; TempY := PntY1;
  LattTempY := LattPntY;
  LattTempX := LattPntX;
  ExtCommand := False;
  ExtLattCommand := False;
  LineCurStartOK := False;
end;

procedure InitCursor;
var i:byte;
begin
  Size  := ImageSize(PntX1,PntY1,PntX2,PntY2);
```

```
  GetMem (CurBuff,Size);
  GetMem (CurPattern,Size);
end;

procedure WireLine(StepX0,StepY0,StepX1,StepY1:Longint);
var X0,Y0,X1,Y1,Mul:integer;
begin
  X0  := (((StepX0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
  Y0  := (((StepY0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
  DataX0 := StepX0;
  DataY0 := StepY0;
  case RubberLineStyle of
  Angl : begin
           X1  := (((StepX1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
           Y1  := (((StepY1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
           DataX1 := StepX1;
           DataY1 := StepY1;
         end;
  Orth : begin
           PosStepX := StepX1-StepX0;
           PosStepY := StepY1-StepY0;
           if abs(PosStepX) >= abs(PosStepY) then
             PosStepY := 0
           else
             PosStepX := 0;
           StepX1 := StepX0 + PosStepX;
           StepY1 := StepY0 + PosStepY;
           X1  := (((StepX1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
           Y1  := (((StepY1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
           DataX1 := StepX1;
           DataY1 := StepY1;
         end;
  Dg45 : begin
           PosStepX := StepX1-StepX0;
           PosStepY := StepY1-StepY0;
           if abs(PosStepX) >= abs(PosStepY) then
           begin
             if abs(PosStepX) <> abs(PosStepY) then
             begin
               if PosStepX < 0 then Mul := -1 else Mul := 1;
               PosStepX := (abs(PosStepX) - abs(PosStepY)) * Mul;
               PosStepY := 0;
             end;
           end
           else
           begin
             if PosStepY < 0 then Mul := -1 else Mul := 1;
             PosStepY := (abs(PosStepY) - abs(PosStepX)) * Mul;
             PosStepX := 0;
           end;
           StepX1 := StepX0 + PosStepX;
           StepY1 := StepY0 + PosStepY;
           X1  := (((StepX1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
           Y1  := (((StepY1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
```

```
            DataX1 := StepX1;
            DataY1 := StepY1;
          end;
    end;
    GetLayerColor(LayerColor);
    SetColor(LayerColor);
    DrawPCBWire(X0,Y0,X1,Y1,Trace);
    StoreLineCommand;
    StartLineX := StepX1;
    StartLineY := StepY1;
end;

{-------------------------}

procedure Redraw;
var X0,Y0,X1,Y1,R : Longint;
    i,St,En : integer;
    SoldColor : integer;
    CompColor : integer;
    SilkColor : integer;
    TempLayer : integer;

procedure SoldRedraw;
var i,Tr : integer;
begin
  SetLayerColor(Sold);
  for i := 1 to SoldPointer do
  begin
    case SoldVar^[i].CdeData of
    LineCode : begin
                 X0 := ((SoldVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((SoldVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 X1 := ((SoldVar^[i].PosData[3]-PanOffsX0) * GridSizeX) div StepLength
                 Y1 := ((SoldVar^[i].PosData[4]-PanOffsY0) * GridSizeY) div StepLength
                 Tr := SoldVar^[i].AssData[1];
                 DrawPCBWire(X0,Y0,X1,Y1,Tr);
               end;
    CircCode : begin
                 X0 := ((SoldVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((SoldVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 R  := (SoldVar^[i].PosData[3] * GridSizeX) div StepLength;
                 Circle(X0,Y0,R);
               end;
    ArcCode  : begin
                 X0 := ((SoldVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((SoldVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 R  := (SoldVar^[i].PosData[3] * GridSizeX) div StepLength;
                 St :=  SoldVar^[i].PosData[4];
                 En :=  SoldVar^[i].AssData[1];
                 Arc(X0,Y0,St,En,R);
               end;
    end;{case}
  end;{for Loop}
end;
```

```
procedure CompRedraw;
var i,Tr : integer;
begin
  SetLayerColor(Comp);
  for i := 1 to CompPointer do
  begin
    case CompVar^[i].CdeData of
    LineCode : begin
                 X0 := ((CompVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((CompVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 X1 := ((CompVar^[i].PosData[3]-PanOffsX0) * GridSizeX) div StepLength
                 Y1 := ((CompVar^[i].PosData[4]-PanOffsY0) * GridSizeY) div StepLength
                 Tr := CompVar^[i].AssData[1];
                 DrawPCBWire(X0,Y0,X1,Y1,Tr);
               end;
    CircCode : begin
                 X0 := ((CompVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((CompVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 R  := (CompVar^[i].PosData[3] * GridSizeX) div StepLength;
                 Circle(X0,Y0,R);
               end;
    ArcCode  : begin
                 X0 := ((CompVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((CompVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 R  :=  (CompVar^[i].PosData[3] * GridSizeX) div StepLength;
                 St :=   CompVar^[i].PosData[4];
                 En :=   CompVar^[i].AssData[1];
                 Arc(X0,Y0,St,En,R);
               end;
    end;{case}
  end;{for Loop}
end;
procedure SilkRedraw;
var i,Tr : integer;
    LayerI : byte;
begin
  SetLayerColor(Silk);
  for i := 1 to SilkPointer do
  begin
    case SilkVar^[i].CdeData of
    LineCode : begin
                 X0 := ((SilkVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((SilkVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 X1 := ((SilkVar^[i].PosData[3]-PanOffsX0) * GridSizeX) div StepLength
                 Y1 := ((SilkVar^[i].PosData[4]-PanOffsY0) * GridSizeY) div StepLength
                 Tr := SilkVar^[i].AssData[1];
                 DrawPCBWire(X0,Y0,X1,Y1,Tr);
               end;
    CircCode : begin
                 X0 := ((SilkVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((SilkVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 R  := (SilkVar^[i].PosData[3] * GridSizeX) div StepLength;
                 Circle(X0,Y0,R);
               end;
```

```
    ArcCode  : begin
                 X0 := ((SilkVar^[i].PosData[1]-PanOffsX0) * GridSizeX) div StepLength
                 Y0 := ((SilkVar^[i].PosData[2]-PanOffsY0) * GridSizeY) div StepLength
                 R  := (SilkVar^[i].PosData[3] * GridSizeX) div StepLength;
                 St :=  SilkVar^[i].PosData[4];
                 En :=  SilkVar^[i].AssData[1];
                 Arc(X0,Y0,St,En,R);
               end;
    end;{case}
  end;{for Loop}
end;

procedure TextRedraw;
var i   : integer;
    X,Y : longint;
    TDir,Lyr : byte;
    Str : String[20];
    TempLayer : byte;
begin
  for i := 1 to TextPointer do
  begin
    TDir := TextVar^[i].TxtDir;
    Lyr  := TextVar^[i].TxtLyr;
    X    := TextVar^[i].PosData[1];
    Y    := TextVar^[i].PosData[2];
    Str  := TextVar^[i].Message;
    WriteGraphText(TDir,Lyr,X,Y,Str);
  end;
end;

procedure PadsRedraw;
var i,X0,Y0  : integer;
    PS,PD,RdLyr : byte;
begin
  for i := 1 to PadsPointer do
  begin
    X0 := PadsVar^[i].PosData[1];
    Y0 := PadsVar^[i].PosData[2];
    PS := PadsVar^[i].AssData[1];
    PD := PadsVar^[i].AssData[2];
    if CheckPadLyrRedraw(i,RdLyr) then
        FillPadStyle(X0,Y0,PS,PD)
    else if RdLyr <> Layer then
          FillPadStyle(X0,Y0,PS,PD);
  end;{for Loop}
end;

procedure RedrawPadLayer;
var PS,PD,RdLyr : byte;
    i,X0,Y0 : integer;
begin
  for i := 1 to PadsPointer do
  begin
    X0 := PadsVar^[i].PosData[1];
```

```
    YO := PadsVar^[i].PosData[2];
    PS := PadsVar^[i].AssData[1];
    PD := PadsVar^[i].AssData[2];
    if not CheckPadLyrRedraw(i,RdLyr) then
    begin
      if RdLyr = Layer then
        FillPadStyle(XO,YO,PS,PD);
    end;
  end;{for Loop}
end;

procedure SubRedraw (Var NextR : byte);
begin
  case NextR of
  Sold : begin
           if SoldLayerEn then SoldRedraw;
           NextR := Comp;
         end;
  Comp : begin
           if CompLayerEn then CompRedraw;
           NextR := Silk;
         end;
  Silk : begin
           if SilkLayerEn then SilkRedraw;
           NextR := Sold;
         end;
  end;
end;

begin
  DispGraph;
  LayerTemp  := Layer;
  LayerTemp := LayerTemp + 1;
  if LayerTemp > Silk then
    LayerTemp := Sold;
  DispGraph;
  SubRedraw(LayerTemp);
  DispGraph;
  SubRedraw(LayerTemp);
  DispGraph;
  SubRedraw(LayerTemp);
  if PadsLayerEn then
  begin
    PadsRedraw;
    RedrawPadLayer;
  end;
  if TextLayerEn then
    TextRedraw;
  if OrgLine then  {if LineCursor Occure then Draw}
    DrawStepLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
end;

{-------------------------------------------------------------------}
```

```
procedure DispLibDevice(DrawMode : boolean);
var X0,Y0,X1,Y1,R : Longint;
    i,St,En    : integer;
    TempLayer  : integer;
    LayerI,Tr  : integer;
    PadColTemp : byte;
    DrlColTemp : byte;
    PS,PD,Pc,Pl: byte;
begin
  CursorOff;
  for i := 1 to DevPointer do
  begin
    case DevVar[i].CdeData of
    LineCode : begin
                 X0 := ((PositionCurX+RefX+DevVar[i].PosData[1]-PanOffsX0) * GridSizeX)
                 Y0 := ((PositionCurY+RefY+DevVar[i].PosData[2]-PanOffsY0) * GridSizeY)
                 X1 := ((PositionCurX+RefX+DevVar[i].PosData[3]-PanOffsX0) * GridSizeX)
                 Y1 := ((PositionCurY+RefY+DevVar[i].PosData[4]-PanOffsY0) * GridSizeY)
                 LayerI := DevVar[i].AssData[2];
                 if DrawMode then
                   SetLayerColor(LayerI)
                 else
                   SetColor(Black);
                 Tr := DevVar[i].AssData[1];
                 DrawPCBWire(X0,Y0,X1,Y1,Tr);
               end;
    CircCode : begin
                 X0 := ((PositionCurX+RefX+DevVar[i].PosData[1]-PanOffsX0) * GridSizeX)
                 Y0 := ((PositionCurY+RefY+DevVar[i].PosData[2]-PanOffsY0) * GridSizeY)
                 R  := (DevVar[i].PosData[3] * GridSizeX) div StepLength;
                 LayerI := DevVar[i].AssData[2];
                 if DrawMode then
                   SetLayerColor(LayerI)
                 else
                   SetColor(Black);
                 Circle(X0,Y0,R);
               end;
    ArcCode  : begin
                 X0 := ((PositionCurX+RefX+DevVar[i].PosData[1]-PanOffsX0) * GridSizeX)
                 Y0 := ((PositionCurY+RefY+DevVar[i].PosData[2]-PanOffsY0) * GridSizeY)
                 R  := (DevVar[i].PosData[3] * GridSizeX) div StepLength;
                 St :=  DevVar[i].PosData[4];
                 En :=  DevVar[i].AssData[1];
                 LayerI := DevVar[i].AssData[2];
                 if DrawMode then
                   SetLayerColor(LayerI)
                 else
                   SetColor(Black);
                 Arc(X0,Y0,St,En,R);
               end;
    PadCode  : begin
                 X0 := PositionCurX+RefX+DevVar[i].PosData[1];
                 Y0 := PositionCurY+RefY+DevVar[i].PosData[2];
```

```
                PS := DevVar[i].AssData[1];
                PD := DevVar[i].AssData[2];
                if DrawMode then
                begin
                  FillPadStyle(X0,Y0,PS,PD);
                end
                else
                begin
                  PadColTemp := PadColor;
                  DrlColTemp := DrllColor;
                  PadColor   := Black;
                  DrllColor  := Black;
                  DeCodePadLyr(PD,Pc,Pl);
                  Pl := Black;
                  EnCodePadLyr(PD,Pc,Pl);
                  FillPadStyle (X0,Y0,PS,PD);
                  PadColor  := PadColTemp;
                  DrllColor := DrlColTemp;
                end;
              end;
    end;{case}
  end;{for Loop}
  if DrawMode then
    MoveLibsDataToScrData; {LibData -> Sold,Comp,Silk,Pad}
  GetLayerColor(TempLayer);
  SetColor(TempLayer);
  CursorOn;
end;

procedure DispCurDevice;
var X0,Y0,X1,Y1,R : Longint;
    i,St,En   : integer;
    PS,PD     : byte;
    TempLayer : integer;
    LayerI    : integer;
    ch : char;
begin
  CalPadCurLine(40);
  for i := 1 to DevPointer do
  begin
    case DevVar[i].CdeData of
    LineCode : begin
                 X0 := ((PositionCurX+RefX+DevVar[i].PosData[1]-PanOffsX0) * GridSizeX)
                 Y0 := ((PositionCurY+RefY+DevVar[i].PosData[2]-PanOffsY0) * GridSizeY)
                 X1 := ((PositionCurX+RefX+DevVar[i].PosData[3]-PanOffsX0) * GridSizeX)
                 Y1 := ((PositionCurY+RefY+DevVar[i].PosData[4]-PanOffsY0) * GridSizeY)
                 DrawAngLine(X0,Y0,X1,Y1);
               end;
    CircCode : begin
                 X0 := PositionCurX + RefX + DevVar[i].PosData[1];
                 Y0 := PositionCurY + RefY + DevVar[i].PosData[2];
                 R  := DevVar[i].PosData[3];
                 CalPadCurLine(R);
                 PadDrawCir(X0,Y0,R); {Same as Circle Command}
```

```
                end;
    ArcCode  : begin
                 XO := PositionCurX + RefX + DevVar[i].PosData[1];
                 YO := PositionCurY + RefY + DevVar[i].PosData[2];
                 R  := DevVar[i].PosData[3];
                 CalCIRArc(XO,YO,R);
                 St :=  DevVar[i].PosData[4];
                 En :=  DevVar[i].AssData[1];
                 DrawArcCur(St,En);
               end;
    PadCode  : begin
                 XO := PositionCurX + RefX + DevVar[i].PosData[1];
                 YO := PositionCurY + RefY + DevVar[i].PosData[2];
                 PS := DevVar[i].AssData[1];
                 PD := DevVar[i].AssData[2];
                 DrawPadCur(XO,YO,PS,PD);
               end;
   end;{case}
  end;{for Loop}
  GetLayerColor(TempLayer);
  SetColor(TempLayer);
end;

procedure IncLayer;
begin
  Layer := Layer + 1;
  if Layer > Silk then
    Layer := Sold;
  WriteStatusData(3);
  ActiveTextDirLayer;
  WriteStatusData(11);
  DispGraph;
end;

procedure DecLayer;
begin
  Layer := Layer - 1;
  if Layer < Sold then
    Layer := Silk;
  WriteStatusData(3);
  ActiveTextDirLayer;
  WriteStatusData(11);
  DispGraph;
end;

Procedure MoveCursor(var KeyCommand : char);
var KeyCount : integer;
    oldL     : byte;

procedure RunLineCommand;
begin
  CursorOff;
  if OrgLine and KeyPass then
    DrawStepLine(StartLineX,StartLineY,TpX,TpY);
```

```
  Cursor(PositionCurX,PositionCurY,KeyCommand);
  TpX := PositionCurX; TpY := PositionCurY;
  case KeyCommand of
  #59 : begin
          if StartOrg then
          begin
            StartLineX := PositionCurX;
            StartLineY := PositionCurY;
            StartOrg   := false;
          end;
          OrgLine  := true;
          if not((StartLineX=PositionCurX) and (StartLineY=PositionCurY)) then
          if not StartOrg then
            begin
              WireLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
            end;
        end;
  #60 : begin
          OrgLine := false;
          StartOrg:= true;
        end;
  end;{Case}
  if OrgLine then
  begin
    KeyCount := 0;
    KeyPass := false;
    repeat
      if not KeyPass then
      begin
        if KeyCount = 500 then
        begin
          DrawStepLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
          KeyPass := true;
          CursorON;   {Cursor On For RubberLine Display}
          DispORG;    {Display Cursor Position}
          DispGraph;
        end;
        KeyCount := KeyCount + 1;
      end;
    until KeyPressed;
    if KeyPass then CursorON {Cursor off} ;
  end;
  CursorON;
  DispORG;
  DispGraph;
end;

procedure FillWirePad;
begin
  if OrgLine then
  begin
    CursorOff;
    DrawStepLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
    case RubberLineStyle of
```

```
     Angl : WireLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
    else begin
           WireLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
           WireLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
         end;
    end;
    CursorOn;
  end; {OrgLine}
  StartLineX := PositionCurX;
  StartLineY := PositionCurY;
end;

begin {MoveCursor}
  repeat
    ExtCommand := false;
    FuncKeysOn := false;
    if Keypressed then
    begin
      KeyCommand := ReadKey;
      if KeyCommand = #0 then
      begin
        KeyCommand := ReadKey;
        case KeyCommand of
        #61 : begin  {Function Key F3}
                FillWirePad;
                KeyCommand := 'F'; {Pad Command}
                ExtCommand := true;
                FuncKeysOn := true;
              end;
        #62 : begin  {Function Key F4}
                FillWirePad;
                KeyCommand := 'G'; {Pad Command}
                ExtCommand := true;
                FuncKeysOn := true;
                if Layer = Sold then
                  Layer := Comp
                else
                  if Layer = Comp then
                    Layer := Sold;
                WriteStatusData(3);
                DispGraph;
              end;
        #63 : begin  {Function Key F5 [Circle]}
                KeyCommand := 'H';
                ExtCommand := true;
                FuncKeysOn := true;
              end;
        #64 : begin  {Function Key F6 [Delete Pad]}
                KeyCommand := 'I';
                ExtCommand := true;
                FuncKeysOn := true;
              end;
        #65 : begin  {Function Key F7 [Delete Line]}
                if OrgLine then
```

```
                DrawStepLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
              OrgLine    := false;
              StartOrg   := true;
              KeyCommand := 'J';
              ExtCommand := true;
              FuncKeysOn := true;
            end;
        #66 : begin  {Function Key F8 [Delete Circle]}
              KeyCommand := 'K';
              ExtCommand := true;
              FuncKeysOn := true;
            end;
        #67 : begin  {Function Key F9  [Redraw Screen]}
              KeyCommand := 'B';
              ExtCommand := true;
              FuncKeysOn := true;
            end;
        #68 : begin  {Function Key F10 [Screen PAN]}
              KeyCommand := 'C';
              ExtCommand := true;
              FuncKeysOn := true;
            end;
        #82 : begin  {Function Key Ins [GoStep * 10]}
              if not GoStepStat then
                GoStep := GoStep * 10;
              GoStepStat := On;
            end;
        #83 : begin  {Function Key Del [GoStep  * 1]}
              if GoStepStat then
                GoStep := GoStep Div 10;
              GoStepStat := Off;
            end;
        else RunLineCommand;
        end;
      end
      else
      begin
         case KeyCommand of
         '+'     : IncLayer;
         '-'     : DecLayer;
         else
           begin
             ExtCommand := true;
             if OrgLine then
               DrawStepLine(StartLineX,StartLineY,PositionCurX,PositionCurY);
             OrgLine := false;
             StartOrg := true;
           end;
         end; {case}
      end;
    end;
  Until ExtCommand;
end;
```

```
begin { Main }

end.
```

```
Program : PCB Part III
Programmer : Suttinun Poramatikul

{
              +-----------------------------------+
              |                                   |
              | Unit  : PCB-CAD Unit III          |
              | Programmer : Suttinun Poramatikul |
              | Version    : 1.00A                |
              | Last Updated : 20 Jun 1988        |
              |                                   |
              +-----------------------------------+
}

unit PCBUnit3;

interface

uses Crt,Graph,Declare,PCBUnit1,PCBUnit2,PCBUnit4,PCBUnit5,PCBUnit6;

procedure MainMenu;

procedure PCBSetUp;

procedure PCBMain;

implementation

var
  S         : string;
  MODE      : String;
  Name      : String;
  Ok        : boolean;
  LibLoadOk : boolean;
  LibNameOk : boolean;
  MoveLibOk : boolean;
  ZoomKey   : char;
  DirKey,ch : char;
  Chkey     : char;
  CKey      : char;
  PadTemp   : byte;
  TempLayer : byte;
  PadErr,i  : integer;
  PadHi     : integer;
  PadLo     : integer;
  PadDelPnt : integer;
  LneDelPnt : integer;
  PpntNum   : integer;
```

```
procedure MovingStepAdj(MStep : integer);
var PstX,PstY : real;
begin
  PstX      := PntStepX / MovingStep;
  PstY      := PntStepY / MovingStep;
  PntStepX := trunc(PstX * MStep);
  PntStepY := trunc(PstY * MStep);
  PstX      := TpX / MovingStep;
  PstY      := TpY / MovingStep;
  TpX       := trunc(PstX * MStep);
  TpY       := trunc(PstY * MStep);
  PstX      := StartLineX / MovingStep;
  PstY      := StartLineY / MovingStep;
  StartLineX := trunc(PstX * MStep);
  StartLineY := trunc(PstY * MStep);
  MovingStep := MStep;
end;

procedure CheckZoomFactor(ZoomKeyCom : char);
begin
  Case ZoomKeyCom of
  Left  : begin
            ZoomFactor := ZoomFactor - 1;
            if ZoomFactor < 1 then ZoomFactor := 6;
          end;{ ZoomFactor - 1 }
  Right : begin
            ZoomFactor := ZoomFactor + 1;
            if ZoomFactor > 6 then ZoomFactor := 1;
          end;{ ZoomFactor + 1 }
  end;{case}
end;

procedure CheckDevDir(DirKeyC : char);
begin
  Case DirKeyC of
  Left  : begin
            DeviceDir := DeviceDir - 1;
            if DeviceDir < 1 then DeviceDir := 4;
          end;{ DeviceDir  - 1 }
  Right : begin
            DeviceDir := DeviceDir + 1;
            if DeviceDir > 4 then DeviceDir := 1;
          end;{ DeviceDir + 1 }
  end;{case}
end;

procedure CheckTextDir(DirKeyC : char);
begin
  if Layer = Sold then
  begin
    Case DirKeyC of
    Left  : begin
              TextDir := TextDir - 1;
```

```
            if TextDir < 5 then TextDir := 8;
          end;{ TextDir  - 1 }
    Right : begin
            TextDir := TextDir + 1;
            if TextDir > 8 then TextDir := 5;
          end;{ TextDir + 1 }
    end;{case}
  end
  else
  begin
    Case DirKeyC of
    Left  : begin
            TextDir := TextDir - 1;
            if TextDir < 1 then TextDir := 4;
          end;{ TextDir  - 1 }
    Right : begin
            TextDir := TextDir + 1;
            if TextDir > 4 then TextDir := 1;
          end;{ TextDir + 1 }
    end;{case}
  end;
end;

procedure KeyPadDir(DirKeyC : char);
var LTem,DTem : byte;
begin
    Case DirKeyC of
    Left  : begin
              DeCodePadLyr(PadDir,DTem,LTem);
              if DTem = THor then
              begin
                 DTem := TVer;
                 EnCodePadLyr(PadDir,DTem,LTem);
              end;
              EndBeep;
            end;
    Right : begin
              DeCodePadLyr(PadDir,DTem,LTem);
              if DTem = TVer then
              begin
                DTem := THor;
                EnCodePadLyr(PadDir,DTem,LTem);
              end;
              EndBeep;
            end;
    end;{case}
end;

procedure ReadLibsActDir;
begin
  DispText;
  DeleteLine(2);
  SetColor(TSCColor);
  WriteTextXY(2,2,'LIB Dir : ');
```

```
  ReadTextXY(13,2,Name);
  LibsActDir := Name;
  DeleteLine(2);
  WriteStatusData(10);
  DispText;
end;

procedure ReadFileActDir;
begin
  DispText;
  DeleteLine(2);
  SetColor(TSCColor);
  WriteTextXY(2,2,'FLE Dir : ');
  ReadTextXY(13,2,Name);
  FileActDir := Name;
  DeleteLine(2);
  WriteStatusData(15);
  DispText;
end;

procedure ClearPointer;
begin
  SoldPointer := 0;
  CompPointer := 0;
  SilkPointer := 0;
  PadsPointer := 0;
  TextPointer := 0;
end;

procedure ClearData;
begin
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  SetColor(TSCColor);
  WriteTextXY(2,2,'CLEAR [Y/N]');
  EnterPassKey(Chkey);
  if Chkey in ['Y','y',#13] then
  begin
    ClearPointer;
    SetGraphMode(GraphMode);
    if ORGStat then
    begin
      OpenWindowORG(81,25,89,26);
      WriteORG(2,1);
    end;
    OpenWindowStat(38,25,79,26);
    DispGraph;
    if GridOn then Grid(On);
    EndBeep;
  end
  else DeleteLine(2);
  DispText;
end;
```

```
procedure PCBSetUp;
var
  SYMB_Command : string;
  DEVC_Command : string;
  DELT_Command : string;
  GRID_Command : string;
  STAT_Command : string;
  Ch : char;
  BufStr : string;
  Code : integer;

procedure FuncSetCursor;
begin
  SYMB_Command := 'CURW';
  SetColor(TSCColor);
  DispText;
  Deleteline(2);
  writeTextXY(2,2,'CURW');
  DeleteLine(1);
  MenuSelect := 1;
  MenuBlock(On);
  SetColor(TMCColor);
  WriteTextXY(2,1,'Angl Orth 45dg                ');
  repeat
    SelectMenu;
    case MenuSelect of
    1 : RubberLineStyle := Angl;
    2 : RubberLineStyle := Orth;
    3 : RubberLineStyle := dg45;
    end;
  until MenuSelect in [1..3];
  DeleteLine(1);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DelText(2,2,'CURWp');
  WriteTextXY(2,2,MODE);
  WriteStatusData(1);
  DispGraph;
end;

procedure FuncSetThickness;
begin
  SYMB_Command := 'WThk';
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  writeTextXY(2,2,'WThk');
  DeleteLine(1);
  MenuSelect := 1;
  MenuBlock(On);
  SetColor(TMCColor);
  WriteTextXY(2,1,'T12. T16. T20. T50.           ');
  repeat
```

```
    SelectMenu;
    case MenuSelect of
    1 : Trace := Thck1;
    2 : Trace := Thck2;
    3 : Trace := Thck3;
    4 : Trace := Thck4;
    end;
  until MenuSelect in [1..4];
  DeleteLine(1);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DelText(2,2,'WThkp');
  WriteTextXY(2,2,MODE);
  WriteStatusData(5);
  DispGraph;
end;

procedure CheckPadDir(PadKeyCom : char);
begin
  Case PadKeyCom of
  Left  : begin
            PadHi := PadHi - 1;
            if PadHi < 0 then PadHi := 5;
          end;{ PadHi - 1 }
  Right : begin
            PadHi := PadHi + 1;
            if PadHi > 5 then PadHi := 0;
          end;{ PadFactor + 1 }
  UpK   : begin
            PadLo := PadLo - 1;
            if PadLo < 0 then PadLo := 1;
          end;{ PadLo - 1 }
  Down  : begin
            PadLo := PadLo + 1;
            if PadLo > 1 then PadLo := 0;
          end;{ PadLo + 1 }
  end;{case}
end;

procedure PadSelect(Var PadSel : integer);
var DirKey : char;
begin
  PadHi := 0;
  PadLo := 0;
  PadMenuBlock (PadLo,PadHi,On);
  repeat
    DirKey := ' ';
    if Keypressed then
    begin
      DirKey := ReadKey;
      if DirKey = #0 then
      begin
        PadMenuBlock (PadLo,PadHi,Off);
        DirKey := ReadKey;
```

```
        CheckPadDir(DirKey);
        CursorBeep;
        PadMenuBlock (PadLo,PadHi,On);
      end;
    end;
  until DirKey = #13;
  PadSel := (PadLo * 6) + PadHi + 1;
end;

procedure FuncSetPadSize;
var PadSel : integer;
begin
  SYMB_Command := 'PADs';
  DispText;
  SetColor(TMCColor);
  DeleteLine(1);
  DeleteLine(2);
  if PadAssError then
  begin
    WriteTextXY(2,1,'P52r P52s P60r P60s P72r P72s   ');
    WriteTextXY(2,2,'100r 100s 200r 200s THor TVer   ');
  end
  else
  begin
    for i := 0 to 5 do
      WriteTextXY(2+6*i,1,PadAssData[i+1].PadNme);
    for i := 0 to 5 do
      WriteTextXY(2+6*i,2,PadAssData[i+7].PadNme);
  end;
  PadSelect(PadSel);
  Sound(2500);Delay(100);NoSound;
  PadSize := PadSel;
  DeleteLine(1);
  DeleteLine(2);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  WriteTextXY(2,2,MODE);
  WriteStatusData(2);
  DispGraph;
end;

procedure FuncSetLayer;
begin
  SYMB_Command := 'LAYR';
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  writeTextXY(2,2,'LAYR');
  DeleteLine(1);
  MenuSelect := 1;
  MenuBlock(On);
  SetColor(TMCColor);
  WriteTextXY(2,1,'SOLD COMP SILK                  ');
  repeat
```

```
    SelectMenu;
    case MenuSelect of
    1 : Layer := Sold;
    2 : Layer := Comp;
    3 : Layer := Silk;
    end;
  until MenuSelect in [1..3];
  ActiveTextDirLayer;
  WriteStatusData(11);
  DispText;
  DeleteLine(1);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DeleteLine(2);
  WriteTextXY(2,2,MODE);
  WriteStatusData(3);
  DispGraph;
end;

procedure DispActLayer;
begin
  if SoldLayerEn then SetColor(TMCColor)
  else SetColor(Red);
  WriteTextXY(2,1,'SOLD');
  if CompLayerEn then SetColor(TMCColor)
  else SetColor(Red);
  WriteTextXY(2,1,'     COMP');
  if SilkLayerEn then SetColor(TMCColor)
  else SetColor(Red);
  WriteTextXY(2,1,'          SILK');
  if PadsLayerEn then SetColor(TMCColor)
  else SetColor(Red);
  WriteTextXY(2,1,'               PADS');
  if TextLayerEn then SetColor(TMCColor)
  else SetColor(Red);
  WriteTextXY(2,1,'                    TEXT');
end;

procedure FuncActiveLayer;
begin
  SYMB_Command := 'LAYR';
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  writeTextXY(2,2,'LAYR');
  DeleteLine(1);
  MenuSelect := 1;
  MenuBlock(On);
  SetColor(TMCColor);
  WriteTextXY(2,1,'SOLD COMP SILK PADS TEXT Exit ');
  DispActLayer;
  repeat
    SelectMenu;
    case MenuSelect of
```

```
    1 : SoldLayerEn := not SoldLayerEn;
    2 : CompLayerEn := not CompLayerEn;
    3 : SilkLayerEn := not SilkLayerEn;
    4 : PadsLayerEn := not PadsLayerEn;
    5 : TextLayerEn := not TextLayerEn;
    end;
    DispActLayer;
  until MenuSelect = 6;
  DispText;
  DeleteLine(1);
  DeleteLine(2);
end;

procedure FuncSetGridScreen;
begin
  MenuSelect := 1;
  repeat
    DispText;
    DeleteLine(1);
    MenuBlock(On);
    SetColor(TSCColor);
    writeTextXY(2,2,'SCRN');
    SetColor(TMCColor);
    WriteTextXY(2,1,'GRID ZOOM                EXIT');
    SelectMenu;
    DeleteLine(1);
    case MenuSelect of
    1 : begin
          GRID_Command := 'GRID';
          SetColor(TSCColor);
          DispText;
          DeleteLine(2);
          WriteTextXY(2,2,'GRID');
          DispGraph;
          CursorOff;
          GridStat := not GridStat;
          Grid(GridStat);
          CursorOn;
          DispText;
          DelText(2,2,'GRID');
        end;
     2 : begin
          GRID_Command := 'ZOOM';
          SetColor(TSCColor);
          DispText;
          DeleteLine(2);
          WriteTextXY(2,2,'ZOOM');
          DeleteLine(1);
          MenuSelect := 1;
          MenuBlock(On);
          SetColor(TMCColor);
          WriteTextXY(2,1,' 1    2    3    4    5    6     ');
          repeat
            SelectMenu;
```

```
            SetParaZoom(MenuSelect);
          until MenuSelect in [1..6];
          DeleteLine(1);
          DeleteLine(2);
          WriteStatusData(8);
          MenuOffsX0 := 5;
          MenuOffsY0 := 10;
          MenuOffsX1 := 10;
          MenuOffsY1 := 0;
          OnScreenDisplay;
          DispGraph;
          Redraw;
          CursorOn;
          DispText;
        end;
    6 : begin
          GRID_Command := 'EXIT';
        end;
    end { case };
    MenuSelect := 6;
    DeleteLine(2);
  until GRID_Command = 'EXIT';
  DispText;
  DeleteLine(1);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DeleteLine(2);
  WriteTextXY(2,2,MODE);
  MenuSelect := 3;
  KeyCommand := ' '; {Function Exit}
  DispGraph;
end;

function CheckUserPad(PNum : integer) : boolean;
var PadT,PD,RdLyr : byte;
begin
  CheckUserPad := true;
  PadT := PadsVar^[PNum].AssData[1];
  if PadAssError then
  begin
    if  PadT in [11,12] then
    begin
      DeCodePadLyr(PadsVar^[PNum].AssData[2],PD,RdLyr);
      if RdLyr <> Layer then CheckUserPad := false;
    end;
  end
  else
  begin
    if PadAssData[PadT].PadStl = 2 then
    begin
      DeCodePadLyr(PadsVar^[PNum].AssData[2],PD,RdLyr);
      if RdLyr <> Layer then CheckUserPad := false;
    end;
  end;
```

```
end;

procedure FuncFillPads;
var DTem,LTem : byte;
begin
  SetColor(PadColor);
  DeletePadsData(PositionCurX,PositionCurY,PadDelPnt,PadErr);
  if PadErr = 0 then
  begin
    if CheckUserPad(PadDelPnt) then
    begin
      UnDispPads(PadDelPnt);
      CompressPadsData(PadDelPnt);
    end;
  end;
  DataCenX := PositionCurX;
  DataCenY := PositionCurY;
  DeCodePadLyr(PadDir,DTem,LTem);
  EnCodePadLyr(PadDir,DTem,Layer);
  StorePadsCommand;
  GetLayerColor(SysColor);
  CursorOff;
  FillPadStyle(PositionCurX,PositionCurY,PadSize,PadDir);
  CursorOn;
  SetColor(SysColor);
  WriteStatusData(6);
  DispGraph;
end;

procedure FuncFillLinkPads;
var PadTemp,Tem : byte;
begin
  DeletePadsData(PositionCurX,PositionCurY,PadDelPnt,PadErr);
  if PadErr = 0 then
  begin
    if CheckUserPad(PadDelPnt) then
    begin
      UnDispPads(PadDelPnt);
      CompressPadsData(PadDelPnt);
    end;
  end;
  SetColor(Yellow);
  DataCenX := PositionCurX;
  DataCenY := PositionCurY;
  PadTemp := PadSize;
  PadSize := 1 {P52r};
  PadDir  := 0;
  Tem     := PadDir;
  EnCodePadLyr(PadDir,Tem,Layer);
  StorePadsCommand;
  GetLayerColor(SysColor);
  CursorOff;
  FillPadStyle(PositionCurX,PositionCurY,PadSize,PadDir);
  CursorOn;
```

```
  PadSize := PadTemp;
  SetColor(SysColor);
  WriteStatusData(6);
  DispGraph;
end;

procedure PnZmRdDisp;
begin
  DispText;
  DeleteLine(1);
  DeleteLine(2);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  WriteTextXY(2,2,MODE);
end;

procedure RedrawDataOnScreen;
begin
  DispGraph;
  Redraw;
  CursorOn;
end;

procedure CalCoordPanning;
begin
  {On X Axis}
  PanOffsX0 := PositionCurX-ScreenCenX;
  if PanOffsX0 < 0 then PanOffsX0 := 0;
  if PositionCurX > (MaxScreenX-ScreenLength) then
    PanOffsX0 :=  MaxScreenX-ScreenLength;
  {On Y Axis}
  PanOffsY0 := PositionCurY-ScreenCenY;
  if PanOffsY0 < 0 then PanOffsY0 := 0;
  if PositionCurY > (MaxScreenY-ScreenHeight) then
    PanOffsY0 :=  MaxScreenY-ScreenHeight;
end;

procedure FuncPanScreen;
begin
  CalCoordPanning;
  DispGraph;
  CursorOff;
  OnScreenDisplay;
  if LWStat then LibWindow;
end;

procedure FuncZoomScreen;
begin
  DispGraph;
  CursorOff;
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  WriteTextXY(2,2,'ZOOM');
```

```
  repeat
    ZoomKey := ' ';
    if Keypressed then
    begin
      ZoomKey := ReadKey;
      if ZoomKey = #0 then
      begin
        ZoomKey := ReadKey;
        CheckZoomFactor(ZoomKey);
        WriteStatusData(8);
      end;
    end;
  until ZoomKey = #13;
  SetParaZoom(ZoomFactor);
  CalCoordPanning;
  SetCoordinateData;
  FuncPanScreen;
end;

procedure FuncPadDir;
begin
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  WriteTextXY(2,2,'PAD  Hor<-DIR->Ver');
  DeleteLine(1);
  repeat
    DirKey := ' ';
    if Keypressed then
    begin
      DirKey := ReadKey;
      if DirKey = #0 then
      begin
        DirKey := ReadKey;
        KeyPadDir(DirKey);
        WriteStatusData(14);
      end;
    end;
  until DirKey = #13;
  DispText;
  DeleteLine(1);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DeleteLine(2);
  WriteTextXY(2,2,MODE);
  WriteStatusData(11);
  DispGraph;
end;

procedure FuncSetTextDir;
begin
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
```

```
  WriteTextXY(2,2,'TEXT L<-DIR->R');
  DeleteLine(1);
  repeat
    DirKey := ' ';
    if Keypressed then
    begin
      DirKey := ReadKey;
      if DirKey = #0 then
      begin
        DirKey := ReadKey;
        CheckTextDir(DirKey);
        WriteStatusData(11);
      end;
    end;
  until DirKey = #13;
  DispText;
  DeleteLine(1);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DeleteLine(2);
  WriteTextXY(2,2,MODE);
  DispGraph;
end;

procedure RedrawScreen(i : byte);
begin
  OnScreenDisplay;
  DispGraph;
  if i = 1 then PnZmRdDisp;
  Redraw;
  DispText;
  EndBeep;
end;

procedure SetCurMovementDisp;
begin
  DispText;
  DeleteLine(1);
  DeleteLine(2);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  WriteTextXY(2,2,MODE);
  DispGraph;
end;

procedure FuncSetCurMovement;
begin
  SYMB_Command := 'STEP';
  DispText;
  DeleteLine(1);
  DeleteLine(2);
  SetColor(TMCColor);
  WriteTextXY(2,1,' 100  50   25   10    5    1  ');
  SetColor(TSCColor);
```

```
  WriteTextXY(2,2,'STEP');
  MenuSelect := 1;
  MenuBlock(On);
  repeat
    SelectMenu;
    case MenuSelect of
    1 : begin GoStep := 100;  MovingStepAdj(10); end;
    2 : begin GoStep := 50;   MovingStepAdj(10); end;
    3 : begin GoStep := 25;   MovingStepAdj(10); end;
    4 : begin GoStep := 10;   MovingStepAdj(10); end;
    5 : begin GoStep := 5;    MovingStepAdj(10); end;
    6 : begin GoStep := 1;    MovingStepAdj(10); end;
    end;
  until MenuSelect in [1..6];
  GoStepStat := true;
  WriteStatusData(7);{3}
end;

procedure FuncDrawCircle;
begin
  SYMB_Command := 'CIRC';
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  WriteTextXY(2,2,'CIRC');
  DispGraph;
  Repeat
    MoveCIRCursor(KeyCommand);
  until KeyCommand in ['Q','q','E','e'];
  KeyCommand := ' '; {Function Exit}
  DispText;
  DeleteLine(1);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DeleteLine(2);
  WriteTextXY(2,2,MODE);
  DispGraph;
end;

procedure FuncMoveLibCur;
begin
  SYMB_Command := 'LIB';
  SetColor(TSCColor);
  DispText;
  DeleteLine(2);
  WriteTextXY(2,2,'LIB');
  DispGraph;
  repeat
    MoveLIBCursor(KeyCommand);
  until KeyCommand in ['Q','q','E','e'];
  KeyCommand := ' '; {Function Exit}
  DispText;
  DeleteLine(1);
  SetColor(TMCColor);
```

```
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  DeleteLine(2);
  WriteTextXY(2,2,MODE);
  DispGraph;
end;

procedure FuncWriteTextOnPCB;
begin
  DispText;
  DeleteLine(1);
  DeleteLine(2);
  WriteTextXY(2,2,'Text > ');
  ReadTextXY(9,2,Name);
  DeleteLine(2);
  SetColor(TMCColor);
  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
  WriteTextXY(2,2,MODE);
  DispGraph;
  CursorOff;
  GetLayerColor(SysColor);
  SetColor(SysColor);
  WriteGraphText(TextDir,Layer,PositionCurX,PositionCurY,Name);
  StoreTextCommand(TextDir,Layer,PositionCurX,PositionCurY,Name);
  CursorOn;
end;

procedure DispLibLabel(LPnt : integer);
begin
  DispText;
  DeleteLine(1);
  DeleteLine(1);
  DeleteLine(2);
  SetColor(TSCColor);
  WriteTextXY(2,1,'LABEL : '+ LibLName + '  ['+LibCName+']');
end;

procedure DispLabel(LPnt : integer);
begin
  DispText;
  DeleteLine(2);
  SetColor(TSCColor);
  WriteTextXY(2,2,'LABEL : '+ LibLName + '  ['+LibLName+']');
end;

procedure EnterLibData;
var LibPass : boolean;
    Name    : string;
begin
  DispText;
  repeat
    LibPass := true;
    DeleteLine(1);
    DeleteLine(2);
    SetColor(TSCColor);
```

```
    WriteTextXY(2,2,'LABEL : ');
    ReadTextXY(10,2,Name);
    Name := Copy(Name,1,12);
    if LibPointer > 0 then
    begin
      for i := 1 to LibPointer do
      begin
        if Name = LibVar^[i].LibLabel.Name then
        begin
          DeleteLine(1);
          WriteTextXY(2,1,'LABEL Exist');
          Delay(1500);
          EndBeep;
          LibPass := false;
        end;
      end;
    end;
  until LibPass;
  LibPointer := LibPointer + 1;
  LibActNum := LibPointer;
  LibLName := Name;
  DeleteLine(1);
  SetColor(TSCColor);
  WriteTextXY(2,1,'LABEL : '+Name);
  DeleteLine(2);
  SetColor(TMCColor);
  WriteTextXY(2,2,'COMMENTS : ');
  ReadTextXY(14,2,Name);
  Name := Copy(Name,1,12);
  LibCName := Name;
  StoreDataDevLib(LibActNum);
  DispGraph;
end;

procedure FuncRotDevice;
var DirDevKey : char;
begin
  DispText;
  DeleteLine(1);
  DeleteLine(2);
  SetColor(TSCColor);
  WriteTextXY(2,2,'LIB L<-DIR->R');
  WriteStatusData(13);
  repeat
    DirDevKey := ' ';
    if Keypressed then
    begin
      DirDevKey := ReadKey;
      if DirDevKey = #0 then
      begin
        DirDevKey := ReadKey;
        CheckDevDir(DirDevKey);
        WriteStatusData(13);
      end;
```

```
    end;
  until DirDevKey = #13;
  case DeviceDir of
  1 : RotLibInEast;
  2 : RotLibInNorth;
  3 : RotLibInWest;
  4 : RotLibInSouth;
  end;
  DispText;
  DeleteLine(1);
  DeleteLine(2);
  Sound(1000);Delay(150);NoSound;
  DispGraph;
end;

procedure FuncMoveBoxLIB;
begin
  SYMB_Command := 'PLCE';
  LibCursorOn := true;
  DispGraph;
  CursorOn;
  repeat
    FuncKeysOn := Off;
    DispText;
    DeleteLine(1);
    DeleteLine(2);
    SetColor(TMCColor);
    WriteTextXY(2,1,'Plce Keep DelL RotL AskL Lcur');
    if LibLoadOk then
    begin
      DispLabel(LibActNum);
      DispGraph;
      MoveLibCursor(Keycommand);
    end
    else
    begin
      SetColor(TSCColor);
      WriteTextXY(2,2,SYMB_Command);
      DispGraph;
      CursorControl(KeyCommand);
    end;
    if FuncKeysOn then
    begin
      case KeyCommand of
      'B': begin
             OnScreenDisplay;
             RedrawDataOnScreen;
             EndBeep;
           end;
      'C': begin
             FuncPanScreen;
             DispGraph;
             Redraw;
             CursorOn;
```

```
          end;
       end; {case}
    end
    else
    begin
    case KeyCommand of
    #27     : begin
                if MoveLibOk then
                begin
                  DispLibDevice(On);
                  StoreDataDevLib(LibActNum);
                end;
                LibLoadOk := false;
                MoveLibOk := false;
              end;
    'P','p' : begin
                if LibLoadOk then
                begin
                  if not LibNameOk then
                  begin
                    EnterLibData;
                    DispGraph;
                    DispLibDevice(On);
                    StoreDataDevLib(LibActNum);
                    EndBeep;
                    MoveLibOk := false;
                  end
                  else
                  begin
                    DispLibLabel(LibActNum);
                    DeleteLine(2);
                    SetColor(TSCColor);
                    WriteTextXY(2,2,'Enter: [Y/N]');
                    repeat
                      EndBeep;
                      ch := ReadKey;
                      if ch in ['Y','y'] then
                      begin
                        DispGraph;
                        DispLibDevice(On);
                        StoreDataDevLib(LibActNum);
                        EndBeep;
                        LibLoadOk := false;
                      end;
                    until ch in ['Y','y','N','n'];
                  end;
                end
                else
                begin
                  DispText;
                  DeleteLine(1);
                  DeleteLine(2);
                  SetColor(TSCColor);
                  WriteTextXY(2,2,'LIB not exist !!!');
```

```
                EndBeep;
                EnterKey;
                DispGraph;
              end;
            end;
    'K','k' : begin
              if not LibLoadOk then
              begin
                CheckLibExist(PositionCurX,PositionCurY);
                if CheckLibOk then
                begin
                  MoveLibOk := true;
                  LibLoadOk := true;
                  LibNameOk := true;
                  DispLibLabel(LibActNum);
                  DispGraph;
                  RestoreLibDataToDevData(LibActNum);
                  MoveDataDevToTem;
                  DelDevLib(LibActNum);
                  DispLibDevice(Off);
                  LibActNum := LibPointer;
                end
                else LibLoadOk := false;
              end;
            end;
    'D','d' : begin
              if not LibLoadOk then
              begin
                CheckLibExist(PositionCurX,PositionCurY);
                if CheckLibOk then
                begin
                  RestoreLibDataToDevData(LibActNum);
                  DispLibDevice(Off);
                  DispLibLabel(LibActNum);
                  DeleteLine(2);
                  SetColor(TMCColor);
                  WriteTextXY(2,2,'DELETE [Y/N]');
                  DispGraph;
                  repeat
                    Chkey := ReadKey;
                  until Chkey in ['Y','y','N','n',#13];
                  if Chkey in ['Y','y'] then
                  begin
                    DelDevLib(LibActNum);
                    LibPointer := LibPointer - 1;
                    EndBeep;
                    LibLoadOk := false;
                    LibNameOk := false;
                    MoveLibOk := true;
                  end
                  else DispLibDevice(On);
                end;
              end;
            end;
```

```
    'L','l' : begin
                LibCursorOn := not LibCursorOn;
                Sound(1000);delay(150);noSound;
              end;
    'A','a' : begin
                if not LibLoadOk then
                begin
                  CheckLibExist(PositionCurX,PositionCurY);
                  if CheckLibOk then
                  begin
                    DispLibLabel(LibActNum);
                    EnterKey;
                    DispGraph;
                  end
                  else EndBeep;
                end;
              end;
    'R','r' : FuncRotDevice;
    'Z','z' : begin
                FuncZoomScreen;
                DispGraph;
                Redraw;
                CursorOn;
              end;
    'S','s' : begin
                DispText;DeleteLine(1);
                FuncSetCurMovement;
              end;
    end;
  end;
  until KeyCommand in ['Q','q','E','e'];
  KeyCommand := ' '; {set command in LIBS Mode}
  DispText;
  DeleteLine(1);
  DeleteLine(2);
  DispGraph;
  CursorOff;
end;

{-------------------------------------------------}

procedure FuncLoadBoxLIB;
begin
  WriteStatusData(10);
  DispText;
  repeat
    DeleteLine(2);
    SetColor(TSCColor);
    WriteTextXY(2,2,'Load LIB : ');
    ReadTextXY(15,2,Name);
    if Name <> '' then
    begin
      if CheckFilename(Name) then
      begin
```

```
        if OpenRdLibsFile(LibsActDir+Name) then
        begin
          DeleteLine(2);
          WriteTextXY(2,2,LibsActDir+Name);
          ReadLibsFile(LibsActDir+Name);
          LibLoadOk := true;
          LibNameOk := false;
          Ok := true;
        end;
      end;
    end;
  Until Ok or (Name = '');
  if Name = '' then
  begin
    DeleteLine(2);
    WriteTextXY(2,2,'Load error !!!');
    EndBeep;
    Delay(1000);
  end;
  DeleteLine(2);
end;

procedure FuncLoadPCB;
begin
  WriteStatusData(15);
  DispText;
  repeat
    DeleteLine(2);
    SetColor(TSCColor);
    WriteTextXY(2,2,'Load PCB : ');
    ReadTextXY(15,2,Name);
    if Name <> '' then
    begin
      if CheckFilename(Name) then
      begin
        if OpenRdPCBFile(FileActDir+Name) then
        begin
          DeleteLine(2);
          WriteTextXY(2,2,FileActDir+Name);
          ReadCommandFromFile(FileActDir+Name);
          LibLoadOk := true;
          LibNameOk := false;
          Ok := true;
        end;
      end;
    end;
  Until Ok or (Name = '');
  if Name = '' then
  begin
    DeleteLine(2);
    WriteTextXY(2,2,'Load error !!!');
    EndBeep;
    Delay(1000);
  end;
```

```
  DeleteLine(2);
end;

procedure FuncSaveBoxLIB;
var Ok : boolean;
begin
  WriteStatusData(10);
  DispText;
  repeat
    DeleteLine(2);
    SetColor(TSCColor);
    WriteTextXY(2,2,'Save LIB : ');
    ReadTextXY(15,2,Name);
    if Name <> '' then
    begin
      if CheckFilename(Name) then
      begin
        if OpenWrLibsFile(LibsActDir+Name) then
        begin
          DeleteLine(2);
          WriteTextXY(2,2,LibsActDir+Name);
          WriteLibsFile(LibsActDir+Name);
          Ok := true;
        end;
      end;
    end;
  Until Ok or (Name = '');
  if Name = '' then
  begin
    DeleteLine(2);
    WriteTextXY(2,2,'Save error !!!');
    EndBeep;
    Delay(1000);
  end;
  DeleteLine(2);
end;

procedure FuncSavePCB;
var Ok : boolean;
begin
  WriteStatusData(15);
  DispText;
  repeat
    DeleteLine(2);
    SetColor(TSCColor);
    WriteTextXY(2,2,'Save PCB : ');
    ReadTextXY(15,2,Name);
    if Name <> '' then
    begin
      if CheckFilename(Name) then
      begin
        if OpenWrPCBFile(FileActDir+Name) then
        begin
          DeleteLine(2);
```

```
          WriteTextXY(2,2,FileActDir+Name);
          WriteCommandToFile(FileActDir+Name);
          Ok := true;
        end;
      end;
    end;
  Until Ok or (Name = '');
  if Name = '' then
  begin
    DeleteLine(2);
    WriteTextXY(2,2,'Save error !!!');
    EndBeep;
    Delay(1000);
  end;
  DeleteLine(2);
end;

{-------------------------------------}

procedure DeletePadCodeData;
begin
  DeletePadsData(PositionCurX,PositionCurY,PadDelPnt,PadErr);
  if PadErr = 0 then
  begin
    if CheckUserPad(PadDelPnt) then
    begin
      UnDispPads(PadDelPnt);
      CompressPadsData(PadDelPnt);
      WriteStatusData(6);
    end;
  end
  else
  if PadErr = 1 then
  begin
    DispText;
    DeleteLine(2);
    SetColor(ErrColor);
    WriteTextXY(2,2,'Undo LIBs Pad !!! ');
    EnterKey;
    KeyCommand := ' ';
    DeleteLine(2);
    SetColor(TSCColor);
    WriteTextXY(2,2,MODE);
  end;
  DispGraph;
end;
procedure MoveScreenLine;
var TemX,TemY : integer;
begin
  DispGraph;
  CursorOff;
  case Layer of
  Sold : begin
           DeleteSoldLineData(PositionCurX,PositionCurY,LineOk,LineErr);
```

```
          DelSoldLineActiveOff;
        end;
  Comp : begin
          DeleteCompLineData(PositionCurX,PositionCurY,LineOk,LineErr);
          DelCompLineActiveOff;
        end;
  Silk : begin
          DeleteSilkLineData(PositionCurX,PositionCurY,LineOk,LineErr);
          DelSilkLineActiveOff;
        end;
  end;
  CursorOn;
  if LineOk then
  begin
    TemX := PositionCurX;
    TemY := PositionCurY;
    Repeat
      FuncKeysOn := Off;
      DispGraph;
      MoveLineCursor(KeyCommand);
      case KeyCommand of
      #27 : begin
              CursorOff;
              case Layer of
                Sold : EnterNewSoldLine(TemX,TemY);
                Comp : EnterNewCompLine(TemX,TemY);
                Silk : EnterNewSilkLine(TemX,TemY);
              end;
              CursorOn;
              Sound(1000);
              delay(150);
              NoSound;
              KeyCommand := #13;
            end;
      #13 : begin
              CursorOff;
              case Layer of
                Sold : EnterNewSoldLine(PositionCurX,PositionCurY);
                Comp : EnterNewCompLine(PositionCurX,PositionCurY);
                Silk : EnterNewSilkLine(PositionCurX,PositionCurY);
              end;
              CursorOn;
              Sound(1000);
              delay(150);
              NoSound;
            end;
      'A' : begin
              if FuncKeysOn then
              begin
                CursorOff;
                case Layer of
                  Sold : EnterDelSoldLine;
                  Comp : EnterDelCompLine;
                  Silk : EnterDelSilkLine;
```

```
                end;
                CursorOn;
                KeyCommand := #13; {MoveLine Command Exit}
              end;
            end;
        end;
      until KeyCommand = #13;
    end;
end;

procedure FuncDelCircle;
begin
  DispGraph;
  case Layer of
  Sold : CheckSoldCirData(PositionCurX,PositionCurY,PpntNum,PadErr);
  Comp : CheckCompCirData(PositionCurX,PositionCurY,PpntNum,PadErr);
  Silk : CheckSilkCirData(PositionCurX,PositionCurY,PpntNum,PadErr);
  end;
  case PadErr of
  0 : begin
        case Layer of
        Sold : begin
                 case SoldVar^[PpntNum].CdeData of
                   CircCode : UnDispCirSold(PpntNum);
                   ArcCode  : UnDispArcSold(PpntNum);
                 end;
                 CompressSoldData(PpntNum);
               end;
        Comp : begin
                 case CompVar^[PpntNum].CdeData of
                   CircCode : UnDispCirComp(PpntNum);
                   ArcCode  : UnDispArcComp(PpntNum);
                 end;
                 CompressCompData(PpntNum);
               end;
        Silk : begin
                 case SilkVar^[PpntNum].CdeData of
                   CircCode : UnDispCirSilk(PpntNum);
                   ArcCode  : UnDispArcSilk(PpntNum);
                 end;
                 CompressSilkData(PpntNum);
               end;
        end;
        EndBeep;
      end;
  1 : begin
        DispText;
        DeleteLine(2);
        SetColor(ErrColor);
        WriteTextXY(2,2,'LIBs Error !!! ');
        EnterKey;
        KeyCommand := ' ';
        DeleteLine(2);
        SetColor(TSCColor);
```

```
        WriteTextXY(2,2,MODE);
        DispGraph;
      end;
  end;
end;

procedure FuncDeleteText;
var X,Y,X0,Y0,X1,Y1 : integer;
    DelTNum : integer;
    TextLX,TextLY : integer;
begin
  for i := 1 to textPointer do
  begin
    X := PositionCurX;
    Y := PositionCurY;
    TextArea(i,X0,Y0,X1,Y1);
    if CheckCurInBox(X,Y,X0,Y0,X1,Y1) = 1 then
    begin
      DelTNum := i;
    end;
  end;
  DispGraph;
  TextArea(DelTNum,X0,Y0,X1,Y1);
  TextLX := X1-X0;
  TextLY := Y1-Y0;
  CursorOff;
  TdirTemp := TextVar^[DelTNum].TxtDir;
  TLyrTemp := Black;
  StrMessg := TextVar^[DelTNum].Message;
  WriteGraphText(TdirTemp,TLyrTemp,X0,Y0,StrMessg);
  CursorOn;
  PositionCurX := X0;
  PositionCurY := Y0;
  repeat
    MoveTextCursor(KeyCommand,TextLX,TextLY);
    if not FuncKeysOn then
    begin
      case KeyCommand of
      #27 : begin
              CursorOff;
              TdirTemp := TextVar^[DelTNum].TxtDir;
              TLyrTemp := TextVar^[DelTNum].TxtLyr;
              StrMessg := TextVar^[DelTNum].Message;
              WriteGraphText(TdirTemp,TLyrTemp,X0,Y0,StrMessg);
              CursorOn;
            end;
       #13 : begin
              TdirTemp := TextVar^[DelTNum].TxtDir;
              TLyrTemp := TextVar^[DelTNum].TxtLyr;
              StrMessg := TextVar^[DelTNum].Message;
              TextPosX := PositionCurX;
              TextPosY := PositionCurY;
              RestoreText (DelTNum);
              CursorOff;
```

```
            WriteGraphText(TdirTemp,TLyrTemp,TextPosX,TextPosY,StrMessg);
            CursorOn;
          end;
      end;
    end
    else if KeyCommand = 'A' then CompressTextData(DelTNum);
  until KeyCommand in [#13,#27,'A'];
end;

procedure FunctionControl;
begin
  MoveCursor(KeyCommand);
  if FuncKeysOn then
  begin
    case KeyCommand of
    'H' : FuncDrawCircle;
    'F' : FuncFillPads;
    'G' : FuncFillLinkPads;
    'C' : begin
            FuncPanScreen;
            PnZmRdDisp;
            RedrawDataOnScreen;
            EndBeep;
          end;
    'B' : begin
            OnScreenDisplay;
            PnZmRdDisp;
            RedrawDataOnScreen;
            EndBeep;
          end;
    'I' : DeletePadCodeData;
    'J' : MoveScreenLine;
    'K' : FuncDelCircle;
    end;
  end
  else
  begin
    case KeyCommand of
    'C','c' : FuncSetCursor;
    'D','d' : FuncSetTextDir;
    'G','g' : FuncSetGridScreen;
    'L','l' : FuncSetLayer;
    'M','m' : begin
                FuncWriteTextOnPCB;
                EndBeep;
              end;
    'P','p' : FuncSetPadSize;
    'S','s' : begin
                FuncSetCurMovement;
                SetCurMovementDisp;
              end;
    'U','u' : FuncDeleteText;
    'W','w' : FuncSetThickness;
    'X','x' : FuncPadDir;
```

```
    'Z','z' : begin
                FuncZoomScreen;
                PnZmRdDisp;
                RedrawDataOnScreen;
                EndBeep;
              end;
    end;
  end;
  FuncKeysOn := false;
end;

begin { PCBSetUp }
  DispText;
  DeleteLine(1);
  MenuBlock(On);
  case MenuSelect of
    1 : begin
          Command := 'SYMB';
          MenuSelect := 1;
          repeat
            DispText;
            DeleteLine(1);
            DeleteLine(2);
            MenuBlock(On);
            SetColor(TMCColor);
            WriteTextXY(2,1,'LOAD SAVE EDIT LIBS CLRS EXIT');
            SetColor(TSCColor);
            WriteTextXY(2,2,Command);
            SelectMenu;
            DeleteLine(1);
            case MenuSelect of
              1 : begin
                    SYMB_Command := 'LOAD';
                    FuncLoadPCB;
                    DispGraph;
                    Redraw;
                    DispText;
                  end;
              2 : begin
                    SYMB_Command := 'SAVE';
                    FuncSavePCB;
                    DispText;
                  end;
              3 : begin
                    SYMB_Command := 'EDIT';
                    DispText;
                    DeleteLine(2);
                    SetColor(TSCColor);
                    writeTextXY(2,2,SYMB_Command);
                    DeleteLine(1);
                    SetColor(TMCColor);
                    WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
                    DispGraph;
                    CursorOn;
```

```
            TpX := PntX; TpY := PntY;
            OrgLine := false;
            StartLineX := 0;
            StartLineY := 0;
            StartOrg   := true;
            MODE := 'EDIT';
            Repeat
              FunctionControl;
            until KeyCommand in ['E','e','Q','q'];
            DispGraph;
            CursorOff;
            DispText;
            DeleteLine(2);
          end;
      4 : begin
            SYMB_Command := 'LIBS';
            MenuSelect := 1;
            LibLoadOk := false;
            LibNameOk := false;
            repeat
              WriteStatusData(13);
              DispText;
              DeleteLine(1);
              DeleteLine(2);
              MenuBlock(On);
              SetColor(TMCColor);
              WriteTextXY(2,1,'PLCE LOAD LDIR          EXIT');
              SetColor(TSCColor);
              WriteTextXY(2,2,SYMB_Command);
              SelectMenu;
              DeleteLine(1);
              case MenuSelect of
              1 : FuncMoveBoxLIB;
              2 : FuncLoadBoxLIB;
              3 : ReadLibsActDir;
              6 : SYMB_Command := 'EXIT';
              end { case };
            until SYMB_Command = 'EXIT';
            WriteStatusData(11);
            DispText;
            DeleteLine(2);
            SYMB_Command := 'LIBS';
            MenuSelect := 5;
          end;
      5 : begin
            ClearData;
            SYMB_Command := 'EXIT';
          end;
      6 : begin
            SYMB_Command := 'EXIT';
          end;
    end { case };
    DeleteLine(2);
  until SYMB_Command = 'EXIT';
```

```
        MenuSelect := 1;
      end;
  2 : begin
        Command := 'DEVC';
        MenuSelect := 1;
        ClearPointer;
        PositionCurX := 0;
        PositionCurY := 0;
        TempOffsX0 := MenuOffsX0;
        TempOffsY0 := MenuOffsY0;
        TempOffsX1 := MenuOffsX1;
        TempOffsY1 := MenuOffsY1;
        MenuOffsX0 := LibOffsX0;
        MenuOffsY0 := LibOffsY0;
        MenuOffsX1 := LibOffsX1;
        MenuOffsY1 := LibOffsY1;
        FuncPanScreen;
        RedrawDataOnScreen;
        repeat
          DispText;
          DeleteLine(1);
          MenuBlock(On);
          SetColor(TMCColor);
          WriteTextXY(2,1,'LDIR SAVE EDIT      CLRS EXIT');
          SetColor(TSCColor);
          DeleteLine(2);
          WriteTextXY(2,2,Command);
          SelectMenu;
          Deleteline(2);
          case MenuSelect of
            1 : ReadLibsActDir;
            2 : FuncSaveBoxLIB;
            3 : begin
                  DEVC_Command := 'EDIT';
                  SetColor(TSCColor);
                  DispText;
                  DeleteLine(2);
                  writeTextXY(2,2,DEVC_Command);
                  DeleteLine(1);
                  SetColor(TMCColor);
                  WriteTextXY(2,1,'CURW WThk PADs LAYR SCRN EXIT');
                  DispGraph;
                  TpX := PntX; TpY := PntY;
                  OrgLine := false;
                  StartLineX := 0;
                  StartLineY := 0;
                  StartOrg   := true;
                  MODE := 'EDIT';
                  Repeat
                    FunctionControl;
                  until KeyCommand in ['Q','q','E','e'];
                end;
            5 : begin
                  ClearData;
```

```
                DispGraph;
                CursorOn;
              end;
          6 : begin
                DispText;
                DeleteLine(2);
                WriteTextXY(2,2,'DEVC Exit [Y/N]');
                repeat
                  EnterPassKey(ch);
                  if ch in ['Y','y'] then
                  begin
                    DEVC_Command := 'EXIT';
                    ClearPointer;
                    MenuOffsX0 := TempOffsX0;
                    MenuOffsY0 := TempOffsY0;
                    MenuOffsX1 := TempOffsX1;
                    MenuOffsY1 := TempOffsY1;
                    OnScreenDisplay;
                     if LWStat then LibWindow;
                    EndBeep;
                  end;
                until ch in ['Y','y','N','n',#13];
              end;
        end { case };
      until DEVC_Command = 'EXIT';
      MenuSelect := 2;
    end;
  3 : begin
      DeleteLine(1);
      ReadFileActDir;
    end;
  4 : begin
      DispText;
      DeleteLine(1);
      DeleteLine(2);
      SetColor(TSCColor);
      WriteTextXY(2,2,'PLOT [Y/N]');
      repeat
        Chkey := ReadKey;
      until Chkey in ['Y','y','N','n'];
      if Chkey in ['Y','y'] then
        DrawPlotter;
      DeleteLine(2);
    end;
  5 : begin
      Command := 'STAT';
      MenuSelect := 1;
      repeat
        DispText;
        DeleteLine(1);
        DeleteLine(2);
        MenuBlock(On);
        SetColor(TMCColor);
        WriteTextXY(2,1,'LWOn LWOf LAYR ORGC ZmEn EXIT');
```

```
            SetColor(TSCColor);
            WriteTextXY(2,2,Command);
            SelectMenu;
            DeleteLine(1);
            case MenuSelect of
              1 : begin
                    STAT_Command := 'STAT';
                    if not LWStat then
                    begin
                      MenuOffsX0 := MenuOffsX0 + LibWindowX1;
                      LWStat := On;
                      OnScreenDisplay;
                      DispText;
                      DeleteLine(2);
                      write(^G);
                      LibWindow;
                      DispText;
                    end;
                  end;
              2 : begin
                    STAT_Command := 'STAT';
                    if LwStat then
                    begin
                      MenuOffsX0 := MenuOffsX0 - LibWindowX1;
                      OnScreenDisplay;
                      DispText;
                      LWStat := Off;
                    end;
                  end;
              3 : begin
                    FuncActiveLayer;
                    RedrawScreen(0);
                    MenuSelect := 6;
                  end;
              4 : begin
                    STAT_Command := 'ORGD';
                    DispText;
                    DeleteLine(2);
                    SetColor(TSCColor);
                    WriteTextXY(2,2,STAT_Command);
                    ORGStat := Not(ORGStat);
                    if ORGStat then
                    begin
                      OpenWindowORG(81,25,89,26);
                      WriteORG(2,1);
                    end
                    else
                      CloseWindowORG;
                    DispText;
                    EnterKey;
                    DeleteLine(2);
                  end;
              5 : if ZoomEn then ZoomEn := false
                  else ZoomEn := true;
```

```
            6 : begin
                  STAT_Command := 'EXIT';
                end;
          end { case };
          DeleteLine(2);
        until STAT_Command = 'EXIT';
        MenuSelect := 1;
      end;
    6 : begin
          DispText;
          DeleteLine(1);
          SetColor(TECColor);
          WriteTextXY(2,1,'[EXIT] Please confirm..');
          Ch := ReadKey;
          if Upcase(Ch) = 'Y' then Command := 'EXIT' else Command := '';
        end;
  end { case };
end { PCBSetUp };

procedure MainMenu;
begin { MainMenu }
  DispText;
  SetColor(TMCColor);
  DeleteLine(1);
  WriteTextXY(2,1,'SYMB DEVC DIRS PLOT STAT EXIT');
  MenuBlock(On);
end { MainMenu };

procedure PCBMain;
begin { PCBMain }
  repeat
    MainMenu;
    SelectMenu;
    PCBSetUp;
  until Command = 'EXIT';
  CloseGraph;
end { PCBMain };

begin

end { unit PCBUnit3 }.
```

```
Text File List Processing Program V4.00C Page : 108
File : A:PCBUNIT4.PAS
Current Date : Sunday September 4, 1988
Current Time : 2:04 AM

Program : PCB Part IV
Programmer : Suttinun Poramatikul


{
              +-------------------------------------+
              |                                     |
              | Unit  : PCB-CAD Unit IV             |
              | Programmer : Suttinun Poramatikul   |
              | Version    : 1.00A                  |
              | Last Updated : 23 Jun 1988          |
              |                                     |
              +-------------------------------------+
}

Unit  PCBUnit4;

interface

Uses   Crt,Graph,Declare;

const
  MaxStr = 7;

var
       WindowX0,WindowY0 : word;
       WindowX1,WindowY1 : word;
       Xcom0,Ycom0       : word;
       PD,PL             : byte;
       StepLengthX       : integer;
       StepLengthY       : integer;
       i,LayerTemp       : integer;
       GridRep           : boolean;

procedure EnterKey;

procedure EnterPassKey(var Keys : char);

procedure DrawStepLine(StepX1,StepY1,StepX2,StepY2:Longint);

procedure DrawLine(x1,y1,x2,y2:integer);

procedure DrawAngLine(x1,y1,x2,y2:integer);

procedure DelWire(X0,Y0,X1,Y1,Trce : integer);

procedure DrawPCBWire(X0,Y0,X1,Y1,Trce : integer);
```

```
procedure DrawNewWire(X0,Y0,X1,Y1,Trce : integer);
procedure CursorLine(x1,y1,x2,y2:integer);
procedure DeCodePadLyr(Pd:byte; var PadDr,PadLyr : byte);
procedure EnCodePadLyr(var Pd:byte; PadDr,PadLyr : byte);
procedure AskPadDiameter(PadType : integer;var T,InD,OuD : integer);
procedure EndBeep;
procedure DispORG;
procedure DispRad;
procedure DispAng;
procedure DispGraph;
procedure DispScreen;
procedure DispText;
procedure DispFile;
procedure ClearLibWindow;
procedure DispLibWindow;
procedure OpenWindowORG (X0,Y0,X1,Y1:byte);
procedure OpenWindowStat (X0,Y0,X1,Y1:byte);
procedure CloseWindowORG;
procedure CloseWindowStat;
procedure ORGView;
procedure WriteORG (x,y : byte);
procedure WriteRad (x,y : byte);
procedure WriteStatusData(StatNum:byte);
procedure SetLayerColor(SLayer : integer);
procedure GetLayerColor(var GrColor:integer);
procedure DelText(X,Y : byte; Str : string);
procedure WriteTextXY (x,y : byte; Str : string);
```

```
procedure DeleteLine (Y : byte);

procedure Grid (Status : boolean);

procedure SetCoordinateData;

procedure OnScreenDisplay;

procedure SetPosition;

procedure ResetPosition(PosX,PosY:longint);

function  CheckPadLyrRedraw(PNum : integer;var RdLyr : byte) : boolean;

procedure GridDataSet(GrdX,GrdY,ZmF,StpLngth : integer);

procedure SetParaZoom(ZoomVar : byte);

procedure ActiveTextDirLayer;

procedure WriteGraphText(TxtDir,Lyr: byte;PX,PY : integer;Name : string);

procedure StoreTextCommand(TDir,Lyr: byte; X,Y : integer; str: string);

procedure RotLibInNorth;

procedure RotLibInSouth;

procedure RotLibInEast;

procedure RotLibInWest;

function CheckFilename(Var Name : String) : boolean;

function OpenWrLibsFile(Filename : string) : boolean;

function OpenRdLibsFile(Filename : string) : boolean;

function OpenWrPCBFile(Filename : string) : boolean;

function OpenRdPCBFile(Filename : string) : boolean;


Implementation

{$L DrTHerc}
{$L DrTEGA}
var XcomHerc : word; YcomHerc : word;
    XcomEGA  : word; YcomEGA  : word;
    ColorCom : word;

procedure _DotEGA(Xi,Yi,ColorCom:word); External;

procedure _DotHerc(Xi,Yi:word); External;
```

```
procedure _Dot(X,Y:word);
begin
   if (X > GraphCornerX0-1) and (X < GraphCornerX1+1) then
   if (Y > GraphCornerY0-1) and (Y < GraphCornerY1+1) then
   begin
     case GraphDriver of
     EGA : begin
             XcomEGA := Xcom0; YcomEGA := Ycom0;
             _DotEGA(X,Y,ColorCom);
           end;
     HercMono : begin
             XcomHerc := Xcom0; YcomHerc := Ycom0;
             _DotHerc(X,Y);
           end;
     end;
   end;
end;

procedure DrawAngLine(x1,y1,x2,y2:integer);
var x,y,DeltaX,DeltaY,XStep,YStep,direction:integer;
begin
      x:=x1;
      y:=y1;
      XStep:=1;
      YStep:=1;
      if x1>x2 then XStep:=-1;
      if y1>y2 then YStep:=-1;
      DeltaX:=abs(x2-x1);
      DeltaY:=abs(y2-y1);
      if DeltaX=0 then direction:=-1
      else direction:=0;
      while not ((x=x2) and (y=y2)) do
       begin
        Xcom0 := x;
        Ycom0 := y;
        _Dot(Xcom0,Ycom0);
        if direction<0 then
         begin
          y:=y+YStep;
          direction:=direction+DeltaX;
         end
        else
         begin
          x:=x+XStep;
          direction:=direction-DeltaY;
         end;
       end;
end;

procedure DrawOrthLine(x1,y1,x2,y2:integer);
var x,y,DeltaX,DeltaY,XStep,YStep,direction:integer;
begin
      x:=x1;
```

```
      y:=y1;
      XStep:=1;
      YStep:=1;
      if x1>x2 then XStep:=-1;
      if y1>y2 then YStep:=-1;
      DeltaX:=abs(x2-x1);
      DeltaY:=abs(y2-y1);
      if StepLengthX >= StepLengthY then
      begin
        while not (x = x2) do
        begin
          Xcom0 := x;
          Ycom0 := y;
          _Dot(Xcom0,Ycom0);
          x:=x+XStep;
        end;
        while not (y = y2) do
        begin
          Xcom0 := x;
          Ycom0 := y;
          _Dot(Xcom0,Ycom0);
          y:=y+YStep;
        end;
      end;
      if StepLengthX < StepLengthY then
      begin
        while not (y = y2) do
        begin
          Xcom0 := x;
          Ycom0 := y;
          _Dot(Xcom0,Ycom0);
          y:=y+YStep;
        end;
        while not (x = x2) do
        begin
          Xcom0 := x;
          Ycom0 := y;
          _Dot(Xcom0,Ycom0);
          x:=x+XStep;
        end;
      end;
end;

procedure Draw45dLine(StepX0,StepY0,StepX1,StepY1:integer);
var PosStepX,PosStepY,Mul,StepX2,StepY2:integer;
    X0,Y0,X1,Y1,X2,Y2 : integer;
begin
  PosStepX := StepX1-StepX0;
  PosStepY := StepY1-StepY0;
  if abs(PosStepX) >= abs(PosStepY) then
  begin
    if abs(PosStepX) <> abs(PosStepY) then
    begin
      if PosStepX < 0 then Mul := -1 else Mul := 1;
```

```
      PosStepX := (abs(PosStepX) - abs(PosStepY)) * Mul;
      PosStepY := 0;
    end;
  end
  else
  begin
    if PosStepY < 0 then Mul := -1 else Mul := 1;
    PosStepY := (abs(PosStepY) - abs(PosStepX)) * Mul;
    PosStepX := 0;
  end;
  StepX2 := StepX0 + PosStepX;
  StepY2 := StepY0 + PosStepY;
  X0  := ((StepX0 * GridSizeX) div StepLength) + GraphCornerX0;
  Y0  := ((StepY0 * GridSizeY) div StepLength) + GraphCornerY0;
  X2  := ((StepX2 * GridSizeX) div StepLength) + GraphCornerX0;
  Y2  := ((StepY2 * GridSizeY) div StepLength) + GraphCornerY0;
  X1  := ((StepX1 * GridSizeX) div StepLength) + GraphCornerX0;
  Y1  := ((StepY1 * GridSizeY) div StepLength) + GraphCornerY0;
  DrawAngLine(X0,Y0,X2,Y2);
  DrawAngLine(X2,Y2,X1,Y1);
end;

procedure DrawStepLine(StepX1,StepY1,StepX2,StepY2:Longint);
var  x1,x2,y1,y2 : integer;
begin
  x1 := (((StepX1-PanOffsX0) * GridSizeX) div StepLength) + GraphCornerX0;
  y1 := (((StepY1-PanOffsY0) * GridSizeY) div StepLength) + GraphCornerY0;
  x2 := (((StepX2-PanOffsX0) * GridSizeX) div StepLength) + GraphCornerX0;
  y2 := (((StepY2-PanOffsY0) * GridSizeY) div StepLength) + GraphCornerY0;
  StepLengthX := abs(StepX2-StepX1);
  StepLengthY := abs(StepY2-StepY1);
  case RubberLineStyle of
    Angl : DrawAngLine(x1,y1,x2,y2);
    Orth : DrawOrthLine(x1,y1,x2,y2);
    dg45 : begin
             StepX1 := StepX1-PanOffsX0;
             StepY1 := StepY1-PanOffsY0;
             StepX2 := StepX2-PanOffsX0;
             StepY2 := StepY2-PanOffsY0;
             Draw45dLine(StepX1,StepY1,StepX2,StepY2);
           end;
  end;
end;

procedure DrawLine(x1,y1,x2,y2:integer);
begin
  x1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphCornerX0;
  y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphCornerY0;
  x2 := (((X2-PanOffsX0) * GridSizeX) div StepLength) + GraphCornerX0;
  y2 := (((Y2-PanOffsY0) * GridSizeY) div StepLength) + GraphCornerY0;
  DrawOrthLine(x1,y1,x2,y2);
end;

procedure CalDeltaXY(var DeX,DeY,Trce : integer);
```

```
var Theta : real;
begin
  Theta :=  Arctan (-DeY/DeX);
  DeX   := trunc((Trce) * Sin (Theta));
  DeY   := trunc((Trce) * Cos (Theta));
  DeX   := (DeX * GridSizeX) div StepLength;
  DeY   := (DeY * GridSizeY) div StepLength;
end;

procedure DrawThicknessLine(X0,Y0,X1,Y1,Trce : integer);
var DeX,DeY,Thk,ThkZoom  : integer;
    Theta,ThTrce,ThX,ThY : integer;
begin
  if ZoomFactor = 6 then ThkZoom := 2
  else ThkZoom := 3;
  case Trce of
  1 : ThTrce := 12;
  2 : ThTrce := 16;
  3 : ThTrce := 20;
  4 : ThTrce := 50;
  else ThTrce := 0;
  end;
  if ThTrce <> 0 then
  begin
    DeX := X1-X0;
    DeY := Y1-Y0;
    if DeX = 0 then
    begin
      ThY := 0;
      if ZoomFactor = 6 then
        Thk := 1
      else Thk := 2;
      repeat
        ThX := (Thk * GridSizeX) div StepLength;
        Line(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
        Line(X0-ThX,Y0-ThY,X1-ThX,Y1-ThY);
        Circle(X0,Y0,ThX);
        Circle(X1,Y1,ThX);
        Thk := Thk + ThkZoom;
      until Thk > (ThTrce div 2);
    end;
    if DeY = 0 then
    begin
      Thk := 2;
      repeat
        ThX := 0;
        ThY := (Thk * GridSizeY) div StepLength;
        Line(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
        Line(X0-ThX,Y0-ThY,X1-ThX,Y1-ThY);
        ThX := (Thk * GridSizeX) div StepLength;
        Circle(X0,Y0,ThX);
        Circle(X1,Y1,ThX);
        Thk := Thk + ThkZoom;
      until Thk > (ThTrce div 2);
```

```
    end;
    if (DeX <> 0) and (DeY <> 0) then
    begin
      Thk := 2;
      repeat
        ThX := DeX; ThY := DeY;
        CalDelTaXY(ThX,ThY,Thk);
        Line(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
        Line(X0-ThX,Y0-ThY,X1-ThX,Y1-ThY);
        ThX := (Thk * GridSizeX) div StepLength;
        Circle(X0,Y0,ThX);
        Circle(X1,Y1,ThX);
        Thk := Thk + ThkZoom;
      until Thk > (ThTrce div 2);
    end;
  end;{ThTrce = 0}
end;

procedure DrawThinLine(X0,Y0,X1,Y1,Trce : integer);
var DeX,DeY,Thk,ThkZoom  : integer;
    Theta,ThTrce,ThX,ThY : integer;
begin
  case Trce of
  1 : ThTrce := 12;
  2 : ThTrce := 16;
  3 : ThTrce := 20;
  4 : ThTrce := 50;
  else ThTrce := 0;
  end;
  if ThTrce <> 0 then
  begin
    DeX := X1-X0;
    DeY := Y1-Y0;
    if DeX = 0 then
    begin
      ThY := 0;
      ThX := ((ThTrce div 2) * GridSizeX) div StepLength;
      Line(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
      Line(X0-ThX,Y0-ThY,X1-ThX,Y1-ThY);
      Circle(X0,Y0,ThX);
      Circle(X1,Y1,ThX);
    end;
    if DeY = 0 then
    begin
      ThX := 0;
      ThY := ((ThTrce div 2) * GridSizeY) div StepLength;
      Line(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
      Line(X0-ThX,Y0-ThY,X1-ThX,Y1-ThY);
      ThX := ((ThTrce div 2) * GridSizeX) div StepLength;
      Circle(X0,Y0,ThX);
      Circle(X1,Y1,ThX);
    end;
    if (DeX <> 0) and (DeY <> 0) then
    begin
```

```
      Thk := ThTrce div 2;
      ThX := DeX; ThY := DeY;
      CalDelTaXY(ThX,ThY,Thk);
      Line(X0+ThX,Y0+ThY,X1+ThX,Y1+ThY);
      Line(X0-ThX,Y0-ThY,X1-ThX,Y1-ThY);
      ThX := ((ThTrce div 2) * GridSizeX) div StepLength;
      Circle(X0,Y0,ThX);
      Circle(X1,Y1,ThX);
    end;
  end;{ThTrce = 0}
end;

procedure DrawPCBWire(X0,Y0,X1,Y1,Trce : integer);
begin
  if (ZoomFactor in [5,6]) or (Trce = 4) then
  begin
     if ZoomEn then
       DrawThicknessLine(X0,Y0,X1,Y1,Trce)
     else
       DrawThinLine(X0,Y0,X1,Y1,Trce);
  end
  else Line(X0,Y0,X1,Y1);
end;

procedure DelWire(X0,Y0,X1,Y1,Trce : integer);
begin
  SetColor(Black);
  X0  := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
  Y0  := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
  X1  := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
  Y1  := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
  DrawPCBWire(X0,Y0,X1,Y1,Trce);
end;

procedure DrawNewWire(X0,Y0,X1,Y1,Trce : integer);
begin
  SetLayerColor(Layer);
  X0  := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
  Y0  := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
  X1  := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
  Y1  := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
  DrawPCBWire(X0,Y0,X1,Y1,Trce);
end;

procedure CursorLine(x1,y1,x2,y2:integer);
begin
  x1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphCornerX0;
  y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphCornerY0;
  x2 := (((X2-PanOffsX0) * GridSizeX) div StepLength) + GraphCornerX0;
  y2 := (((Y2-PanOffsY0) * GridSizeY) div StepLength) + GraphCornerY0;
  DrawAngLine(x1,y1,x2,y2);
end;

procedure DecodePadLyr(Pd:byte; var PadDr,PadLyr : byte);
```

```
begin
  PadDr  := Pd and $0F;
  PadLyr := Pd div 16;
end;

procedure EnCodePadLyr(var Pd:byte; PadDr,PadLyr : byte);
begin
  Pd := PadLyr*16 + PadDr;
end;

procedure AskPadDiameter(PadType : integer;var T,InD,OuD : integer);
begin
  if PadAssError then
  begin
     case PadType of
     1 : begin T := 0;InD := 16;  OuD := 32;  end;
     2 : begin T := 1;InD := 16;  OuD := 32;  end;
     3 : begin T := 0;InD := 30;  OuD := 52;  end;
     4 : begin T := 1;InD := 30;  OuD := 52;  end;
     5 : begin T := 0;InD := 40;  OuD := 62;  end;
     6 : begin T := 1;InD := 40;  OuD := 62;  end;
     7 : begin T := 0;InD := 50;  OuD := 75;  end;
     8 : begin T := 1;InD := 50;  OuD := 75;  end;
     9 : begin T := 0;InD := 60;  OuD := 100; end;
    10 : begin T := 1;InD := 60;  OuD := 100; end;
    11 : begin T := 2;InD := 300; OuD := 50;  end;
    12 : begin T := 3;InD := 300; OuD := 100; end;
     end;
  end
  else
  begin
    case PadAssData[PadType].PadStl of
    0 {R}     : begin
                  T := 0;
                  InD := PadAssData[PadType].InDia;
                  OuD := PadAssData[PadType].OutDia;
                end;
    1 {S}     : begin
                  T := 1;
                  InD := PadAssData[PadType].InDia;
                  OuD := PadAssData[PadType].OutDia;
                end;
    2 {THor}  : begin
                  T := 2;
                  InD := PadAssData[PadType].InDia;
                  OuD := PadAssData[PadType].OutDia;
                end;
    end;
  end;
end;

procedure EndBeep;
begin { CursorBeep }
  Sound (2000);
```

```
  Delay (100);
  NoSound;
end { CursorBeep };

procedure DispGraph;
begin { DispGraph }
  SetViewPort(AxisX0,AxisY0,AxisX1,AxisY1,ClipOn);
end { DispGraph };

procedure DispScreen;
begin { DispScreen }
  SetViewPort(0,0,MaxX,MaxY,ClipOn);
end { DispScreen };

procedure DispText;
begin { DispText };
  SetViewPort(TextAxisX0,TextAxisY0,TextAxisX1,TextAxisY1, ClipOn);
end { DispText };

procedure DispFile;
begin { DispFile };
  SetViewPort(FileAxisX0,FileAxisY0,FileAxisX1,FileAxisY1, ClipOn);
end { DispFile };

procedure DispStat;
begin
  SetViewPort(StatWindowX0,StatWindowY0,StatWindowX1,StatWindowY1,ClipOn);
end;

procedure DispLibWindow;
begin
  SetViewPort(LibWindowX0,LibWindowY0,LibWindowX1,LibWindowY1,ClipOn);
end;

procedure ClearLibWindow;
begin
  DispLibWindow;
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar(LibWindowX0,LibWindowY0,LibWindowX1,LibWindowY1);
end;
procedure OpenWindowORG (X0,Y0,X1,Y1 : byte);
begin { WindowORG }
  Dec(X0);
  Dec(Y0);
  ORGWindowX0 := X0 * GraphTextWidth;
  ORGWindowY0 := Y0 * GraphTextHeight;
  ORGWindowX1 := X1 * GraphTextWidth;
  ORGWindowY1 := Y1 * GraphTextHeight;
  SetViewPort(ORGWindowX0,ORGWindowY0,ORGWindowX1,ORGWindowY1,ClipOn);
  GetImage(0,0,(X1-X0) * GraphTextWidth,(Y1-Y0) * GraphTextHeight,ORGWindow^);
  if GraphMode <> HercMonoHi then
```

```
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar(0,0,(X1-X0) * GraphTextWidth,(Y1-Y0) * GraphTextHeight);
  LayerTemp := Layer;
  Layer := Sold;
  GetLayerColor(LayerColor);
  SetColor(LayerColor);
  SetColor(TECColor);
  Rectangle(0,0,(X1-X0) * GraphTextWidth,(Y1-Y0) * GraphTextHeight);
  Layer := LayerTemp;
end;

procedure CloseWindowORG;
begin
  SetViewPort(ORGWindowX0,ORGWindowY0,ORGWindowX1,ORGWindowY1,ClipOn);
  PutImage(0,0,ORGWindow^,NormalPut);
end;

procedure OpenWindowStat (X0,Y0,X1,Y1 : byte);
begin { WindowStat }
  Dec(X0);
  Dec(Y0);
  StatWindowX0 := X0 * GraphTextWidth;
  StatWindowY0 := Y0 * GraphTextHeight;
  StatWindowX1 := X1 * GraphTextWidth;
  StatWindowY1 := Y1 * GraphTextHeight;
  SetViewPort(StatWindowX0,StatWindowY0,StatWindowX1,StatWindowY1,ClipOn);
  GetImage(0,0,(X1-X0) * GraphTextWidth,(Y1-Y0) * GraphTextHeight,StatWindow^);
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar(0,0,(X1-X0) * GraphTextWidth,(Y1-Y0) * GraphTextHeight);
  LayerTemp := Layer;
  Layer := Sold;
  GetLayerColor(LayerColor);
  SetColor(LayerColor);
  SetColor(TECColor);
  Rectangle(0,0,(X1-X0) * GraphTextWidth,(Y1-Y0) * GraphTextHeight);
  Layer := LayerTemp;
  WriteStatusData(1);
  WriteStatusData(2);
  WriteStatusData(3);
  WriteStatusData(4);
  WriteStatusData(5);
  WriteStatusData(6);
  WriteStatusData(7);
  WriteStatusData(8);
  WriteStatusData(9);
  WriteStatusData(10);
  WriteStatusData(11);
end;
```

```
procedure CloseWindowStat;
begin
  SetViewPort(StatWindowX0,StatWindowY0,StatWindowX1,StatWindowY1,ClipOn);
  PutImage(0,0,StatWindow^,NormalPut);
end;

procedure ORGView;
begin
  SetViewPort(ORGWindowX0,ORGWindowY0,ORGWindowX1,ORGWindowY1,ClipOn);
end { ORGView };

procedure WriteORG (x,y : byte);
var NumPX ,NumPY : byte;
    PntXStr,PntYStr : string;
begin { WriteORG }
  Dec (X);
  Dec (Y);
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar (X * GraphTextWidth, Y * GraphTextHeight + 4,
       (X+MaxStr) * GraphTextWidth + 6 ,Y * GraphTextHeight + 20);
  Str(PositionCurX,PntXStr);
  NumPX := Length(PntXStr);
  Str(PositionCurY,PntYStr);
  NumPY := Length(PntYStr);
  SetColor(White);
  OutTextXY((X+MaxStr-NumPX) * GraphTextWidth,Y * GraphTextHeight + 4,PntXStr);
  OutTextXY((X+MaxStr-NumPY) * GraphTextWidth,Y * GraphTextHeight + 14,PntYStr);
end { WriteORG };

procedure WriteAng (x,y : byte);
var NumStA ,NumEnA : byte;
    StAStr,EnAStr : string;
begin { WriteORG }
  Dec (X);
  Dec (Y);
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar (X * GraphTextWidth, Y * GraphTextHeight + 4,
       (X+MaxStr) * GraphTextWidth + 6 ,Y * GraphTextHeight + 20);
  Bar (X * GraphTextWidth, Y * GraphTextHeight + 4,
       (X+MaxStr) * GraphTextWidth + 6 ,Y * GraphTextHeight + 20);
  Str(StAngl,StAStr);
  NumStA := Length(StAStr);
  Str(EnAngl,EnAStr);
  NumEnA := Length(EnAStr);
  SetColor(White);
  OutTextXY((X+MaxStr-NumStA-1) * GraphTextWidth,Y * GraphTextHeight + 4,StAStr);
  OutTextXY((X+MaxStr-NumEnA-1) * GraphTextWidth,Y * GraphTextHeight + 14,EnAStr);
  OutTextXY(X * GraphTextWidth,Y * GraphTextHeight + 4,'S');
```

```
  OutTextXY(X * GraphTextWidth,Y * GraphTextHeight + 14,'E');
  OutTextXY((X+MaxStr) * GraphTextWidth,Y * GraphTextHeight + 4,chr(248));
  OutTextXY((X+MaxStr) * GraphTextWidth,Y * GraphTextHeight + 14,chr(248));
end { WriteORG };

procedure WriteRad (x,y : byte);
var NumPR  : byte;
    RadStr : string;
begin { WriteRad }
  Dec (X);
  Dec (Y);
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar (X * GraphTextWidth, Y * GraphTextHeight + 4,
      (X+MaxStr) * GraphTextWidth + 6 ,Y * GraphTextHeight + 20);
  Bar (X * GraphTextWidth, Y * GraphTextHeight + 4,
      (X+MaxStr) * GraphTextWidth + 6 ,Y * GraphTextHeight + 20);
  Str(CIRRadius,RadStr);
  RadStr := 'R'+RadStr;
  NumPR := Length(RadStr);
  SetColor(White);
  OutTextXY((X+MaxStr-NumPR) * GraphTextWidth,Y * GraphTextHeight + 14,RadStr);
end { WriteRad };

procedure DelStat(x,y:byte;Str:string);
begin
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar (X * GraphTextWidth, Y * GraphTextHeight+2,
      (X+Length(str)) * GraphTextWidth+4,Y * GraphTextHeight + 12);
end;

procedure WriteStat (x,y : byte; TextColor:integer; Str:string);
begin { WriteStat }
  Dec(X);
  SetColor(TextColor);
  DelStat(X,Y,Str);
  case Y of
   0 : OutTextXY(X * GraphTextWidth,Y * GraphTextHeight+4 ,Str);
   1 : OutTextXY(X * GraphTextWidth,Y * GraphTextHeight+2, Str);
  end;
end { WriteStat };

procedure WriteStatusData(StatNum : byte);
var StatStr,TotalHoldStr : string;
    StrNL : byte;
    k : integer;
begin
   DispStat;
   case StatNum of
```

```
   1 : begin
         case RubberLineStyle of
          Angl : StatStr := 'Angl';
          Orth : StatStr := 'Orth';
          Dg45 : StatStr := '45Dg';
          else   StatStr := '    ';
         end;
         WriteStat( 2,0,TSCColor,StatStr);
       end;
   2 : begin
         if PadAssError then
         begin
           case PadSize of
            P52r : StatStr := 'P52r';
            P52s : StatStr := 'P52s';
            P60r : StatStr := 'P60r';
            P60s : StatStr := 'P60s';
            P72r : StatStr := 'P72r';
            P72s : StatStr := 'P72s';
           P100r : StatStr := '100e';
           P100s : StatStr := '100s';
           P200r : StatStr := '200e';
           P200s : StatStr := '200s';
            THor : StatStr := 'THor';
            TVer : StatStr := 'TVer';
            else   StatStr := '    ';
           end;
         end
         else
           StatStr := PadAssData[Padsize].PadNme;
         WriteStat( 8,0,TSCColor,StatStr);
       end;
   3 : begin
         case Layer of
          Sold : StatStr := 'Sold';
          Comp : StatStr := 'Comp';
          Silk : StatStr := 'Silk';
          else   StatStr := '    ';
         end;
         GetLayerColor(SysColor);
         WriteStat(2,1,SysColor,StatStr);
       end;
   7 : begin
         str(GoStep,StatStr);
         WriteStat(22,0,TSCColor,'    ');
         WriteStat(22,0,TSCColor,'G'+StatStr);
       end;
   5 : begin
         case Trace of
          Thck1 : StatStr := 'T12';
          Thck2 : StatStr := 'T16';
          Thck3 : StatStr := 'T20';
          Thck4 : StatStr := 'T50';
         end;
```

```
        WriteStat(18,0,TSCColor,StatStr);
      end;
  6 : begin
        Str(PadsPointer,TotalHoldStr);;
        TotalHoldStr := TotalHoldStr;
        StrNL := Length(TotalHoldStr);
        WriteStat(37-StrNl,0,TSCColor,TotalHoldStr);
        WriteStat(38,0,TSCColor,'Hld');
      end;
  8 : begin
        str(ZoomFactor,StatStr);
        WriteStat(14,0,TSCColor,'   ');
        WriteStat(14,0,TSCColor,'Zm'+StatStr);
      end;
  9 : begin
        str(PanOffsX0,StatStr);
        WriteStat(8,1,TSCColor,'         ');
        WriteStat(8,1,TSCColor,'X'+StatStr);
        str(PanOffsY0,StatStr);
        WriteStat(15,1,TSCColor,'         ');
        WriteStat(15,1,TSCColor,'Y'+StatStr);
      end;
 10 : begin
        StatStr := LibsActDir;
        if Length(LibsActDir) > 14 then
          StatStr := Copy (LibsActDir,1,14);
        WriteStat(22,1,TSCColor,'                ');
        WriteStat(22,1,TSCColor,'LDir '+ StatStr);
      end;
 11 : begin
        case TextDir of
         1 : StatStr := '[N]';
         2 : StatStr := '[S]';
         3 : StatStr := '[E]';
         4 : StatStr := '[W]';
         5 : StatStr := '[n]';
         6 : StatStr := '[s]';
         7 : StatStr := '[e]';
         8 : StatStr := '[w]';
         else StatStr:= '   ';
        end;
        WriteStat(27,0,TSCColor,StatStr);
      end;
 12 : begin
        str(CdeC,StatStr);
        WriteStat(14,0,TSCColor,'   ');
        WriteStat(14,0,TSCColor,'C'+StatStr);
      end;
 13 : begin
        case DeviceDir of
         1 : StatStr := '270';
         2 : StatStr := '000';
         3 : StatStr := '090';
         4 : StatStr := '180';
```

```
          else StatStr := '   ';
          end;
          WriteStat(27,0,TSCColor,StatStr);
        end;
  14 : begin
          DeCodePadLyr(PadDir,PD,PL);
          case PD of
          THor : StatStr := 'Hor';
          TVer : StatStr := 'Ver';
          else StatStr := '   ';
          end;
          WriteStat(27,0,TSCColor,StatStr);
        end;
  15 : begin
          StatStr := FileActDir;
          if Length(FileActDir) > 14 then
            StatStr := Copy (FileActDir,1,14);
          WriteStat(22,1,TSCColor,'                    ');
          WriteStat(22,1,TSCColor,'FDir '+ StatStr);
        end;

   end;
end;

procedure DispORG;
begin
   if ORGStat then
   begin
     ORGView;
     WriteORG(2,1);
   end
end;

procedure DispRad;
begin
  ORGView;
  WriteRad(2,1);
end;

procedure DispAng;
begin
  ORGView;
  WriteAng(2,1);
end;
procedure SetLayerColor(SLayer : integer);
begin
   Case GraphDriver of
   EGA : begin
           case SLayer of
             Sold : SetColor(SolLayer);
             Comp : SetColor(ComLayer);
             Silk : SetColor(SilLayer);
             Drll : SetColor(DrlLayer);
             BIack : SetColor(Black);
```

```
          end;
        end;
   HercMono : begin
          SetColor(White);
          case SLayer of
            Sold : SetLineStyle(SolidLn,$FFFF,NormWidth);
            Comp : SetLineStyle(DottedLn,$FFFF,NormWidth);
            Silk : SetLineStyle(DashedLn,$FFFF,NormWidth);
            Drll : SetLineStyle(CenterLn,$FFFF,NormWidth);
            Black : SetColor(Black);
          end;
        end;
   end;
end;
procedure GetLayerColor(Var GrColor:integer);
begin
   Case GraphDriver of
   EGA : begin
          SetLineStyle(SolidLn,$FFFF,NormWidth);
          case Layer of
            Sold : GrColor := SolLayer;
            Comp : GrColor := ComLayer;
            Silk : GrColor := SilLayer;
            Drll : GrColor := DrlLayer;
          end;
        end;
   HercMono : begin
          GrColor := White;
          case Layer of
            Sold : SetLineStyle(SolidLn,$FFFF,NormWidth);
            Comp : SetLineStyle(DottedLn,$FFFF,NormWidth);
            Silk : SetLineStyle(DashedLn,$FFFF,NormWidth);
            Drll : SetLineStyle(CenterLn,$FFFF,NormWidth);
          end;
        end;
   end;
end;

procedure DelText(X,Y : byte; Str : string);
begin
  Dec (X);
  Dec (Y);
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  Bar (X * GraphTextWidth, Y * GraphTextHeight,
       X + Length(Str) * GraphTextWidth + 2, Y * GraphTextHeight + 10);
end;

procedure WriteTextXY (x,y : byte; Str : string);
begin { WriteTextXY }
  Dec (X);
  Dec (Y);
```

```
  OutTextXY(X * GraphTextWidth,Y * GraphTextHeight,Str);
end { WriteTextXY };

procedure DeleteLine (Y : byte);
begin { DeleteLine }
  Dec(Y);
  if GraphMode <> HercMonoHi then
    SetFillPattern(Hidden,DMCColor)
  else
    SetFillPattern(DelMask,DMCColor);
  case Y of
  0 : Bar (0,0,(NumDelStr+1) * GraphTextWidth,10);
  1 : begin Bar (0,12,(NumDelStr+1) * GraphTextWidth,22);
            Bar (0,12,(NumDelStr+1) * GraphTextWidth,22);
      end;
  end;
end { DeleteLine };

procedure Grid (Status : boolean);
var
  StepPixelX,StepPixelY : word;
  Count,CountX,CountY : byte;
  LayerTemp : integer;

begin { Grid }
  GridOn := Status;
  StepPixelY := GraphScreenY0;
  CountY := 0;
  if GraphDriver <> EGA then
    SetLayerColor(Sold)
  else
    SetColor(TECColor);
  Rectangle(0,0,GraphScreenWidth + GridSizeX,GraphScreenHigth + GridSizeY);
  begin
    if GridOn then
    begin
      SetColor(White);
    end
    else
    begin
      SetColor(Black);
    end;
    if ZoomFactor <> 1 then
    while CountY <= StepGridOnScreenY do
    begin
      StepPixelX := GraphScreenX0;
      CountX := 0;
      Count := 0;
      while CountX <= StepGridOnScreenX do
      begin
        if GridOn then
          PutPixel (StepPixelX,StepPixelY,LightBlue)
        else
          PutPixel (StepPixelX,StepPixelY,Black);
```

```
        Inc(StepPixelX,GridSizeX);
        Inc(CountX);
        Inc(Count);
      end;
      Inc(StepPixelY,GridSizeY);
      Inc(CountY);
    end;
  end;
end { Grid };

procedure SetCoordinateData;
var Adj : integer;
begin { DrawAxis }
  MaxXGraph := MaxX - (GridSizeX div 2);
  MaxYGraph := MaxY - (GridSizeY div 2) - (GraphTextHeight * 3);
  StepGridOnScreenX := ((MaxXGraph - MenuOffsX0 - MenuOffsX1) div GridSizeX) - 1;
  StepGridOnScreenY := ((MaxYGraph - MenuOffsY0 - MenuOffsY1) div GridSizeY) - 1;
  AxisX0 := MenuOffsX0;
  AxisY0 := MenuOffsY0;
  TextAxisX0     := 0;
  TextAxisY0     := MaxYGraph + GraphTextHeight - 6;
  TextAxisX1     := MaxX;
  TextAxisY1     := MaxY;
  FileAxisX0     := 0;
  FileAxisY0     := 0;
  FileAxisX1     := GraphTextWidth * FileLenght;
  FileAxisY1     := GraphTextHeight * FileLine;
  GraphScreenX0  := GridSizeX div 2;
  GraphCornerX0  := AxisX0 + GraphScreenX0;
  GraphScreenY0  := GridSizeY div 2;
  GraphCornerY0  := AxisY0 + GraphScreenY0;
  GraphScreenWidth := StepGridOnScreenX * GridSizeX;
  GraphScreenHigth := StepGridOnScreenY * GridSizeY;
  GraphCornerX1  := GraphCornerX0 + GraphScreenWidth;
  GraphCornerY1  := GraphCornerY0 + GraphScreenHigth;
  GraphScreenX1  := GraphScreenX0 + GraphScreenWidth;
  GraphScreenY1  := GraphScreenY0 + GraphScreenHigth;
  ScreenLength   := StepGridOnScreenX * StepLength;
  ScreenHeight   := StepGridOnScreenY * StepLength;
  ScreenCenX     := (ScreenLength div 2);
  ScreenCenY     := (ScreenHeight div 2);
  Adj := 0;
  if (GridSizeX mod 2) <> 0 then Adj := 1;
  AxisX1 := GraphCornerX1 + Adj + (GridSizeX div 2);
  Adj := 0;
  if (GridSizeY mod 2) <> 0 then Adj := 1;
  AxisY1 := GraphCornerY1 + Adj + (GridSizeY div 2);
end { DrawAxis };

procedure OnScreenDisplay;
begin
  SetCoordinateData;
  SetGraphMode(GraphMode);
  OpenWindowStat(38,25,79,26);
```

```
  OpenWindowORG(81,25,89,26);
  WriteORG(2,1);
  DispGraph;
  Grid(GridStat);
end;

procedure EnterKey;
var Ch : char;
begin
  repeat
    Sound(1000);
    Delay(200);
    NoSound;
    Ch := ReadKey;
  until Ch = #13;
end;

procedure EnterPassKey(var Keys : char);
begin
  repeat
    Sound(1000);
    Delay(200);
    NoSound;
    Keys := ReadKey;
  until Keys in [#13,#27,'Y','y','N','n'];
end;

procedure SetPosition;
begin
  PositionCurX := (PntStepX*StepLength) div MovingStep;
  PositionCurY := (PntStepY*StepLength) div MovingStep;
end;

procedure ResetPosition(PosX,PosY:longint);
begin
  PntStepX := (PosX * MovingStep) div StepLength;
  PntStepY := (PosY * MovingStep) div StepLength;
end;

procedure GridDataSet(GrdX,GrdY,ZmF,StpLngth : integer);
begin
  ZoomFactor := ZmF;
  GridSizeX  := GrdX;
  GridSizeY  := GrdY;
  StepLength := StpLngth;
end;

procedure SetParaZoom(ZoomVar : byte);
begin
  case ZoomVar of
  1 : GridDataSet( 4, 3,1,100);  {15400x9800}
  2 : GridDataSet( 8, 6,2,100);  {7600x4800}
  3 : GridDataSet( 6, 5,3,50);   {5100x2900}
  4 : GridDataSet( 8, 6,4,50);   {3800x2400}
```

```
  5 : GridDataSet(12,10,5,50);    {2500x1400}
  6 : GridDataSet(10, 8,6,20);    {1200x700 }
  end;
end;

procedure ActiveTextDirLayer;
begin
  if Layer = Sold then
  begin
    case TextDir of
    1 : TextDir := 5;
    2 : TextDir := 6;
    3 : TextDir := 7;
    4 : TextDir := 8;
    end;
  end
  else
  begin
    case TextDir of
    5 : TextDir := 1;
    6 : TextDir := 2;
    7 : TextDir := 3;
    8 : TextDir := 4;
    end;
  end;
end;

{-------------------------}

function CheckPadLyrRedraw(PNum : integer;var RdLyr : byte) : boolean;
var PadT,PD : byte;
begin
  CheckPadLyrRedraw := true;
  PadT := PadsVar^[PNum].AssData[1];
  if PadAssError then
  begin
    if  PadT in [11,12] then
    begin
      CheckPadLyrRedraw := false;
      DeCodePadLyr(PadsVar^[PNum].AssData[2],PD,RdLyr);
    end;
  end
  else
  begin
    if PadAssData[PadT].PadStl = 2 then
    begin
      CheckPadLyrRedraw := false;
      DeCodePadLyr(PadsVar^[PNum].AssData[2],PD,RdLyr);
    end;
  end;
end;

{-------------------------------}
```

```
procedure WriteGraphText(TxtDir,Lyr : byte;PX,PY : integer;Name : string);
var ch    : char;
    buf   : string[1];
    s,i,l : integer;
    TX,TY : Longint;
    PCurX,PCurY : integer;
    X0,Y0,X1,Y1 : Longint;


procedure WriteTextInEast;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataX0;
    Y0 := PCurY + TxtComData[s].TxtCom[l].DataY0;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataX1;
    Y1 := PCurY + TxtComData[s].TxtCom[l].DataY1;
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX + 100;
end;

procedure WriteTextInEastInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataX0;
    Y0 := PCurY + TxtComData[s].TxtCom[l].DataY0 * -1;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataX1;
    Y1 := PCurY + TxtComData[s].TxtCom[l].DataY1 * -1;
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX + 100;
end;

procedure WriteTextInNorth;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataY0;
    Y0 := PCurY + (TxtComData[s].TxtCom[l].DataX0) * -1;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataY1;
    Y1 := PCurY + (TxtComData[s].TxtCom[l].DataX1) * -1;
```

```
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY - 100;
end;

procedure WriteTextInNorthInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataY0 * -1;
    Y0 := PCurY + TxtComData[s].TxtCom[l].DataX0 * -1;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataY1 * -1;
    Y1 := PCurY + TxtComData[s].TxtCom[l].DataX1 * -1;
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY - 100;
end;

procedure WriteTextInWestInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataX0 * -1;
    Y0 := PCurY + TxtComData[s].TxtCom[l].DataY0;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataX1 * -1;
    Y1 := PCurY + TxtComData[s].TxtCom[l].DataY1;
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX - 100;
end;

procedure WriteTextInWest;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataX0 * -1;
    Y0 := PCurY + TxtComData[s].TxtCom[l].DataY0 * -1;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataX1 * -1;
    Y1 := PCurY + TxtComData[s].TxtCom[l].DataY1 * -1;
```

```
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurX := PCurX - 100;
end;

procedure WriteTextInSouthInv;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataY0;
    Y0 := PCurY + TxtComData[s].TxtCom[l].DataX0;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataY1;
    Y1 := PCurY + TxtComData[s].TxtCom[l].DataX1;
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY + 100;
end;

procedure WriteTextInSouth;
var l : integer;
begin
  for l := 1 to TxtComData[s].TComNum do
  begin
    X0 := PCurX + TxtComData[s].TxtCom[l].DataY0 * -1;
    Y0 := PCurY + TxtComData[s].TxtCom[l].DataX0;
    X1 := PCurX + TxtComData[s].TxtCom[l].DataY1 * -1;
    Y1 := PCurY + TxtComData[s].TxtCom[l].DataX1;
    X0 := (((X0-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y0 := (((Y0-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    X1 := (((X1-PanOffsX0) * GridSizeX) div StepLength) + GraphScreenX0;
    Y1 := (((Y1-PanOffsY0) * GridSizeY) div StepLength) + GraphScreenY0;
    Line(X0,Y0,X1,Y1);
  end;
  PCurY := PCurY + 100;
end;

begin   {WriteGraphText}
  case TxtDir of
  1 : begin             {North}
        PCurX := PX - 80;
        PCurY := PY;
      end;
  2 : begin             {South}
        PCurX := PX + 80;
        PCurY := PY;
```

```
      end;
  3 : begin              {East}
        PCurX := PX;
        PCurY := PY - 80;
      end;
  4 : begin              {West}
        PCurX := PX;
        PCurY := PY + 80;
      end;
  5 : begin              {NorthInv}
        PCurX := PX + 80;
        PCurY := PY;
      end;
  6 : begin              {SouthInv}
        PCurX := PX - 80;
        PCurY := PY;
      end;
  7 : begin              {EastInv}
        PCurX := PX;
        PCurY := PY + 80;
      end;
  8 : begin              {WestInv}
        PCurX := PX;
        PCurY := PY - 80;
      end;
end;
SetLayerColor(Lyr);
for i := 1 to Length(Name) do
begin
  buf := copy(Name,i,1);
  if buf = ' ' then
  begin
    case TxtDir of
    1 : PCurY := PCurY - 100;
    2 : PCurY := PCurY + 100;
    3 : PCurX := PCurX + 100;
    4 : PCurX := PCurX - 100;
    5 : PCurY := PCurY - 100;
    6 : PCurY := PCurY + 100;
    7 : PCurX := PCurX + 100;
    8 : PCurX := PCurX - 100;
    end;
  end
  else
  begin
    for s := 1 to TxtNum do
    begin
      if TxtComData[s].KeyChar = buf then
      begin
        case TxtDir of
        1 : WriteTextInNorth;
        2 : WriteTextInSouth;
        3 : WriteTextInEast;
        4 : WriteTextInWest;
```

```
          5 : WriteTextInNorthInv;
          6 : WriteTextInSouthInv;
          7 : WriteTextInEastInv;
          8 : WriteTextInWestInv;
          end;
        end;
      end;
    end;
  end;
end;

procedure StoreTextCommand(TDir,Lyr: byte; X,Y : integer; str: string);
begin
  TextPointer := TextPointer + 1;
  TextVar^[TextPointer].TxtDir  := TDir;
  TextVar^[TextPointer].TxtLyr  := Lyr;
  TextVar^[TextPointer].PosData[1] := X;
  TextVar^[TextPointer].PosData[2] := Y;
  TextVar^[TextPointer].Message := Str;
end;

procedure RotLibInEast;
var l : integer;
    Dtem,Ltem,Ptem : byte;
begin
  RefX := RefTemX;
  RefY := RefTemY;
  MinDataX := MinTemDataX;
  MinDataY := MinTemDataY;
  MaxDataX := MaxTemDataX;
  MaxDataY := MaxTemDataY;
  for l := 1 to TemPointer do
  begin
    DevVar[l].PosData[1] := TemVar[l].PosData[1];
    DevVar[l].PosData[2] := TemVar[l].PosData[2];
    if TemVar[l].CdeData = LineCode then
    begin
      DevVar[l].PosData[3] := TemVar[l].PosData[3];
      DevVar[l].PosData[4] := TemVar[l].PosData[4];
    end;
    if TemVar[l].CdeData = ArcCode then
    begin
      DevVar[l].PosData[4] := TemVar[l].PosData[4];
      DevVar[l].AssData[1] := TemVar[l].AssData[1];
    end;
    if TemVar[l].CdeData = PadCode then
    begin
      DeCodePadLyr(TemVar[l].AssData[2],Dtem,Ltem);
      case Dtem of
      THor : Dtem := THor;
      TVer : Dtem := TVer;
      else Dtem := THor;
      end;
      EnCodePadLyr(Ptem,Dtem,Ltem);
```

```
      DevVar[1].AssData[2] := Ptem;
    end;
  end;
end;

procedure RotLibInNorth;
var 1 : integer;
    TemAngSt,TemAngEn : integer;
    Dtem,Ltem,Ptem : byte;
begin
  RefX := RefTemY;
  RefY := RefTemX * -1;
  MinDataX := MinTemDataX;
  MinDataY := MinTemDataY;
  MaxDataX := MaxTemDataY;
  MaxDataY := MaxTemDataX * -1;
  for 1 := 1 to TemPointer do
  begin
    DevVar[1].PosData[1] := TemVar[1].PosData[2];
    DevVar[1].PosData[2] := TemVar[1].PosData[1] * -1;
    if TemVar[1].CdeData = LineCode then
    begin
      DevVar[1].PosData[3] := TemVar[1].PosData[4];
      DevVar[1].PosData[4] := TemVar[1].PosData[3] * -1;
    end;
    if TemVar[1].CdeData = ArcCode then
    begin
      TemAngSt := TemVar[1].PosData[4] + 90;
      if TemAngSt >= 360 then
        TemAngSt := TemAngSt - 360;
      DevVar[1].PosData[4] := TemAngSt;
      TemAngEn := TemVar[1].AssData[1] + 90;
      if TemAngEn >= 360 then
        TemAngEn := TemAngEn - 360;
      DevVar[1].AssData[1] := TemAngEn;
    end;
    if TemVar[1].CdeData = PadCode then
    begin
      DeCodePadLyr(TemVar[1].AssData[2],Dtem,Ltem);
      case Dtem of
      THor : Dtem := TVer;
      TVer ; Dtem := THor;
      else Dtem := THor;
      end;
      EnCodePadLyr(Ptem,Dtem,Ltem);
      DevVar[1].AssData[2] := Ptem;
    end;
  end;
end;

procedure RotLibInWest;
var 1 : integer;
    TemAngSt,TemAngEn : integer;
    Dtem,Ltem,Ptem : byte;
```

```
begin
  RefX := RefTemX * -1;
  RefY := RefTemY * -1;
  MinDataX := MinTemDataX;
  MinDataY := MinTemDataY;
  MaxDataX := MaxTemDataX * -1;
  MaxDataY := MaxTemDataY * -1;
  for l := 1 to TemPointer do
  begin
    DevVar[l].PosData[1] := TemVar[l].PosData[1] * -1;
    DevVar[l].PosData[2] := TemVar[l].PosData[2] * -1;
    if TemVar[l].CdeData = LineCode then
    begin
      DevVar[l].PosData[3] := TemVar[l].PosData[3] * -1;
      DevVar[l].PosData[4] := TemVar[l].PosData[4] * -1;
    end;
    if TemVar[l].CdeData = ArcCode then
    begin
      TemAngSt := TemVar[l].PosData[4] + 180;
      if TemAngSt >= 360 then
        TemAngSt := TemAngSt - 360;
      DevVar[l].PosData[4] := TemAngSt;
      TemAngEn := TemVar[l].AssData[1] + 180;
      if TemAngEn >= 360 then
        TemAngEn := TemAngEn - 360;
      DevVar[l].AssData[1] := TemAngEn;
    end;
    if TemVar[l].CdeData = PadCode then
    begin
      DeCodePadLyr(TemVar[l].AssData[2],Dtem,Ltem);
      case Dtem of
      THor : Dtem := THor;
      TVer : Dtem := TVer;
      else Dtem := THor;
      end;
      EnCodePadLyr(Ptem,Dtem,Ltem);
      DevVar[l].AssData[2] := Ptem;
    end;
  end;
end;
procedure RotLibInSouth;
var l : integer;
    TemAngSt,TemAngEn : integer;
    Dtem,Ltem,Ptem : byte;
begin
  RefX := RefTemY * -1;
  RefY := RefTemX;
  MinDataX := MinTemDataX;
  MinDataY := MinTemDataY;
  MaxDataX := MaxTemDataY * -1;
  MaxDataY := MaxTemDataX;
  for l := 1 to TemPointer do
  begin
    DevVar[l].PosData[1] := TemVar[l].PosData[2] * -1;
```

```
    DevVar[1].PosData[2] := TemVar[1].PosData[1];
    if TemVar[1].CdeData = LineCode then
    begin
      DevVar[1].PosData[3] := TemVar[1].PosData[4] * -1;
      DevVar[1].PosData[4] := TemVar[1].PosData[3];
    end;
    if TemVar[1].CdeData = ArcCode then
    begin
      TemAngSt := TemVar[1].PosData[4] + 270;
      if TemAngSt >= 360 then
        TemAngSt := TemAngSt - 360;
      DevVar[1].PosData[4] := TemAngSt;
      TemAngEn := TemVar[1].AssData[1] + 270;
      if TemAngEn >= 360 then
        TemAngEn := TemAngEn - 360;
      DevVar[1].AssData[1] := TemAngEn;
    end;
    if TemVar[1].CdeData = PadCode then
    begin
      DeCodePadLyr(TemVar[1].AssData[2],Dtem,Ltem);
      case Dtem of
      THor : Dtem := TVer;
      TVer : Dtem := THor;
      else Dtem := THor;
      end;
      EnCodePadLyr(Ptem,Dtem,Ltem);
      DevVar[1].AssData[2] := Ptem;
    end;
  end;
end;

{-------------------------------------------------------------------}

function CheckFilename(Var Name : String) : boolean;
var  ChCount    : byte;
     Ch         : string[1];
     Ext        : boolean;
     NameOk     : boolean;
     NameBuffer : string;
     LogDrive   : string[2];
     ExtName    : String[3];
begin
    Ext := False;
    NameOk := False;
    NameBuffer := '';
    LogDrive := '';
    ExtName := '';
    i := 0;
    for ChCount := 1 to length(Name) do
     begin
        Ch := Copy(Name,ChCount,1);
        if Ch = '.' then Ext := True;
        if Ext then
           ExtName := concat(ExtName,Ch)
```

```
          else
            NameBuffer := concat(NameBuffer,Ch)
      end;
      if copy(NameBuffer,2,1) = ':' then
         begin
           logdrive := Copy(NameBuffer,1,2);
           NameBuffer := Copy(NameBuffer,3,Length(NameBuffer));
         end
      else
         NameBuffer := Copy(NameBuffer,1,Length(NameBuffer));
      Name := '';
      CheckFilename := true;
      for ChCount := 1 to Length(NameBuffer) do
      begin
        Ch := Copy(NameBuffer,ChCount,1);
        if Ch = '\' then CheckFilename := false;
      end;
      Name := Copy(NameBuffer,1,8);
end;

procedure FileExistDisp;
begin
  DeleteLine(2);
  SetColor(TMCColor);
  WriteTextXY(2,2,'File exist : Overwrite [Y/N]');
end;

function OpenWrLibsFile(Filename : string) : boolean;
var Ok,WrOk : boolean;Choice : char;
begin
  Assign (LibsFile,Filename+'.LIB');
  {$I-} reset (LibsFile); {$I+}
  Ok := (IOresult = 0);
  OpenWrLibsFile := true;
  WrOk := true;
  if Ok then
  begin
    FileExistDisp;
    EnterPassKey(Choice);
    if Choice in ['N','n'] then
    begin
     Close(LibsFile);
     OpenWrLibsFile := false;
     WrOk := false;
    end;
  end;
  if WrOk = true then
    rewrite (LibsFile);
end;

procedure FileNotExistDisp;
begin
  DeleteLine(2);
  SetColor(TMCColor);
```

```
  WriteTextXY(2,2,'File not exist');
end;

function OpenRdLibsFile(Filename : string) : boolean;
var Ok : boolean;Choice : char;
begin
  Assign (LibsFile,Filename+'.LIB');
  {$I-} reset (LibsFile); {$I+}
  Ok := (IOresult = 0);
  OpenRdLibsFile := true;
  if not Ok then
  begin
    FileNotExistDisp;
    repeat
      EnterPassKey(Choice);
    until Choice = #13;
    OpenRdLibsFile := false;
  end;
end;

function PCBFileOk (Filename : string) : boolean;
var Ok : boolean;
begin
  Assign (PCBFile,Filename);
  {$I-} reset (PCBFile); {$I+}
  Ok := (IOresult = 0);
  PCBFileOk := Ok;
  if Ok then Close(PCBFile);
end;

function OpenWrPCBFile(Filename : string) : boolean;
var Ok,WrOk : boolean;Choice : char;
begin
  if PCBFileOk(Filename+'SLK') then
  if PCBFileOk(Filename+'CMP') then
  if PCBFileOk(Filename+'SLD') then
  if PCBFileOk(Filename+'PAD') then
  if PCBFileOk(Filename+'MSG') then
  if PCBFileOk(Filename+'DEV') then
  begin
    OpenWrPCBFile := false;
    FileExistDisp;
  end
  else
    OpenWrPCBFile := true;
end;

function OpenRdPCBFile(Filename : string) : boolean;
var Ok : boolean;Choice : char;
begin
  if PCBFileOk(Filename+'SLK') then
  if PCBFileOk(Filename+'CMP') then
  if PCBFileOk(Filename+'SLD') then
  if PCBFileOk(Filename+'PAD') then
```

```
  if PCBFileOk(Filename+'MSG') then
  if PCBFileOk(Filename+'DEV') then
    OpenRdPCBFile := true
  else
  begin
    FileNotExistDisp;
    OpenRdPCBFile := false;
  end;
end;

{-------------------------------------------------------}

begin { Main }
   ColorCom := 7;
   GridRep := false;
end.
```

```
Program : PCB Part V
Programmer : Suttinun Poramatikul


Unit  PCBUnit5;

interface

Uses   Declare;

var   i,p : integer;
      ok  : boolean;

procedure CheckMinData;

procedure CheckMaxData;

procedure StoreSoldLineCommand;

procedure StoreSoldCircCommand;

procedure StoreSoldArcCommand;

procedure StoreCompCircCommand;

procedure StoreCompLineCommand;

procedure StoreCompArcCommand;

procedure StoreSilkCircCommand;

procedure StoreSilkLineCommand;

procedure StoreSilkArcCommand;

procedure StorePadsCommand;

procedure StoreLineCommand;

procedure StoreCircCommand;

procedure StoreArcCommand;

procedure WriteCommandToFile(filename : string);

procedure ReadCommandFromFile(filename : string);

procedure StoreDataDevLib(Pnt : integer);
```

```
procedure MoveDataDevToTem;

procedure MoveLibsDataToScrData;

procedure WriteLibsFile(filename : string);

procedure ReadLibsFile(filename : string);

procedure ReadTextCommandFile;

procedure RestoreText(TNum: integer);

procedure StoreTextCommand(TDir,Lyr: byte; X,Y : integer; str: string);

procedure TextArea(Tnum : integer; var X0,Y0,X1,Y1 : integer);

Implementation

procedure CheckMinData;
var ch : char; i      : integer;
    Tx0,Ty0,Tx1,Ty1   : integer;
    SoldMinX,SoldMinY : integer;
    CompMinX,CompMinY : integer;
    SilkMinX,SilkMinY : integer;
    PadsMinX,PadsMinY : integer;
    TextMinX,TextMinY : integer;

begin
  SoldMinX := 30000;
  SoldMinY := 30000;
  CompMinX := 30000;
  CompMinY := 30000;
  SilkMinX := 30000;
  SilkMinY := 30000;
  PadsMinX := 30000;
  PadsMinY := 30000;
  TextMinX := 30000;
  TextMinY := 30000;
  if SoldPointer > 0 then
  for i := 1 to SoldPointer do
  begin
    if SoldMinX >  SoldVar^[i].PosData[1] then
       SoldMinX := SoldVar^[i].PosData[1];
    if SoldMinY >  SoldVar^[i].PosData[2] then
       SoldMinY := SoldVar^[i].PosData[2];
    if SoldVar^[i].CdeData = LineCode then
    begin
      if SoldMinX >  SoldVar^[i].PosData[3] then
         SoldMinX := SoldVar^[i].PosData[3];
      if SoldMinY >  SoldVar^[i].PosData[4] then
         SoldMinY := SoldVar^[i].PosData[4];
    end;
  end;
```

```
  if CompPointer > 0 then
  for i := 1 to CompPointer do
  begin
    if CompMinX >  CompVar^[i].PosData[1] then
       CompMinX := CompVar^[i].PosData[1];
    if CompMinY >  CompVar^[i].PosData[2] then
       CompMinY := CompVar^[i].PosData[2];
   if CompVar^[i].CdeData = LineCode then
    begin
      if CompMinX >  CompVar^[i].PosData[3] then
         CompMinX := CompVar^[i].PosData[3];
      if CompMinY >  CompVar^[i].PosData[4] then
         CompMinY := CompVar^[i].PosData[4];
    end;
  end;
  if SilkPointer > 0 then
  for i := 1 to SilkPointer do
  begin
    if SilkMinX >  SilkVar^[i].PosData[1] then
       SilkMinX := SilkVar^[i].PosData[1];
    if SilkMinY >  SilkVar^[i].PosData[2] then
       SilkMinY := SilkVar^[i].PosData[2];
    if SilkVar^[i].CdeData = LineCode then
    begin
      if SilkMinX >  SilkVar^[i].PosData[3] then
         SilkMinX := SilkVar^[i].PosData[3];
      if SilkMinY >  SilkVar^[i].PosData[4] then
         SilkMinY := SilkVar^[i].PosData[4];
    end;
  end;
  if PadsPointer > 0 then
  for i := 1 to PadsPointer do
  begin
    if PadsMinX >  PadsVar^[i].PosData[1] then
       PadsMinX := PadsVar^[i].PosData[1];
    if PadsMinY >  PadsVar^[i].PosData[2] then
       PadsMinY := PadsVar^[i].PosData[2];
  end;
  if TextPointer > 0 then
  for i := 1 to TextPointer do
  begin
    TextArea (i,Tx0,Ty0,Tx1,Ty1);
    if TextMinX  > Tx0 then
       TextMinX := Tx0;
    if TextMinY  > Ty0 then
       TextMinY := Ty0;
    if TextMinX  > Tx1 then
       TextMinX := Tx1;
    if TextMinY  > Ty1 then
       TextMinY := Ty1;
  end;
  MinDataX := SoldMinX;
  if MinDataX > CompMinX then MinDataX := CompMinX;
  if MinDataX > SilkMinX then MinDataX := SilkMinX;
```

```
  if MinDataX > PadsMinX then MinDataX := PadsMinX;
  if MinDataX > TextMinX then MinDataX := TextMinX;
  MinDataY := SoldMinY;
  if MinDataY > CompMinY then MinDataY := CompMinY;
  if MinDataY > SilkMinY then MinDataY := SilkMinY;
  if MinDataY > PadsMinY then MinDataY := PadsMinY;
  if MinDataY > TextMinY then MinDataY := TextMinY;
end;

procedure CheckMaxData;
var ch : char; i      : integer;
    Tx0,Ty0,Tx1,Ty1   : integer;
    SoldMaxX,SoldMaxY : integer;
    CompMaxX,CompMaxY : integer;
    SilkMaxX,SilkMaxY : integer;
    PadsMaxX,PadsMaxY : integer;
    TextMaxX,TextMaxY : integer;
begin
  SoldMaxX := 0;
  SoldMaxY := 0;
  CompMaxX := 0;
  CompMaxY := 0;
  SilkMaxX := 0;
  SilkMaxY := 0;
  PadsMaxX := 0;
  PadsMaxY := 0;
  TextMaxX := 0;
  TextMaxY := 0;
  if SoldPointer > 0 then
  for i := 1 to SoldPointer do
  begin
    if SoldMaxX <  SoldVar^[i].PosData[1] then
       SoldMaxX := SoldVar^[i].PosData[1];
    if SoldMaxY <  SoldVar^[i].PosData[2] then
       SoldMaxY := SoldVar^[i].PosData[2];
    if SoldVar^[i].CdeData = LineCode then
    begin
      if SoldMaxX <  SoldVar^[i].PosData[3] then
         SoldMaxX := SoldVar^[i].PosData[3];
      if SoldMaxY <  SoldVar^[i].PosData[4] then
         SoldMaxY := SoldVar^[i].PosData[4];
    end;
  end;
  if CompPointer > 0 then
  for i := 1 to CompPointer do
  begin
    if CompMaxX <  CompVar^[i].PosData[1] then
       CompMaxX := CompVar^[i].PosData[1];
    if CompMaxY <  CompVar^[i].PosData[2] then
       CompMaxY := CompVar^[i].PosData[2];
    if CompVar^[i].CdeData = LineCode then
    begin
      if CompMaxX <  CompVar^[i].PosData[3] then
         CompMaxX := CompVar^[i].PosData[3];
```

```
      if CompMaxY < CompVar^[i].PosData[4] then
         CompMaxY := CompVar^[i].PosData[4];
    end;
  end;
  if SilkPointer > 0 then
  for i := 1 to SilkPointer do
  begin
    if SilkMaxX < SilkVar^[i].PosData[1] then
       SilkMaxX := SilkVar^[i].PosData[1];
    if SilkMaxY < SilkVar^[i].PosData[2] then
       SilkMaxY := SilkVar^[i].PosData[2];
    if SilkVar^[i].CdeData = LineCode then
    begin
      if SilkMaxX < SilkVar^[i].PosData[3] then
         SilkMaxX := SilkVar^[i].PosData[3];
      if SilkMaxY < SilkVar^[i].PosData[4] then
         SilkMaxY := SilkVar^[i].PosData[4];
    end;
  end;
  if PadsPointer > 0 then
  for i := 1 to PadsPointer do
  begin
    if PadsMaxX < PadsVar^[i].PosData[1] then
       PadsMaxX := PadsVar^[i].PosData[1];
    if PadsMaxY < PadsVar^[i].PosData[2] then
       PadsMaxY := PadsVar^[i].PosData[2];
  end;
  if TextPointer > 0 then
  for i := 1 to TextPointer do
  begin
    TextArea (i,Tx0,Ty0,Tx1,Ty1);
    if TextMaxX < Tx0 then
       TextMaxX := Tx0;
    if TextMaxY < Ty0 then
       TextMaxY := Ty0;
    if TextMaxX < Tx1 then
       TextMaxX := Tx1;
    if TextMaxY < Ty1 then
       TextMaxY := Ty1;
  end;
  MaxDataX := SoldMaxX;
  if MaxDataX < CompMaxX then MaxDataX := CompMaxX;
  if MaxDataX < SilkMaxX then MaxDataX := SilkMaxX;
  if MaxDataX < PadsMaxX then MaxDataX := PadsMaxX;
  if MaxDataX < TextMaxX then MaxDataX := TextMaxX;
  MaxDataY := SoldMaxY;
  if MaxDataY < CompMaxY then MaxDataY := CompMaxY;
  if MaxDataY < SilkMaxY then MaxDataY := SilkMaxY;
  if MaxDataY < PadsMaxY then MaxDataY := PadsMaxY;
  if MaxDataY < TextMaxY then MaxDataY := TextMaxY;
end;

procedure StoreSoldLineCommand;
begin
```

```
  SoldPointer := SoldPointer + 1;
  SoldVar^[SoldPointer].CdeData     := LineCode;
  SoldVar^[SoldPointer].LibPnt      := notLib;
  SoldVar^[SoldPointer].PosData[1] := DataX0;
  SoldVar^[SoldPointer].PosData[2] := DataY0;
  SoldVar^[SoldPointer].PosData[3] := DataX1;
  SoldVar^[SoldPointer].PosData[4] := DataY1;
  SoldVar^[SoldPointer].AssData[1] := Trace;
  SoldVar^[SoldPointer].AssData[2] := Layer;
end;
procedure StoreSoldCircCommand;
begin
  SoldPointer := SoldPointer + 1;
  SoldVar^[SoldPointer].CdeData     := CircCode;
  SoldVar^[SoldPointer].LibPnt      := notLib;
  SoldVar^[SoldPointer].PosData[1] := DataCenX;
  SoldVar^[SoldPointer].PosData[2] := DataCenY;
  SoldVar^[SoldPointer].PosData[3] := DataRadius;
  SoldVar^[SoldPointer].PosData[4] := 0;
  SoldVar^[SoldPointer].AssData[1] := 0;
  SoldVar^[SoldPointer].AssData[2] := Layer;
end;
procedure StoreSoldArcCommand;
begin
  SoldPointer := SoldPointer + 1;
  SoldVar^[SoldPointer].CdeData     := ArcCode;
  SoldVar^[SoldPointer].LibPnt      := notLib;
  SoldVar^[SoldPointer].PosData[1] := DataCenX;
  SoldVar^[SoldPointer].PosData[2] := DataCenY;
  SoldVar^[SoldPointer].PosData[3] := DataRadius;
  SoldVar^[SoldPointer].PosData[4] := StAngl;
  SoldVar^[SoldPointer].AssData[1] := EnAngl;
  SoldVar^[SoldPointer].AssData[2] := Layer;
end;
procedure WriteCommandToSoldFile(Pnt : integer);
begin
  write(SoldFile,SoldVar^[Pnt].CdeData:2);
  write(SoldFile,SoldVar^[Pnt].LibPnt:4);
  write(SoldFile,SoldVar^[Pnt].PosData[1]:6);
  write(SoldFile,SoldVar^[Pnt].PosData[2]:6);
  write(SoldFile,SoldVar^[Pnt].PosData[3]:6);
  write(SoldFile,SoldVar^[Pnt].PosData[4]:6);
  write(SoldFile,SoldVar^[Pnt].AssData[1]:4);
  writeln(SoldFile,SoldVar^[Pnt].AssData[2]:4);
end;
procedure WriteSoldToLibsFile(Pnt : integer);
var X0,X1,Y0,Y1 : integer;
begin
  if SoldVar^[Pnt].CdeData = LineCode then
  begin
    X0 := SoldVar^[Pnt].PosData[1] - MinDataX;
    Y0 := SoldVar^[Pnt].PosData[2] - MinDataY;
    X1 := SoldVar^[Pnt].PosData[3] - MinDataX;
    Y1 := SoldVar^[Pnt].PosData[4] - MinDataY;
```

```
  end
  else
  begin
    X0 := SoldVar^[Pnt].PosData[1] - MinDataX;
    Y0 := SoldVar^[Pnt].PosData[2] - MinDataY;
    X1 := SoldVar^[Pnt].PosData[3];
    Y1 := SoldVar^[Pnt].PosData[4];
  end;
  write(LibsFile,SoldVar^[Pnt].CdeData:2);
  write(LibsFile,X0:6);
  write(LibsFile,Y0:6);
  write(LibsFile,X1:6);
  write(LibsFile,Y1:6);
  write(LibsFile,SoldVar^[Pnt].AssData[1]:4);
  writeln(LibsFile,SoldVar^[Pnt].AssData[2]:4);
end;
procedure ReadSoldFromLibsFile(Pnt : integer);
begin
  Read(LibsFile,DevVar[Pnt].CdeData);
  Read(LibsFile,DevVar[Pnt].PosData[1]);
  Read(LibsFile,DevVar[Pnt].PosData[2]);
  Read(LibsFile,DevVar[Pnt].PosData[3]);
  Read(LibsFile,DevVar[Pnt].PosData[4]);
  Read(LibsFile,DevVar[Pnt].AssData[1]);
  Readln(LibsFile,DevVar[Pnt].AssData[2]);
end;
procedure ReadCommandFromSoldFile(Pnt : integer);
begin
  read(SoldFile,SoldVar^[Pnt].CdeData);
  read(SoldFile,SoldVar^[Pnt].LibPnt);
  read(SoldFile,SoldVar^[Pnt].PosData[1]);
  read(SoldFile,SoldVar^[Pnt].PosData[2]);
  read(SoldFile,SoldVar^[Pnt].PosData[3]);
  read(SoldFile,SoldVar^[Pnt].PosData[4]);
  read(SoldFile,SoldVar^[Pnt].AssData[1]);
  readln(SoldFile,SoldVar^[Pnt].AssData[2]);
end;
procedure StoreCompLineCommand;
begin
  CompPointer := CompPointer + 1;
  CompVar^[CompPointer].CdeData    := LineCode;
  CompVar^[CompPointer].LibPnt     := notLib;
  CompVar^[CompPointer].PosData[1] := DataX0;
  CompVar^[CompPointer].PosData[2] := DataY0;
  CompVar^[CompPointer].PosData[3] := DataX1;
  CompVar^[CompPointer].PosData[4] := DataY1;
  CompVar^[CompPointer].AssData[1] := Trace;
  CompVar^[CompPointer].AssData[2] := Layer;
end;
procedure StoreCompCircCommand;
begin
  CompPointer := CompPointer + 1;
  CompVar^[CompPointer].CdeData    := CircCode;
  CompVar^[CompPointer].LibPnt     := notLib;
```

```
  CompVar^[CompPointer].PosData[1] := DataCenX;
  CompVar^[CompPointer].PosData[2] := DataCenY;
  CompVar^[CompPointer].PosData[3] := DataRadius;
  CompVar^[CompPointer].PosData[4] := 0;
  CompVar^[CompPointer].AssData[1] := 0;
  CompVar^[CompPointer].AssData[2] := Layer;
end;
procedure StoreCompArcCommand;
begin
  CompPointer := CompPointer + 1;
  CompVar^[CompPointer].CdeData    := ArcCode;
  CompVar^[CompPointer].LibPnt     := notLib;
  CompVar^[CompPointer].PosData[1] := DataCenX;
  CompVar^[CompPointer].PosData[2] := DataCenY;
  CompVar^[CompPointer].PosData[3] := DataRadius;
  CompVar^[CompPointer].PosData[4] := StAngl;
  CompVar^[CompPointer].AssData[1] := EnAngl;
  CompVar^[CompPointer].AssData[2] := Layer;
end;
procedure WriteCommandToCompFile(Pnt : integer);
begin
  write(CompFile,CompVar^[Pnt].CdeData:2);
  write(CompFile,CompVar^[Pnt].LibPnt:4);
  write(CompFile,CompVar^[Pnt].PosData[1]:6);
  write(CompFile,CompVar^[Pnt].PosData[2]:6);
  write(CompFile,CompVar^[Pnt].PosData[3]:6);
  write(CompFile,CompVar^[Pnt].PosData[4]:6);
  write(CompFile,CompVar^[Pnt].AssData[1]:4);
  writeln(CompFile,CompVar^[Pnt].AssData[2]:4);
end;
procedure WriteCompToLibsFile(Pnt : integer);
var X0,X1,Y0,Y1 : integer;
begin
  if CompVar^[Pnt].CdeData = LineCode then
  begin
    X0 := CompVar^[Pnt].PosData[1] - MinDataX;
    Y0 := CompVar^[Pnt].PosData[2] - MinDataY;
    X1 := CompVar^[Pnt].PosData[3] - MinDataX;
    Y1 := CompVar^[Pnt].PosData[4] - MinDataY;
  end
  else
  begin
    X0 := CompVar^[Pnt].PosData[1] - MinDataX;
    Y0 := CompVar^[Pnt].PosData[2] - MinDataY;
    X1 := CompVar^[Pnt].PosData[3];
    Y1 := CompVar^[Pnt].PosData[4];
  end;
  write(LibsFile,CompVar^[Pnt].CdeData:2);
  write(LibsFile,X0:6);
  write(LibsFile,Y0:6);
  write(LibsFile,X1:6);
  write(LibsFile,Y1:6);
  write(LibsFile,CompVar^[Pnt].AssData[1]:4);
  writeln(LibsFile,CompVar^[Pnt].AssData[2]:4);
```

```
end;
procedure ReadCompFromLibsFile(Pnt : integer);
begin
  Read(LibsFile,DevVar[Pnt].CdeData);
  Read(LibsFile,DevVar[Pnt].PosData[1]);
  Read(LibsFile,DevVar[Pnt].PosData[2]);
  Read(LibsFile,DevVar[Pnt].PosData[3]);
  Read(LibsFile,DevVar[Pnt].PosData[4]);
  Read(LibsFile,DevVar[Pnt].AssData[1]);
  Readln(LibsFile,DevVar[Pnt].AssData[2]);
end;
procedure ReadCommandFromCompFile(Pnt : integer);
begin
  read(CompFile,CompVar^[Pnt].CdeData);
  read(CompFile,CompVar^[Pnt].LibPnt);
  read(CompFile,CompVar^[Pnt].PosData[1]);
  read(CompFile,CompVar^[Pnt].PosData[2]);
  read(CompFile,CompVar^[Pnt].PosData[3]);
  read(CompFile,CompVar^[Pnt].PosData[4]);
  read(CompFile,CompVar^[Pnt].AssData[1]);
  readln(CompFile,CompVar^[Pnt].AssData[2]);
end;
procedure StoreSilkLineCommand;
begin
  SilkPointer := SilkPointer + 1;
  SilkVar^[SilkPointer].CdeData    := LineCode;
  SilkVar^[SilkPointer].LibPnt     := notLib;
  SilkVar^[SilkPointer].PosData[1] := DataX0;
  SilkVar^[SilkPointer].PosData[2] := DataY0;
  SilkVar^[SilkPointer].PosData[3] := DataX1;
  SilkVar^[SilkPointer].PosData[4] := DataY1;
  SilkVar^[SilkPointer].AssData[1] := Trace;
  SilkVar^[SilkPointer].AssData[2] := Layer;
end;
procedure StoreSilkCircCommand;
begin
  SilkPointer := SilkPointer + 1;
  SilkVar^[SilkPointer].CdeData    := CircCode;
  SilkVar^[SilkPointer].LibPnt     := notLib;
  SilkVar^[SilkPointer].PosData[1] := DataCenX;
  SilkVar^[SilkPointer].PosData[2] := DataCenY;
  SilkVar^[SilkPointer].PosData[3] := DataRadius;
  SilkVar^[SilkPointer].PosData[4] := 0;
  SilkVar^[SilkPointer].AssData[1] := 0;
  SilkVar^[SilkPointer].AssData[2] := Layer;
end;
procedure StoreSilkArcCommand;
begin
  SilkPointer := SilkPointer + 1;
  SilkVar^[SilkPointer].CdeData    := ArcCode;
  SilkVar^[SilkPointer].LibPnt     := notLib;
  SilkVar^[SilkPointer].PosData[1] := DataCenX;
  SilkVar^[SilkPointer].PosData[2] := DataCenY;
  SilkVar^[SilkPointer].PosData[3] := DataRadius;
```

```
  SilkVar^[SilkPointer].PosData[4] := StAngl;
  SilkVar^[SilkPointer].AssData[1] := EnAngl;
  SilkVar^[SilkPointer].AssData[2] := Layer;
end;
procedure WriteCommandToSilkFile(Pnt : integer);
begin
  write(SilkFile,SilkVar^[Pnt].CdeData:2);
  write(SilkFile,SilkVar^[Pnt].LibPnt:4);
  write(SilkFile,SilkVar^[Pnt].PosData[1]:6);
  write(SilkFile,SilkVar^[Pnt].PosData[2]:6);
  write(SilkFile,SilkVar^[Pnt].PosData[3]:6);
  write(SilkFile,SilkVar^[Pnt].PosData[4]:6);
  write(SilkFile,SilkVar^[Pnt].AssData[1]:4);
  writeln(SilkFile,SilkVar^[Pnt].AssData[2]:4);
end;
procedure WriteSilkToLibsFile(Pnt : integer);
var X0,X1,Y0,Y1 : integer;
begin
  if SilkVar^[Pnt].CdeData = LineCode then
  begin
    X0 := SilkVar^[Pnt].PosData[1] - MinDataX;
    Y0 := SilkVar^[Pnt].PosData[2] - MinDataY;
    X1 := SilkVar^[Pnt].PosData[3] - MinDataX;
    Y1 := SilkVar^[Pnt].PosData[4] - MinDataY;
  end
  else
  begin
    X0 := SilkVar^[Pnt].PosData[1] - MinDataX;
    Y0 := SilkVar^[Pnt].PosData[2] - MinDataY;
    X1 := SilkVar^[Pnt].PosData[3];
    Y1 := SilkVar^[Pnt].PosData[4];
  end;
  write(LibsFile,SilkVar^[Pnt].CdeData:2);
  write(LibsFile,X0:6);
  write(LibsFile,Y0:6);
  write(LibsFile,X1:6);
  write(LibsFile,Y1:6);
  write(LibsFile,SilkVar^[Pnt].AssData[1]:4);
  writeln(LibsFile,SilkVar^[Pnt].AssData[2]:4);
end;
procedure ReadSilkFromLibsFile(Pnt : integer);
begin
  Read(LibsFile,DevVar[Pnt].CdeData);
  Read(LibsFile,DevVar[Pnt].PosData[1]);
  Read(LibsFile,DevVar[Pnt].PosData[2]);
  Read(LibsFile,DevVar[Pnt].PosData[3]);
  Read(LibsFile,DevVar[Pnt].PosData[4]);
  Read(LibsFile,DevVar[Pnt].AssData[1]);
  Readln(LibsFile,DevVar[Pnt].AssData[2]);
end;
procedure ReadCommandFromSilkFile(Pnt : integer);
begin
  read(SilkFile,SilkVar^[Pnt].CdeData);
  read(SilkFile,SilkVar^[Pnt].LibPnt);
```

```
  read(SilkFile,SilkVar^[Pnt].PosData[1]);
  read(SilkFile,SilkVar^[Pnt].PosData[2]);
  read(SilkFile,SilkVar^[Pnt].PosData[3]);
  read(SilkFile,SilkVar^[Pnt].PosData[4]);
  read(SilkFile,SilkVar^[Pnt].AssData[1]);
  readln(SilkFile,SilkVar^[Pnt].AssData[2]);
end;
procedure StorePadsCommand;
begin
  PadsPointer := PadsPointer + 1;
  PadsVar^[PadsPointer].CdeData    := PadCode;
  PadsVar^[PadsPointer].LibPnt     := notLib;
  PadsVar^[PadsPointer].PosData[1] := DataCenX;
  PadsVar^[PadsPointer].PosData[2] := DataCenY;
  PadsVar^[PadsPointer].AssData[1] := Padsize;
  PadsVar^[PadsPointer].AssData[2] := PadDir;
end;
procedure WriteCommandToPadsFile(Pnt : integer);
begin
  write(PadsFile,PadsVar^[Pnt].CdeData:2);
  write(PadsFile,PadsVar^[Pnt].LibPnt:4);
  write(PadsFile,PadsVar^[Pnt].PosData[1]:6);
  write(PadsFile,PadsVar^[Pnt].PosData[2]:6);
  write(PadsFile,PadsVar^[Pnt].AssData[1]:6);
  writeln(PadsFile,PadsVar^[Pnt].AssData[2]:6);
end;
procedure WritePadsToLibsFile(Pnt : integer);
var X0,Y0 : integer;
begin
  X0 := PadsVar^[Pnt].PosData[1] - MinDataX;
  Y0 := PadsVar^[Pnt].PosData[2] - MinDataY;
  write(LibsFile,PadsVar^[Pnt].CdeData:2);
  write(LibsFile,X0:6);
  write(LibsFile,Y0:6);
  write(LibsFile,PadsVar^[Pnt].AssData[1]:6);
  writeln(LibsFile,PadsVar^[Pnt].AssData[2]:6);
end;
procedure ReadPadsFromLibsFile(Pnt : integer);
begin
  Read(LibsFile,DevVar[Pnt].CdeData);
  Read(LibsFile,DevVar[Pnt].PosData[1]);
  Read(LibsFile,DevVar[Pnt].PosData[2]);
  Read(LibsFile,DevVar[Pnt].AssData[1]);
  Readln(LibsFile,DevVar[Pnt].AssData[2]);
end;
procedure ReadCommandFromPadsFile(Pnt : integer);
begin
  read(PadsFile,PadsVar^[Pnt].CdeData);
  read(PadsFile,PadsVar^[Pnt].LibPnt);
  read(PadsFile,PadsVar^[Pnt].PosData[1]);
  read(PadsFile,PadsVar^[Pnt].PosData[2]);
  read(PadsFile,PadsVar^[Pnt].AssData[1]);
  readln(PadsFile,PadsVar^[Pnt].AssData[2]);
end;
```

```
procedure StoreLineCommand;
begin
  Case Layer of
    Sold : StoreSoldLineCommand;
    Comp : StoreCompLineCommand;
    Silk : StoreSilkLineCommand;
  end;
end;
procedure StoreCircCommand;
begin
  Case Layer of
    Sold : StoreSoldCircCommand;
    Comp : StoreCompCircCommand;
    Silk : StoreSilkCircCommand;
  end;
end;
procedure StoreArcCommand;
begin
  Case Layer of
    Sold : StoreSoldArcCommand;
    Comp : StoreCompArcCommand;
    Silk : StoreSilkArcCommand;
  end;
end;
procedure WriteSoldFile(filename : string);
begin
   Assign(SoldFile,filename + '.SLD');
   rewrite(SoldFile);
   writeln(SoldFile,SoldPointer);
   for i := 1 to SoldPointer do
   begin
     WriteCommandToSoldFile(i);
   end;
   close(SoldFile);
end;
procedure WriteCompFile(filename : string);
begin
   Assign(CompFile,filename + '.CMP');
   rewrite(CompFile);
   writeln(CompFile,CompPointer);
   for i := 1 to CompPointer do
   begin
     WriteCommandToCompFile(i);
   end;
   close(CompFile);
end;
procedure WriteSilkFile(filename : string);
begin
   Assign(SilkFile,filename + '.SLK');
   rewrite(SilkFile);
   writeln(SilkFile,SilkPointer);
   for i := 1 to SilkPointer do
   begin
     WriteCommandToSilkFile(i);
```

```
    end;
    close(SilkFile);
end;
procedure WritePadsFile(filename : string);
begin
    Assign(PadsFile,filename + '.PAD');
    rewrite(PadsFile);
    writeln(PadsFile,PadsPointer);
    for i := 1 to PadsPointer do
    begin
      WriteCommandToPadsFile(i);
    end;
    close(PadsFile);
end;

procedure StoreDataDevLib(Pnt : integer);
begin
  LibVar^[Pnt].LibLabel.Name := LibLName;
  LibVar^[Pnt].LibCommt.Name := LibCName;
  LibVar^[Pnt].XLibScn := PositionCurX;
  LibVar^[Pnt].YLibScn := PositionCurY;
  LibVar^[Pnt].XLibRef := RefX;
  LibVar^[Pnt].YLibRef := RefY;
  LibVar^[Pnt].XLibMin := MinDataX;
  LibVar^[Pnt].YLibMin := MinDataY;
  LibVar^[Pnt].XLibMax := MaxDataX;
  LibVar^[Pnt].YLibMax := MaxDataY;
end;

procedure WriteDevLibDataToFile(Pnt : integer);
begin
  writeln(DevLibFile,LibVar^[Pnt].LibLabel.Name);
  writeln(DevLibFile,LibVar^[Pnt].LibCommt.Name);
  write   (DevLibFile,LibVar^[Pnt].XLibScn:6);
  write   (DevLibFile,LibVar^[Pnt].YLibScn:6);
  write   (DevLibFile,LibVar^[Pnt].XLibRef:6);
  write   (DevLibFile,LibVar^[Pnt].YLibRef:6);
  write   (DevLibFile,LibVar^[Pnt].XLibMin:6);
  write   (DevLibFile,LibVar^[Pnt].YLibMin:6);
  write   (DevLibFile,LibVar^[Pnt].XLibMax:6);
  writeln(DevLibFile,LibVar^[Pnt].YLibMax:6);
end;

procedure ReadDevLibDataToFile(Pnt : integer);
begin
  Readln(DevLibFile,LibVar^[Pnt].LibLabel.Name);
  Readln(DevLibFile,LibVar^[Pnt].LibCommt.Name);
  Read   (DevLibFile,LibVar^[Pnt].XLibScn);
  Read   (DevLibFile,LibVar^[Pnt].YLibScn);
  read   (DevLibFile,LibVar^[Pnt].XLibRef);
  Read   (DevLibFile,LibVar^[Pnt].YLibRef);
  Read   (DevLibFile,LibVar^[Pnt].XLibMin);
  Read   (DevLibFile,LibVar^[Pnt].YLibMin);
  Read   (DevLibFile,LibVar^[Pnt].XLibMax);
```

```
  Readln(DevLibFile,LibVar^[Pnt].YLibMax);
end;

procedure WriteUseDevLibsFile(filename : string);
begin
   Assign(DevLibFile,filename + '.DEV');
   rewrite(DevLibFile);
   writeln(DevLibFile,LibPointer);
   if LibPointer > 0 then
   begin
     for i := 1 to LibPointer do
     begin
       WriteDevLibDataToFile(i);
     end;
   end;
   close(DevLibFile);
end;

procedure ReadUseDevLibsFile(filename : string);
begin
   Assign(DevLibFile,filename + '.DEV');
   {$i-} reset(DevLibFile); {$i+}
   ok := (IOResult = 0);
   if Ok then
   begin
     readln(DevLibFile,LibPointer);
     if LibPointer > 0 then
     begin
       for i := 1 to LibPointer do
         ReadDevLibDataToFile(i);
     end;
     close(DevLibFile);
   end;
end;

procedure WriteLibsFile(filename : string);
begin
   CheckMinData;
   CheckMaxData;
   RefX := MinDataX - PositionCurX;
   RefY := MinDataY - PositionCurY;
   MaxDataX := MaxDataX - MinDataX;
   MaxDataY := MaxDataY - MinDataY;
   writeln(LibsFile,MinDataX);
   writeln(LibsFile,MinDataY);
   writeln(LibsFile,MaxDataX);
   writeln(LibsFile,MaxDataY);
   writeln(LibsFile,RefX);
   writeln(LibsFile,RefY);
   writeln(LibsFile,PadsPointer);
   for i := 1 to PadsPointer do
   begin
     WritePadsToLibsFile(i);
   end;
```

```
    writeln(LibsFile,SoldPointer);
    for i := 1 to SoldPointer do
    begin
      WriteSoldToLibsFile(i);
    end;
    writeln(LibsFile,CompPointer);
    for i := 1 to CompPointer do
    begin
      WriteCompToLibsFile(i);
    end;
    writeln(LibsFile,SilkPointer);
    for i := 1 to SilkPointer do
    begin
      WriteSilkToLibsFile(i);
    end;
    close(LibsFile);
end;

procedure MoveDataDevToTem;
var i : integer;
begin
  RefTemX     := RefX;
  RefTemY     := RefY;
  MinTemDataX := MinDataX;
  MinTemDataY := MinDataY;
  MaxTemDataX := MaxDataX;
  MaxTemDataY := MaxDataY;
  TemPointer := DevPointer;
  for i := 1 to TemPointer do
  begin
    TemVar[i].CdeData    := DevVar[i].CdeData;
    TemVar[i].PosData[1] := DevVar[i].PosData[1];
    TemVar[i].PosData[2] := DevVar[i].PosData[2];
    TemVar[i].PosData[3] := DevVar[i].PosData[3];
    TemVar[i].PosData[4] := DevVar[i].PosData[4];
    TemVar[i].AssData[1] := DevVar[i].AssData[1];
    TemVar[i].AssData[2] := DevVar[i].AssData[2];
  end;
end;

procedure MoveLibToScrnData(i : integer);
begin
    Case DevVar[i].AssData[2] of
    Sold : begin
             SoldPointer := SoldPointer + 1;
             SoldVar^[SoldPointer].CdeData    := DevVar[i].CdeData;
             SoldVar^[SoldPointer].LibPnt     := LibPointer;
             if DevVar[i].CdeData = LineCode then
             begin
                SoldVar^[SoldPointer].PosData[1] := PositionCurX+RefX+DevVar[i].PosData[1];
                SoldVar^[SoldPointer].PosData[2] := PositionCurY+RefY+DevVar[i].PosData[2];
                SoldVar^[SoldPointer].PosData[3] := PositionCurX+RefX+DevVar[i].PosData[3];
                SoldVar^[SoldPointer].PosData[4] := PositionCurY+RefY+DevVar[i].PosData[4];
             end
```

```
            else
            begin
               SoldVar^[SoldPointer].PosData[1] := PositionCurX+RefX+DevVar[i].PosData[1];
               SoldVar^[SoldPointer].PosData[2] := PositionCurY+RefY+DevVar[i].PosData[2];
               SoldVar^[SoldPointer].PosData[3] := DevVar[i].PosData[3];
               SoldVar^[SoldPointer].PosData[4] := DevVar[i].PosData[4];
            end;
            SoldVar^[SoldPointer].AssData[1] := DevVar[i].AssData[1];
            SoldVar^[SoldPointer].AssData[2] := DevVar[i].AssData[2];
          end;
    Comp : begin
            CompPointer := CompPointer + 1;
            CompVar^[CompPointer].CdeData    := DevVar[i].CdeData;
            CompVar^[CompPointer].LibPnt     := LibPointer;
            if DevVar[i].CdeData = LineCode then
            begin
               CompVar^[CompPointer].PosData[1] := PositionCurX+RefX+DevVar[i].PosData[1];
               CompVar^[CompPointer].PosData[2] := PositionCurY+RefY+DevVar[i].PosData[2];
               CompVar^[CompPointer].PosData[3] := PositionCurX+RefX+DevVar[i].PosData[3];
               CompVar^[CompPointer].PosData[4] := PositionCurY+RefY+DevVar[i].PosData[4];
            end
            else
            begin
               CompVar^[CompPointer].PosData[1] := PositionCurX+RefX+DevVar[i].PosData[1];
               CompVar^[CompPointer].PosData[2] := PositionCurY+RefY+DevVar[i].PosData[2];
               CompVar^[CompPointer].PosData[3] := DevVar[i].PosData[3];
               CompVar^[CompPointer].PosData[4] := DevVar[i].PosData[4];
            end;
            CompVar^[CompPointer].AssData[1] := DevVar[i].AssData[1];
            CompVar^[CompPointer].AssData[2] := DevVar[i].AssData[2];
          end;
    Silk : begin
            SilkPointer := SilkPointer + 1;
            SilkVar^[SilkPointer].CdeData    := DevVar[i].CdeData;
            SilkVar^[SilkPointer].LibPnt     := LibPointer;
            if DevVar[i].CdeData = LineCode then
            begin
               SilkVar^[SilkPointer].PosData[1] := PositionCurX+RefX+DevVar[i].PosData[1];
               SilkVar^[SilkPointer].PosData[2] := PositionCurY+RefY+DevVar[i].PosData[2];
               SilkVar^[SilkPointer].PosData[3] := PositionCurX+RefX+DevVar[i].PosData[3];
               SilkVar^[SilkPointer].PosData[4] := PositionCurY+RefY+DevVar[i].PosData[4];
            end
            else
            begin
               SilkVar^[SilkPointer].PosData[1] := PositionCurX+RefX+DevVar[i].PosData[1];
               SilkVar^[SilkPointer].PosData[2] := PositionCurY+RefY+DevVar[i].PosData[2];
               SilkVar^[SilkPointer].PosData[3] := DevVar[i].PosData[3];
               SilkVar^[SilkPointer].PosData[4] := DevVar[i].PosData[4];
            end;
            SilkVar^[SilkPointer].AssData[1] := DevVar[i].AssData[1];
            SilkVar^[SilkPointer].AssData[2] := DevVar[i].AssData[2];
          end;
    end;
end;
```

```
procedure MoveLibsDataToScrData;
begin
  for i := 1 to DevPointer do
  begin
    case DevVar[i].CdeData of
    PadCode : begin
            PadsPointer := PadsPointer + 1;
            PadsVar^[PadsPointer].CdeData    := DevVar[i].CdeData;
            PadsVar^[PadsPointer].LibPnt     := LibPointer;
            PadsVar^[PadsPointer].PosData[1] := PositionCurX+RefX+DevVar[i].PosData[1];
            PadsVar^[PadsPointer].PosData[2] := PositionCurY+RefY+DevVar[i].PosData[2];
            PadsVar^[PadsPointer].AssData[1] := DevVar[i].AssData[1];
            PadsVar^[PadsPointer].AssData[2] := DevVar[i].AssData[2];
          end;
    else  MoveLibToScrnData(i);
    end;
  end;
end;

procedure ReadLibsFile(filename : string);
var DPnt : integer;
begin
   DevPointer := 0;
   Readln(LibsFile,MinDataX);
   Readln(LibsFile,MinDataY);
   Readln(LibsFile,MaxDataX);
   Readln(LibsFile,MaxDataY);
   Readln(LibsFile,RefX);
   Readln(LibsFile,RefY);
   Readln(LibsFile,DPnt);
   for i := 1 to DPnt do
   begin
     ReadPadsFromLibsFile(DevPointer+i);
   end;
   DevPointer := DevPointer + DPnt;
   Readln(LibsFile,DPnt);
   for i := 1 to DPnt do
   begin
     ReadSoldFromLibsFile(i);
   end;
   DevPointer := DevPointer + DPnt;
   Readln(LibsFile,DPnt);
   for i := 1 to Dpnt do
   begin
     ReadCompFromLibsFile(DevPointer+i);
   end;
   DevPointer := DevPointer + DPnt;
   Readln(LibsFile,DPnt);
   for i := 1 to DPnt do
   begin
     ReadSilkFromLibsFile(DevPointer+i);
   end;
   DevPointer := DevPointer + DPnt;
```

```
   MoveDataDevToTem;
   close(LibsFile);
end;

procedure ReadTextData(i : integer);
begin
  read  (TextDtaFile,TextVar^[i].TxtDir);
  read  (TextDtaFile,TextVar^[i].TxtLyr);
  read  (TextDtaFile,TextVar^[i].PosData[1]);
  readln(TextDtaFile,TextVar^[i].PosData[2]);
  readln(TextDtaFile,TextVar^[i].Message);
end;

procedure WriteTextData(i : integer);
begin
  write  (TextDtaFile,TextVar^[i].TxtDir:2);
  write  (TextDtaFile,TextVar^[i].TxtLyr:2);
  write  (TextDtaFile,TextVar^[i].PosData[1]:6);
  writeln(TextDtaFile,TextVar^[i].PosData[2]:6);
  writeln(TextDtaFile,TextVar^[i].Message);
end;

procedure ReadTextFile(filename : string);
begin
   Assign(TextDtaFile,filename + '.MSG');
   {$i-} reset(TextDtaFile); {$i+}
   ok := (IOResult = 0);
   if Ok then
   begin
     Readln(TextDtaFile,TextPointer);
     for i := 1 to TextPointer do
       ReadTextData(i);
     close(TextDtaFile);
   end;
end;

procedure WriteTextFile(filename : string);
begin
   Assign(TextDtaFile,filename + '.MSG');
   rewrite(TextDtaFile);
   writeln(TextDtaFile,TextPointer);
   for i := 1 to TextPointer do
     WriteTextData(i);
   close(TextDtaFile);
end;

procedure WriteCommandToFile (filename : string);
begin
   WriteSoldFile(Filename);
   WriteCompFile(filename);
   WriteSilkFile(filename);
   WritePadsFile(filename);
   WriteTextFile(filename);
   WriteUseDevLibsFile(filename);
```

```
end;

procedure ReadSoldFile(filename : string);
begin
   Assign(SoldFile,filename + '.SLD');
   {$i-} reset(SoldFile); {$i+}
   ok := (IOResult = 0);
   if Ok then
   begin
     Readln(SoldFile,SoldPointer);
     for i := 1 to SoldPointer do
       ReadCommandFromSoldFile(i);
     close(SoldFile);
   end;
end;
procedure ReadCompFile(filename : string);
begin
   Assign(CompFile,filename + '.CMP');
   {$i-} reset(CompFile); {$i+}
   ok := (IOResult = 0);
   if Ok then
   begin
     Readln(CompFile,CompPointer);
     for i := 1 to CompPointer do
       ReadCommandFromCompFile(i);
     close(CompFile);
   end;
end;
procedure ReadSilkFile(filename : string);
begin
   Assign(SilkFile,filename + '.SLK');
   {$i-} reset(SilkFile); {$i+}
   ok := (IOResult = 0);
   if Ok then
   begin
     Readln(SilkFile,SilkPointer);
     for i := 1 to SilkPointer do
       ReadCommandFromSilkFile(i);
     close(SilkFile);
   end;
end;
procedure ReadPadsFile(filename : string);
begin
   Assign(PadsFile,filename + '.PAD');
   {$i-} reset(PadsFile); {$i+}
   ok := (IOResult = 0);
   if Ok then
   begin
     Readln(PadsFile,PadsPointer);
     for i := 1 to PadsPointer do
       ReadCommandFromPadsFile(i);
     close(PadsFile);
   end;
end;
```

```
procedure ReadCommandFromFile (filename : string);
begin
   ReadSoldFile(Filename);
   ReadCompFile(filename);
   ReadSilkFile(filename);
   ReadPadsFile(filename);
   ReadTextFile(filename);
   ReadUseDevLibsFile(filename);
end;

procedure ReadTextCommandFile;
var StrBuff : string[3];
    StrCh   : String[1];
    ok      : boolean;
begin
   Assign(TextComFile,'PCBCAD.CHR');
   {$i-} reset(TextComFile); {$i+}
   ok := (IOResult = 0);
   if ok then
   begin
     Readln(TextComFile,TxtNum);
     for p := 1 to TxtNum do
     begin
       ReadLn(TextComFile,TxtComData[p].TComNum);
       ReadLn(TextComFile,TxtComData[p].KeyChar);
       {writeln(TxtComData[p].KeyChar);} {Display for Testing}
       for i := 1 to TxtComData[p].TComNum do
       begin
         Readln(TextComFile,TxtComData[p].TxtCom[i].DataX0,
                            TxtComData[p].TxtCom[i].DataY0,
                            TxtComData[p].TxtCom[i].DataX1,
                            TxtComData[p].TxtCom[i].DataY1);
       end;
     end;
     close(TextComFile);
   end;
end;

procedure RestoreText(TNum: integer);
begin
   TextVar^[TNum].TxtDir     := TdirTemp;
   TextVar^[TNum].TxtLyr     := TLyrTemp;
   TextVar^[TNum].Message    := StrMessg;
   TextVar^[TNum].PosData[1] := TextPosX;
   TextVar^[TNum].PosData[2] := TextPosY;
end;

procedure StoreTextCommand(TDir,Lyr: byte; X,Y : integer; str: string);
begin
  TextPointer := TextPointer + 1;
  TextVar^[TextPointer].TxtDir  := TDir;
  TextVar^[TextPointer].TxtLyr  := Lyr;
```

```
  TextVar^[TextPointer].PosData[1] := X;
  TextVar^[TextPointer].PosData[2] := Y;
  TextVar^[TextPointer].Message := Str;
end;

procedure TextArea(Tnum : integer; var X0,Y0,X1,Y1 : integer);
begin
   Case TextVar^[Tnum].TxtDir of
   1 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 - 80;
         Y1 := Y0 - Length(TextVar^[Tnum].Message) * 100;
       end;
   2 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 + 80;
         Y1 := Y0 + Length(TextVar^[Tnum].Message) * 100;
       end;
   3 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 + Length(TextVar^[Tnum].Message) * 100;
         Y1 := Y0 - 80;
       end;
   4 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 - Length(TextVar^[Tnum].Message) * 100;
         Y1 := Y0 + 80;
       end;
   5 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 + 80;
         Y1 := Y0 - Length(TextVar^[Tnum].Message) * 100;
       end;
   6 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 - 80;
         Y1 := Y0 + Length(TextVar^[Tnum].Message) * 100;
       end;
   7 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 + Length(TextVar^[Tnum].Message) * 100;
         Y1 := Y0 + 80;
       end;
   8 : begin
         X0 := TextVar^[Tnum].PosData[1];
         Y0 := TextVar^[Tnum].PosData[2];
         X1 := X0 - Length(TextVar^[Tnum].Message) * 100;
```

```
        Y1 := Y0 - 80;
      end;
   end;
end;

begin

end.
```

```
Program : PCB Part VI
Programmer : Suttinun Poramatikul

{
              +-----------------------------------+
              |                                   |
              | Unit  : PCB-CAD Unit IV           |
              | Programmer : Suttinun Poramatikul |
              | Version    : 1.00A                |
              | Last Updated : 12 Jun 1988        |
              |                                   |
              +-----------------------------------+

}

unit PCBUnit6;

interface

uses Declare,PCBUnit4;

function  CheckCurInBox(X,Y,X0,Y0,X1,Y1 : integer) : byte;

procedure CompressPadsData(SPoint : integer);

procedure CompressTextData(SPoint : integer);

procedure CompressSoldData(SPoint : integer);

procedure CheckSoldCirData(X,Y : integer;var PNum,Err : integer);

procedure CompressCompData(SPoint : integer);

procedure CheckCompCirData(X,Y : integer;var PNum,Err : integer);

procedure CompressSilkData(SPoint : integer);

procedure CheckSilkCirData(X,Y : integer;var PNum,Err : integer);

procedure CheckLibExist(X,Y : integer);

procedure DeletePadsData(X,Y : integer;var PNum,Err : integer);

procedure DeleteSoldLineData(X,Y : integer;var LineOk,LineErr : boolean);

procedure DelSoldLineActiveOff;
```

```
procedure EnterNewSoldLine(PCX,PCY : integer);

procedure MoveDelSoldLineOn(PCX,PCY : integer);

procedure EnterDelSoldLine;

procedure DeleteCompLineData(X,Y : integer;var LineOk,LineErr : boolean);

procedure DelCompLineActiveOff;

procedure EnterNewCompLine(PCX,PCY : integer);

procedure MoveDelCompLineOn(PCX,PCY : integer);

procedure EnterDelCompLine;

procedure DeleteSilkLineData(X,Y : integer;var LineOk,LineErr : boolean);

procedure DelSilkLineActiveOff;

procedure EnterNewSilkLine(PCX,PCY : integer);

procedure MoveDelSilkLineOn(PCX,PCY : integer);

procedure EnterDelSilkLine;

procedure DelDevLib(PntS : integer);

procedure MoveDelLineOn(PCX,PCY : integer);

procedure RestoreLibDataToDevData(LibNum:integer);

implementation

var  DelPnt : integer;
     DelArr : array [1..200] of integer;


procedure CompressPadsData(SPoint : integer);
var i : integer;
begin
  for i := SPoint to PadsPointer - 1 do
  begin
    PadsVar^[i].CdeData    := PadsVar^[i+1].CdeData;
    PadsVar^[i].LibPnt     := PadsVar^[i+1].LibPnt;
    PadsVar^[i].PosData[1] := PadsVar^[i+1].PosData[1];
    PadsVar^[i].PosData[2] := PadsVar^[i+1].PosData[2];
    PadsVar^[i].AssData[1] := PadsVar^[i+1].AssData[1];
    PadsVar^[i].AssData[2] := PadsVar^[i+1].AssData[2];
  end;
  PadsPointer := PadsPointer - 1;
end;

procedure CompressTextData(SPoint : integer);
```

```
var i : integer;
begin
  for i := SPoint to TextPointer - 1 do
  begin
    TextVar^[i].TxtDir := TextVar^[i+1].TxtDir;
    TextVar^[i].TxtLyr := TextVar^[i+1].TxtLyr;
    TextVar^[i].PosData[1] := TextVar^[i+1].PosData[1];
    TextVar^[i].PosData[2] := TextVar^[i+1].PosData[2];
    TextVar^[i].Message := TextVar^[i+1].Message;
  end;
  TextPointer := TextPointer - 1;
end;

procedure CompressSoldData(SPoint : integer);
var i : integer;
begin
  for i := SPoint to SoldPointer - 1 do
  begin
    SoldVar^[i].CdeData    := SoldVar^[i+1].CdeData;
    SoldVar^[i].LibPnt     := SoldVar^[i+1].LibPnt;
    SoldVar^[i].PosData[1] := SoldVar^[i+1].PosData[1];
    SoldVar^[i].PosData[2] := SoldVar^[i+1].PosData[2];
    SoldVar^[i].PosData[3] := SoldVar^[i+1].PosData[3];
    SoldVar^[i].PosData[4] := SoldVar^[i+1].PosData[4];
    SoldVar^[i].AssData[1] := SoldVar^[i+1].AssData[1];
    SoldVar^[i].AssData[2] := SoldVar^[i+1].AssData[2];
  end;
  SoldPointer := SoldPointer - 1;
end;

procedure CompressCompData(SPoint : integer);
var i : integer;
begin
  for i := SPoint to CompPointer - 1 do
  begin
    CompVar^[i].CdeData    := CompVar^[i+1].CdeData;
    CompVar^[i].LibPnt     := CompVar^[i+1].LibPnt;
    CompVar^[i].PosData[1] := CompVar^[i+1].PosData[1];
    CompVar^[i].PosData[2] := CompVar^[i+1].PosData[2];
    CompVar^[i].PosData[3] := CompVar^[i+1].PosData[3];
    CompVar^[i].PosData[4] := CompVar^[i+1].PosData[4];
    CompVar^[i].AssData[1] := CompVar^[i+1].AssData[1];
    CompVar^[i].AssData[2] := CompVar^[i+1].AssData[2];
  end;
  CompPointer := CompPointer - 1;
end;

procedure CompressSilkData(SPoint : integer);
var i : integer;
begin
  for i := SPoint to SilkPointer - 1 do
  begin
    SilkVar^[i].CdeData    := SilkVar^[i+1].CdeData;
    SilkVar^[i].LibPnt     := SilkVar^[i+1].LibPnt;
```

```
    SilkVar^[i].PosData[1] := SilkVar^[i+1].PosData[1];
    SilkVar^[i].PosData[2] := SilkVar^[i+1].PosData[2];
    SilkVar^[i].PosData[3] := SilkVar^[i+1].PosData[3];
    SilkVar^[i].PosData[4] := SilkVar^[i+1].PosData[4];
    SilkVar^[i].AssData[1] := SilkVar^[i+1].AssData[1];
    SilkVar^[i].AssData[2] := SilkVar^[i+1].AssData[2];
  end;
  SilkPointer := SilkPointer - 1;
end;

procedure DeletePadsData(X,Y : integer;var PNum,Err : integer);
var i : integer;
begin
  Err := 2;
  for i := 1 to PadsPointer do
  begin
    if PadsVar^[i].PosData[1] = X then
    if PadsVar^[i].PosData[2] = Y then
    begin
      PNum := i;
      if PadsVar^[i].LibPnt = notLib then
        Err := 0
      else Err := 1;
    end;
  end; {for}
end;

procedure CheckSoldCirData(X,Y : integer;var PNum,Err : integer);
var i : integer;
begin
  Err := 2;
  for i := 1 to SoldPointer do
  begin
    if SoldVar^[i].CdeData in [CircCode,ArcCode] then
    begin
      if SoldVar^[i].PosData[1] = X then
      if SoldVar^[i].PosData[2] = Y then
      begin
        PNum := i;
        if SoldVar^[i].LibPnt = notLib then Err := 0
        else Err := 1;
      end;
    end;
  end; {for}
end;

procedure CheckCompCirData(X,Y : integer;var PNum,Err : integer);
var i : integer;
begin
  Err := 2;
  for i := 1 to CompPointer do
  begin
    if CompVar^[i].CdeData in [CircCode,ArcCode] then
    begin
```

```
      if CompVar^[i].PosData[1] = X then
      if CompVar^[i].PosData[2] = Y then
      begin
        PNum := i;
        if CompVar^[i].LibPnt = notLib then Err := 0
        else Err := 1;
      end;
    end;
  end; {for}
end;

procedure CheckSilkCirData(X,Y : integer;var PNum,Err : integer);
var i : integer;
begin
  Err := 2;
  for i := 1 to SilkPointer do
  begin
    if SilkVar^[i].CdeData in [CircCode,ArcCode] then
    begin
      if SilkVar^[i].PosData[1] = X then
      if SilkVar^[i].PosData[2] = Y then
      begin
        PNum := i;
        if SilkVar^[i].LibPnt = notLib then Err := 0
        else Err := 1;
      end;
    end;
  end; {for}
end;

procedure SwapData(var Xs,Xe : integer);
var TempX : integer;
begin
  if Xs > Xe then
  begin
    TempX := Xs;
    Xs := Xe;
    Xe := TempX;
  end;
end;

function CheckCurInBox(X,Y,X0,Y0,X1,Y1 : integer) : byte;
begin
  CheckCurInBox := 0;
  SwapData(X0,X1);
  SwapData(Y0,Y1);
  if ((X0 <= X) and (Y0 <= Y)) and ((X1 >= X) and (Y1 >= Y)) then
  begin
    CheckCurInBox := 1;
  end
end;

procedure CheckLibExist(X,Y : integer);
var X0,Y0,X1,Y1,i : integer;
```

```
begin
  CheckLibOk := false;
  if LibPointer > 0 then
  begin
    for i := 1 to LibPointer do
    begin
      X0 := LibVar^[i].XLibScn + LibVar^[i].XLibRef;
      Y0 := LibVar^[i].YLibScn + LibVar^[i].YLibRef;
      X1 := LibVar^[i].XLibScn + LibVar^[i].XLibRef + LibVar^[i].XLibMax;
      Y1 := LibVar^[i].YLibScn + LibVar^[i].YLibRef + LibVar^[i].YLibMax;
      if CheckCurInBox(X,Y,X0,Y0,X1,Y1) = 1 then
      begin
        LibActNum := i;
        CheckLibOk := true;
        write(^g);
      end;
    end;
  end;
end;

function CheckXYInLine(X,Y,X0,Y0,X1,Y1 : integer) : byte;
begin
  CheckXYInLine := 0;
  if (X0 = X) and (Y0 = Y) then
  begin
    CheckXYInLine := 1;
  end
  else {1}
  begin
    if (X1 = X) and (Y1 = Y) then
    begin
      CheckXYInLine := 2;
    end
    else
    begin {2}
      SwapData(X0,X1);
      if (X0 = X1) or (Y0 = Y1) then
      begin
        if (((X0 < X) and (X < X1)) and ((Y0 = Y) or (Y1 = Y))) then
        begin
          CheckXYInLine := 3;
        end
        else
        begin {3}
          SwapData(Y0,Y1);
          if (((Y0 < Y) and (Y < Y1)) and ((X0 = X) or (X1 = X))) then
          begin
            CheckXYInLine := 4;
          end;
        end;  {3}
      end;
    end;  {2}
  end; {1}
end;
```

```
procedure DeleteSoldLineData(X,Y : integer;var LineOk,LineErr : boolean);
var i,PcL : integer;
    X0,Y0,X1,Y1,TempX,TempY : integer;
begin
  RootLcnt := 0;
  LineErr  := false;
  LineOk   := false;
  for i := 1 to SoldPointer do
  begin
    if SoldVar^[i].CdeData = LineCode then
    begin
      X0 := SoldVar^[i].PosData[1];
      Y0 := SoldVar^[i].PosData[2];
      X1 := SoldVar^[i].PosData[3];
      Y1 := SoldVar^[i].PosData[4];
      PcL := CheckXYInLine(X,Y,X0,Y0,X1,Y1);
      if PcL in [1,2,3,4] then
      begin
        if SoldVar^[i].LibPnt = notLib then
        begin
          RootLcnt := RootLcnt + 1;
          LineOk := true;
          POnCur[RootLcnt] := PcL;
          RootLine[RootLcnt] := i;
          if PcL = 3 then begin
                            LX0[RootLcnt] := X0-X;
                            LX1[RootLcnt] := X1-X;
                          end;
          if PcL = 4 then begin
                            LY0[RootLcnt] := Y0-Y;
                            LY1[RootLcnt] := Y1-Y;
                          end;
        end;
        {Check for Lib-Line if data exit then LineErr = true}
      end;
    end; {if LineCode}
  end;
end;

procedure DeleteCompLineData(X,Y : integer;var LineOk,LineErr : boolean);
var i,PcL : integer;
    X0,Y0,X1,Y1,TempX,TempY : integer;
begin
  RootLcnt := 0;
  LineErr  := false;
  LineOk   := false;
  for i := 1 to CompPointer do
  begin
    if CompVar^[i].CdeData = LineCode then
    begin
      X0 := CompVar^[i].PosData[1];
      Y0 := CompVar^[i].PosData[2];
      X1 := CompVar^[i].PosData[3];
```

```
      Y1 := CompVar^[i].PosData[4];
      PcL := CheckXYInLine(X,Y,X0,Y0,X1,Y1);
      if PcL in [1,2,3,4] then
      begin
        if CompVar^[i].LibPnt = notLib then
        begin
          RootLcnt := RootLcnt + 1;
          LineOk := true;
          POnCur[RootLcnt] := PcL;
          RootLine[RootLcnt] := i;
          if PcL = 3 then begin
                           LX0[RootLcnt] := X0-X;
                           LX1[RootLcnt] := X1-X;
                          end;
          if PcL = 4 then begin
                           LY0[RootLcnt] := Y0-Y;
                           LY1[RootLcnt] := Y1-Y;
                          end;
        end;
        {Check for Lib-Line if data exit then LineErr = true}
      end;
    end; {if LineCode}
  end;
end;

procedure DeleteSilkLineData(X,Y : integer;var LineOk,LineErr : boolean);
var i,PcL : integer;
    X0,Y0,X1,Y1,TempX,TempY : integer;
begin
  RootLcnt := 0;
  LineErr  := false;
  LineOk   := false;
  for i := 1 to SilkPointer do
  begin
    if SilkVar^[i].CdeData = LineCode then
    begin
      X0 := SilkVar^[i].PosData[1];
      Y0 := SilkVar^[i].PosData[2];
      X1 := SilkVar^[i].PosData[3];
      Y1 := SilkVar^[i].PosData[4];
      PcL := CheckXYInLine(X,Y,X0,Y0,X1,Y1);
      if PcL in [1,2,3,4] then
      begin
        if SilkVar^[i].LibPnt = notLib then
        begin
          RootLcnt := RootLcnt + 1;
          LineOk := true;
          POnCur[RootLcnt] := PcL;
          RootLine[RootLcnt] := i;
          if PcL = 3 then begin
                           LX0[RootLcnt] := X0-X;
                           LX1[RootLcnt] := X1-X;
                          end;
          if PcL = 4 then begin
```

```
                              LY0[RootLcnt] := Y0-Y;
                              LY1[RootLcnt] := Y1-Y;
                            end;
          end;
          {Check for Lib-Line if data exit then LineErr = true}
        end;
      end; {if LineCode}
    end;
  end;

  procedure DelSoldLineActiveOff;
  var X0,Y0,X1,Y1,p,T : integer;
  begin
    if LineOk then
    for i := 1 to RootLcnt do
    begin
      p  := RootLine[i];
      X0 := SoldVar^[p].PosData[1];
      Y0 := SoldVar^[p].PosData[2];
      X1 := SoldVar^[p].PosData[3];
      Y1 := SoldVar^[p].PosData[4];
      T  := SoldVar^[p].AssData[1];
      DelWire(X0,Y0,X1,Y1,T);
    end;
  end;

  procedure DelCompLineActiveOff;
  var X0,Y0,X1,Y1,p,T : integer;
  begin
    if LineOk then
    for i := 1 to RootLcnt do
    begin
      p  := RootLine[i];
      X0 := CompVar^[p].PosData[1];
      Y0 := CompVar^[p].PosData[2];
      X1 := CompVar^[p].PosData[3];
      Y1 := CompVar^[p].PosData[4];
      T  := CompVar^[p].AssData[1];
      DelWire(X0,Y0,X1,Y1,T);
    end;
  end;

  procedure DelSilkLineActiveOff;
  var X0,Y0,X1,Y1,p,T : integer;
  begin
    if LineOk then
    for i := 1 to RootLcnt do
    begin
      p  := RootLine[i];
      X0 := SilkVar^[p].PosData[1];
      Y0 := SilkVar^[p].PosData[2];
      X1 := SilkVar^[p].PosData[3];
      Y1 := SilkVar^[p].PosData[4];
      T  := SilkVar^[p].AssData[1];
```

```
    DelWire(X0,Y0,X1,Y1,T);
  end;
end;

procedure MoveDelSoldLineOn(PCX,PCY : integer);
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  for i := 1 to RootLcnt do
  begin
    p  := RootLine[i];
    X0 := SoldVar^[p].PosData[1];
    Y0 := SoldVar^[p].PosData[2];
    X1 := SoldVar^[p].PosData[3];
    Y1 := SoldVar^[p].PosData[4];
    if POnCur[i] = 1 then CursorLine(PCX,PCY,X1,Y1);
    if POnCur[i] = 2 then CursorLine(X0,Y0,PCX,PCY);
    if POnCur[i] = 3 then CursorLine(PCX+LX0[i],PCY,PCX+LX1[i],PCY);
    if POnCur[i] = 4 then CursorLine(PCX,PCY+LY0[i],PCX,PCY+LY1[i]);
  end;
end;

procedure MoveDelCompLineOn(PCX,PCY : integer);
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  for i := 1 to RootLcnt do
  begin
    p  := RootLine[i];
    X0 := CompVar^[p].PosData[1];
    Y0 := CompVar^[p].PosData[2];
    X1 := CompVar^[p].PosData[3];
    Y1 := CompVar^[p].PosData[4];
    if POnCur[i] = 1 then CursorLine(PCX,PCY,X1,Y1);
    if POnCur[i] = 2 then CursorLine(X0,Y0,PCX,PCY);
    if POnCur[i] = 3 then CursorLine(PCX+LX0[i],PCY,PCX+LX1[i],PCY);
    if POnCur[i] = 4 then CursorLine(PCX,PCY+LY0[i],PCX,PCY+LY1[i]);
  end;
end;

procedure MoveDelSilkLineOn(PCX,PCY : integer);
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  for i := 1 to RootLcnt do
  begin
    p  := RootLine[i];
    X0 := SilkVar^[p].PosData[1];
    Y0 := SilkVar^[p].PosData[2];
    X1 := SilkVar^[p].PosData[3];
    Y1 := SilkVar^[p].PosData[4];
    if POnCur[i] = 1 then CursorLine(PCX,PCY,X1,Y1);
    if POnCur[i] = 2 then CursorLine(X0,Y0,PCX,PCY);
    if POnCur[i] = 3 then CursorLine(PCX+LX0[i],PCY,PCX+LX1[i],PCY);
```

```
    if POnCur[i] = 4 then CursorLine(PCX,PCY+LY0[i],PCX,PCY+LY1[i]);
  end;
end;

procedure MoveDelLineOn(PCX,PCY : integer);
begin
  case Layer of
  Sold : MoveDelSoldLineOn(PCX,PCY);
  Comp : MoveDelCompLineOn(PCX,PCY);
  Silk : MoveDelSilkLineOn(PCX,PCY);
  end;
end;
procedure EnterNewSoldLine(PCX,PCY : integer);
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  for i := 1 to RootLcnt do
  begin
    p  := RootLine[i];
    X0 := SoldVar^[p].PosData[1];
    Y0 := SoldVar^[p].PosData[2];
    X1 := SoldVar^[p].PosData[3];
    Y1 := SoldVar^[p].PosData[4];
    if POnCur[i] = 1 then begin
                            SoldVar^[p].PosData[1] := PCX;
                            SoldVar^[p].PosData[2] := PCY;
                          end;
    if POnCur[i] = 2 then begin
                            SoldVar^[p].PosData[3] := PCX;
                            SoldVar^[p].PosData[4] := PCY;
                          end;
    if POnCur[i] = 3 then begin
                            SoldVar^[p].PosData[1] := PCX+LX0[i];
                            SoldVar^[p].PosData[2] := PCY;
                            SoldVar^[p].PosData[3] := PCX+LX1[i];
                            SoldVar^[p].PosData[4] := PCY;
                          end;
    if POnCur[i] = 4 then begin
                            SoldVar^[p].PosData[1] := PCX;
                            SoldVar^[p].PosData[2] := PCY+LY0[i];
                            SoldVar^[p].PosData[3] := PCX;
                            SoldVar^[p].PosData[4] := PCY+LY1[i];
                          end;
     DrawNewWire(SoldVar^[p].PosData[1],SoldVar^[p].PosData[2],
                 SoldVar^[p].PosData[3],SoldVar^[p].PosData[4],
                 SoldVar^[p].AssData[1]);
  end;
end;

procedure EnterNewCompLine(PCX,PCY : integer);
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  for i := 1 to RootLcnt do
```

```
  begin
    p  := RootLine[i];
    X0 := CompVar^[p].PosData[1];
    Y0 := CompVar^[p].PosData[2];
    X1 := CompVar^[p].PosData[3];
    Y1 := CompVar^[p].PosData[4];
    if POnCur[i] = 1 then begin
                            CompVar^[p].PosData[1] := PCX;
                            CompVar^[p].PosData[2] := PCY;
                          end;
    if POnCur[i] = 2 then begin
                            CompVar^[p].PosData[3] := PCX;
                            CompVar^[p].PosData[4] := PCY;
                          end;
    if POnCur[i] = 3 then begin
                            CompVar^[p].PosData[1] := PCX+LX0[i];
                            CompVar^[p].PosData[2] := PCY;
                            CompVar^[p].PosData[3] := PCX+LX1[i];
                            CompVar^[p].PosData[4] := PCY;
                          end;
    if POnCur[i] = 4 then begin
                            CompVar^[p].PosData[1] := PCX;
                            CompVar^[p].PosData[2] := PCY+LY0[i];
                            CompVar^[p].PosData[3] := PCX;
                            CompVar^[p].PosData[4] := PCY+LY1[i];
                          end;
     DrawNewWire(CompVar^[p].PosData[1],CompVar^[p].PosData[2],
                 CompVar^[p].PosData[3],CompVar^[p].PosData[4],
                 CompVar^[p].AssData[1]);

  end;
end;

procedure EnterNewSilkLine(PCX,PCY : integer);
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  for i := 1 to RootLcnt do
  begin
    p  := RootLine[i];
    X0 := SilkVar^[p].PosData[1];
    Y0 := SilkVar^[p].PosData[2];
    X1 := SilkVar^[p].PosData[3];
    Y1 := SilkVar^[p].PosData[4];
    if POnCur[i] = 1 then begin
                            SilkVar^[p].PosData[1] := PCX;
                            SilkVar^[p].PosData[2] := PCY;
                          end;
    if POnCur[i] = 2 then begin
                            SilkVar^[p].PosData[3] := PCX;
                            SilkVar^[p].PosData[4] := PCY;
                          end;
    if POnCur[i] = 3 then begin
                            SilkVar^[p].PosData[1] := PCX+LX0[i];
                            SilkVar^[p].PosData[2] := PCY;
```

```
                              SilkVar^[p].PosData[3] := PCX+LX1[i];
                              SilkVar^[p].PosData[4] := PCY;
                            end;
    if POnCur[i] = 4 then begin
                              SilkVar^[p].PosData[1] := PCX;
                              SilkVar^[p].PosData[2] := PCY+LY0[i];
                              SilkVar^[p].PosData[3] := PCX;
                              SilkVar^[p].PosData[4] := PCY+LY1[i];
                            end;
     DrawNewWire(SilkVar^[p].PosData[1],SilkVar^[p].PosData[2],
                 SilkVar^[p].PosData[3],SilkVar^[p].PosData[4],
                 SilkVar^[p].AssData[1]);
  end;
end;

procedure SortMaxforRoot;
var Ok : boolean;
    RootTemp : integer;
begin
  repeat
   Ok := true;
   for i := 1 to RootLcnt-1 Do
   begin
     if RootLine[i] < RootLine[i+1] then
     begin
       RootTemp := RootLine[i];
       RootLine[i] := RootLine[i+1];
       RootLine[i+1] := RootTemp;
       Ok := false;
     end;
   end;
  until Ok;
end;

procedure EnterDelSoldLine;
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  begin
    SortMaxforRoot;
    for i := 1 to RootLcnt do
    begin
      p := RootLine[i];
      CompressSoldData(p);
    end;
  end;
end;

procedure EnterDelCompLine;
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  begin
    SortMaxforRoot;
```

```
    for i := 1 to RootLcnt do
    begin
      p := RootLine[i];
      CompressCompData(p);
    end;
  end;
end;

procedure EnterDelSilkLine;
var X0,Y0,X1,Y1,p : integer;
begin
  if LineOk then
  begin
    SortMaxforRoot;
    for i := 1 to RootLcnt do
    begin
      p := RootLine[i];
      CompressSilkData(p);
    end;
  end;
end;

procedure CheckLibDataInSold (LibNum : integer);
begin
  if SoldPointer > 0 then
  begin
    for i := 1 to SoldPointer do
    begin
      if SoldVar^[i].LibPnt = LibNum then
      begin
        DevPointer := DevPointer+1;
        DevVar[DevPointer].CdeData := SoldVar^[i].CdeData;
        if SoldVar^[i].CdeData = LineCode then
        begin
          DevVar[DevPointer].PosData[1] := SoldVar^[i].PosData[1]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[2] := SoldVar^[i].PosData[2]-LibVar^[LibNum].YLibScn
          DevVar[DevPointer].PosData[3] := SoldVar^[i].PosData[3]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[4] := SoldVar^[i].PosData[4]-LibVar^[LibNum].YLibScn
        end
        else
        begin
          DevVar[DevPointer].PosData[1] := SoldVar^[i].PosData[1]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[2] := SoldVar^[i].PosData[2]-LibVar^[LibNum].YLibScn
          DevVar[DevPointer].PosData[4] := SoldVar^[i].PosData[3]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[4] := SoldVar^[i].PosData[4]-LibVar^[LibNum].YLibScn
        end;
        DevVar[DevPointer].AssData[1] := SoldVar^[i].AssData[1];
        DevVar[DevPointer].AssData[2] := SoldVar^[i].AssData[2];
        DelPnt := DelPnt + 1;
        DelArr[DelPnt]  := i;
      end;
    end;{for}
  end;{SoldPointer = 0}
end;
```

```
procedure CheckLibDataInComp (LibNum : integer);
begin
  if CompPointer > 0 then
  begin
    for i := 1 to CompPointer do
    begin
      if CompVar^[i].LibPnt = LibNum then
      begin
        DevPointer := DevPointer+1;
        DevVar[DevPointer].CdeData := CompVar^[i].CdeData;
        if CompVar^[i].CdeData = LineCode then
        begin
          DevVar[DevPointer].PosData[1] := CompVar^[i].PosData[1]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[2] := CompVar^[i].PosData[2]-LibVar^[LibNum].YLibScn
          DevVar[DevPointer].PosData[3] := CompVar^[i].PosData[3]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[4] := CompVar^[i].PosData[4]-LibVar^[LibNum].YLibScn
        end
        else
        begin
          DevVar[DevPointer].PosData[1] := CompVar^[i].PosData[1]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[2] := CompVar^[i].PosData[2]-LibVar^[LibNum].YLibScn
          DevVar[DevPointer].PosData[3] := CompVar^[i].PosData[3];
          DevVar[DevPointer].PosData[4] := CompVar^[i].PosData[4];
        end;
        DevVar[DevPointer].AssData[1] := CompVar^[i].AssData[1];
        DevVar[DevPointer].AssData[2] := CompVar^[i].AssData[2];
        DelPnt := DelPnt + 1;
        DelArr[DelPnt]  := i;
      end;
    end;{for}
  end;{CompPointer = 0}
end;

procedure CheckLibDataInSilk (LibNum : integer);
begin
  if SilkPointer > 0 then
  begin
    for i := 1 to SilkPointer do
    begin
      if SilkVar^[i].LibPnt = LibNum then
      begin
        DevPointer := DevPointer+1;
        DevVar[DevPointer].CdeData := SilkVar^[i].CdeData;
        if SilkVar^[i].CdeData = LineCode then
        begin
          DevVar[DevPointer].PosData[1] := SilkVar^[i].PosData[1]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[2] := SilkVar^[i].PosData[2]-LibVar^[LibNum].YLibScn
          DevVar[DevPointer].PosData[3] := SilkVar^[i].PosData[3]-LibVar^[LibNum].XLibScn
          DevVar[DevPointer].PosData[4] := SilkVar^[i].PosData[4]-LibVar^[LibNum].YLibScn
        end
        else
        begin
          DevVar[DevPointer].PosData[1] := SilkVar^[i].PosData[1]-LibVar^[LibNum].XLibScn
```

```
          DevVar[DevPointer].PosData[2] := SilkVar^[i].PosData[2]-LibVar^[LibNum].YLibScn
          DevVar[DevPointer].PosData[3] := SilkVar^[i].PosData[3];
          DevVar[DevPointer].PosData[4] := SilkVar^[i].PosData[4];
        end;
        DevVar[DevPointer].AssData[1] := SilkVar^[i].AssData[1];
        DevVar[DevPointer].AssData[2] := SilkVar^[i].AssData[2];
        DelPnt := DelPnt + 1;
        DelArr[DelPnt]  := i;
      end;
    end;{for}
  end;{SilkPointer = 0}

end;

procedure CheçkLibDataInPads (LibNum : integer);
begin
  if PadsPointer > 0 then
  begin
    for i := 1 to PadsPointer do
    begin
      if PadsVar^[i].LibPnt = LibNum then
      begin
        DevPointer := DevPointer+1;
        DevVar[DevPointer].CdeData := PadsVar^[i].CdeData;
        DevVar[DevPointer].PosData[1] := PadsVar^[i].PosData[1]-LibVar^[LibNum].XLibScn
        DevVar[DevPointer].PosData[2] := PadsVar^[i].PosData[2]-LibVar^[LibNum].YLibScn
        DevVar[DevPointer].AssData[1] := PadsVar^[i].AssData[1];
        DevVar[DevPointer].AssData[2] := PadsVar^[i].AssData[2];
        DelPnt := DelPnt + 1;
        DelArr[DelPnt]  := i;
      end;
    end;{for}
  end;{PadsPointer = 0}
end;

procedure SortMaxDelArr;
var Ok : boolean;
    DelTemp : integer;
begin
  repeat
   Ok := true;
   for i := 1 to DelPnt-1 Do
   begin
     if DelArr[i] < DelArr[i+1] then
     begin
       DelTemp := DelArr[i];
       DelArr[i] := DelArr[i+1];
       DelArr[i+1] := DelTemp;
       Ok := false;
     end;
   end;
  until Ok;
end;
```

```
procedure DelDevLib(PntS : integer);
var Pnt : integer;
begin
  if LibPointer > 0 then
  begin
    for Pnt := PntS to LibPointer - 1 do
    begin
      LibVar^[Pnt].LibLabel.Name := LibVar^[Pnt+1].LibLabel.Name;
      LibVar^[Pnt].LibCommt.Name := LibVar^[Pnt+1].LibCommt.Name;
      LibVar^[Pnt].XLibScn := LibVar^[Pnt+1].XLibScn;
      LibVar^[Pnt].YLibScn := LibVar^[Pnt+1].YLibScn;
      LibVar^[Pnt].XLibRef := LibVar^[Pnt+1].XLibRef;
      LibVar^[Pnt].YLibRef := LibVar^[Pnt+1].YLibRef;
      LibVar^[Pnt].XLibMin := LibVar^[Pnt+1].XLibMin;
      LibVar^[Pnt].YLibMin := LibVar^[Pnt+1].YLibMin;
      LibVar^[Pnt].XLibMax := LibVar^[Pnt+1].XLibMax;
      LibVar^[Pnt].YLibMax := LibVar^[Pnt+1].YLibMax;
    end;
  end;
end;

procedure RestoreLibDataToDevData(LibNum:integer);
var Bp,Dp : integer;
begin
  DevPointer := 0;
  LibLName := LibVar^[LibNum].LibLabel.Name;
  LibCName := LibVar^[LibNum].LibCommt.Name;
  RefX     := LibVar^[LibNum].XLibRef;
  RefY     := LibVar^[LibNum].YLibRef;
  MinDataX := LibVar^[LibNum].XLibMin;
  MinDataY := LibVar^[LibNum].YLibMin;
  MaxDataX := LibVar^[LibNum].XLibMax;
  MaxDataY := LibVar^[LibNum].YLibMax;
  PositionCurX := LibVar^[LibNum].XLibScn;
  PositionCurY := LibVar^[LibNum].YLibScn;

  {-------------------------PADS}
  DelPnt := 0;
  CheckLibDataInPads(LibNum);
  if DelPnt > 0 then
  begin
    SortMaxDelArr;
    for Bp := 1 to DelPnt do
    begin
      Dp := DelArr[Bp];
      CompressPadsData(Dp);
    end;
    for Bp := 1 to PadsPointer do
    begin
      if LibNum < PadsVar^[Bp].LibPnt then
        PadsVar^[Bp].LibPnt := PadsVar^[Bp].LibPnt - 1;
    end;
  end;
  {-------------------------SOLD}
```

```
    delPnt := 0;
    CheckLibDataInSold(LibNum);
    if DelPnt > 0 then
    begin
      SortMaxDelArr;
      for Bp := 1 to DelPnt do
      begin
        Dp := DelArr[Bp];
        CompressSoldData(Dp);
      end;
      for Bp := 1 to SoldPointer do
      begin
        if LibNum < SoldVar^[Bp].LibPnt then
          SoldVar^[Bp].LibPnt := SoldVar^[Bp].LibPnt - 1;
      end;
    end;
    {-------------------------COMP}
    DelPnt := 0;
    CheckLibDataInComp(LibNum);
    if DelPnt > 0 then
    begin
      SortMaxDelArr;
      for Bp := 1 to DelPnt do
      begin
        Dp := DelArr[Bp];
        CompressCompData(Dp);
      end;
      for Bp := 1 to CompPointer do
      begin
        if LibNum < CompVar^[Bp].LibPnt then
          CompVar^[Bp].LibPnt := CompVar^[Bp].LibPnt - 1;
      end;
    end;
    {-------------------------SILK}
    DelPnt := 0;
    CheckLibDataInSilk(LibNum);
    if DelPnt > 0 then
    begin
      SortMaxDelArr;
      for Bp := 1 to DelPnt do
      begin
        Dp := DelArr[Bp];
        CompressSilkData(Dp);
      end;
      for Bp := 1 to SilkPointer do
      begin
        if LibNum < SilkVar^[Bp].LibPnt then
          SilkVar^[Bp].LibPnt := SilkVar^[Bp].LibPnt - 1;
      end;
    end;
    {-------------------------}
  end;

  begin { PCBUnit6 }
```