โปรแกรมจำลองการทำงานอุปกรณ์ดิจิตอล

(Digital Circuit Simulator)

นาย กร พวงนาค

นาย จิรวุฒิ ยุวเบญจพล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

โปรแกรมจำลองการทำงานอุปกรณ์ดิจิตอล

(Digital Circuit Simulator)

โดย

นาย กร พวงนาค

นาย จิรวุฒิ ยุวเบญจพล

อาจารย์ที่ปรึกษา

อ. เจริญ วงษ์ชุ่มเย็น

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมจำลองการทำงานวงจรอุปกรณ์ดิจิตอล

(Digital Circuit Simulator)

ผู้จัดทำ

1. นาย กร พวงนาค รหัสนักศึกษา 46015333

2. นาย จิรวุฒิ ยุวเบญจพล รหัสนักศึกษา 46015341

_____อาจารย์ที่ปรึกษา

(อ.เจริญ วงษ์ชุ่มเย็น)

โปรแกรมจำลองการทำงานอุปกรณ์ดิจิตอล

นายกร	พวงนาค	46015333
นายจิรวุฒิ	ยุวเบญจพล	46015341
อ. เจริญ	วงษ์ชุ่มเย็น	อาจารย์ที่ปรึกษา

ปีการศึกษา 2548

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้เสนอโปรแกรมจำลองการทำงานของวงจรดิจิตอลแบบใหม่ให้มีการใช้งานโปรแกรมที่ง่ายยิ่งขึ้น ให้ผู้ใช้งานสามารถจำลองการทำงานของวงจรดิจิตอลได้อย่างง่ายดายกว่าเดิมที่เป็นอยู่ และ สามารถใช้งานได้ตั้งแต่ผู้ใช้ที่หัดเริ่มออกแบบวงจรดิจิตอล ไปจนถึงผู้ที่มีประสบการณ์ในการทำงานออกแบบวงจรมาแล้วได้ ซึ่งจะยังคงความถูกต้องของการจำลองการทำงาน และ ความรวดเร็วใจการจำลองการทำงานเอาไว้ไม่ให้ลดด้อยลงไปกว่าเดิม พร้อมทั้งมีลักษณะของหน้าตาโปรแกรมที่ใช้งานได้สะดวก และ มีความคล่องตัวในการทำงานสูง ซึ่งเหมาะแก่การนำไปออกแบบ และ พัฒนาฮาร์ดแวร์ที่มีหลักการทำงานเป็นดิจิตอล หรือ นำไปใช้ในการศึกษาของนักศึกษาที่กำลังศึกษาทางด้านดิจิตอลอยู่ได้ ซึ่งจะทำให้ประหยัดงบประมาณกว่าการนำอุปกรณ์จริงไปต่อเพื่อทดลอง

(Digital Circuit Simulator)

Korn Puangnark 46015333

Jirawut Yubenjapol 46015341

Charoen Vongchumyen Advisor

Academic Year 2004

ABSTRACT

This thesis proposes a program digital circuit simulator new version. Which easy to use more than Old version. For user can simulation digital circuit Design with easier. And can use if user is beginner or user is professional and resault of simulate is still Wrong and Speed is still Fast. And User interface is design to convenient to used. And Flexible to User Too. Which appropriate to design hardware device. Or use it teach student to leane digital curcuit design. Which will economize money more than use real digital device.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้อย่างดี ด้วยคำแนะนำ และคำปรึกษาจาก อ.เจริญ วงษ์ชุ่มเย็น ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ข้าพเจ้ารู้สึกทราบซึ้งในความอนุเคราะห์จากท่านอาจารย์และขอขอบพระคุณเป็นอย่างสูง

ขอกราบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า

ขอขอบคุณเพื่อนๆ พี่ๆ น้องๆ ในภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยี พระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุกคนที่ให้คำแนะนำต่างๆ และคอยให้กำลังใจเสมอมา

ขอขอบคุณบัณฑิตศึกษาและบัณฑิตวิทยาลัย คณะวิศวกรรมศาสตร์ที่ให้ความช่วยเหลือในเรื่องต่างๆ

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจและให้การสนับสนุนในทุกเรื่องๆ ทำให้ข้าพเจ้าสามารถทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

คุณค่าและประโยชน์อันพึงมาจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

นาย กร พวงนาค

นาย จิรวุฒิ ยุวเบญจพล

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

ปัจจุบันเทคโนโลยีทางฮาร์ดแวร์ได้พัฒนารุดหน้าไปอย่างรวดเร็วมาก และการออกแบบฮาร์ดแวร์แต่ละครั้งนั้นจะต้องสิ้นเปลืองงบประมาณในการทดลองไปมากมาย ซึ่งเป็นการสิ้นเปลืองงบประมาณไปเป็นส่วนมากกับการทดลองดังกล่าวนี้กับการทดลองกับอุปกรณ์จริง ดั้งนั้นจึงมีการสร้างโปรแกรมขึ้นมาเพื่อที่จะใช้ในการจำลองการทำงานของอุปกรณ์เหล่านั้นขึ้นมาเพื่อความประหยัด ซึ่งในการออกแบบและพัฒนาอุปกรณ์ฮาร์ดแวร์ในปัจจุบันได้มีโปรแกรมจำลองการออกแบบ และ การพัฒนาเกิดขึ้นมากมาย โดยโปรแกรมเหล่านี้มีศักยภาพในการประมวลผลและจำลองการทำงานที่ซับซ้อนได้อย่างรวดเร็วและแม่นยำ แต่ทั้งนี้ผู้พัฒนาต้องมีความรู้และเข้าใจในระบบการเชื่อมต่อและใช้งานอย่างชัดเจน

จากเนื้อหาที่กล่าวข้นต้น จะแสดงให้เห็นว่าผู้พัฒนาต้องเป็นผู้พัฒนาที่มีประสบการณ์มาแล้วพอสมควร ดังนั้นจะเป็นการยากแก่ผู้เริ่มต้นพัฒนา ดังนั้นผู้จัดทำโครงงานเล็งเห็นว่าสมควรจะพัฒนาโปรแกรมจำลองการทำงานวงจรดิจิตอล แบบเบื้องต้น หรือที่เรียกว่าแบบทางอุดมคติ ซึ่งจะเหมาะแก่ผู้เริ่มต้นศึกษาการพัฒนาฮาร์ดแวร์ และรวมไปถึง ผู้พัฒนาฮาร์ดแวร์ที่ระบบมีความซับซ้อนที่ต้องการออกแบบระบบเบื้องต้นทาง อุดมคติ เป็นต้น

ซึ่งโปรแกรมที่เราจะออกแบบมานี้จะเน้นให้ลักษณะง่ายดายต่อการใช้งานของผู้ใช้เพื่อจะให้ผู้ใช้ระดับเริ่มต้นได้ใช้งานได้อย่างสะดวกยิ่งขึ้น

1.2 วัตถุประสงค์ของโครงงาน

ปริญญานิพนธ์ฉบับนี้มุ่งหวังเพื่อที่จะสร้างโปรแกรมขึ้นมาเพื่อที่จะใช้ในการทดลองการทำงานของวงจรดิจิตอล เพื่อลดต้นทุนในการนำวงจรจริง ๆ มาทดลองในการออกแบบ และ ลดปัญญาหาเรื่องอุปกรณ์เสียหายขณะทำการทดลอง พร้อมทั้งทำให้การทดลองออกแบบทางด้านฮาร์ดแวร์เป็นไปได้อย่างสะดวกรวดเร็วมายิ่งขึ้น และ ทำให้งานออกแบบมีความคล่องตัวกว่าการใช้วงจรจริงมาทดลอง ดังนั้นปริญญานิพนธ์นี้จึงเสนอโปรแกรมจำลองการทำงาน (Digital Circuit Simulator) ซึ่งจะทำให้การออกแบบวงจรทางด้านดิจิตอล ในงานออกแบบฮาร์ดแวร์ให้มีความสะดวกรวดเร็วในการออกแบบ และ มีการใช้งานที่ง่ายขึ้น

1.3 ขอบเขตของโครงงาน

ในปริญญานิพนธ์ฉบับนี้ได้นำเสนอโปรแกรมจำลองการทำงานของวงจรดิจิตอล ซึ่งจะสามารถจำลองการทำงานทางดิจิตอลได้ และ สามารถจำลองการทำงานของอุปกรณ์ดิจิตอลจำพวก Gate พื้นฐานต่าง ๆ ได้ เช่น AND Gate , OR Gate , NOR Gate , NAND Gate , NOT Gate , XOR Gate และ XNOR Gate และ สามารถจะลองการทำงานของอุปกรณ์ที่มีลักษณะการทำงานเป็น State อย่างพวก Flip Flop ต่างๆ ได้ เช่น D Flip Flop , T Flip Flop , RS Flip Flop และ JK Flip Flop และ รองรับการจำลองการทำงานของอุปกรณ์อินพุตต่าง ๆ ได้ เช่น Switch , Dip Switch และ KeyPad พร้อมทั้งจำลองการทำงานของอุปกรณ์ที่เป็นเอาต์พุตได้ดังนี้ คือ LED แสดงผล , 7SEGMENT และ LED METRIX ซึ่งการจำลองการทำงานทั้งหมดนี้จะเน้นที่การใช้งานให้ง่าย และ สะดวกต่อผู้ใช้งาน

1.4 วิธีการดำเนินการ

1. ศึกษาการทำงานของอุปกรณ์ต่าง ๆ ที่เกี่ยวข้องเพื่อเข้าใจการทำงานของอุปกรณ์
2. ศึกษาการโครงสร้างของระบบฐานข้อมูลที่จะนำมาเก็บข้อมูลอุปกรณ์
3. ศึกษาเกี่ยวกับโครงสร้างของไฟล์ที่จะใช้ในการเก็บวงจรที่จะจำลองการทำงาน
4. จัดหาอุปกรณ์ที่จะนำมาจำลองการทำงานเพื่อนำมาทดลองก่อนจะจำลองการทำงาน
5. วิเคราะห์ และออกแบบระบบของโปรแกรมจำลองการทำงาน
6. พัฒนาโปรแกรมที่ใช้ในการจำลองการทำงานวงจรดิจิตอล
7. วิเคราะห์ผลการทำงานของโปรแกรมจำลองการทำงาน และ แก้ไขโปรแกรมให้มีการใช้งานได้ง่ายที่สุด และ ทำงานได้ใกล้เคียงกับความเป็นจริงให้มากที่สุด

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้รับความรู้ความเข้าใจเกี่ยวกับการทำงานของอุปกรณ์ต่าง ๆ
2. ได้รับความรู้ความเข้าใจเกี่ยวกับการสร้างระบบดาต้าเบสที่ใช้เก็บข้อมูลอุปกรณ์
3. ได้รับความรู้ความเข้าใจเกี่ยวกับการสร้างโปรแกรมจำลองการทำงานวงจรดิจิตอล
4. โปรแกรมจำลองการทำงานวงจรดิจิตอลที่ใช้งานง่าย และ สะดวกในการใช้งาน

1.6 ส่วนประกอบของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 5 บทด้วยกันคือ

บทที่ 1 กล่าวถึงความสำคัญและที่มาของโครงงาน วัตถุประสงค์ของโครงงาน ขอบเขตของโครงงาน วิธีการดำเนินการ ประโยชน์ที่คาดว่าจะได้รับ และส่วนประกอบของปริญญานิพนธ์

บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานที่ใช้ในโครงงาน ซึ่งประกอบด้วยทฤษฎีอะไรบ้าง ให้บรรยายทฤษฎีทั้งหมดโดยละเอียด

บทที่ 3 กล่าวถึงชิ้นงานของโครงงานนี้ ส่วนที่ได้พัฒนาขึ้น การทำงานของระบบหรือชิ้นงานบรรยายโดยละเอียด

บทที่ 4 กล่าวถึงการทดลองและผลการทดลอง การหาค่าสมรรถนะของระบบ การวัดประสิทธิภาพของระบบ พารามิเตอร์ที่ใช้และผลที่ได้จากการจำลองระบบ ผลการทดลองหรือผลการทำงานทั้งหมด

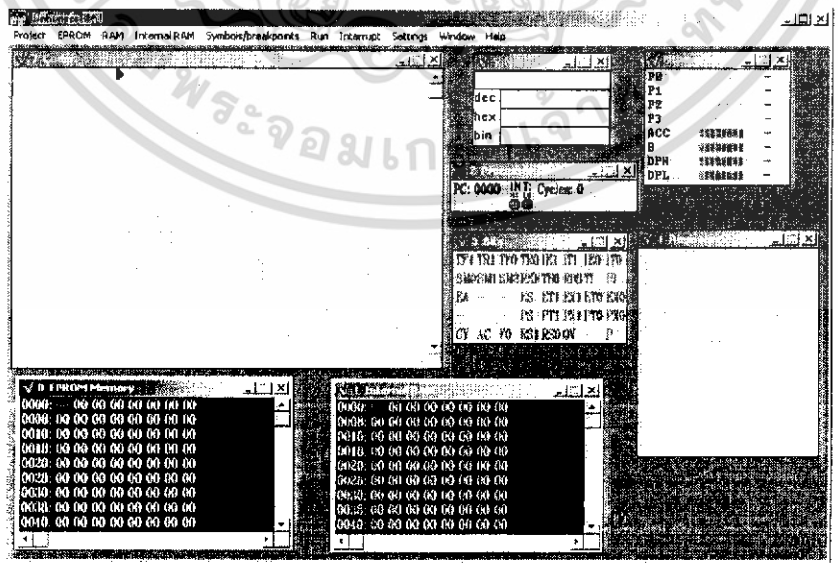
บทที่ 5 เป็นบทวิจารณ์และสรุป ซึ่งกล่าวถึงบทสรุปของโครงงาน วิจารณ์สิ่งที่ได้รับจากโครงงาน และข้อเสนอแนะสำหรับเป็นแนวทางในการพัฒนาต่อ

บทที่ 2
ทฤษฎีพื้นฐานที่ใช้ในโครงงาน

ในหัวข้อนี้จะกล่าวถึงทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องในการวิจัย และ พื้นฐานของการทำงานของอุปกรณ์ต่าง ๆ ที่จะนำมาทำการจำลองการทำงานเป็นโปรแกรมจำลองการทำงาน ซึ่งในเนื้อหาในบทนี้จะกล่าวถึง หลักการทำงานของอุปกรณ์ต่าง และ คุณลักษณะของอุปกรณ์ต่าง ๆ ที่จะนำมาจำลองการทำงาน ซึ่งเนื้อหาทั้งหมดนี้จำเป็นสำหรับการออกแบบ และ พัฒนาโปรแกรมจำลองการทำงานทั้งสิ้น

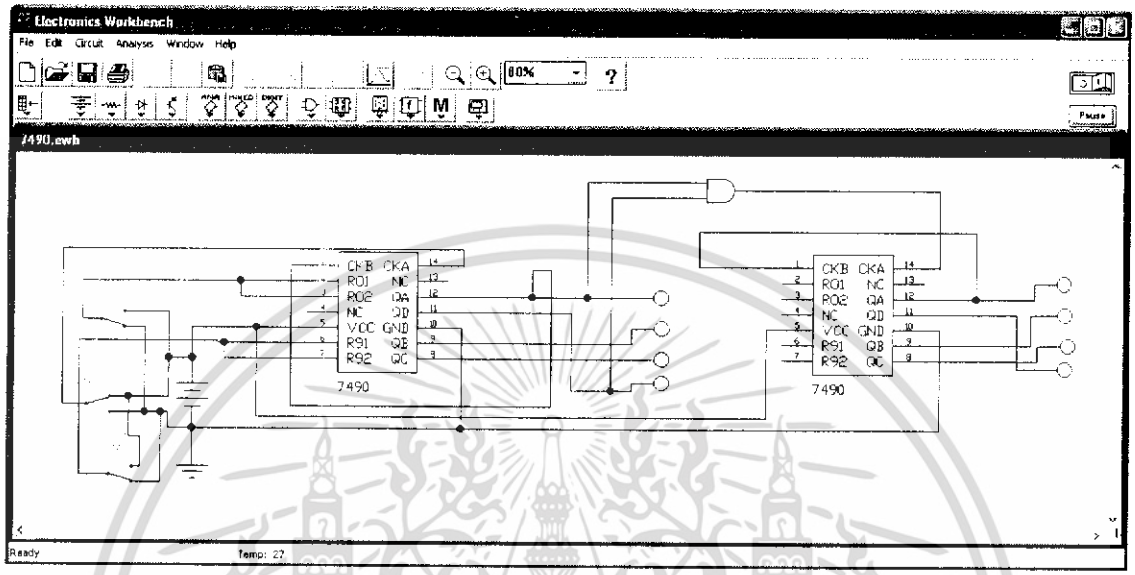
2.1 ความหมายของโปรแกรมจำลองการทำงาน

โปรแกรมจำลองการทำงาน หมายถึง โปรแกรมที่สามารถจำลองผลของการทำงานของสิ่งต่าง ๆ บนคอมพิวเตอร์ขึ้นมาให้มีการแสดงผล แบบเสมือนจริงขึ้นมาที่หน้าจอคอมพิวเตอร์ เพื่อที่จะทดลองการทำงานของอะไรบางอย่างขึ้นมา ซึ่งการจำลองนั้นก็จะจำลองมาจากการทำงานจริงของสิ่งเหล่านั้น ซึ่งจะเป็นการจำลองในทางอุดมคติซะเป็นส่วนใหญ่ หรือ บางตัวก็อาจจะแสดงผลการจำลองการทำงานออกมาเป็นในทางปฏิบัติได้ ดังนั้นโปรแกรมจำลองการทำงานของอุปกรณ์ดิจิตอลก็คือ โปรแกรมที่ใช้จำลองการทำงานของวงจรดิจิตอลขึ้นมาให้แสดงการทำงานแบบจำลอง ๆ ขึ้นมา ให้เราได้เห็นผลการทำงานของวงจรดิจิตอลนั้น ๆ เพื่อที่จะได้สามารถนำไปออกแบบและประยุกต์ใช้งานได้โดยไม่ต้องนำอุปกรณ์จริง ๆ มาต่อเพื่อดูการทำงาน ซึ่งลักษณะหน้าตาของโปรแกรมจะมีลักษณะดังรูปที่ 2.1



รูปที่ 2.1 แสดง หน้าตาโปรแกรมจำลองการทำงาน

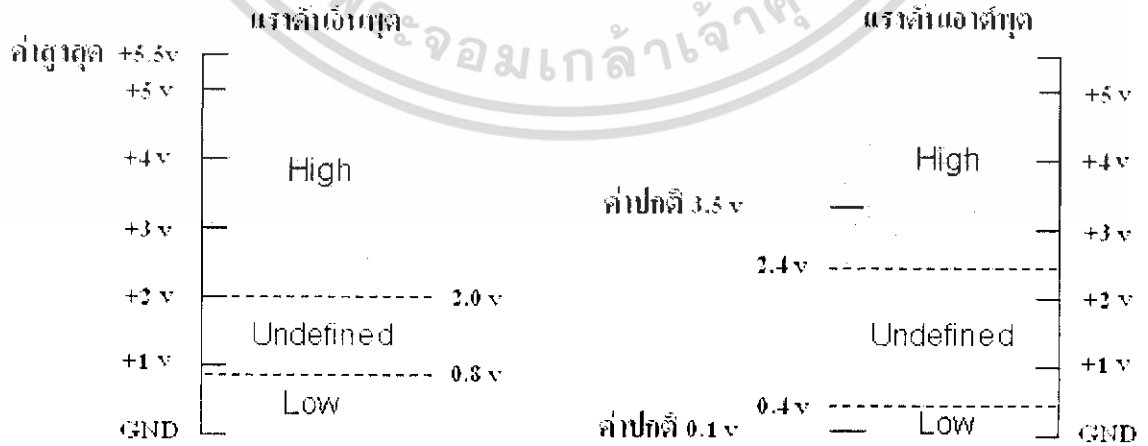
จากรูปที่ 1 ในหน้าที่ผ่านมาเป็นโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ ซึ่งเป็นโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ขึ้นมาให้ผู้ใช้ได้เห็นผลการทำงานของไมโครคอนโทรลเลอร์ตัวนั้น ๆ ส่วนโปรแกรมจำลองการทำงานของวงจรดิจิตอลก็จะมีลักษณะหน้าตาดังรูปที่ 2.2



รูปที่ 2.2 แสดงแสดง หน้าตาโปรแกรมจำลองการทำงานวงจรดิจิตอล

2.2 ข้อมูลการทำงานของไอซี ทีทีแอล และ ซีมอส

TTL ย่อมาจาก Transistor Transister Logic ซึ่งเป็นวงจรลอจิกที่สร้างขึ้นมาจากวงจรสวิตซ์ที่ทำงานด้วยทรานซิสเตอร์ ซึ่งประกอบกันเป็นเกตแบบต่าง ๆ เช่น แอนด์เกต ออร์เกต เป็นต้น โดยระดับแรงดันลอจิกของเกตแบบทีทีแอลชนิดมาตรฐาน เป็นดังรูปที่ 2.3



รูปที่ 2.3 แสดงระดับแรงดันอินพุต และ เอาต์พุตของวงจรรวมทีทีแอลชนิดมาตรฐาน

วงจรรวมทีทีแอลมีหลายตระกูล เช่น ทีทีแอลมาตรฐาน,ทีทีแอลกำลังงานต่ำ (L),ทีทีแอลซอตต์กี (S) , ทีทีแอลซอตต์กีกำลังต่ำ (LS) และ แบบทีทีแอลความเร็วสูง (H) แต่ละชนิดจะมีโครงสร้างของอุปกรณ์ภายในวงจรที่แตกต่างกัน แต่ยังคงใช้ทรานซิสเตอร์ทำงานเป็นสวิตซ์ ให้วงจรรวมชนิดทีทีแอลทำงานตามทีทได้ออกแบบ และ สร้างไว้เป็นเกตต่าง ๆ ละจิกเกตพื้นฐานที่สำคัญ ๆ มีดังนี้

1. AND GATE มีสัญลักษณ์ และ การทำงานดังตารางความจริงต่อไปนี้

สมการการทำงาน Y = A*B

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

รูปที่ 2.4 แสดงสัญลักษณ์ สมการการทำงาน และ ตารางความจริง AND GATE

2. OR GATE มีสัญลักษณ์ และ การทำงานดังตารางความจริงต่อไปนี้

สมการการทำงาน Y = A+B

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

รูปที่ 2.5 แสดงสัญลักษณ์ สมการการทำงาน และ ตารางความจริง OR GATE

3. NAND GATE มีสัญลักษณ์ และ การทำงานดังตารางความจริงต่อไปนี้

สมการการทำงาน Y = NOT (A*B)

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

รูปที่ 2.6 แสดงสัญลักษณ์ สมการการทำงาน และ ตารางความจริง NAND GATE

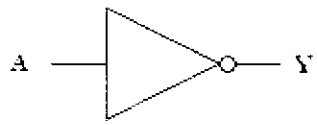
4. NOR GATE มีสัญลักษณ์ และ การทำงานดังตารางความจริงต่อไปนี้

สมการการทำงาน Y = NOT (A*B)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

รูปที่ 2.7 แสดงสัญลักษณ์ สมการการทำงาน และ ตารางความจริง NOR GATE

5. NOT GATE



สมการการทำงาน Y = NOT (Y)

A	Y
0	1
1	0

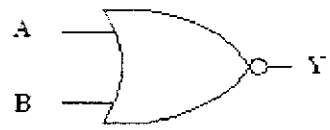
รูปที่ 2.8 แสดงสัญลักษณ์ สมการการทำงาน และ ตารางความจริง NOT GATE

6. EXCUSIVE OR GATE

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

รูปที่ 2.9 แสดงสัญลักษณ์ สมการการทำงาน และ ตารางความจริง EXCUSIVE OR GATE

7. EXCUSIVE NOR GATE



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

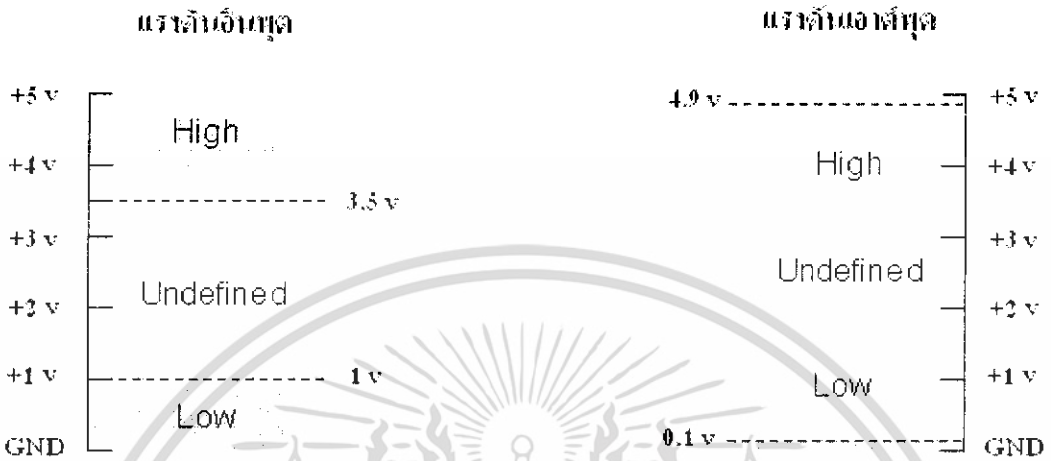
รูปที่ 2.10 แสดงสัญลักษณ์ สมการการทำงาน และ ตารางความจริง EXCUSIVE NOR GATE

ลอจิกเกตที่มีโครงสร้างภายในประกอบด้วยทรานซิสเตอร์ คือ ลอจิกเกตในตระกูลทีทีแอลนั้น มีขีดจำกัดในเรื่องการสร้างให้มีความสามารถในการรวมวงจรขนาดใหญ่ (ISL,Large Scale Integrated Circuit) จึงมีการพัฒนาเพื่อสร้างลอจิกจากมอสเฟส เรียกว่า วงจรรวมแบบมอสเฟส และ ซีมอส ซึ่งสามารถสร้างให้ทำงานได้ตามฟังก์ชั่นเหมือนกับวงจรรวมทีทีแอล แต่ต่างกันที่วงจรภายในของวงจรรวมแบบมอสและ ซีมอส ใช้มอสเฟต (MOSFET) แทนทรานซิสเตอร์ ทำให้คุณลักษณะทางแรงดันของลอจิกอินพุต และ เอาต์พุตเปลี่ยนแปลงไป แต่คุณสมบัติที่เด่นมากของมอสและซีมอส คือ กินกระแสอินพุตต่ำกว่าวงจรรวมที่เป็นทีทีแอลประมาณ 40 เท่า เลยทีเดียว ดังรูปที่ 2.11

ตระกูลของวงจรรวม		กระแสเอาต์พุต	กระแสอินพุต
TTL	Standard TTL	I = 400uA I = 16mA	I = 40 uA I = 1.6 mA
	Low Power Schottky	I = 400uA I = 8mA	I = 20 uA I = 400 mA
	Advanced Low Power Schottky	I = 400uA I = 8mA	I = 20 uA I = 100 mA
CMOS	4000 Series	I = 400uA I = 400uA	I = 1 uA I = 1 uA
	74HC00 Series	I = 4 mA I = 4 mA	I = 1 uA I = 1 uA

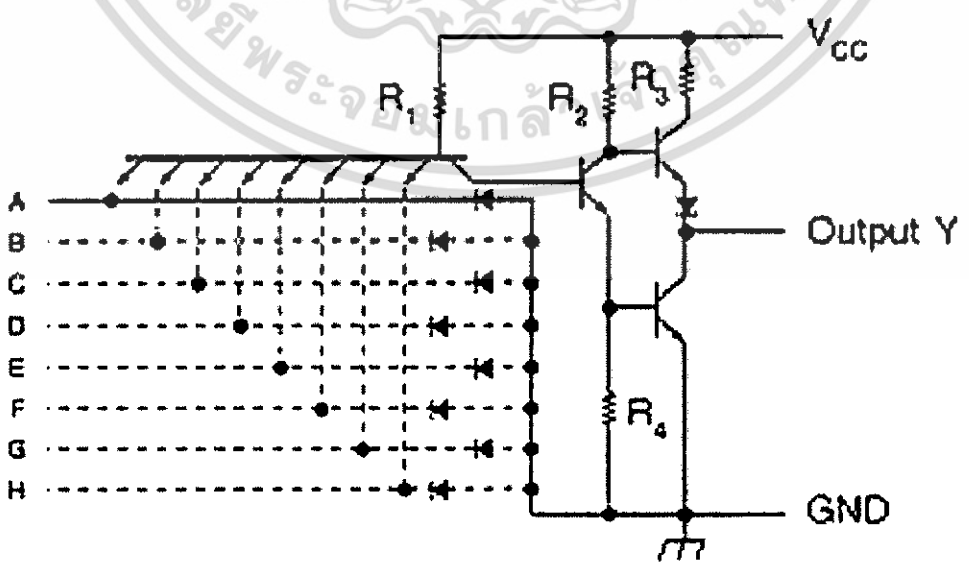
รูปที่ 2.11 ตารางเปรียบเทียบลักษณะสมบัติทางด้านกระแสและ แรงดันอินพุต / เอาต์พุตของทั้งสองชนิด

ระดับแรงดันอินพุต และ เอาต์พุตของวงจรรวมแบบซีมอส แตกต่างจากทีทีแอล คือ ระดับอินพุต HI ประมาณ 3.5-5 V และ ระดับอินพุต LOW ประมาณ 1.5-0 V และ ระดับแรงดันเอาต์พุต HI ประมาณ 4.95 – 5V และ เอาต์พุต LOW ประมาณ 0.05-0 V ตามรูปที่2.12

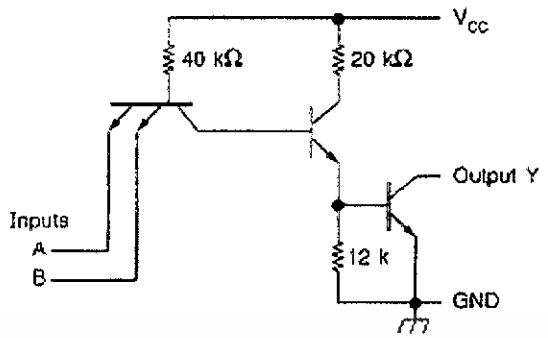


รูปที่ 2.12 แสดงระดับแรงดันอินพุต และ เอาต์พุตของวงจรรวมแบบซีมอส

ลอจิกเกตทั้งสองชนิดนั้น แบ่งตามลักษณะของวงจรภาคเอาต์พุตออกเป็น 2 แบบ คือ 1. เอาต์พุตแบบโทเทมโพล และ 2. เอาต์พุตแบบคอลเลกเตอร์เปิด ซึ่งแตกต่างกันที่ทรานซิสเตอร์ตัวสุดท้ายที่เอาต์พุต ซึ่งใช้ต่อเพื่อขับไดโอดเปล่งแสง หรือ โหลดว่าต่อแบบใด สังเกตความแตกต่างที่ชัดเจนจากแนนด์เกตเบอร์ 7400 ซึ่งเป็นเอาต์พุตแบบโทเทมโพล ดังรูปที่ 2.13 และแนนด์เกตเบอร์ 7401 ซึ่งเป็นเอาต์พุตคอลเลกเตอร์เปิด ดังรูปที่ 2.14

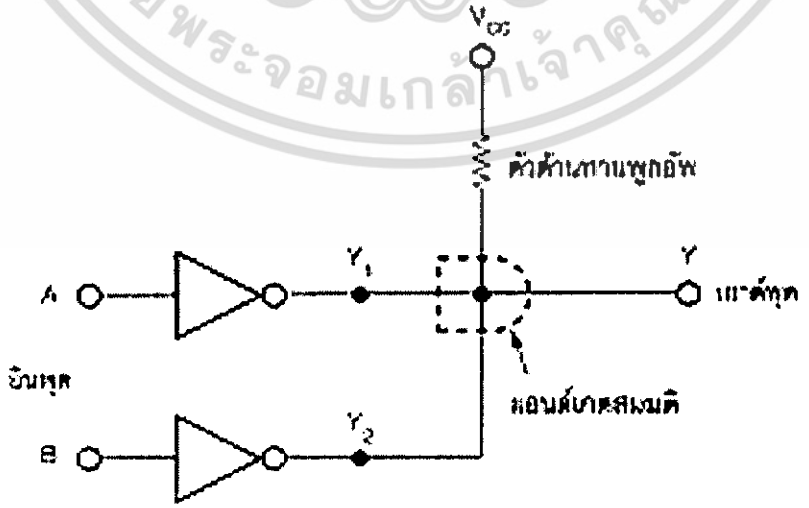


รูปที่ 2.13 แสดงวงจรภายในที่เป็น Output โทเทมโพล



รูปที่ 2.14 แสดงวงจรภายในที่เป็น Output คอลเลกเตอร์เปิด

เมื่อเทียบการทำงานของเอาต์พุตแบบโทเทมโพล และ แบบคอลเลกเตอร์เปิดแล้วจะให้ลอจิกที่เอาต์พุตตรงกัน เมื่อใช้โวลต์มิเตอร์วัดค่าแรงดันต่ำ และ สูง แต่ถ้านำไปขับไดโอดเปล่งแสงแล้วจะต่อวงจรต่างกัน กล่าวคือ เกตแบบเอาต์พุตแบบโทเทมโพลต้องต่อเอาต์พุตกับแอโนดของไดโอดเปล่งแสง และ ต่อแคโทดของไดโอดเปล่งแสงลงกราวด์ การติดดับของไดโอดเปล่งแสงจะตรงกันกับสภาวะลอจิกเอาต์พุต คือ ลอจิก "1" ไดโอดเปล่งแสงจะติด ลอจิก "0" ไดโอดเปล่งแสงจะดับ แต่แบบคอลเลกเตอร์เปิดนั้นต้องต่อเอาต์พุตกับแคโทดของไดโอดเปล่งแสง และ ต่อแอโนดของไดโอดเปล่งแสงเข้ากับแหล่งจ่าย โดยต่ออนุกรมกับตัวต้านทานตัวหนึ่ง เรียกว่าตัวต้านทานพูลอัพ เกตแบบคอลเลกเตอร์เปิดจึงจะทำงานได้ และ ถ้าวัดสภาวะลอจิกเอาต์พุตของเกต จะตรงกันข้ามกับการติดดับของไดโอดเปล่งแสง ประโยชน์ของลอจิกเกตแบบคอลเลกเตอร์เปิด คือ การนำไปใช้งานในวงจรที่มีชื่อว่า Wire AND Gate โดยนำเอาต์พุตของเกตชนิดเอาต์พุตคอลเลกเตอร์เปิดทุกตัวต่อร่วมกัน ดังรูปที่ 2.15

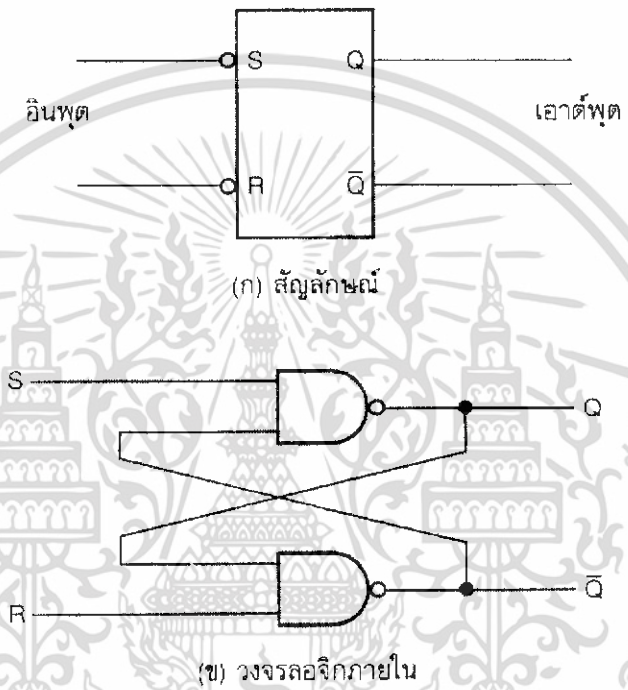


รูปที่ 2.15 แสดงการใช้งานลอจิกเกตแบบคอลเลกเตอร์เปิดมาสร้างแอนเกตสมมุติ

2.3 ข้อมูลการทำงานของ Flip Flop

ฟลิปฟลอป เป็นอุปกรณ์ลอจิกพวกไบสเตเบิลที่มีอินพุตเดียวหรือ มากกว่า และ มีเอาต์พุต 2 เอาต์พุต เอาต์พุตทั้งสองนี้จะต้องแสดงสภาวะลอจิกตรงกันข้ามกัน ฟลิปฟลอป เป็นวงจรพื้นฐานที่จะนำไปใช้ในการสร้างวงจรดิจิตอลต่าง ๆ ที่เกี่ยวกับวงจรลอจิกเชิงลำดับ ฟลิปฟลอปมีอยู่หลายชนิดดังต่อไปนี้

1. ฟลิปฟลอปชนิดอาร์เอส มี 2 อินพุตเรียกว่า เซต (Set S) และ รีเซต (Reset R) มี 2 เอาต์พุต คือ Q กับ /Q มีสัญลักษณ์ และ วงจรภายในดังรูปที่ 2.16



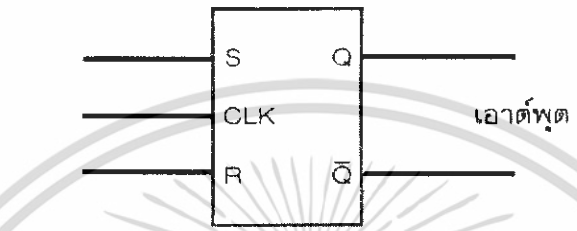
รูปที่ 2.16 แสดงสัญลักษณ์(ก) และ วงจรภายใน(ข) ของ RS FLIP FLOP

การทำงานของฟลิปฟลอปชนิดอาร์เอสที่ใช้แนนด์เกตมี 4 สภาวะ เป็นไปตามตารางความจริงของฟลิปฟลอปชนิดอาร์เอสที่แสดงในรูปที่ 2.17

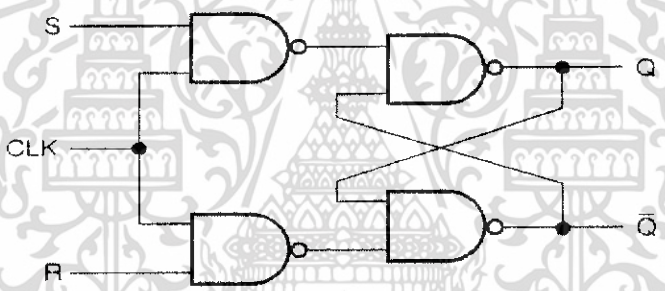
โมดการทำงาน	อินพุต		เอาต์พุต		
	S	R	Q	$\bar{Q}$	ผลของเอาต์พุต Q
Prohibited	0	0	1	1	ห้ามใช้งาน
Set	0	1	1	0	เซตให้ Q = 1
Reset	1	0	0	1	รีเซตให้ Q = 0
Hold	1	1	Q	$\bar{Q}$	ไม่เปลี่ยนแปลง

รูปที่ 2.17 ตารางความจริงของ RS Flip Flop

2. ฟลิปฟลอปชนิดอาร์เอสควบคุมด้วยสัญญาณนาฬิกา เป็นฟลิปฟลอปที่มีขาอินพุต 3 ขา คือ ขา เซต รีเซต และ คล็อก ทำหน้าที่เป็นขาควบคุมเอาต์พุต มี 2 ขา คือ Q และ /Q ขาคล็อกจะเป็นขาควบคุมการทำงานของฟลิปฟลอป ซึ่งถ้าไม่มีการป้อนสัญญาณนาฬิกาเข้าขาคล็อก ก็จะทำให้ฟลิปฟลอปไม่ทำงานแม้จะได้รับสัญญาณลอจิกเข้าที่ขา S และ R ฟลิปฟลอปชนิดอาร์เอสควบคุมด้วยสัญญาณนาฬิกา (R-S Flip-Flop With Clock) มีสัญลักษณ์และวงจรภายในดังรูปที่ 2.18 และ มีการทำงานตามตารางความจริงในรูปที่ 2.19



(ก) สัญลักษณ์



(ข) วงจรภายใน

รูปที่ 2.18 แสดงสัญลักษณ์(ก) และ วงจรภายใน(ข) ของ RS FLIP FLOP With Clock

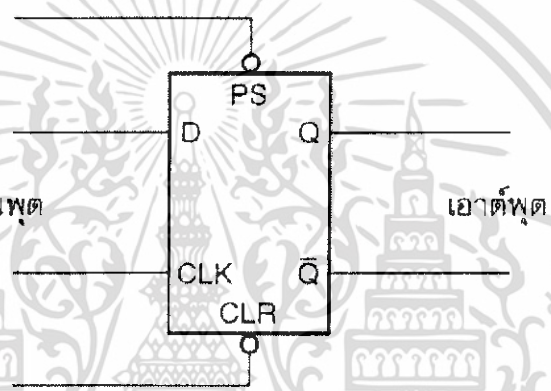
โมดการทำงาน	อินพุต			เอาต์พุต		
	CLK	S	R	Q	$\bar{Q}$	ผลของเอาต์พุต Q
Hold	⎍	0	0	ไม่เปลี่ยนแปลง		ไม่เปลี่ยนแปลง
Reset	⎍	0	1	0	1	รีเซตให้ Q = 0
Set	⎍	1	0	1	0	เซตให้ Q = 1
Prohibited	⎍	1	1	1	1	ห้ามใช้งาน

รูปที่ 2.19 แสดงตารางความจริงของ FLIP FLOP With Clock

3.ฟลิปฟลอปชนิดดี เป็นฟลิปฟลอปที่มีอินพุต 2 ชุด คือ

- อินพุตซิงโครนัส (Synchronous Input) ประกอบด้วยขาข้อมูล (D) และขาคล็อก (CLK) ซึ่งทำงานร่วมกัน เมื่อป้อนข้อมูลสัญญาณลอจิกเข้าที่ขา D ข้อมูลสัญญาณลอจิกจะถูกส่งผ่านไปยังที่เอาต์พุต Q ได้ก็ต่อเมื่อมีสัญญาณพัลส์ป้อนเข้าขา CLK มาควบคุมการทำงานเท่านั้น

- อินพุตอะซิงโครนัส (Asynchronous Input) ประกอบด้วยขาพรีเซต (PS) และขาเคลียร์ ซึ่งทำงานด้วยลอจิก "0" กล่าวคือ เมื่อป้อน "0" ให้ขา PS จะทำให้ Q = "1" เมื่อป้อน "0" ให้ขา CLK จะทำให้ Q = "0" ฟลิปฟลอปชนิดดีที่นิยมนำมาใช้งาน คือ เบอร์ 7474 ดังแสดงในรูปที่ 2.20



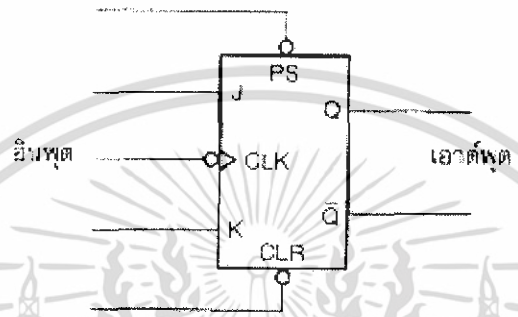
โมดการทำงาน	อินพุต				เอาต์พุต	
	อะซิงโครนัส		ซิงโครนัส			
	PS	CLR	CLK	D	Q	$\bar{Q}$
Asynchronous	0	1	x	x	1	0
Asynchronous reset	1	0	x	x	0	1
Prohibited	0	0	x	x	1	1
Set	1	1	↑	1	1	0
Reset	1	1	↑	0	0	1

รูปที่ 2.20 แสดงสัญลักษณ์ และ ตารางความจริง ของ D Flip Flip

4. ฟลิปฟลอปชนิดเจเค เป็นฟลิปฟลอปที่ใช้ประโยชน์กันอย่างกว้างขวางในงานดิจิตอล ฟลิปฟลอปชนิดเจเคที่นิยมใช้ คือ เบอร์ 7476 มีอินพุต 2 ชุด คือ

- อินพุตซิงโครนัส ประกอบด้วยขา J K และ CLK

- อินพุตอะซิงโครนัส ประกอบด้วยขา PS และ ขา CLR ทำงานเหมือนกับ PS และ ขา CLR ของฟลิปฟลอปชนิดดี สัญลักษณ์ และ ตารางความจริงแสดงการทำงานของฟลิปฟลอปชนิดเจเคเบอร์ 7476 แสดงดังรูปที่ 2.21



โมดการทำงาน	อินพุต					เอาต์พุต	
	อะซิงโครนัส		ซิงโครนัส				
	PS	CLR	CLK	J	K	Q	$\bar{Q}$
Asynchronous	0	1	x	x	x	1	0
Asynchronous reset	1	0	x	x	x	0	1
Prohibited	0	0	x	x	x	1	1
Hold	1	1	⎍	0	0	ไม่เปลี่ยนแปลง	
Reset	1	1	⎍	0	1	0	1
Set	1	1	⎍	1	0	1	0
Toggle	1	1	⎍	1	1.	สภาวะตรงข้าม	

รูปที่ 2.21 แสดงสัญลักษณ์ และ ตารางความจริง ของ JK Flip Flip

2.4 ข้อมูลการทำงานของอุปกรณ์อินพุต

อุปกรณ์อินพุตนั้นหมายถึง อุปกรณ์ที่ทำหน้าที่ในการรับอินพุต จากผู้ใช้งานเพื่อส่งต่อไปให้แก่อุปกรณ์ทางลอจิกต่าง ๆ เช่น พวกเกตต่าง ๆ หรือ ฟลิปฟลอปต่าง ๆ หรือ ไม่ว่าจะเป็นอุปกรณ์จำพวกที่เป็นอุปกรณ์เอาต์พุตเลยก็ได้ ซึ่งอุปกรณ์อินพุตที่กล่าวมานี้ก็ได้แก่ อุปกรณ์จำพวกสวิตซ์ต่าง ๆ ซึ่งจะแบ่งออกเป็น 2 ชนิดด้วยกัน คือ

- สวิตซ์ชนิดกดติดปล่อยดับ

ซึ่งเป็นสวิตซ์ที่มีการทำงาน คือ หากกดก็จะเป็นการซ๊อตวงจรเข้าด้วยกัน และหากปล่อยก็จะตัดวงจรออกจากกัน ซึ่งจะใช้ในการจำลองสัญญาณที่มีลักษณะเป็นพัลซ์ได้ ซึ่งมีสัญลักษณ์เป็นรูปที่ 2.22

รูปที่ 2.22 แสดงสัญลักษณ์ของสวิตซ์ชนิดกดติดปล่อยดับ

- สวิตซ์ชนิดกดติดกดดับ

ซึ่งเป็นสวิตซ์ที่มีการทำงาน คือ หากหนึ่งครั้งจะเป็นการซ๊อตวงจรเข้าด้วยกัน ซึ่งหากปล่อยแล้วก็ยังคงซ๊อตวงจรอยู่อย่างนั้นไปเรื่อย ๆ จนกว่าจะมีการกดอีกครั้งหนึ่งจึงจะเป็นการตัดวงจรออกจากกัน ซึ่งจะมีสัญลักษณ์ดังรูปที่ 2.23

รูปที่ 2.23 แสดงสัญลักษณ์ของสวิตซ์ชนิดกดติดกดดับ

2.5 ข้อมูลการทำงานของอุปกรณ์เอาต์พุต

อุปกรณ์เอาต์พุตนั้นก็คือ อุปกรณ์เป็นอุปกรณ์ที่ส่งผลการจำลองการทำงานมาให้แก่ผู้ใช้โปรแกรม โดยรับอินพุตของตัวอุปกรณ์มาจากอุปกรณ์จำพวกเกตต่าง ๆ หรือ อุปกรณ์จำพวกฟลิปฟลอป ซึ่งอุปกรณ์เอาต์พุตเหล่านี้ก็ได้แก่ LED และ 7SEGMENT ซึ่งมีหลักการทำงานดังนี้

- LED ผลการทำงานของมันก็ คือ หลอดจะสว่างเมื่อมีอินพุตเป็นลอจิก "1" และ หลอดจะดับเมื่อลอจิกเป็น "0" ซึ่ง LED นั่นจะมีสัญลักษณ์ในโปรแกรมจำลองการทำงานดังรูปที่ 2.24

รูปที่ 2.24 แสดงสัญลักษณ์ของ LED ในโปรแกรมจำลองการทำงาน

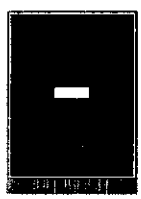
- 7 Segment นั้นมีอยู่ 2 แบบ คือ
  - ○ แบบที่มีอินพุตเป็นแบบ BCD

    แบบนี้การทำงานจะเป็นแบบรับรหัส BCD เข้าไปเพื่อแสดงออกเป็นตัวเลขฐาน16 ออกมาที่เอาต์พุต มีสัญลักษณ์ดังรูปที่ 2.25

รูปที่ 2.25 แสดงสัญลักษณ์ของ 7 Segment แบบรับอินพุตเป็น BCD

  - ○ แบบที่มีอินพุตเป็นแบบ 8 Bits

    แบบนี้การทำงานจะเป็นแบบรับอินพุตแบบ 8 บิตเข้าไปเพื่อแสดงออกเป็นการติดดับของหลอด แต่ละจุดบนตัว 7 Segment ซึ่งจะมีสัญลักษณ์ในโปรแกรมจำลองการทำงาน ดังรูปที่ 2.26



รูปที่ 2.26 แสดงสัญลักษณ์ของ 7 Segment แบบรับอินพุต 8 บิต

บทที่ 3
วิธีการออกแบบโปรแกรมจำลองการทำงานวงจรดิจิตอล

ข้อด้อยของโปรแกรมจำลองการทำงานแบบที่มีอยู่นั้น คือ ใช้งานได้ยาก ด้วยการที่ใช้งานได้ยากนี้ จึงทำให้การพัฒนาฮาร์ดแวร์เป็นไปได้อย่างไม่ราบรื่นนัก และ เป็นเหตุให้การออกแบบ และ พัฒนาฮาร์ดแวร์นั้นเป็นไปได้อย่างช้า ๆ ไม่รวดเร็วเท่าที่ควร

การแก้ปัญหาข้างต้นนี้ เราจึงออกแบบ และ พัฒนาโปรแกรมจำลองการทำงานขึ้นมาใหม่ ให้มีความสะดวก และ คล่องตัวในการใช้งานที่มากขึ้น พร้อมทั้งทำให้ ผู้ใช้ สามารถใช้งานได้อย่างง่ายดาย ไม่ว่าจะเป็น User ที่มีความชำนาญในการออกแบบอยู่แล้ว หรือ ผู้ใช้ที่เป็นผู้ใช้ระดับเริ่มต้น ซึ่งไม่เคยออกแบบ หรือ พัฒนาฮาร์ดแวร์ใด ๆ มาเลย ก็สามารถใช้งานได้อย่างราบรื่น

ในบทที่ 3 นี้จะกล่าวถึงการออกแบบโปรแกรมจำลองการทำงานของวงจรดิจิตอล และแนวคิดต่าง ๆ ที่นำมาใช้ในการออกแบบ พร้อมทั้งแนวคิดต่าง ๆ ในการจำลองการทำงานวงจรดิจิตอล

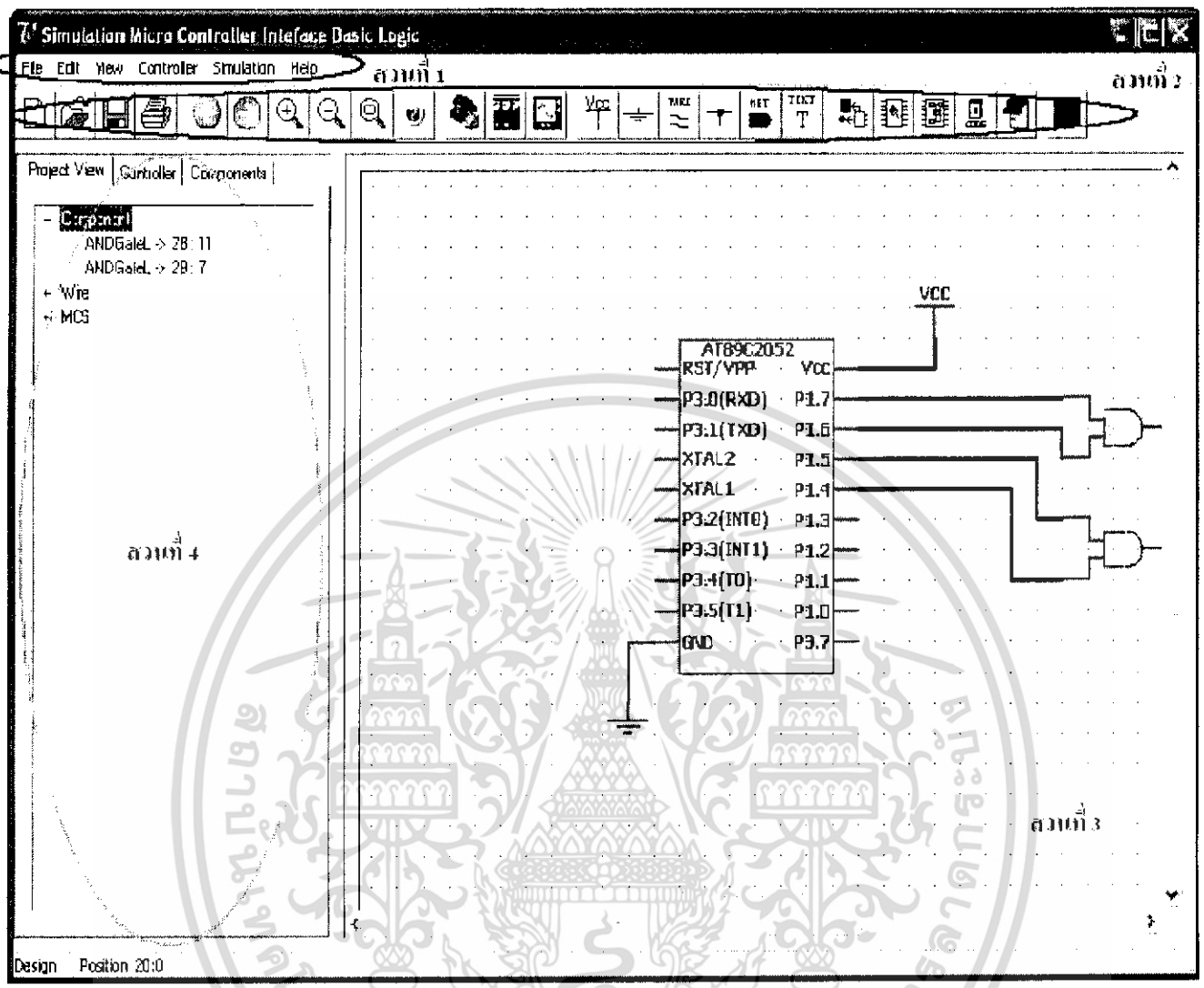
3.1 การออกแบบโครงสร้างการทำงานทางโปรแกรม

3.1.1 การออกแบบส่วนติดต่อผู้ใช้(User interface)

ในการออกแบบส่วนติดต่อผู้ใช้นั้น ได้แบ่ง ออกเป็นส่วน ๆ ได้ 4 ส่วนดังนี้ คือ

1. ส่วนของ Menu Bar (ส่วนที่ 1)
2. ส่วนของ Tool Bar (ส่วนที่ 2)
3. ส่วนของ Circuit Board (ส่วนที่ 3)
4. ส่วนของ Component Bar (ส่วนที่ 4)

ซึ่งลักษณะหน้าตาของโปรแกรมจะมีลักษณะดังรูปที่ 3.1



รูปที่ 3.1 แสดงลักษณะหน้าตาของโปรแกรมจำลองการทำงานที่ได้ออกแบบไว้

3.1.1.1 ส่วนที่ 1 (เมนูบาร์) จะมีเมนูให้เลือกดังนี้

- File
  - ○ New — ใช้ในการสร้าง Project ใหม่
  - ○ Open — ใช้ในการเปิด Project เดิมมาพัฒนาต่อ
  - ○ Save — ใช้ในการเก็บ Project ไว้พัฒนาต่อ
  - ○ Save as — ใช้ในการเก็บ Project ไว้พัฒนาต่อโดยไม่ Save ทับต้นฉบับเดิม
  - ○ Print — ใช้ในการพิมพ์วงจรที่ได้พัฒนาไว้
  - ○ Exit — ออกจากโปรแกรม

- Edit
  - ○ Routing ใช้ในการเรียกหน้าต่างแสดงสมการการเชื่อมต่อ
  - ○ Error Report ใช้ในการเรียกหน้าต่างแสดงความผิดพลาดของวงจร

- View
  - ○ Zoom In ใช้ในการขยายวงจรที่ได้กำลังพัฒนา
  - ○ Zoom Out ใช้ในการย่อขนาดวงจรที่กำลังพัฒนา
  - ○ Fit Screen ใช้ในการปรับเป็นขนาดที่พอดีหน้าจอ
  - ○ Simulation ใช้ในการเปิดหน้าจอจำลองการทำงานขึ้นมา
  - ○ Memory ใช้ในการเปิดหน้าต่างแสดงข้อมูลในหน่วยความจำของ MCS
  - ○ Register ใช้ในการเปิดหน้าต่างแสดงข้อมูลใน Register ออกมาดู
  - ○ Hex Code ใช้ในการเปิดหน้าต่างสำหรับจัดการกับการโหลดไฟล์ .Hex
  - ○ Report ใช้ในการเปิดหน้าต่างสำหรับแสดงรายงานการทำงานของ MCS
  - ○ Show All ใช้เปิดหน้าต่างทั้งหมดที่กล่าวมา

- Controller
  - ○ Properties ใช้ในการกำหนดค่าการทำงานและคุณสมบัติของ Controller

- Simulation
  - ○ Run ใช้ในการจำลองการทำงานแบบต่อเนื่อง
  - ○ Step Run ใช้ในการจำลองการทำแบบเป็น Step
  - ○ Pause ใช้ในการหยุดการจำลองการทำงานแบบชั่วคราว
  - ○ Stop ใช้ในการยกเลิกการจำลองการทำงาน
  - ○ Reset ใช้ในการเริ่มการจำลองการทำงานใหม่ อีกครั้ง

- Help
  - ○ Simulate Help ใช้ในการดูข้อมูลอธิบายการใช้ Simulate
  - ○ Controller Help ใช้ในการดูข้อมูลอธิบายการใช้ Controller

3.1.1.2 ส่วนที่สอง (Tool Bar)

 ใช้ในการสร้าง Project ใหม่

 ใช้ในการเปิด Project เดิมมาพัฒนาต่อ

 ใช้ในการเก็บ Project ไว้พัฒนาต่อ

 ใช้ในการพิมพ์วงจรที่ได้พัฒนาไว้

 ใช้ในการขยายวงจรที่ได้กำลังพัฒนา

 ใช้ในการย่อขนาดวงจรที่กำลังพัฒนา

 ใช้ในการปรับเป็นขนาดที่พอดีหน้าจอ

 ใช้ในการดูข้อมูลอธิบายการใช้งานโปรแกรม

 ใช้เรียกใช้งาน Function Generator

 ใช้เรียกใช้งาน Scope

 ใช้เลือกใช้งาน แหล่งจ่ายไฟ

 ใช้เลือกใช้งาน กราวด์

 ใช้ในการเลือกอุปกรณ์ในการเชื่อมต่อสาย

 เลือกใช้ Junction ในการเชื่อมต่อสายเข้าด้วยกัน

 เลือกใช้ Net เพื่อเป็นทางเชื่อมต่อแทนสาย

 ใช้ในการสร้างลาเบลเพื่อคอมเม็นอุปกรณ์

 ใช้ในการเปิดหน้าจอจำลองการทำงานขึ้นมา

 ใช้ในการเปิดหน้าต่างแสดงข้อมูลในหน่วยความจำของ MCS

 ใช้ในการเปิดหน้าต่างแสดงข้อมูลใน Register ออกมาดู

 ใช้ในการเปิดหน้าต่างสำหรับจัดการกับการโหลดไฟล์ .Hex

 ใช้ในการเปิดหน้าต่างสำหรับแสดงรายงานการทำงานของ MCS

 ใช้ในการเลือกสีของสายที่ใช้เชื่อมต่อวงจร

3.1.1.3 ส่วนที่ 3 Circuit Board

จะเป็นส่วนที่ใช้ในการสร้างวงจร เพื่อที่จะนำมาทำการ Simulate การทำงานของวงจร จะเป็นหน้าว่างเปล่า ซึ่งมีจุด Grid เพื่อช่วยบอกพิกัดในการวางอุปกรณ์ และ เชื่อมต่อสาย ซึ่งอุปกรณ์นั้นจะสามารถหามาได้จากส่วนที่ 4 ทางซ้ายมือ หรือ จาก Tool Bars บางตัว มาใช้งานได้

3.1.1.4 ส่วนที่ 4 (Component Bar)

นั้นเป็นส่วนที่ใช้ในการเลือกอุปกรณ์ลงมาวางเรียงกันในส่วนที่ 3 นั้นเอง ซึ่งจะแบ่งเป็น 3 ช่อง ย่อย ๆ ได้แก่

- Project View

  ในส่วนนี้จะเป็นส่วนที่บอกรายละเอียดในการเชื่อมต่อ และ บอกรายละเอียดของอุปกรณ์ที่เรานำมาวางว่ามีอะไรบ้าง และ อยู่ที่พิกัดไหน

- Controller

  ในส่วนนี้จะเป็นส่วนที่จะใช้ในการเลือกอุปกรณ์ที่เป็นคอนโทรลเลอร์ ซึ่งทำไว้เผื่อพัฒนาต่อ ในภายภาคหน้า

- Components

  ในส่วนนี้จะเป็นส่วนที่จะใช้ในการเลือกอุปกรณ์ที่เป็นอุปกรณ์ที่นำมาต่อร่วม ซึ่งจะแบ่งออกเป็น 5 ประเภทย่อย คือ

  - Basic Gate ได้แก่อุปกรณ์จำพวก Gate ทั่วไป
  - Flip Flop ได้แก่อุปกรณ์จำพวก ที่มีการเก็บสถานะได้
  - TTL ได้แก่อุปกรณ์จำพวกที่เป็นรูปแบบ IC
  - I/P Device ได้แก่อุปกรณ์จำพวกที่เป็นอุปกรณ์ที่รับข้อมูลจากUser
  - O/P Device ได้แก่อุปกรณ์จำพวกที่แสดงผลการทำงานให้ User ทราบ

3.1.2 ลำดับการทำงานโดยรวมของระบบ

- สร้างและกำหนด Library กับอุปกรณ์เชื่อมต่ออื่นๆ
- สร้างวงจรและทำการเชื่อมต่อความสัมพันธ์
- สร้างเส้นทางการเชื่อมต่อ
- สร้างสมการการทำงานจากการเชื่อมต่อ
- จำลองการทำงานและแสดงผล

3.2 หลักการออกแบบ library

การออกแบบ library นั้นมีแนวคิด และ ขอบเขตว่าอุปกรณ์ ชนิดใด ๆ ที่ต้องการสร้าง ใช้งาน ต้องประกอบไปด้วย ข้อมูลเบื้องต้น ตำแหน่งการเชื่อมต่อ และ รูปแบบการทำงานโดย ส่วนประกอบแต่ละส่วนนั้นจะมีโครงสร้าง และ หลักการดังต่อไปนี้

3.2.1 ข้อมูลเบื้องต้นของอุปกรณ์ (Library Component)

- Name ชื่ออุปกรณ์ โดยมีความสัมพันธ์เป็น ตัวบ่งชี้ประจำอุปกรณ์ ซึ่งใช้ในการอ้างถึง
- Type ชนิดของอุปกรณ์ เพื่อจำแนกประเภท หรือ จัดกลุ่ม
- Pin จำนวนขาที่สามารถใช้ได้ในการแสดงผล เพื่อออกแบบ
- BmpName ไฟล์ภาพที่ใช้ในการแสดงผล เพื่อออกแบบ
- Description ข้อมูลเพิ่มเติมเกี่ยวกับอุปกรณ์ชนิดนี้

3.2.2 ตำแหน่งสำหรับการเชื่อมต่อ (Pin table)

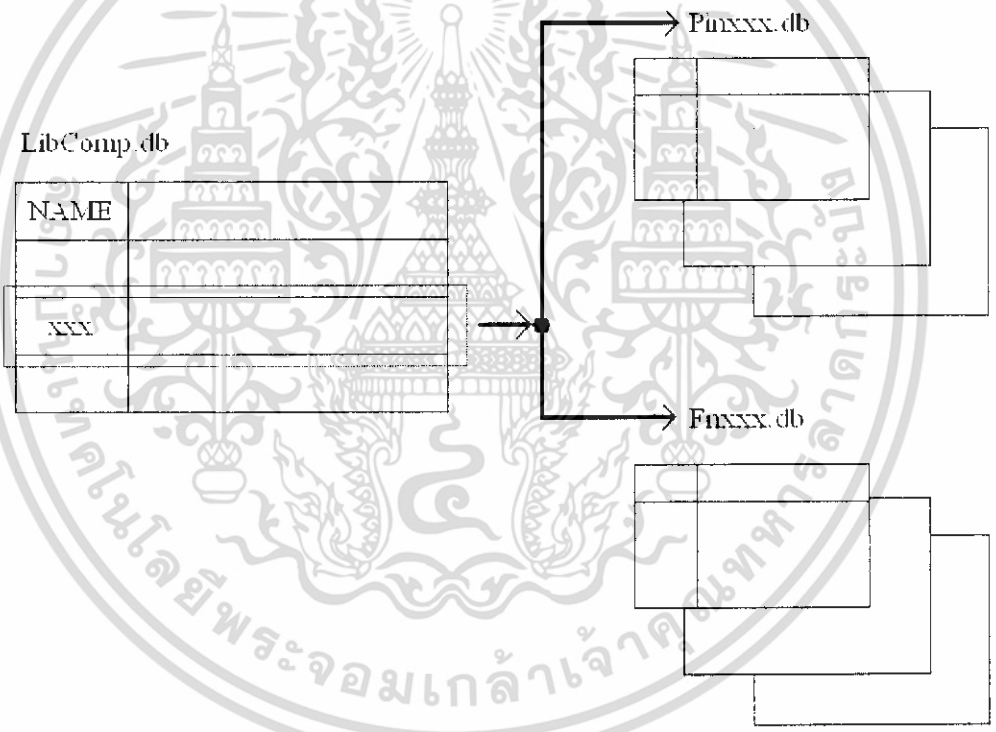
-PinName ชื่อประจำขาอุปกรณ์

- Type ชนิดการเชื่อมต่อของขาอุปกรณ์ โดยมี
  - NC >> ไม่ใช้งาน
  - IN >> ขารับสัญญาณ
  - OUT >> ขาส่งสัญญาณออก
  - Bidirection >> ขาที่สามารถรับ และ ส่งได้พร้อมกัน
- X ตำแหน่งอ้างอิงบนภาพทางแนวนอน
- Y ตำแหน่งอ้างอิงบนภาพทางแนวตั้ง
- Description ข้อมูลเพิ่มเติมประจำขาอุปกรณ์

3.2.3 รูปแบบการทำงาน (Function table)

- PinName [Pins]	ชื่อขาอุปกรณ์ มี Field จำนวนเท่ากับจำนวนขาเชื่อมต่อ
- PresentState	ค่าปัจจุบัน ใช้สำหรับตรวจสอบสถานะ
- NextState	ค่าในอนาคต ใช้กำหนดสถานะใหม่ให้แก่อุปกรณ์
- EventState	การกระทำจากผู้ใช้โดยมี
	MOUSE ON >> เป็นจริงเมื่อ mouse ชี้อยู่บนอุปกรณ์ชนิดนี้
	MOUSE CLICK >> เป็นจริงเมื่อ mouse click บนอุปกรณ์
- Picturename	ภาพแสดงผลในสถานะปัจจุบัน

ซึ่งความสัมพันธ์ทางฐานข้อมูลของการจัดเก็บ Libraly จะมีลักษณะดังรูปที่ 3.2

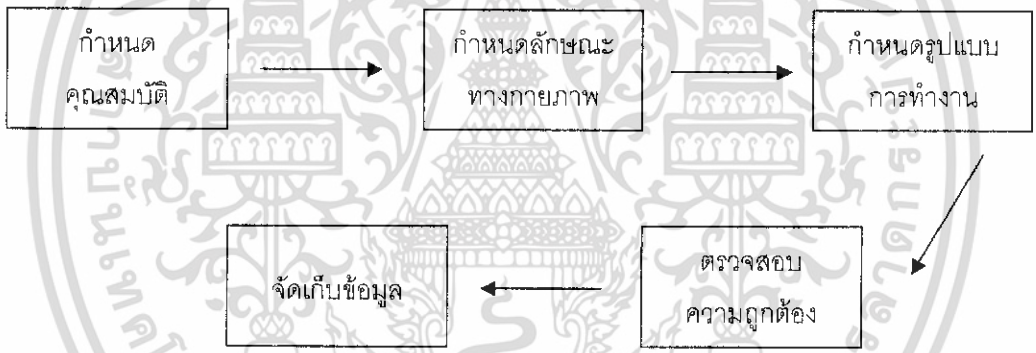


รูปที่ 3.2 แสดงความสัมพันธ์ของข้อมูลในฐานข้อมูลที่ใช้เก็บ Libraly

จากนั้นจึงนำข้อมูลดังกล่าวมาทำตามกระบวนการดังนี้

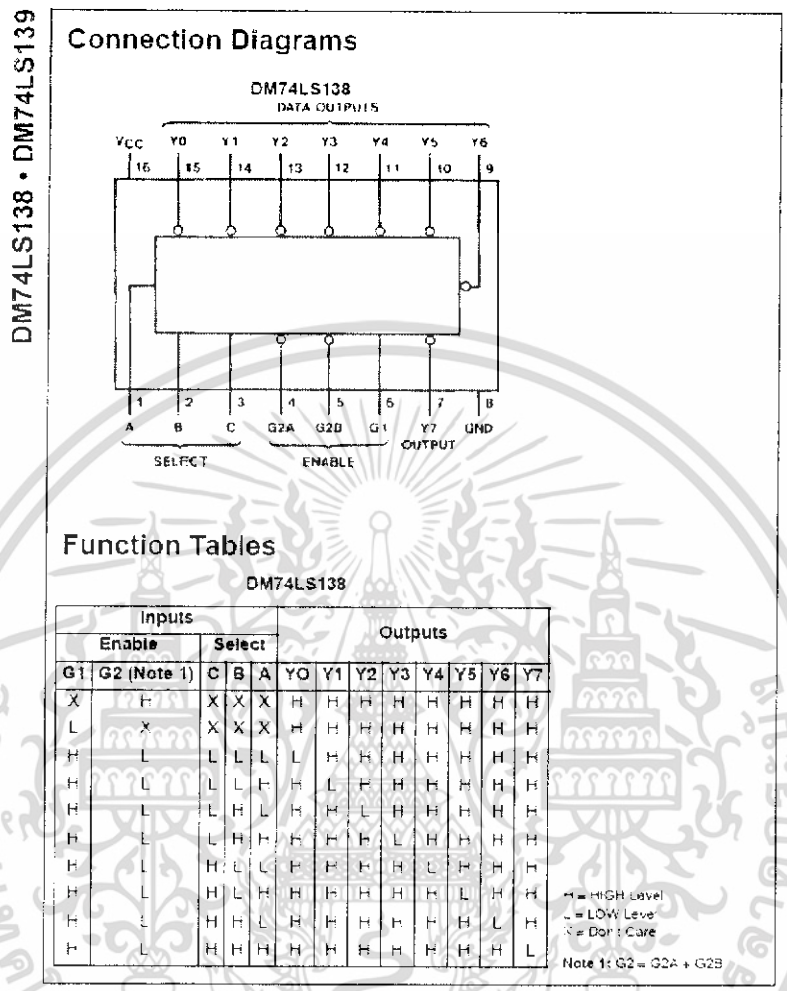
1. กำหนดคุณสมบัติทางกายภาพเบื้องต้น
2. กำหนดคุณสมบัติประจำขาอุปกรณ์ทุกขา
3. สร้างตารางการทำงาน (truth table) เพื่อกำหนดคุณสมบัติการทำงานของอุปกรณ์
4. ตรวจสอบ การทำงานเพื่อตรวจสอบความถูกต้อง
5. จัดเก็บข้อมูล library

ซึ่งการทำงานทั้งหมดที่กล่าวมานี้จะแสดงออกมาให้เห็นได้ดังรูปที่ 3.3



รูปที่ 3.3 แสดงการทำงานของกระบวนการทำงานเบื้องต้นของโปรแกรมจำลองการทำงาน

ตัวอย่าง การสร้างอุปกรณ์ 74138

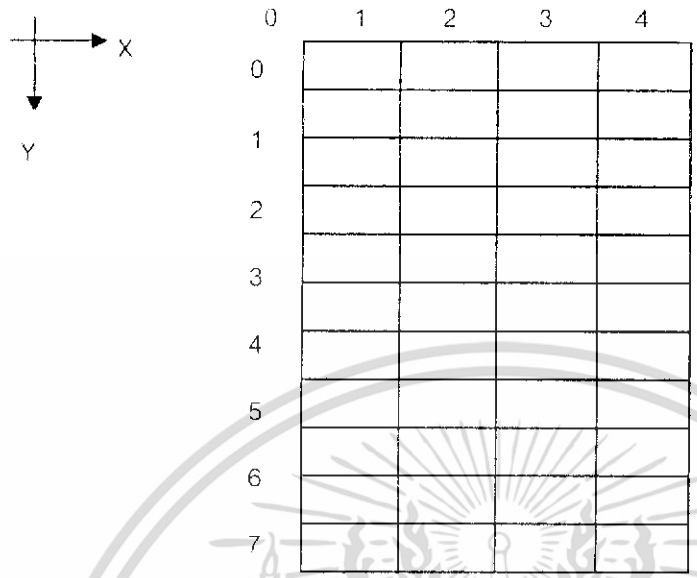


รูปที่ 3.4 แสดงภาพ Datasheet เบอร์ 74138

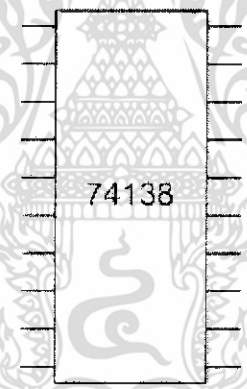
กระบวนการที่ 1 กำหนดคุณสมบัติทางกายภาพเบื้องต้น

Id	=	00001
Type	=	IC
Name	=	DM74LS138
Picture	=	library/Pic/74138.bmp
Pins	=	16
Description	=	3 – Line – to – 8 – Line Decoder

จากกระบวนการที่ 1 จะได้

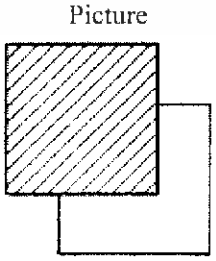


Grid layer



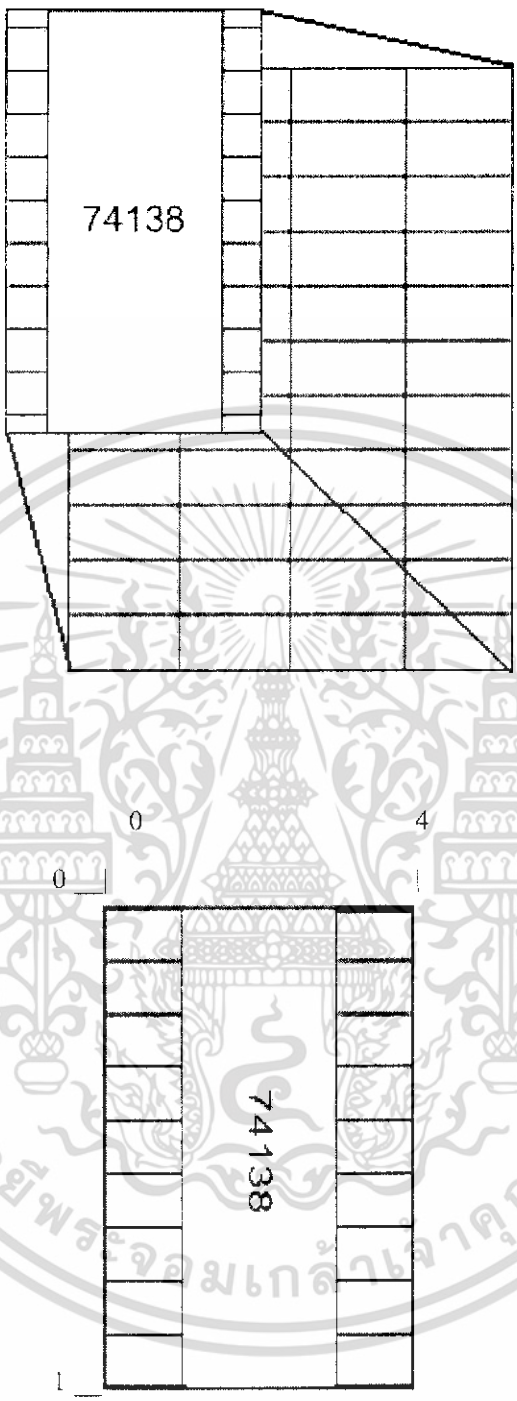
Picture = library / Pic / 74138.bmp

จากนั้นนำภาพมาซ้อน เพื่อให้ได้ซึ่งตำแหน่งขาอุปกรณ์



Grid layer

รูปที่ 3.5 แสดงการหาตำแหน่งพิกัดของขาอุปกรณ์



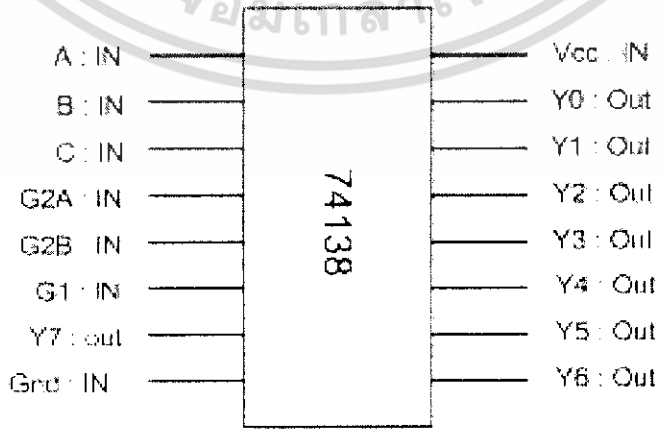
รูปที่ 3.6 แสดงการหาตำแหน่งพิกัดของขาอุปกรณ์(ต่อ)

กระบวนการที่ 2 กำหนดคุณสมบัติของขาอุปกรณ์ทางกายภาพ

Number	Name	Type	Pin X	Pin Y
1	A	IN	0	1
2	B	IN	0	2
3	C	IN	0	3
4	G2A	IN	0	4
5	G2B	IN	0	5
6	G1	IN	0	6
7	Y7	OUT	0	7
8	GND	NC	0	8
9	Y6	OUT	4	1
10	Y5	OUT	4	2
11	Y4	OUT	4	3
12	Y3	OUT	4	4
13	Y2	OUT	4	5
14	Y1	OUT	4	6
15	Y0	OUT	4	7
16	VCC	NC	4	8

รูปที่ 3.7 แสดงตารางคุณสมบัติของขาอุปกรณ์ทางกายภาพ

ถ้าขา VCC กับ GND กำหนด type เป็น NC เพราะโปรแกรมทำงานทางอุดมคติ



รูปที่ 3.8 แสดงภาพของไอซีที่นำมาเก็บเป็นค่าในตารางในรูปที่ 3.7 ที่ผ่านมา

กระบวนการที่ 3 สร้างรูปแบบการทำงานของอุปกรณ์

Inputs					Output							
Enable		Select										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

รูปที่ 3.9 แสดงตารางที่เก็บ Function การทำงานของไอซีที่จะเก็บใน Library

Enable		Select			State Message			
G1	G2*	C	B	A	PresentState	NextState	EventState	Picturename
X	H	X	X	X	0	0	-	-
L	X	X	X	X	0	0	-	-
H	L	L	L	L	0	0	-	-
H	L	L	L	H	0	0	-	-
H	L	L	H	L	0	0	-	-
H	L	L	H	H	0	0	-	-
H	L	H	L	L	0	0	-	-
H	L	H	L	H	0	0	-	-
H	L	H	H	L	0	0	-	-
H	L	H	H	H	0	0	-	-

รูปที่ 3.10 แสดงตารางความสัมพันธ์พิเศษทาง State และ Messageของไอซีที่จะเก็บใน Library

กระบวนการที่ 4

- ตรวจสอบความถูกต้องของ truth table โดย ทำการสร้าง Input ทุกๆเหตุการณ์แล้วทำการ Test เพื่อหา Output หากมีเหตุการณ์ใดไม่สามารถแจกแจงหรือแสดงผลของ Output ได้เพียงเหตุการณ์เดียว แสดงว่าเกิดความผิดพลาด ระบบจะใช่ค่า default ให้แทนพร้อมกับแจ้งข้อผิดพลาด

กระบวนการที่ 5

- เก็บข้อมูลลง library เพื่อจบสมการนี้

3.3 การออกแบบวงจร

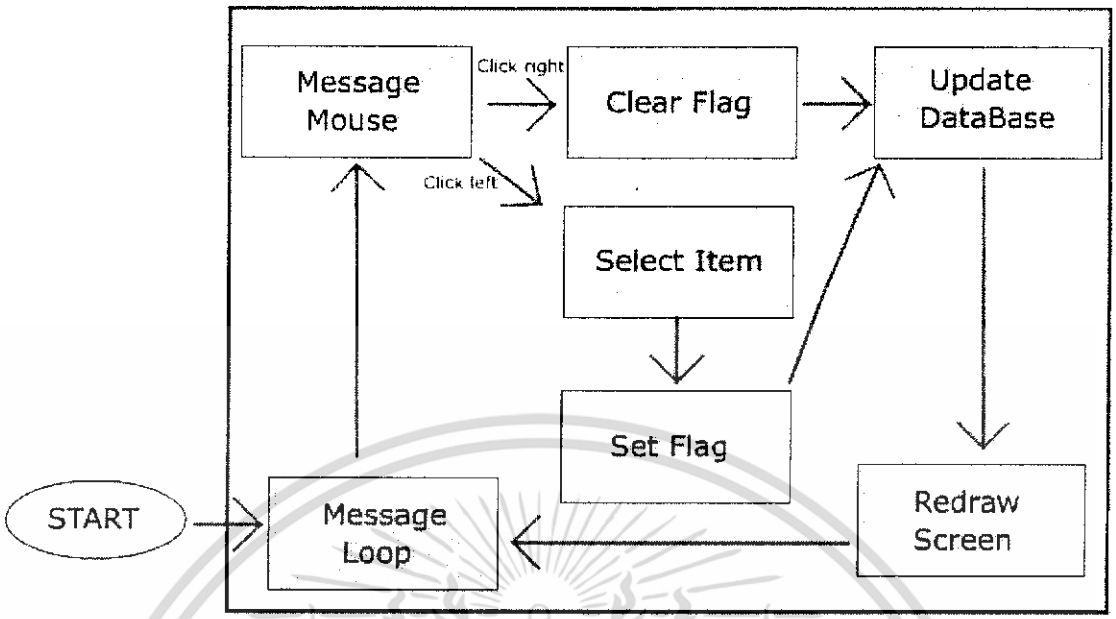
กระบวนการออกแบบวงจรนั้นต้องประกอบไปด้วยข้อมูลหลักดังนี้

- ข้อมูลการจัดวางตำแหน่งและการเชื่อมต่ออุปกรณ์
- หน้าจอเสมือนเพื่อนำอุปกรณ์มาวาดก่อนแสดงผลจริง
- หน้าจอแสดงผลหลัก เพื่อนำภาพจากหน้าจอเสมือนมาแสดงผล

ซึ่งข้อมูลการจัดวางอุปกรณ์ประกอบไปด้วย

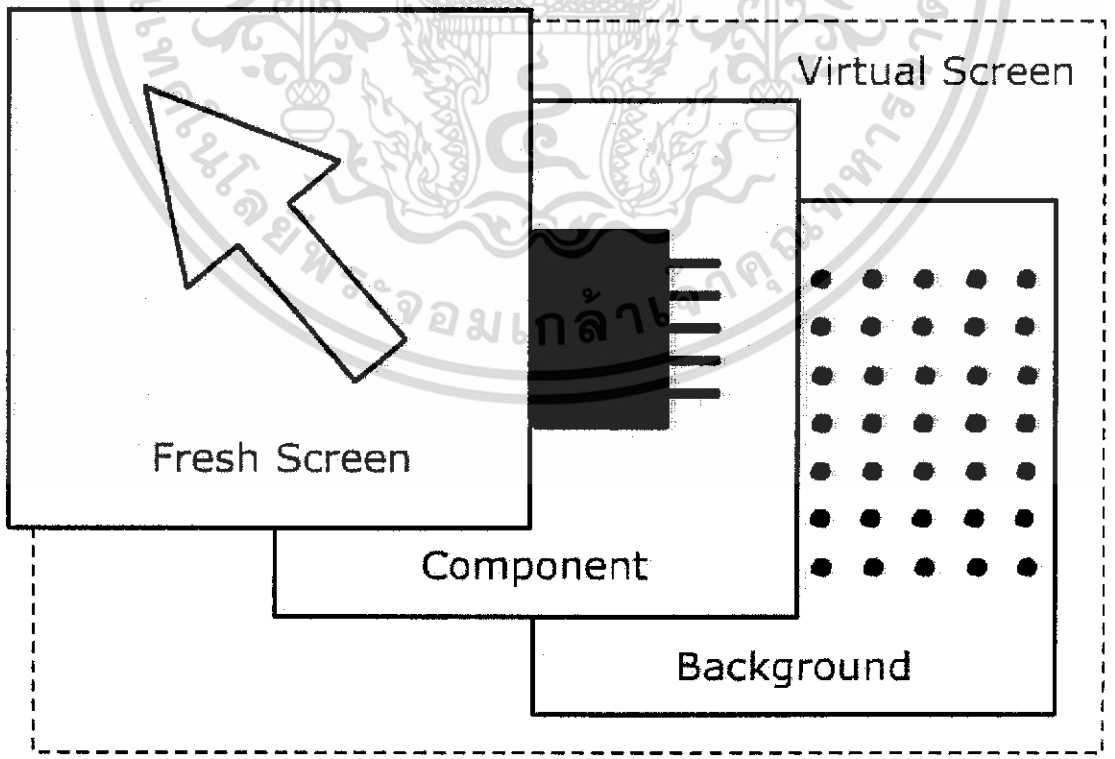
- Serial รหัสประจำตัวอุปกรณ์มีหน้าที่เป็นตัวบ่งชี้อุปกรณ์ในวงจร
- LName ชื่อประจำตัวอุปกรณ์เพื่อระบุอุปกรณ์
- X1 ตำแหน่งอุปกรณ์ทางแนวนอน
- X2 ตำแหน่งอุปกรณ์ทางแนวตั้ง
- X2 ตำแหน่งอุปกรณ์ปลายทางแนวนอน
- Y2 ตำแหน่งอุปกรณ์ปลายทางแนวตั้ง
- Flag ตำแหน่งอุปกรณ์แสดงหมายเลขของอุปกรณ์
    1. อุปกรณ์จาก Library
    2. Vcc
    3. GND
    4. Line จากการลากเส้น
    5. Junction
    6. Net
    7. Text
    8. AT89C52
    9. AT89C2051
- History ลำดับการจัดวางอุปกรณ์

ส่วนในการเลือกลงอุปกรณ์จะมีแนวคิดในการทำงานของโปรแกรมดังในรูปที่ 3.11



รูปที่ 3.11 แสดงแนวคิดการทำงานของโปรแกรมในส่วนของการลากอุปกรณ์มาวาง

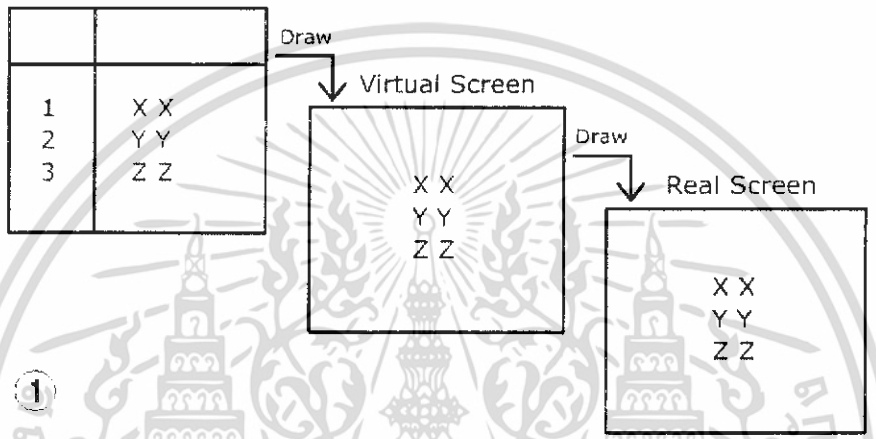
3.3.1 หน้าจอเสมือน



รูปที่ 3.12 แสดงแนวคิดในการออกแบบในส่วนของหน้าจอเสมือน

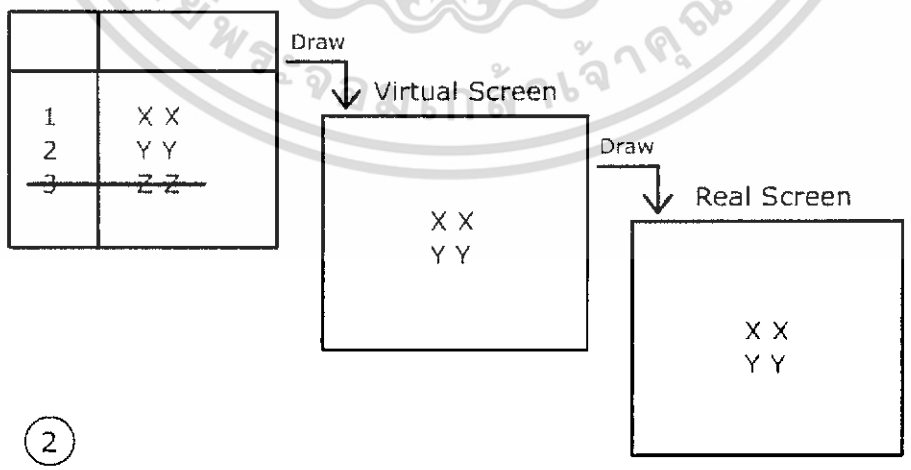
หน้าจอเสมือนทำหน้าที่เป็น หน่วยความจำหน้าจอสำรองก่อนที่จะทำการวาดวงจรขึ้นหน้าจอใหม่อีกครั้งโดยจะมีการตรวจสอบ การวาดอยู่ 2 รูปแบบใหญ่ๆ โดยมีการวาดทั้งหน้าจอใหม่จากฐานข้อมูลการจัดวางอุปกรณ์ และการวาดหน้าจอใหม่จากหน่วยความจำล่าสุดซึ่งมีหลักการทำงานดังนี้

- การวาดหน้าจอใหม่จากฐานข้อมูล ใช้ในเพื่อแสดงอุปกรณ์ที่เหลือหลังจากมีการเปลี่ยนแปลงข้อมูลของอุปกรณ์ในฐานข้อมูล



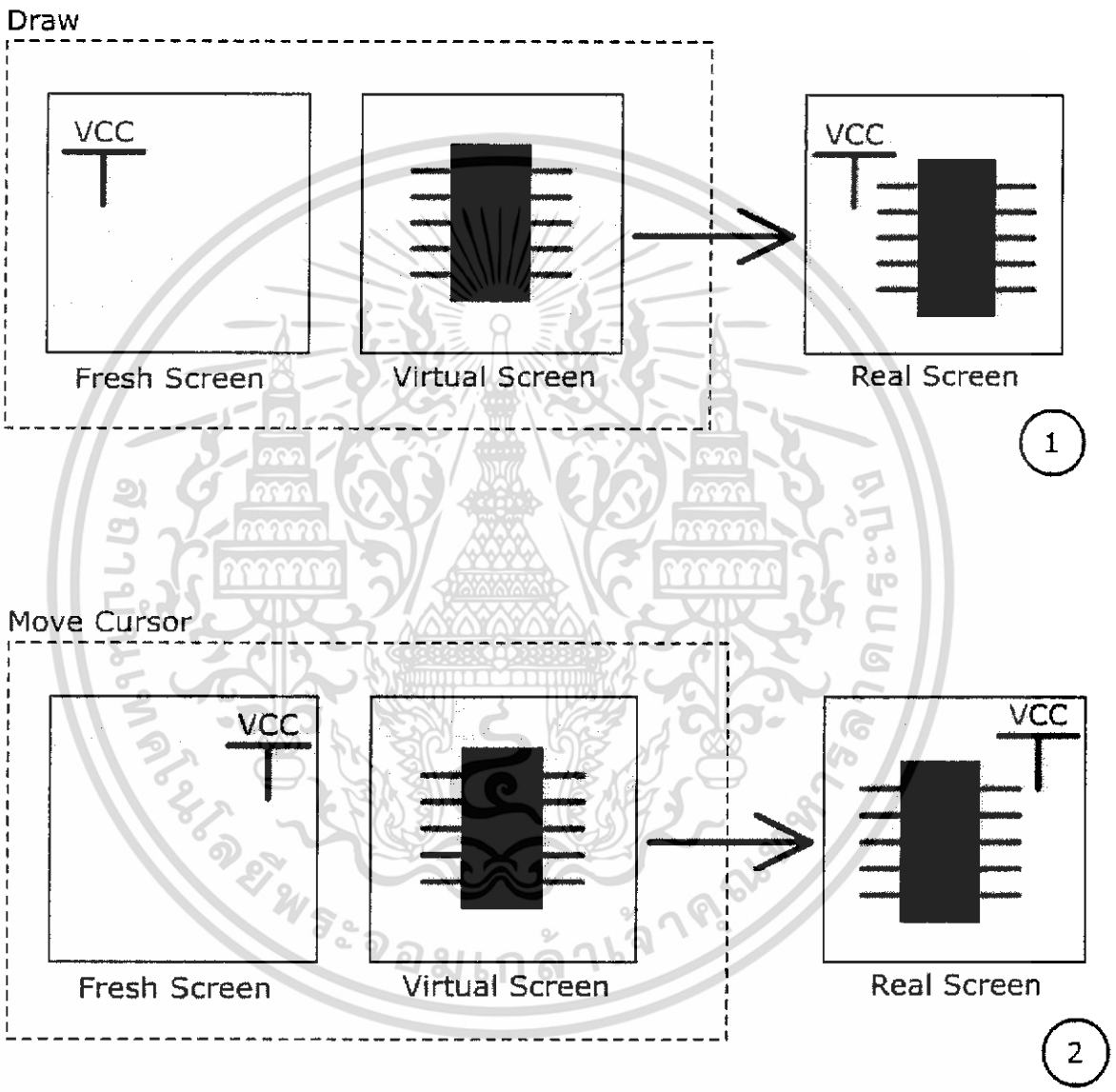
รูปที่ 3.13 แสดงภาพการทำงานของหน้าจอเสมือน

- การวาดหน้าจอใหม่จากหน่วยความจำล่าสุด ใช้เพื่อแสดงผลหลังจากการวาดภาพเดิมๆ ที่ตำแหน่งใหม่ สังเกตุได้ว่าที่ Virtual Screen ไม่มีความจำเป็นในการวาดหน้าจอใหม่อีกครั้ง



รูปที่ 3.14 แสดงภาพการทำงานของหน้าจอเสมือน (ต่อ)

เมื่อได้ใช้การออกแบบในส่วนของหน้าจอเสมือนมาช่วยแล้ว ก็จะทำให้การทำงานในส่วนของการวาดภาพใหม่เมื่อมีการย้ายที่อุปกรณ์จากที่หนึ่งไปยังอีกที่หนึ่งเป็นไปได้อย่างราบรื่นยิ่งกว่าเดิมดังภาพที่ 3.15

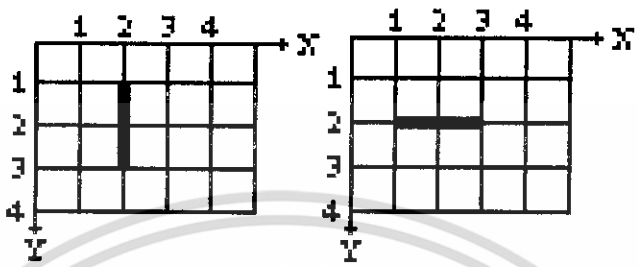


รูปที่ 3.15 แสดงผลการทำงานของหน้าจอเสมือน เมื่อมีการย้ายอุปกรณ์

3.4 การสร้างสมการการเชื่อมต่อ (การค้นหาเส้นทาง)

3.4.1 การสร้างเส้นทาง

สามารถสร้างได้จากการลากสายที wire โดยการสร้างเส้นทางมีเงื่อนไข การสร้างเส้นทางว่าจะต้องลากเป็นเส้นตรงเพียงเท่านั้น ซึ่งการเชื่อมต่อแบบเส้นตรงนั้นสามารถตรวจสอบจุดต่อ (Junction) ได้อย่างง่ายและชัดเจน โดยมีโครงร่างการใช้งานดังนี้

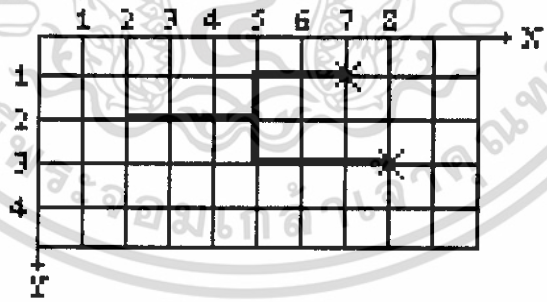


รูปที่ 3.16 แสดงการลากเส้นย่อยที่จะเกิดขึ้นได้เมื่อต่อวงจร

Name line (color)	Name Line (color)
X1 = 2	X1 = 1
Y1 = 1	Y1 = 2
X2 = 2	X2 = 3
Y2 = 3	Y2 = 2

รูปที่ 3.17 แสดงข้อมูลการลากเส้นที่ได้จากแต่ละเส้นในรูป 3.16

การค้นหาเส้นทางแบบหลายเส้นทาง



(ก)

Line (2,2 , 5,2)

Line (5,2 , 5,3)

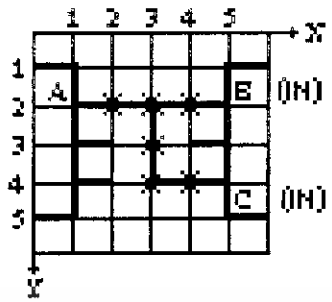
Line (5,3 , 8,3)

Line (5,1 , 7,1)

(ข)

รูปที่ 3.18 (ก)แสดงการลากเส้นในวงจร (ข) แสดงข้อมูลการลากเส้นที่เกิดขึ้น

3.4.2 การค้นหาเส้นทาง

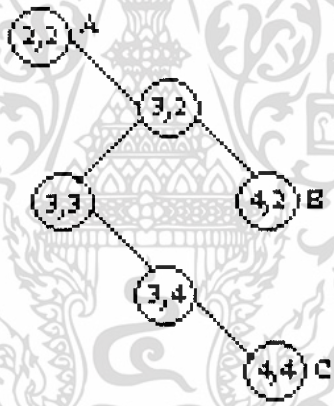


รูปที่ 3.19 แสดงการลากเส้นที่จะเกิดขึ้นได้เมื่อต่อวงจร

จากเส้นทางสามารถนำมาแตก Tree โดยมีเงื่อนไขดังนี้

- การแตก Tree จะไม่ย้อนกลับทางเดิม
- การแตก Tree จะไม่แตก Node ที่เคยมีอยู่แล้ว

Tree ของการค้นหาเส้นทาง



รูปที่ 3.20 แสดงTree ที่เกิดขึ้นจากการต่อวงจรในรูปที่ 3.19

การแตก Tree จะเห็นได้ว่า

A -> B

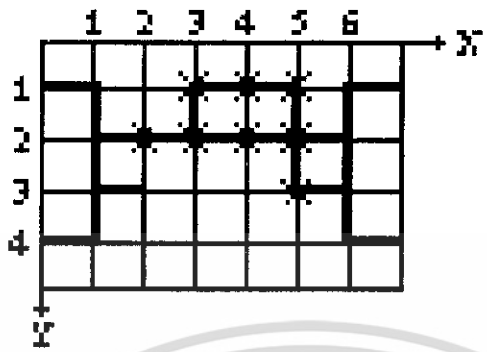
A -> C

นำการแตก Tree มาเก็บเป็นข้อมูลลงฐานข้อมูล

PinName	Relation
B	A
C	A

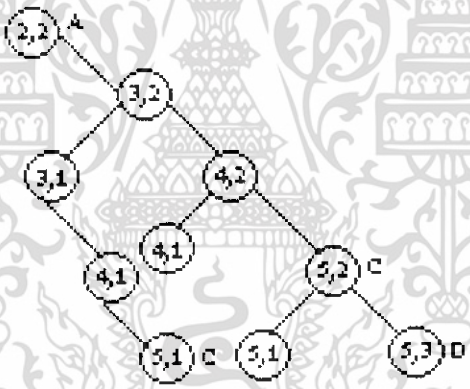
รูปที่ 3.21 แสดงตารางที่จะใช้ในการเก็บข้อมูลการต่อวงจรลงในฐานข้อมูลจริง

3.4.3 การค้นหาเส้นทางแบบมี Loop



รูปที่ 3.22 แสดงการลากเส้นที่จะเกิดขึ้นได้เมื่อต่อวงจรที่มีลักษณะเป็น Loop

ตามเงื่อนไขในการหาเส้นทางในหน้าที่ผ่านมาการแตก Tree จะไม่ทำย้อนกลับไปทางที่เคยทำผ่านมาแล้วก็จึงทำให้ Tree ที่ได้ออกมามีลักษณะดังภาพที่ 3.23



รูปที่ 3.23 แสดงTree ที่เกิดขึ้นจากการต่อวงจรในรูปที่ 3.22

สมการหาเส้นทาง

A->C

A->D

ฐานข้อมูลสมการเส้นทาง

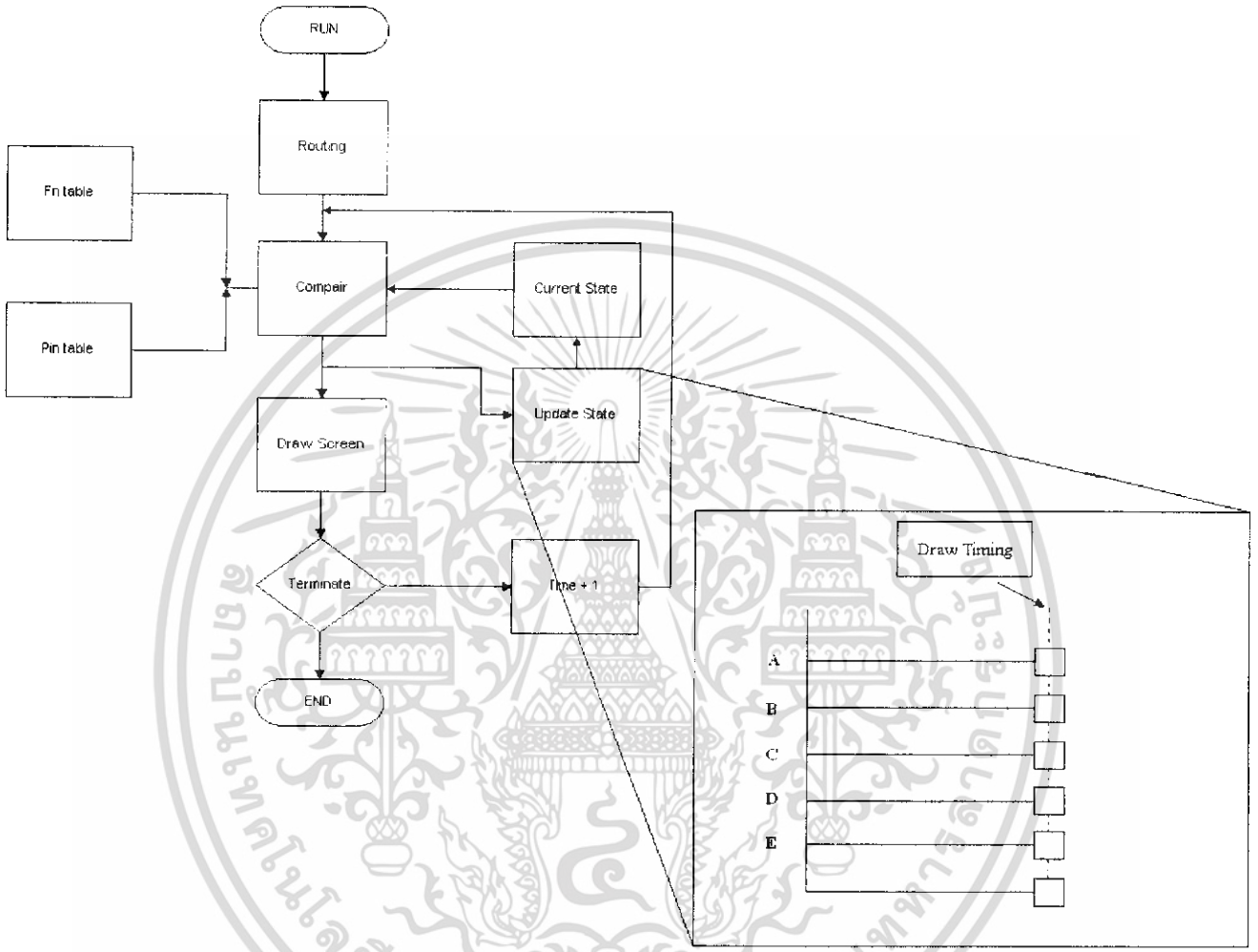
Serial	Pin Name	Relation
2	C	A
2	D	A
1	B	1

สถานะขาลอยมีค่าเป็น 1

รูปที่ 3.24 แสดงข้อมูลที่เกิดขึ้นจากการต่อวงจรในรูปที่ 3.22

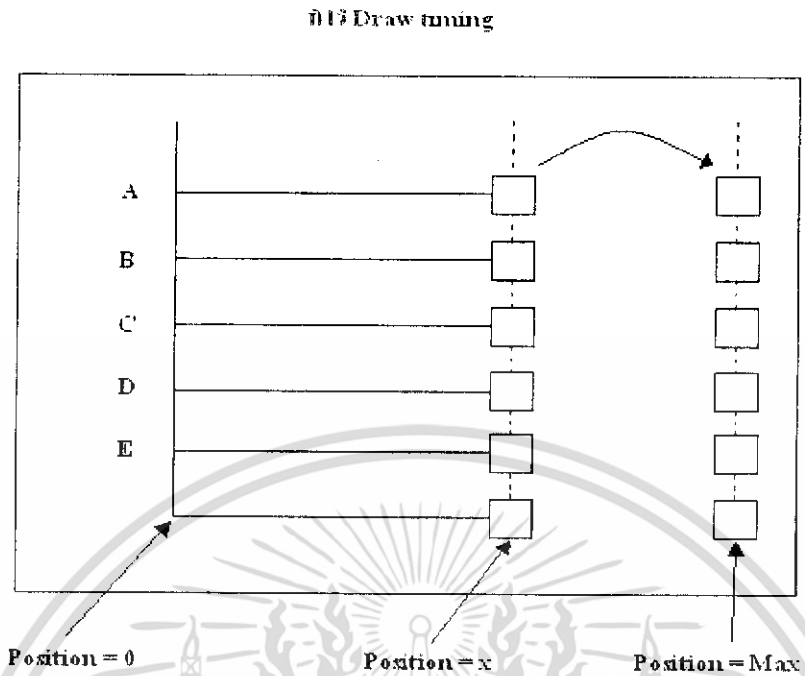
3.5 Timing

ในส่วนของ Timing นั้นจะมีลักษณะการทำงานดังรูปที่ 3.25



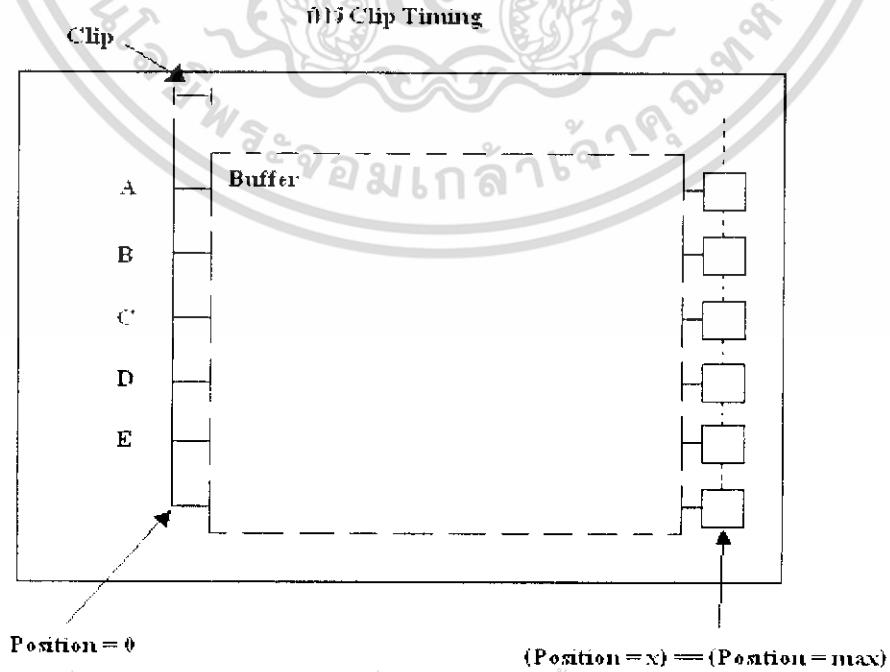
รูปที่ 3.25 แสดงโฟลวชาร์ตการทำงานเมื่อมีการรัน

ซึ่งเมื่อมีการรันจะมีขั้นตอนการทำงานดังนี้ คือ จะทำการเร้าหาเส้นทางว่าอะไรต่อกับอะไรมั่ง จากนั้นก็จะนำไปเปรียบเทียบกับตารางความจริงของผลการทำงานในระบบฐานข้อมูล และ Current State เพื่อจะได้ผลออกมาแล้วผลที่ได้นั้นก็จะถูกนำไปอัพเดทState และนำมาวาดลงบนจอไปพร้อม ๆ กัน แล้วเช็คดูว่ามีการจบโปรแกรมหรือไม่ หากมีการหยุดก็จะหลุดออกไปที่ End ก็เป็นการจบโปรแกรมย่อยส่วนนี้ และ ในขณะเดียวกันที่มีการ Update State เราก็ยังสามารถนำเอาข้อมูลขณะช่วงเวลานั้นมาแสดงผลเป็น Timing Diagram ได้อีกด้วย ซึ่งลักษณะการวาด Timing Diagram จะมีลักษณะดังรูปที่ 3.26



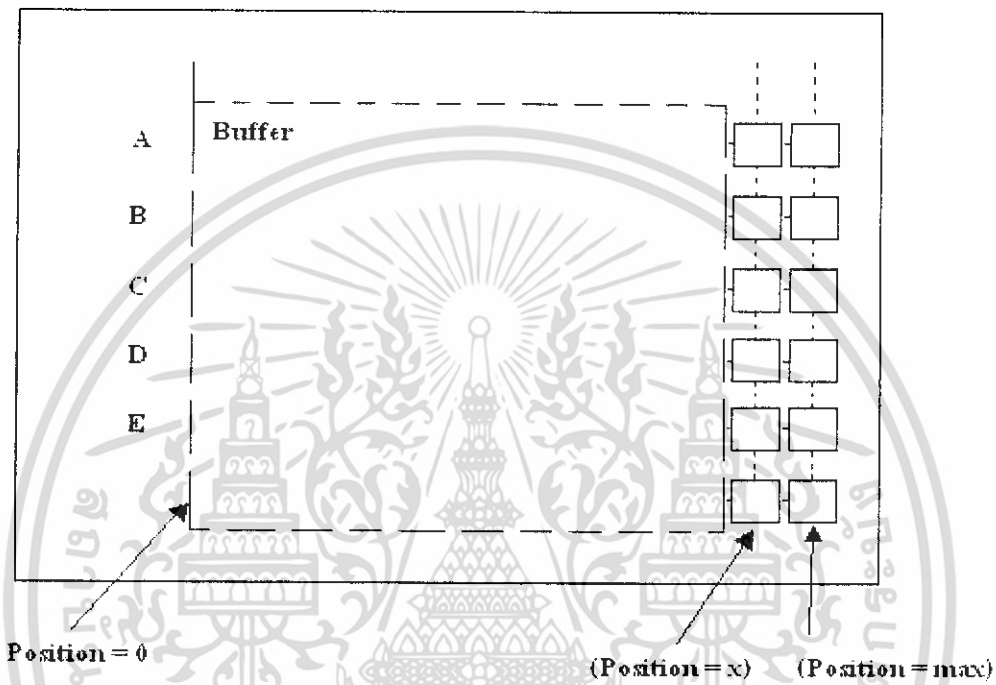
รูปที่ 3.26 แสดงการวาด timing Diagram

ซึ่งแนวคิดในการวาด Timing Diagram ก็คือ จะวาดข้อมูล ณ เวลาต่าง ๆ โดยข้อมูลเหล่านั้นก็คือ Output ที่ขาอุปกรณ์ต่าง ๆ โดยจะวาดเป็นช่วงเวลาไปเรื่อย ๆ ซึ่งข้อมูล Timing Diagram นี้จะถูกเก็บไว้ในเมมโมรี่ ที่จำกัดขนาดไว้ ดังนั้นเมมโมรี่นี้จึงมีขนาดที่จำกัด ด้วยเหตุนี้เองเมื่อวาดไปจะเมมโมรี่เต็มแล้วจึงต้องมีการจัดการเมมโมรี่ของ Timing Diagram ด้วย ซึ่งจะมีวิธีการดังรูปที่ 3.27



รูปที่ 3.27 แสดงการจัดการเมมโมรี่ของ Timing

ซึ่งขั้นตอนในการจัดการเมโมรี่ส่วนนี้ก็คือ จะทำการเขียนเมโมรี่ ณ ช่วงเวลาต่าง ๆ ไปเรื่อย ๆ จนเมื่อ Position x เท่ากับ Position max ก็จำทำการก๊อบปี้ส่วนที่ผ่าน ๆ มาตั้งแต่ Position 1 ไปจนถึง Position x แล้วทำการเลื่อนทั้งหมดไปเริ่มใหม่ที่ Position 0 จากนั้นก็จะเหลือพื้นที่สำหรับเขียนเมโมรี่เพิ่มขึ้นมาดังรูปที่ 3.28



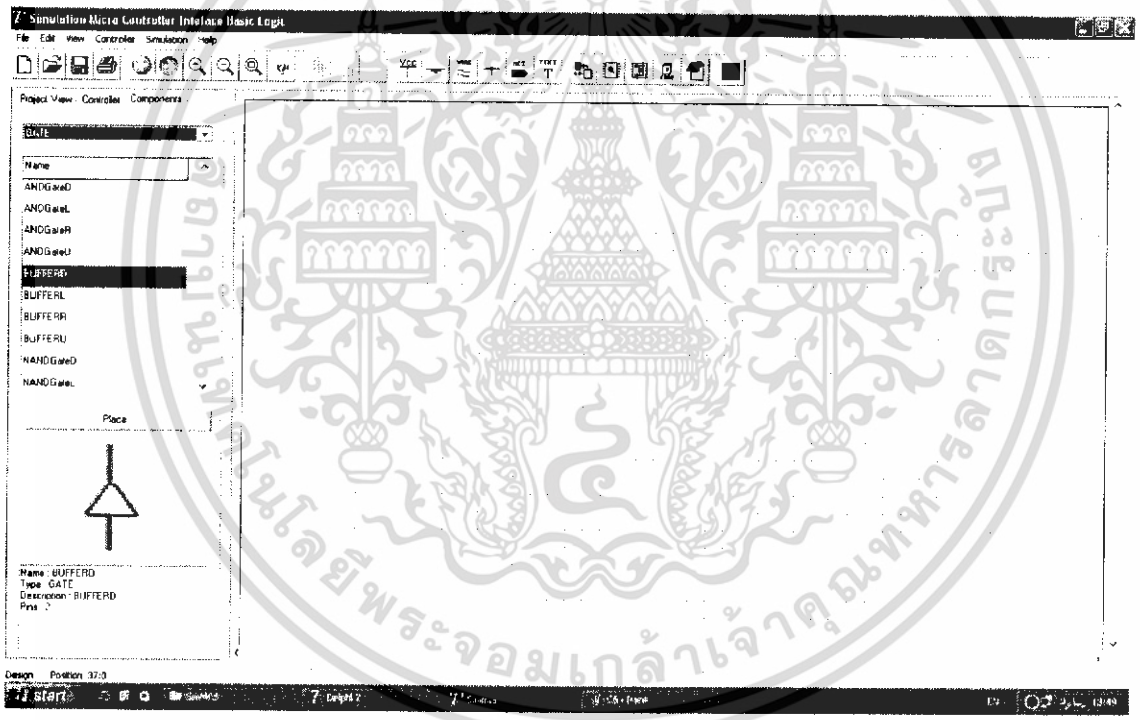
รูปที่ 3.28 แสดงการจัดการเมมโมรี่ของ Timing เมื่อเมโมรี่เต็ม

บทที่ 4
การทดลองและผลการทดลองโปรแกรม

ในบทนี้จะกล่าวถึงหน้าตาของโปรแกรม และ ผลการทำงานของโปรแกรมจำลองการทำงานที่ได้ออกแบบ และ พัฒนาขึ้นมาจนสมบูรณ์แบบ

4.1 รูปร่างลักษณะหน้าตาโปรแกรมจำลองการทำงานวงจรดิจิตอล

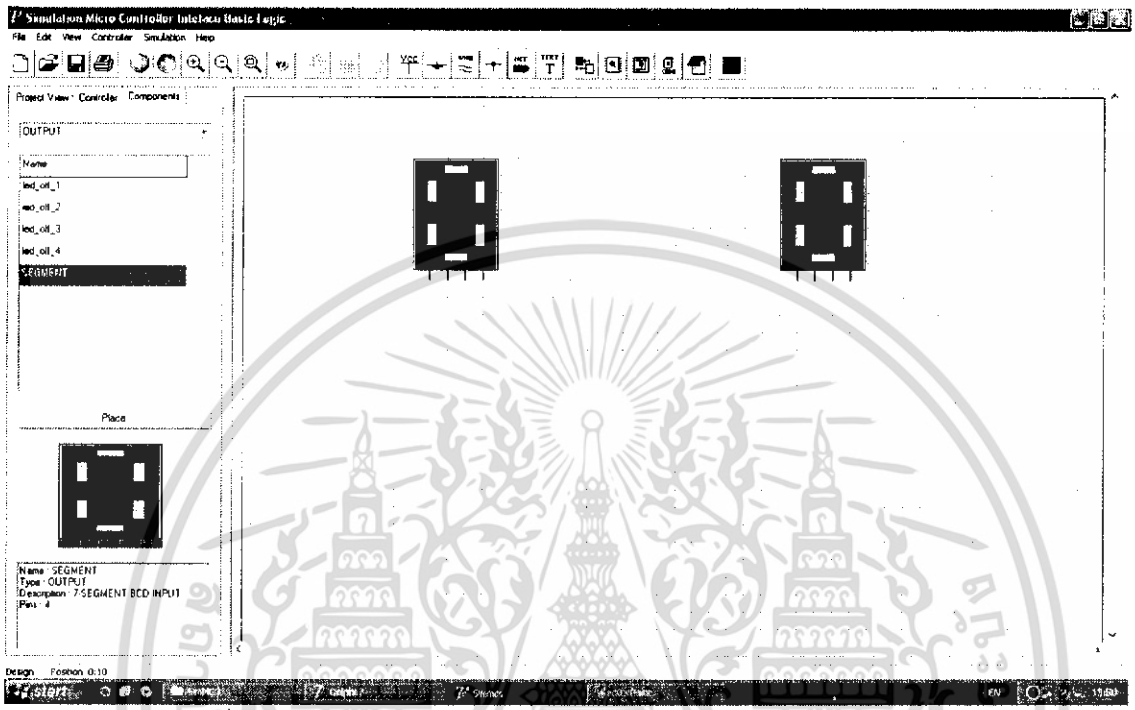
รูปร่างหน้าตาของโปรแกรมจำลองการทำงานที่ได้พัฒนาขึ้นมาเพื่อการใช้งานที่สะดวก และ มีความคล่องตัวในการทำงาน ที่ได้ทำกันมาจะมีลักษณะดังรูปที่ 4.1



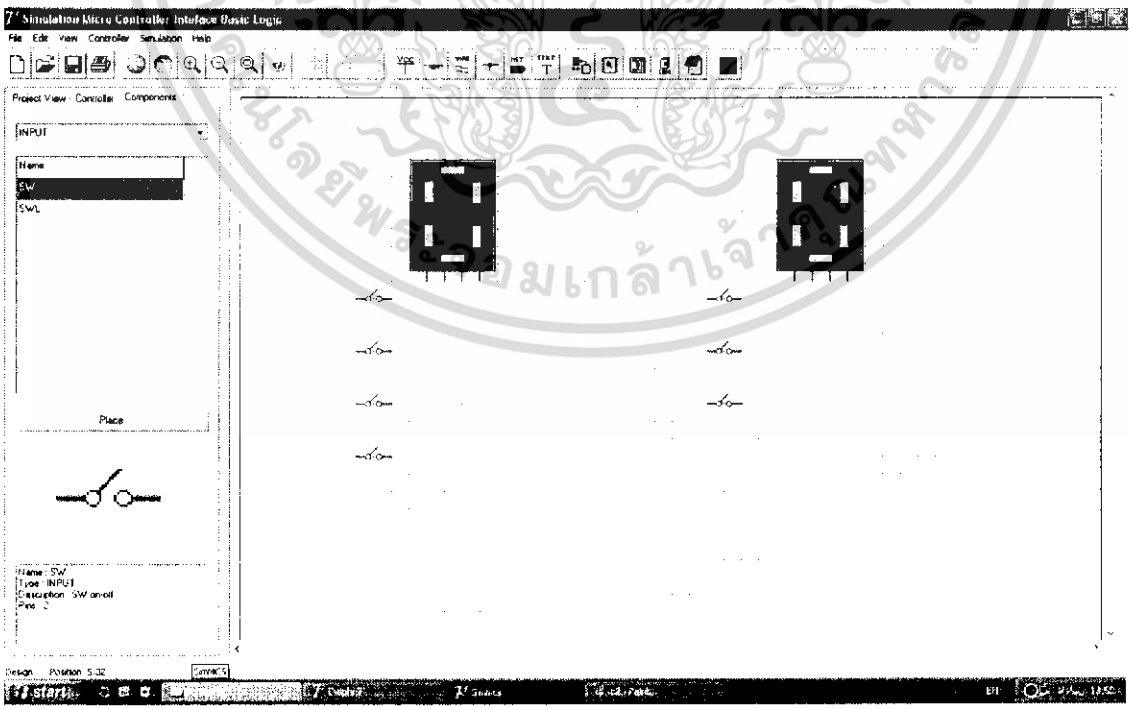
รูปที่ 4.1 แสดง หน้าตาโปรแกรมจำลองการทำงานของโปรแกรมที่ได้ออกแบบและพัฒนา

4.2 ผลการทดลองการทำงานโปรแกรมจำลองการทำงาน

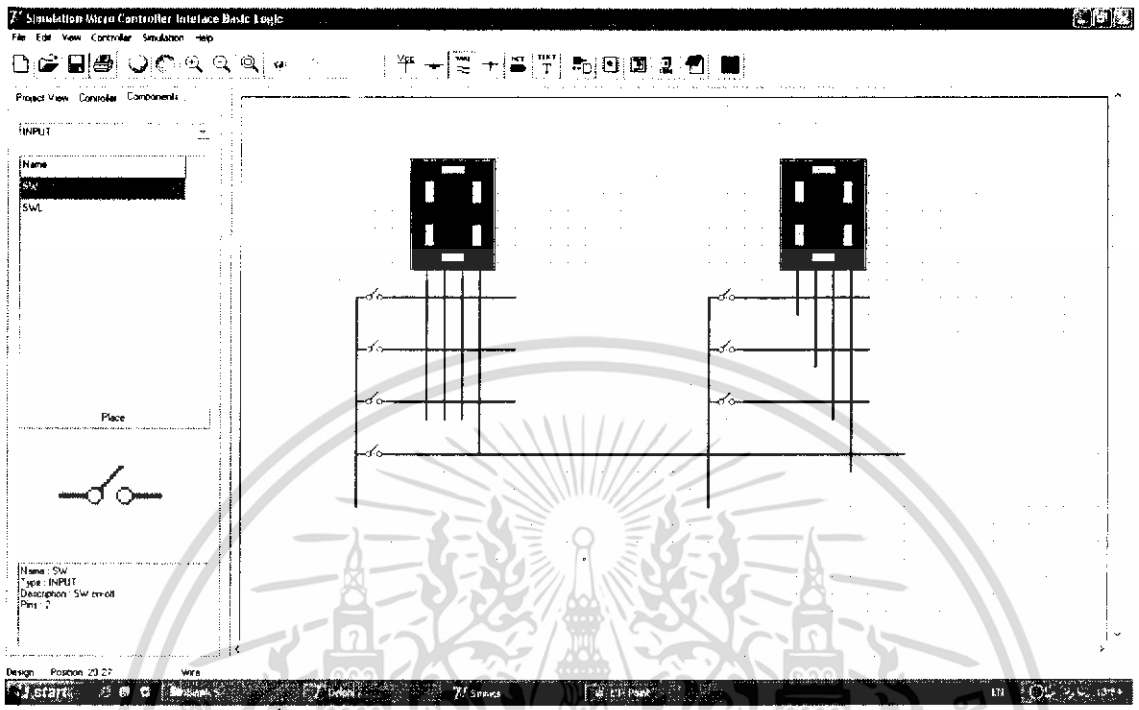
จากการทดลองใช้งานโปรแกรมจำลองการทำงานของโปรแกรมจำลองการทำงานที่ได้พัฒนาขึ้นมาได้ผลการทำงานออกมาโดยเริ่มจากการจำอุปกรณ์มาว่า โดยได้ผลดังนี้



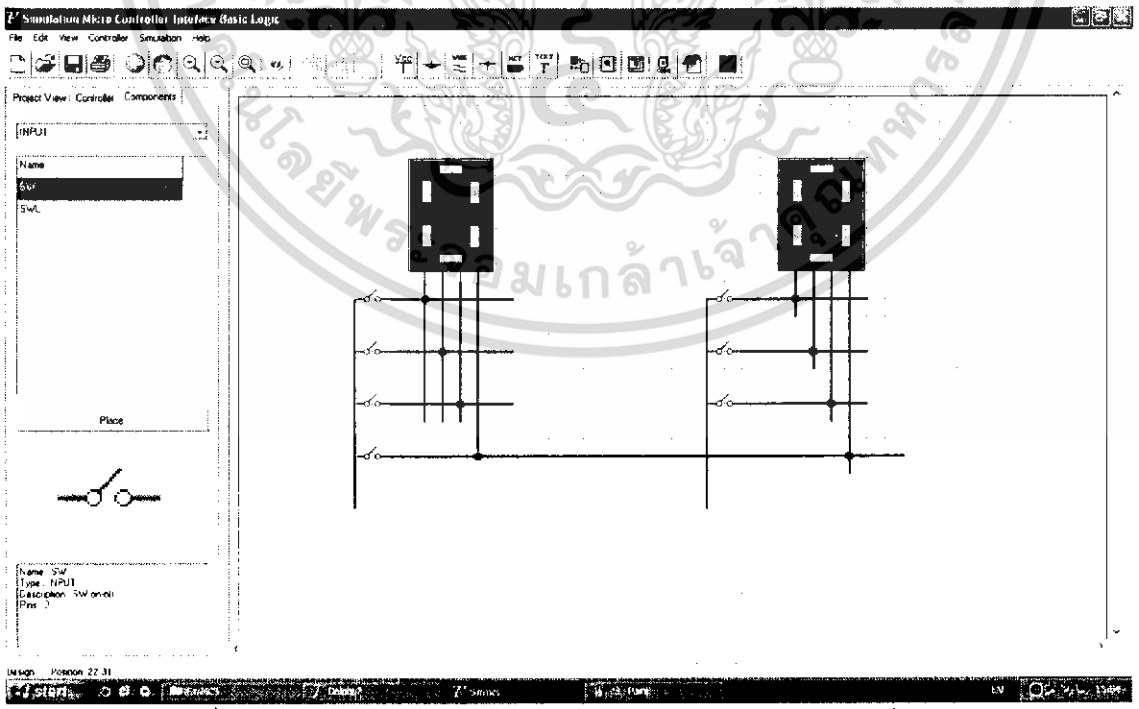
รูปที่ 4.2 แสดง ผลจากการทดสอบการนำอุปกรณ์ 7 Segment มาวาง



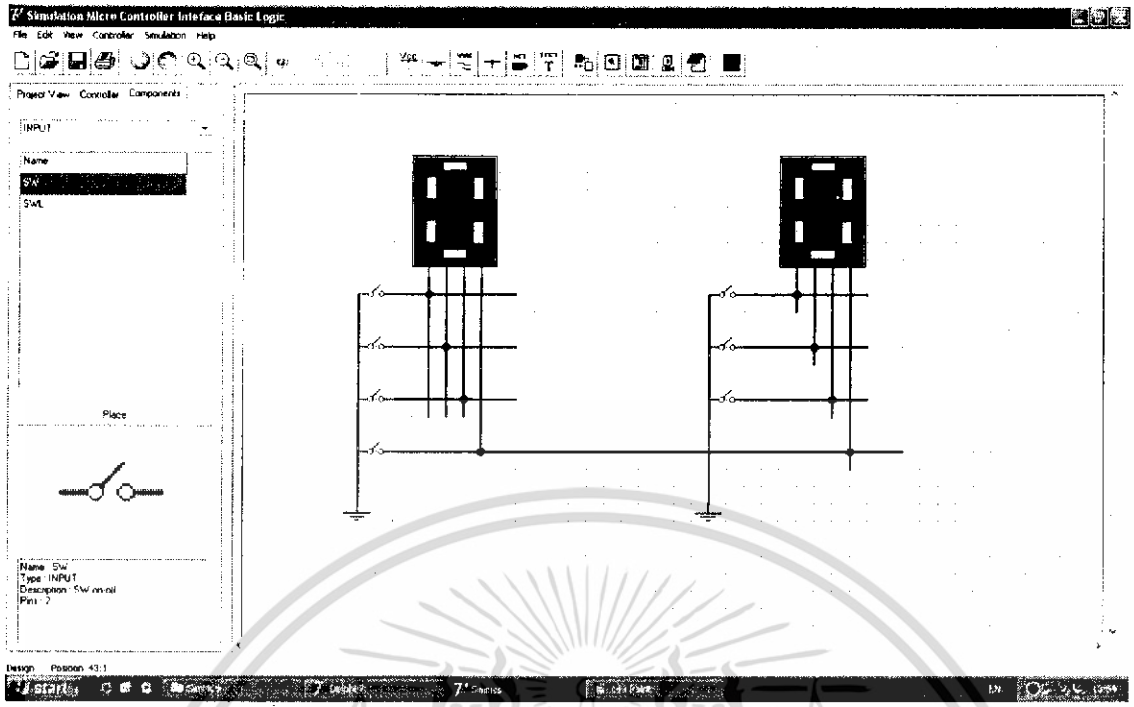
รูปที่ 4.3 แสดง ผลจากการทดสอบการนำอุปกรณ์ Switch มาวาง



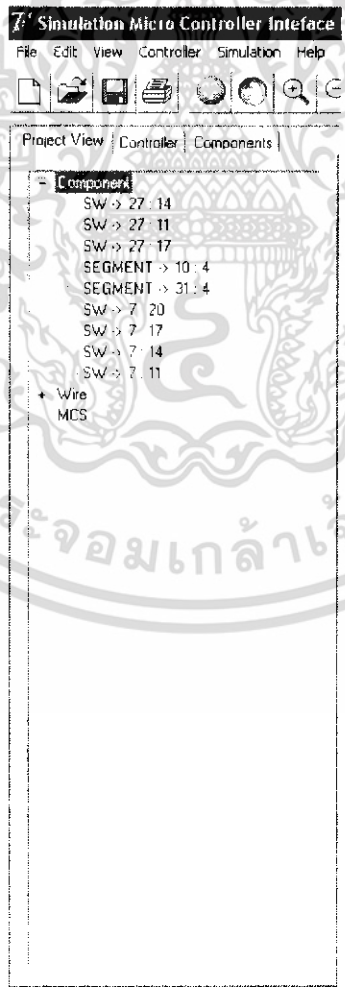
รูปที่ 4.4 แสดง ผลจากการทดสอบการลากสายเพื่อต่อวงจร



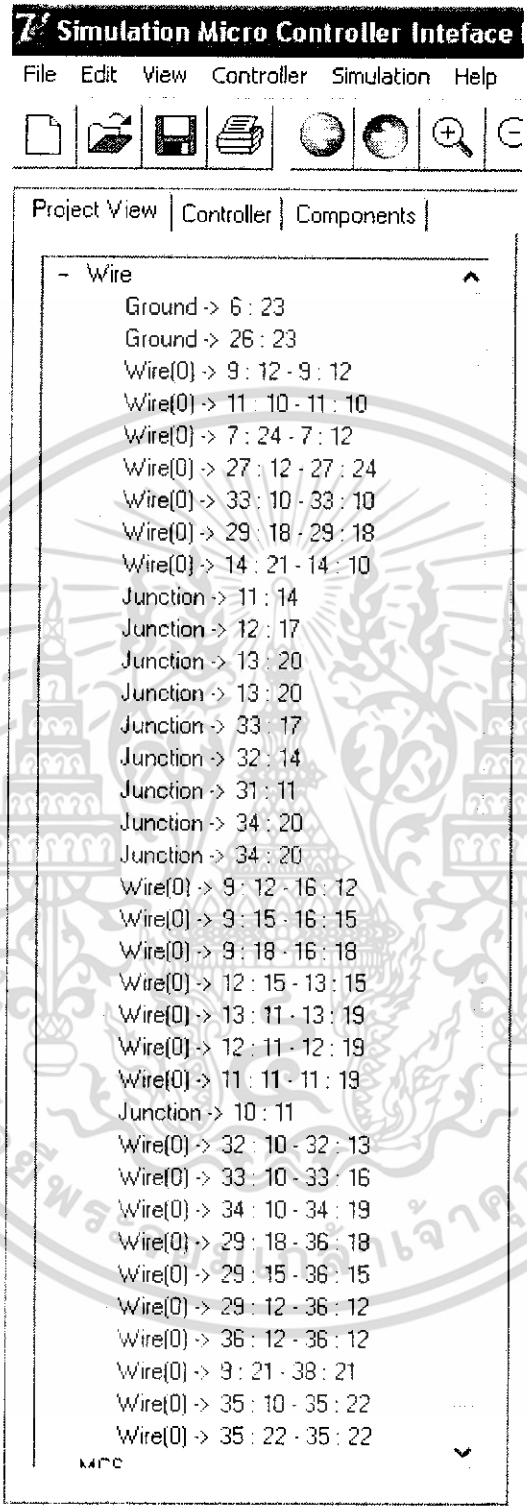
รูปที่ 4.5 แสดง ผลจากการทดสอบการเชื่อมต่อสายเพื่อต่อวงจรด้วยจุดเชื่อมต่อ



รูปที่ 4.6 แสดง ผลจากการทดสอบการเชื่อมวงจรด้วยกราวด์

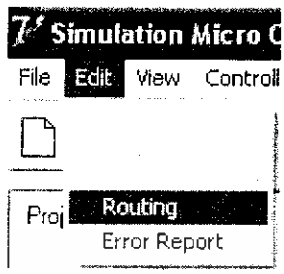


รูปที่ 4.7 แสดง ผลจากการทดสอบการดูอุปกรณ์ที่ใช้ในการSimulate



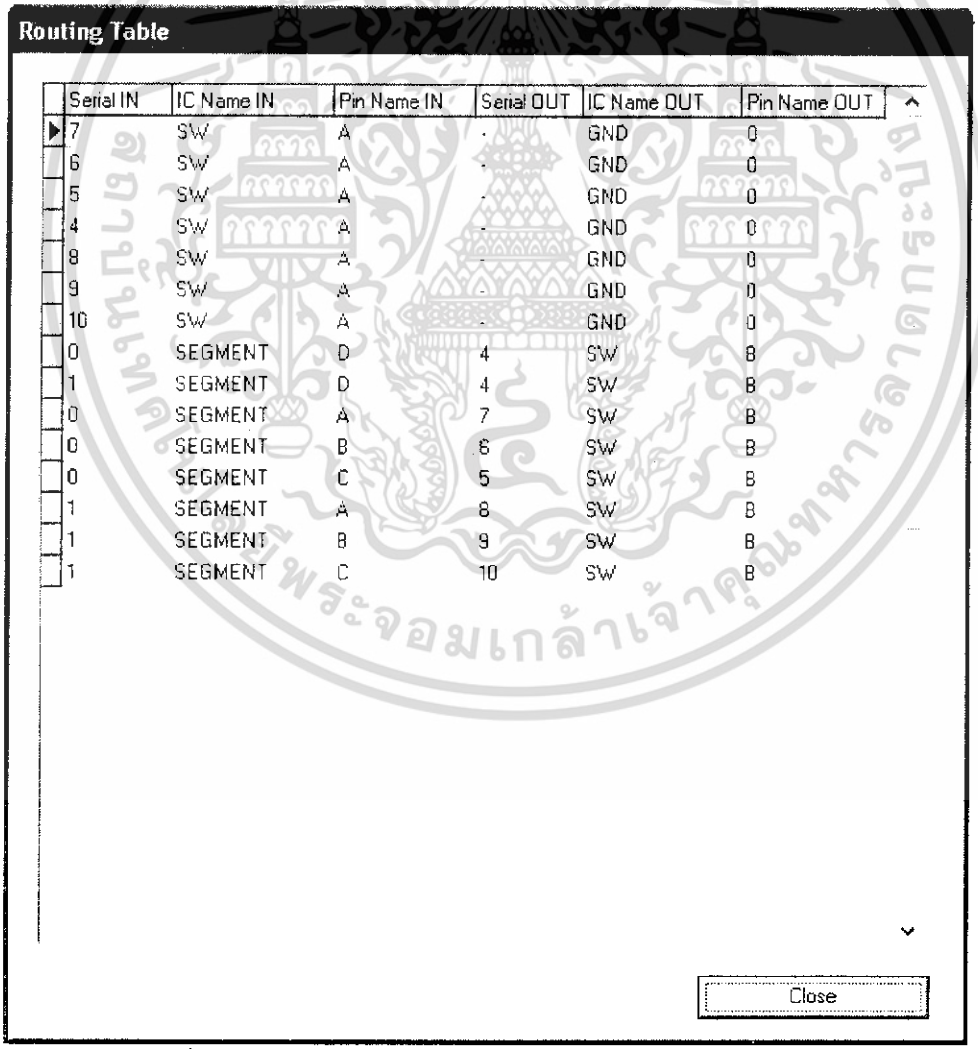
รูปที่ 4.8 แสดง ผลจากการทดสอบดูการเชื่อมต่อของวงจรจากช่องรายละเอียดการเชื่อมต่อ

จากนั้นเมื่อได้ทำการต่อวงจรทุกอย่างเสร็จแล้วก็จะต้องทำการ Routing โดยเลือกที่เมนู Edit แล้วเลือกที่ Routing ดังรูปที่ 4.9



รูปที่ 4.9 แสดง การเลือกใช้เมนู Routing

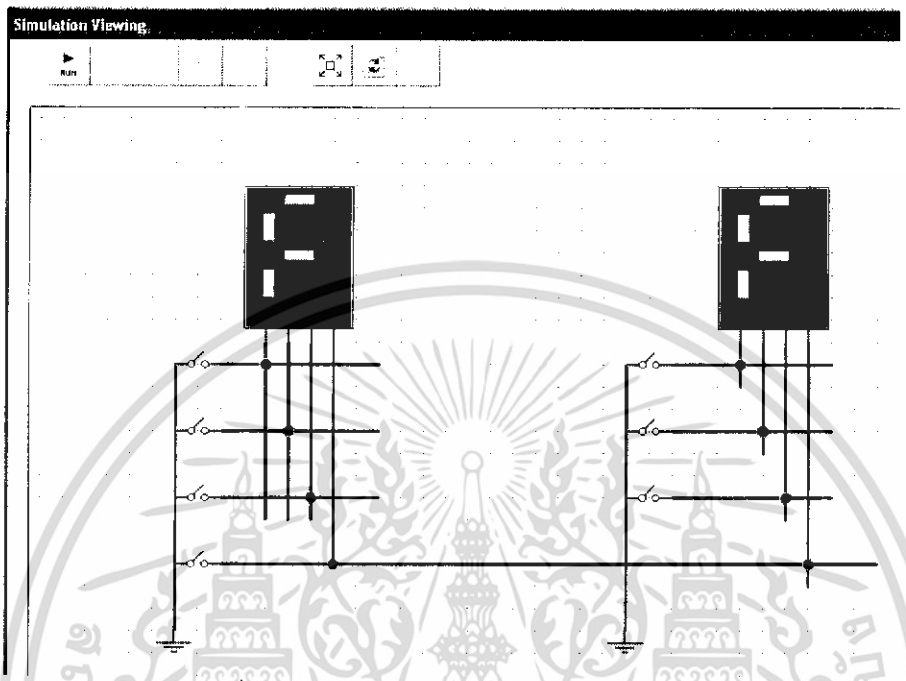
แล้วเราก็จะได้ผลจากการ Routing ออกมา ซึ่งจะแสดงรายละเอียดการเชื่อมต่อของขาต่าง ๆ ว่าขาใดต่อกับ ขาใดอยู่บ้างดังรูปที่ 4.10



Serial IN	IC Name IN	Pin Name IN	Serial OUT	IC Name OUT	Pin Name OUT
7	SW	A	-	GND	0
6	SW	A	-	GND	0
5	SW	A	-	GND	0
4	SW	A	-	GND	0
8	SW	A	-	GND	0
9	SW	A	-	GND	0
10	SW	A	-	GND	0
0	SEGMENT	D	4	SW	B
1	SEGMENT	D	4	SW	B
0	SEGMENT	A	7	SW	B
0	SEGMENT	B	6	SW	B
0	SEGMENT	C	5	SW	B
1	SEGMENT	A	8	SW	B
1	SEGMENT	B	9	SW	B
1	SEGMENT	C	10	SW	B

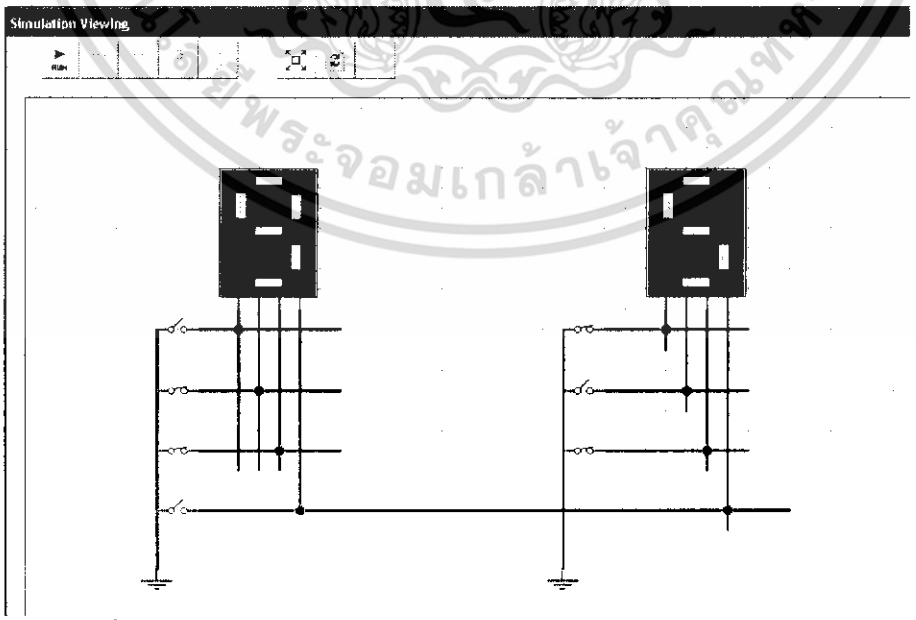
รูปที่ 4.10 แสดงข้อมูลการเชื่อมต่อของขาต่าง ๆ ของอุปกรณ์

จากนั้นเมื่อผ่านการ Routing ก็จะสามารถทำการ Run ได้ซึ่งผลจากการ Run ก็จะได้ผลการทำงานดังรูปที่ 4.11 ซึ่งเป็นผลการทำงานออกมาให้ User ได้เห็นถึงการทำงาน

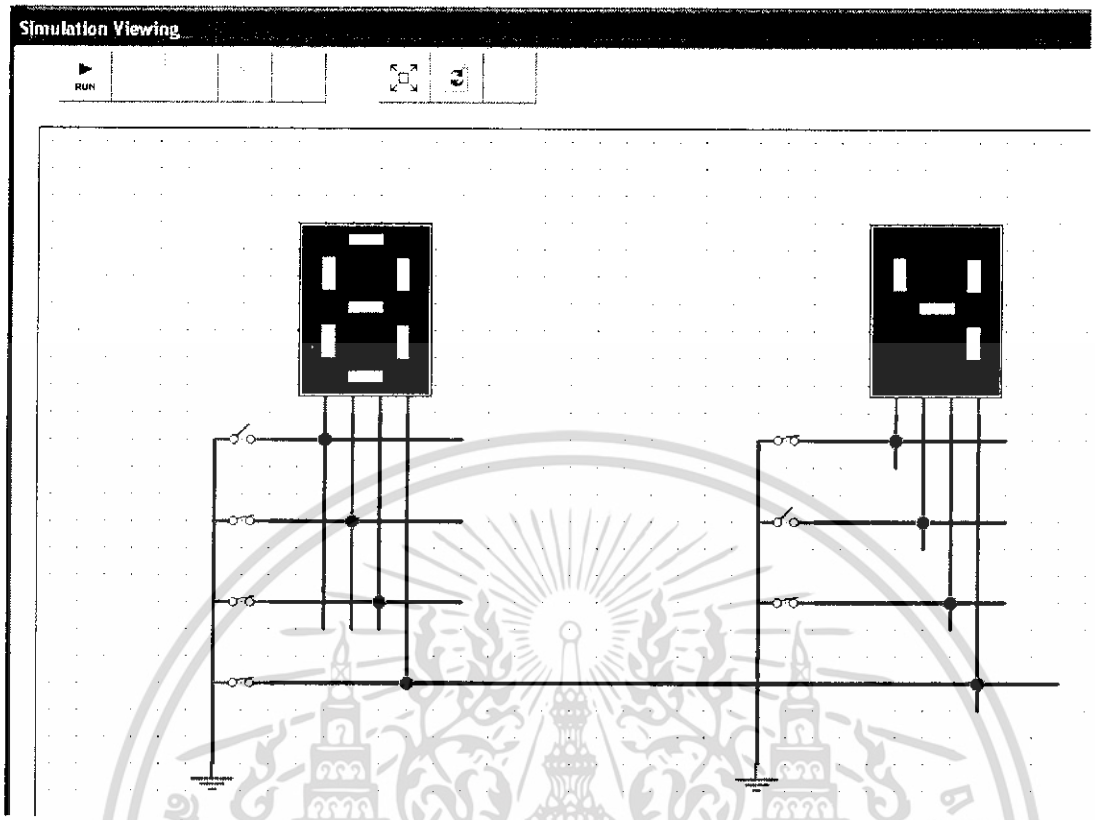


รูปที่ 4.11 แสดงผลการทำงานเมื่อกดปุ่ม Run

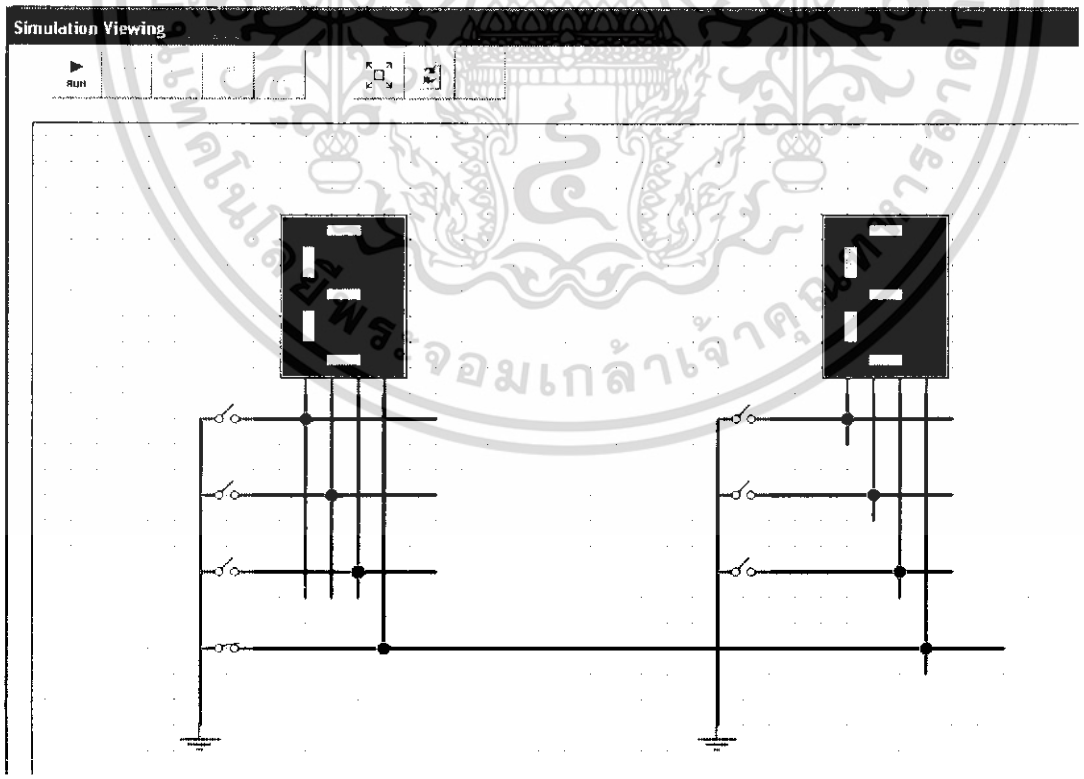
และ เมื่อ User ได้ทดลองกดปุ่มที่อุปกรณ์อินพุต ก็จะเห็นถึงการเปลี่ยนแปลงของการจำลองการทำงานดังรูปที่ 4.12 , 4.13 และ 4.14



รูปที่ 4.12 แสดงผลการทำงานเมื่อมีการเปลี่ยนแปลงอินพุตของวงจร

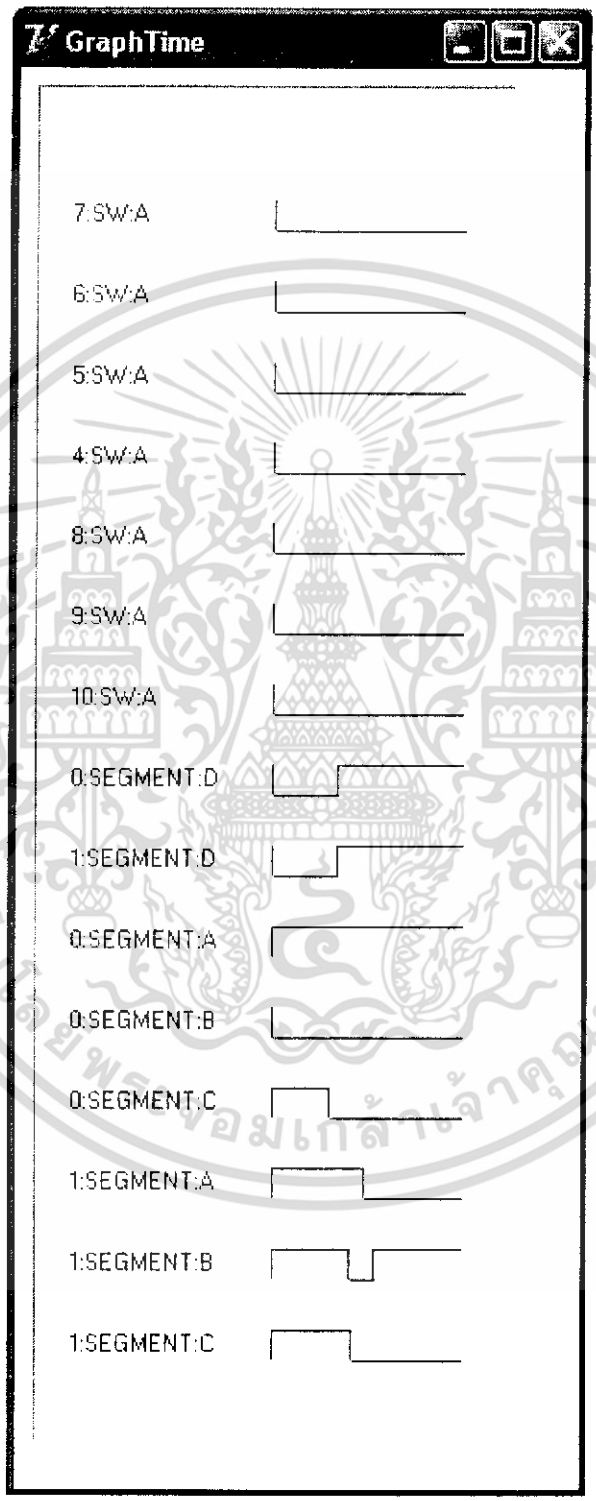


รูปที่ 4.13 แสดงผลการทำงานเมื่อมีการเปลี่ยนแปลงอินพุตของวงจร(ต่อ)

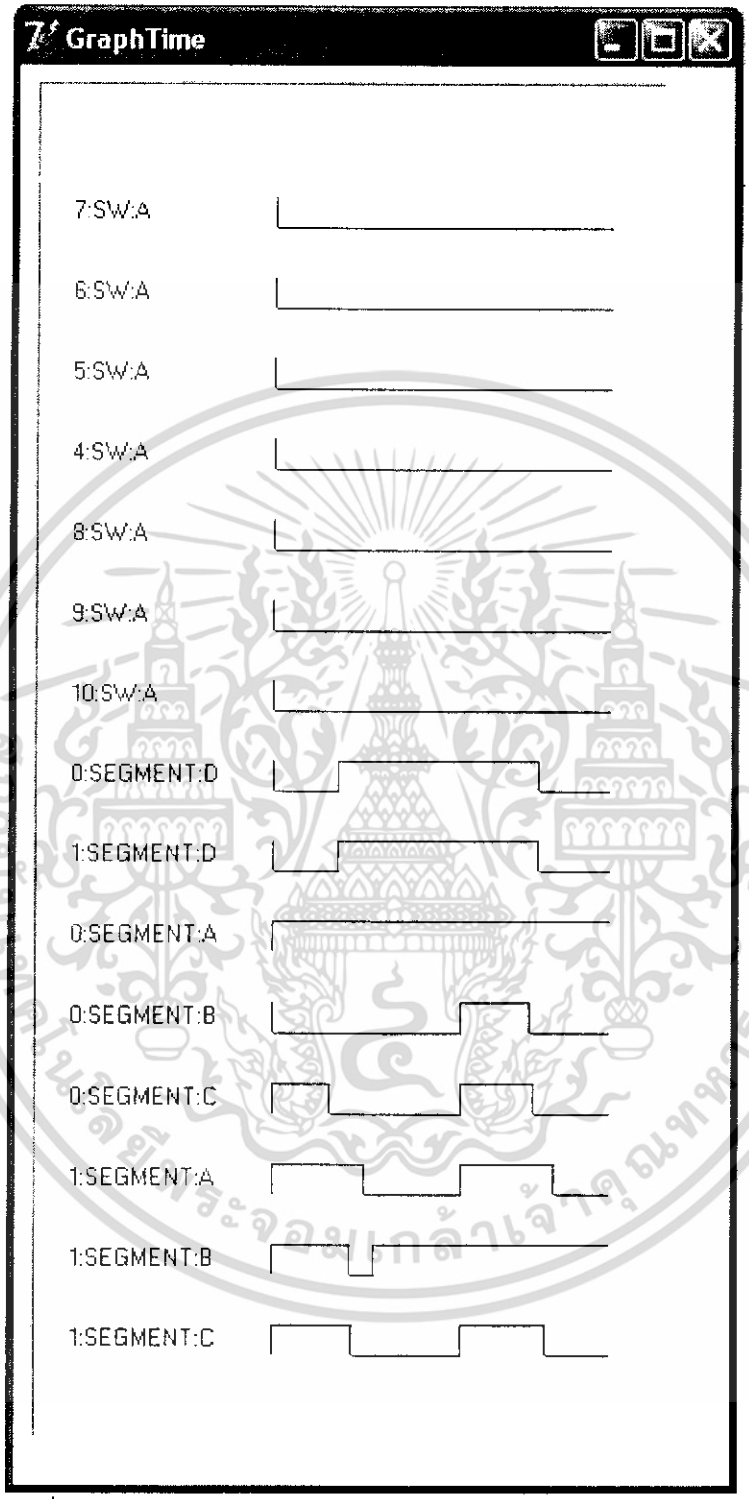


รูปที่ 4.14 แสดงผลการทำงานเมื่อมีการเปลี่ยนแปลงอินพุตของวงจร(ต่อ)

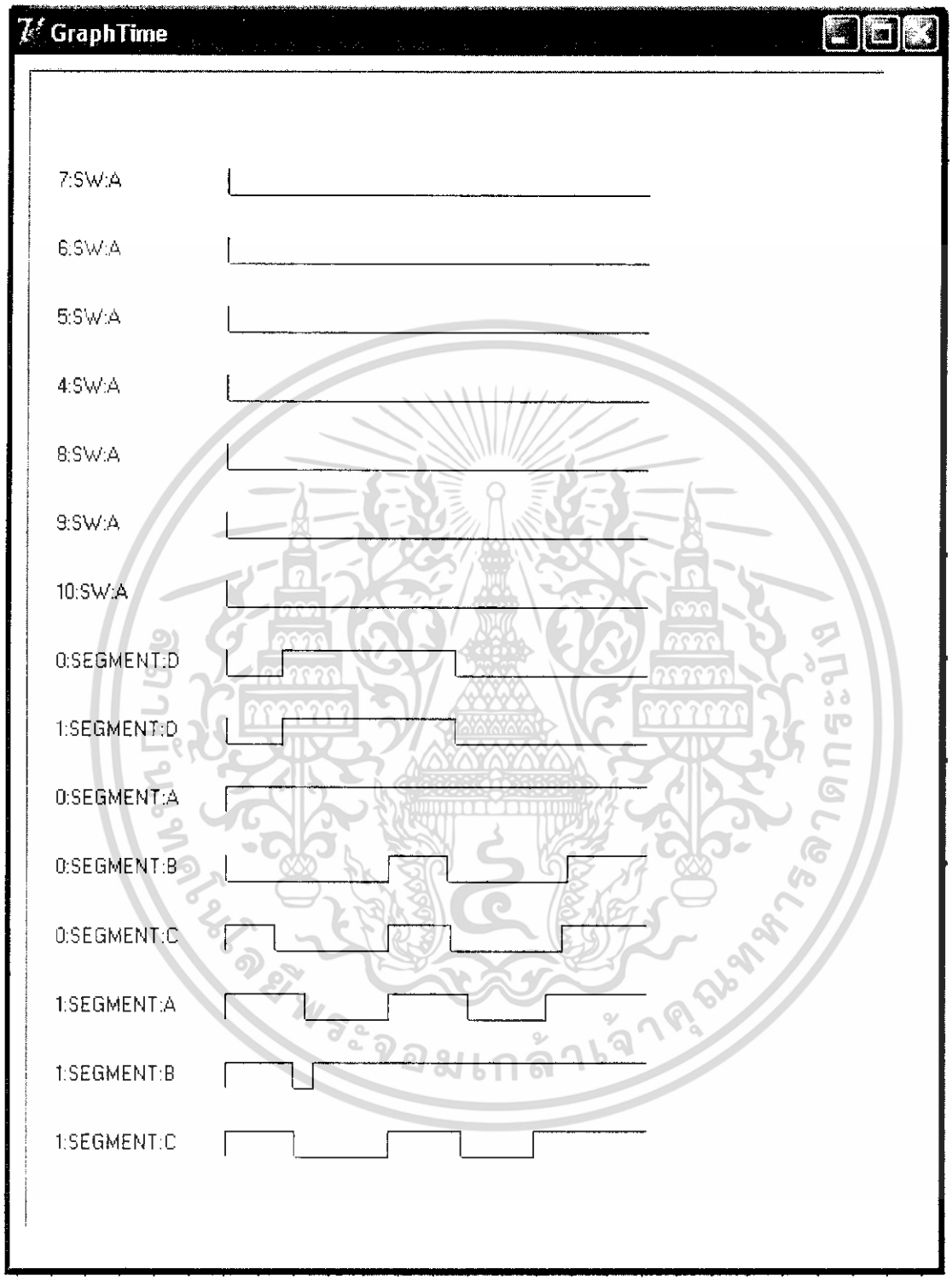
และ ถ้าหากว่า User ต้องการดูการทำงานของ Timing Diagram ของวงจรที่ผ่านมาก็สามารถเปิดหน้าต่าง Timing Diagram ขึ้นมาดูได้ ก็จะได้เห็นผลการทำงานแต่ละช่วงเวลาออกมาซึ่งเป็นสัญญาณดิจิตอลของแต่ละขาของอุปกรณ์ ดังรูปที่ 4.15 , 4.16 และ 4.17



รูปที่ 4.15 แสดงผลการทำงาน Timing Diagram ของช่วงเวลาที่ 1



รูปที่ 4.16 แสดงผลการทำงาน Timing Diagram ของช่วงเวลาที่ 2



รูปที่ 4.16 แสดงผลการทำงาน Timing Diagram ของช่วงเวลาที่ 3

4.3 สมรรถนะของโปรแกรมจำลองการทำงานวงจรดิจิตอล

จากการทดลองที่ผ่านมาได้ผลสรุปสมรรถนะของโปรแกรมจำลองการทำงานที่ทำขึ้นดังนี้

1. สามารถลากอุปกรณ์มาวางลงส่วนที่ใช้ในการจำลองการทำงานได้อย่างราบรื่น และสะดวกรวดเร็วในการวางอุปกรณ์ พร้อมทั้งค้นหาอุปกรณ์ที่จะนำมาใช้งานได้อย่างง่ายดาย และ รวดเร็ว

2. สามารถหาเส้นทางการเชื่อมต่อของอุปกรณ์ได้อย่างถูกต้อง แต่จะใช้เวลาในการหาเส้นทางการเชื่อมต่อนานนิดหน่อย เนื่องจากการหาเส้นทางการเชื่อมต่อยังใช้ Algorithm ที่ยังไม่ดีพอ และ ไม่ได้ทำการหาเส้นทางตั้งแต่ User ลากสายเชื่อมโยงกัน จึงเป็นผลให้ใช้เวลาในส่วนนี้ไม่เร็วพอ

3. สามารถดูข้อมูลการเชื่อมต่อได้ ซึ่งจะเป็นผลดีแก่ User ซึ่งจะทำให้ทราบได้ว่าจุดไหนต่อกัน หรือ จุดไหนไม่ต่อกัน ทำให้การ Debug วงจรนั้นเป็นไปได้สะดวกยิ่งขึ้น

4. สามารถจำลองการทำงานได้อย่างไม่มีข้อผิดพลาด แต่การจำลองการทำงานยังไม่ค่อยราบเรียบดีนัก เนื่องจากการหาเอาต์พุต แต่ละจุดนั้นจะต้องไปค้นหาในตารางความจริงในระบบฐานข้อมูล ซึ่งกว่าจะค้นหาเอาท์พุตครบทุกจุดก็จะกินเวลาไปไม่น้อยเลยที่เดียวเลยทำให้การทำงานดูกระตุก ๆ เล็กน้อย ซึ่งปัญหานี้ก็เป็นแนวทางในการพัฒนาต่อไปในภายหน้า

5. สามารถเก็บผลการทดลองไว้เป็น Timing Diagram ได้ โดยจะมีขนาดการเก็บ Timing Diagram ที่จำกัด เพราะฉะนั้นหากมีผลที่เกินไปจาก Timing Diagram จะเก็บไว้ได้ก็จะถูกตัดไป โดยจะตัดข้อมูลส่วนที่ผ่านไปแล้วทิ้งไป แล้วจะเก็บข้อมูลล่าสุดเอาไว้

บทที่ 5
บทวิจารณ์และสรุป

5.1 บทสรุป

จากที่กลุ่มของข้าพเจ้าได้ศึกษา และ วิเคราะห์การทำงานของอุปกรณ์ดิจิตอลในเทอมที่ผ่านมาทำให้ข้าพเจ้าได้ข้อสรุปในการออกแบบโปรแกรมจำลองการทำงานของวงจรดิจิตอลว่า จะออกแบบให้มีลักษณะการใช้งานที่สะดวกต่อการใช้งานยิ่งขึ้น โดยจะพยายามให้มีลักษณะการใช้งานให้คล้ายกับโปรแกรมจำลองการทำงานของโปรแกรม Electronic Workbench ซึ่งสะดวกในการใช้งานพอสมควร และ ในส่วนของการวางอุปกรณ์นั้นก็จะใช้การว่างอุปกรณ์ลงบน Grid เอาเพื่อที่จะได้ทำการเชื่อมต่อวงจรในโปรแกรมจำลองการทำงานได้ง่ายขึ้น และสะดวกในการหาเส้นทางเป็นสมการได้ง่ายขึ้น พร้อมทั้งขณะประมวลผลเพื่อจำลองการทำงานก็จะสามารถทำได้ง่ายขึ้นอีกด้วย และ ในส่วนของการจำลองทำงานจำพวก Logic Gate นั้นข้าพเจ้าจะเลือกที่จะใช้การนำอินพุท ทุกขามาเปรียบเทียบกับตารางความจริงของอุปกรณ์นั้น ๆ ไป จากนั้นก็จะเอาเอาท์พุตจากตารางความจริงมาแสดงออกที่ขาเอาท์พุตของอุปกรณ์ โดยที่ตารางความจริงนั้นจะเก็บอยู่ในระบบฐานข้อมูล และ ในส่วนของการลากสายอุปกรณ์นั้นจะเป็นแบบ Manual ในการลาก เพื่อความเข้าใจของผู้ใช้งาน โดยสายไหนที่จะให้ต่อกันก็จะเอาจุดต่อมาวางทับลงไป ที่ให้ผู้ใช้เข้าใจในวงจรที่ต่อมากขึ้น และ ไม่เป็นการสับสนต่อผู้ใช้อีกด้วย

5.2 วิจารณ์สิ่งที่ได้จากโครงงาน

สิ่งที่ได้จากการทำโครงงานนี้ นั้นไม่เพียงแต่จะทำให้ได้ความรู้ความเข้าใจในการทำงานของอุปกรณ์ดิจิตอลอย่างถ่องแท้แล้ว ยังทำได้เราได้ทูลในการพัฒนาวงจรดิจิตอลมาอีกด้วย ซึ่งจะทำให้การพัฒนาออกแบบอุปกรณ์ฮาร์ดแวร์ต่าง ๆ เป็นไปได้โดยง่ายขึ้น สะดวกขึ้น และ เป็นการประหยัดงบประมาณในการพัฒนาไปในตัว ถึงแม้ว่าโปรแกรมของเราจะมีความสามารถไม่เท่ากับโปรแกรมที่หาโหลดเอาในอินเตอร์เน็ตได้ตัวอื่น ๆ ได้ แต่โปรแกรมของเราก็สามารถใช้งานได้ง่ายคายจะสามารถใช้โปรแกรมนี้ได้อย่างทะลุปรุโปร่งได้ในเวลาไม่นาน และอาจใช้งานได้อย่างมีประสิทธิภาพเทียบเท่ากับโปรแกรมที่หาดาวน์โหลดมาจากอินเตอร์เน็ตอื่น ๆ และหากมีการพัฒนาต่อไปให้สามารถจำลองการทำงานของไมโครคอนโทรลเลอร์ได้ด้วย ก็จะทำให้โปรแกรมนี้มีประสิทธิภาพมากขึ้นไปกว่าโปรแกรมที่ดาวน์โหลดมาจากอินเตอร์เน็ตได้ และ ก็ไม่ต้องเสียค่าใช้จ่ายในส่วนที่จะต้องไปซื้อโปรแกรมจำลองการทำงานวงจรดิจิตอลมาใช้งานอีกด้วย

5.3 ปัญหาอุปสรรคและแนวทางในการแก้ไข

1. จำนวนของข้อมูลในระบบฐานข้อมูลนั้นมีขนาดใหญ่ จึงทำให้การทำงานของโปรแกรมค่อนข้างจะช้า เนื่องจากการจำลองการทำงานนี้เป็นการจำลองการทำงานของอุปกรณ์โดยการนำไปเทียบตารางความจริงกับข้อมูลในระบบฐานข้อมูล ซึ่งการค้นหาข้อมูลนั้นทำได้ไม่รวดเร็วนักเมื่อใช้กับฐานข้อมูลที่มีขนาดใหญ่ ซึ่งแนวทางในการแก้ปัญหานี้ คือ การออกแบบระบบฐานข้อมูลให้มีความซ้ำซ้อนของข้อมูลน้อยลง และ จัดการให้ข้อมูลที่เก็บมีขนาดที่กะทัดรัดมากยิ่งขึ้น
2. การหาสมการเส้นทางการเชื่อมต่อยังมีข้อบกพร่องอยู่มาก เนื่องจากจำนวนของเส้นทางที่ต่อยิ่งมากก็จะทำให้มีโอกาสที่จะมีการผิดพลาดสูงตามขึ้นไป ซึ่งแนวทางการแก้ปัญหานี้คือ จะต้องหาอัลกอริทึมในการหาเส้นทางระหว่างอุปกรณ์ที่ดีกว่านี้มาใช้งาน
3. ขนาดของหน้าจอที่ใช้ในการจำลองการทำงานมีขนาดที่จำกัด ไม่สามารถที่จะทำการจำลองการทำงานของวงจรดิจิตอลได้ใหญ่เกินกว่าขนาดของหน้าจอ เนื่องจากการรีเฟรชทั้งหน้าจอนั้นจะทำให้การทำงานของโปรแกรมดูไม่ราบรื่นนัก แนวทางการแก้ปัญหานี้คือ จำกัดขนาดทั้งหน้าจอไว้ แล้วรีเฟรชเฉพาะส่วนที่มีการเปลี่ยนแปลงเท่านั้น
4. การประมวลผลของการจำลองการทำงานนั้นยังทำได้ยากอยู่ เนื่องจากสมการการทำงานของอุปกรณ์แต่ละตัวนั้นจะต้องคิดขึ้นมาเองให้เข้ากับการทำงานจริง ๆ เช่นการจะเตรียมข้อมูลการทำงานของไอซีสักตัวหนึ่งในระบบฐานข้อมูล เราก็จะต้องคิด สมการการทำงานให้แก่มันด้วยซึ่งขั้นตอนนี้กินเวลาไปข้อนข้างเยอะ ซึ่งแนวทางการแก้ปัญหานี้ก็ คือ การใช้โปรแกรมในการช่วย GENERATE สมการการทำงาน ขึ้นมาให้แทนที่จะคิดเองก็จะทำให้ขั้นตอนนี้ใช้เวลาน้อยลง แต่โปรแกรมในส่วนนี้ก็ต้องพัฒนาขึ้นมาเอง ไม่สามารถดาวโหลดที่ไหนได้
5. การจำลองผลการทำงานของอุปกรณ์ดิจิตอลนั้นยังมีความผิดพลาดอยู่บ้าง เนื่องจากการหาสมการการเชื่อมต่อได้ไม่ครบถ้วน จึงทำให้ผลการจำลองการทำงานมีความผิดพลาดได้ ซึ่งแนวการทางแก้ไขก็คือ ทำให้ส่วนของการหาสมการการเชื่อมต่อเป็นไปได้อย่างถูกต้องและ ครบถ้วน

5.4 แนวทางการพัฒนาต่อ

1. พัฒนาในส่วนของความเร็วของการจำลองการทำงานให้มีมากขึ้น โดยการหาวิธีการจัดการระบบฐานข้อมูลให้มีความสมบูรณ์สูงที่สุด มีความซ้ำซ้อนที่น้อยลง ก็จะทำให้การจำลองการทำงานมีความเร็วที่เพิ่มขึ้นได้
2. พัฒนาในส่วนของการหาเส้นทางให้มีความสมบูรณ์ ไม่มีข้อบกพร่องใด ๆ เหลือ อยู่เลย โดยการหาแนวคิดในการหาสมการการเชื่อมต่อที่ดีกว่านี้มาใช้งาน
3. พัฒนาในส่วนของขนาดหน้าจอให้มีขนาดที่ปรับเปลี่ยนได้ ซึ่งก็จะทำให้การจำลองการทำงานเป็นไปได้อย่างสะดวกสบายต่อผู้ใช้มากยิ่ง ๆ ขึ้นไป
4. พัฒนาในส่วนของโปรแกรมย่อยที่จะใช้ในการคิดสมการการทำงานเพื่อเก็บลงในระบบฐานข้อมูลให้เองโดยผู้ใช้ไม่ต้องมา กำหนดสมการการทำงานเองลงในระบบฐานข้อมูลโดยตรง ซึ่งเป็นการเสียเวลาอย่างมาก
5. พัฒนาในส่วนของการจำลองการทำงานในส่วนของไมโครคอนโทรลเลอร์เพิ่มเติม เพราะไมโครคอนโทรลเลอร์นั้นมีบทบาทในงานด้านฮาร์ดแวร์มากพอสมควร และ มีการทำงานได้หลากหลาย ซึ่งถ้าหาสามารถจำลองการทำงานของไมโครคอนโทรลเลอร์ได้ด้วยแล้วโปรแกรมนี้จะเป็นโปรแกรมที่เรียกได้ว่าเหมาะสำหรับผู้ที่ต้องการออกแบบฮาร์ดแวร์ขึ้นมาใช้งานเป็นอย่างมากเลยทีเดียว
6. พัฒนาในส่วนของหน้าตาของโปรแกรมให้มีความสวยงามขึ้น และ มีตำแหน่งการเรียกใช้ทูลต่าง ๆ ให้อยู่ในตำแหน่งที่เหมาะสม เพื่อความสะดวกแก่การใช้งานของผู้ใช้ยิ่ง ๆ ขึ้นไป

บทที่ 6
สรุปผลและวิจารณ์

เมื่อทำการออกแบบและทำการ simulate แล้วการทำงานเป็นไปอย่างน่าพอใจ ซึ่งอาจจะนำการออกแบบนี้ไปใช้ประโยชน์ต่อไปในอนาคต เช่น ทำการ Optimize และดาวน์โหลดลง FPGA เพื่อทดสอบการทำงานและพัฒนาต่อไปในอนาคตได้ รวมไปถึงการออกแบบในเชิงของเลย์เอาท์ของวงจรเพื่อจะนำเข้าสู่กระบวนการผลิตได้ต่อไปในอนาคตโดยถ้ากำหนด jitter , f_{max} , f_{min} จะสามารถหาพารามิเตอร์ต่างๆของ adpll นี้ได้

ปัญหาในการออกแบบก็คือ การออกแบบที่ค่อนข้างมีความซับซ้อน ซึ่งจะต้องออกแบบตั้งแต่ส่วนประกอบย่อยๆ ให้ดีที่สุดแล้วนำมาประกอบเข้าด้วยกันเป็นส่วนของ Top-Level ซึ่งเมื่อประกอบเข้าด้วยกันแล้วก็จะเกิดปัญหาของส่วนประกอบย่อยต่างๆ ไม่สามารถทำงานร่วมกันได้ ซึ่งต้องกลับมาแก้ไขในส่วนประกอบย่อยเพื่อให้ได้ All Digital Phase-Locked Loop ที่ดีที่สุด

บรรณานุกรม

[1] ยืน ภู่วรวรรณ, 2536 ,"ทฤษฎีและการใช้งานอิเล็กทรอนิกส์ เล่ม 2" กรุงเทพมหานครฯ ซีเอ็ดยูเคชั่น

[2] สัจจะ จรัสรุ่งรวีวร และ จักรพงษ์ สุขประเสริฐ, 2547 , "เริ่มต้นอย่างมืออาชีพด้วย Delphi 7 ฉบับสมบูรณ์" นนทบุรี บริษัท ไอดีซี อินโฟ ดิสทริบิวเตอร์ เซ็นเตอร์

[3] Milos D.Ercegovac, Tomas Lang, Jaime H.Moreno, 1954, "Introduction to Digital Systems" John Wiley & Sons, Inc.

[4] Victor P.Nelson, H. Troy Nagle, Bill D. Carroll, J. David Irwin, 1974 , "Digital Logic Circuit Analysis and Design" Prentice Hall Englewood Cliffs

ภาคผนวก

ADPLL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity adpll_24bit is
        generic(n : integer:=24;nn : integer:=3;m : integer := 4);
--n=bit of dco , nn=bit of phase_compa , m = bit of register
    Port (
        fqq             : out std_logic;                    --output  fq
        fq_f            : out std_logic;                    --outputของfqที่กว้างเท่ากับ1คาบfs
        lead2,lag2,high2,low2 : out std_logic;              --outputจากlead,lag,high,low
        k2              : out std_logic_vector(n downto 0);  --output ค่า K
        fdco2           : out std_logic_vector(n downto 0);  --output ของ dco แบบ n บิท
        fdco            : out std_logic;                    -- output ของ dco ที่ msb
        fref            : in std_logic;                     --inputของfref
        clkregp         : in std_logic;                     --inputของclkที่ให้p register
        dataregp        : in std_logic_vector(m downto 0);  --ค่า p
        clkregq         : in std_logic;                     --inputของclkที่ให้q register
        dataregq        : in std_logic_vector(m downto 0);  --ค่า q
        load            : in std_logic;                     --สัญาณโหลดค่า qเข้าวงจรedge
        resetp,resetq   : in std_logic;                     --inputค่าreset p,qregister
        reset           : in std_logic;                 --inputค่าresetวงจรภายในทั้งหมด
        fs              : in std_logic);                --input ของfs
end adpll_24bit;

architecture Behavioral of adpll_24bit is
        COMPONENT dco2                                  --เพิ่มcomponent dco
        generic(n : integer);
        PORT(
                k : IN std_logic_vector(n downto 0);
                sync : IN std_logic;
                fs : IN std_logic;
                dcoout : OUT std_logic_vector(n downto 0)
                );
        END COMPONENT;
        COMPONENT fqqq3                                 --เพิ่มcomponent fq
        generic(m:integer);
        PORT(
                rin_reg : IN std_logic_vector(m downto 0);
                clk_reg : IN std_logic;
                reset_reg : IN std_logic;
```

```
        load : IN std_logic;
        fref : IN std_logic;
        reset_edge : IN std_logic;
        fs: IN std_logic;
        q_f : out std_logic;
        fq2 : out std_logic;
        fq : OUT std_logic
        );
END COMPONENT;
COMPONENT k_control3                          --เพิ่มcomponent kcontrol
generic(n : integer);
PORT(
        reset : in std_logic;
        lead : IN std_logic;
        lag : IN std_logic;
        high : IN std_logic;
        low : IN std_logic;
        clk : in std_logic;
        nbit : OUT std_logic_vector(n downto 0)
        );
END COMPONENT;
COMPONENT p_f_compa2                          --เพิ่มcomponent
                                              phasewindowscomparater
generic(n : integer;nn : integer;m : integer );
PORT(
        fdco : IN std_logic_vector(n downto 0);
        fq : IN std_logic;
        reg : IN std_logic_vector(m downto 0);
        reset : in std_logic;
        resetreg : IN std_logic;
        clkreg : IN std_logic;
        resetzero : in std_logic;
        lead : OUT std_logic;
        lag : OUT std_logic;
        high : OUT std_logic;
        low : OUT std_logic
        );
END COMPONENT;
signal fq : std_logic;                        --สัญญาณรับค่าfq
signal buffdco : std_logic_vector(n downto 0);  --สัญญาณรับค่าdco
signal lead,lag,high,low :std_logic;          --สัญญาณรับค่า
                                              --lead,lag,high,low
signal k : std_logic_vector(n downto 0);      --สัญญาณรับค่าk
signal q_f : std_logic;                       --สัญญาณรับค่าfq_f
```

```
        signal fq2 : std_logic;                          --สัญญาณรับค่าfqที่ห่างจากfq
                                                         --ไปอีก 1 คาบfs


begin
        stage2 : p_f_compa2 generic map(n,nn,m)          --เชื่อมต่อวงจรphase
                            port map (buffdco,fq,dataregp,reset
                                      ,resetp,clkregp,fq2,lead,lag,high,low);
        stage3 : k_control3 generic map(n)               --เชื่อมต่อวงจร kcontrol
                            port map (reset,lead,lag,high,low,q_f,k);
        stage4 : dco2 generic map(n)                     --เชื่อมต่อวงจร dco
                            port map (k,fq,fs,buffdco);
        stage1: fqqq3 generic map(m)                     --เชื่อมต่อวงจร fq
         PORT MAP(fq =>fq,rin_reg=>dataregq,clk_reg=>clkregq,
         reset_reg =>resetq ,load =>load ,fref =>fref ,reset_edge =>reset ,
         fs =>fs,q_f => q_f,fq2=>fq2);
        fdco <= buffdco(n);                              --แสดงผลfdcoออกมา
        fqq <= fq;                                       --แสดงผลfqออกมา
        lead2 <= lead;                                   --แสดงผลleadออกมา
        lag2 <= lag;                                     --แสดงผลlagออกมา
        high2 <= high;                                   --แสดงผลhighออกมา
        low2 <= low;                                     --แสดงผลlowออกมา
        k2 <= k;                                         --แสดงผลkออกมา
        fdco2 <= buffdco;                                --แสดงผลfdcoออกมา
        fq_f <= q_f;                                     --แสดงผลfq_fออกมา
end Behavioral;
```

DCO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DCO2 is
        generic(n : integer := 3 );                      --จำนวนบิทที่ใช้
        Port (
                k : in std_logic_vector(n downto 0);     --ค่าที่จะบวก
                sync,fs : in std_logic;                  --syncคือสัญญานบวกค่าใหม่
                                                         --fsคือความถี่ระบบ
                dcoout : out std_logic_vector(n downto 0) --ค่าที่บวกได้
                );
```

```
end DCO2;

architecture Behavioral of DCO2 is
        COMPONENT dff2
        generic(n : integer  );
        PORT(

                din : IN std_logic_vector(n downto 0);
                clk : IN std_logic;
                reset : IN std_logic;
                dout : OUT std_logic_vector(n downto 0)
                );
        END COMPONENT;
        COMPONENT adder
        generic(n : integer  );
        PORT(
                x : IN std_logic_vector(n downto 0);
                y : IN std_logic_vector(n downto 0);
                o : OUT std_logic_vector(n downto 0)
                );
        END COMPONENT;
        signal dcoout2 : std_logic_vector(n downto 0);      --ตัวแปรเชื่อมต่อรับค่าdco
        signal latch  : std_logic_vector(n downto 0);       --ตัวแปรเชื่อมต่อรับค่าที่บวก
begin
        stageadder : adder      generic map (n => n)        --เชื่อมต่อวงจรadder
                                        port map (k,dcoout2,latch);
        stageFF    : dff2       generic map (n => n)        --เชื่อมต่อวงจรflipflop
                                        port map (latch,fs,sync,dcoout2);
        dcoout <= dcoout2;                                  --เชื่อมต่อoutput
end Behavioral;
```

DFF

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dff2 is                                          --เป็นวงจรDflipflopแบบ n บิท
        generic(n : integer :=3 );                      --จำนานบิท
        Port (
                din : in std_logic_vector(n downto 0) := (others => '0');
                dout : out std_logic_vector(n downto 0);
                clk : in std_logic;
                reset : in std_logic
               );
```

```
end dff2;

architecture Behavioral of dff2 is

begin
process (CLK)
begin
  if CLK'event and CLK='1' then  --CLK rising edge
    if RESET='1' then   --synchronous RESET active High
      DOUT <= (others => '0');
    else
      DOUT <= DIN;
    end if;
  end if;
end process;

end Behavioral;
```

ADDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity adder is                                    --วงจรบวกค่า n บิท
        generic(n : integer := 3  );               --adder แบบ  n  บิท
        Port ( x : in std_logic_vector(n downto 0);    --inputที่ใช้บวกค่าที่1
        y : in std_logic_vector(n downto 0);           --inputที่ใช้บวกค่าที่2
        o : out std_logic_vector(n downto 0));         --outputบวกค่าเสร็จ
end adder;

architecture Behavioral of adder is
        COMPONENT fulladder                        -- เพิ่มcomponent fulladder
        PORT(
                cin,x,y : IN std_logic;
                s,cout : OUT std_logic
                );
        END COMPONENT;
        signal c : std_logic_vector(n+1 downto 0);          -- สัญญาณที่ใช้รับค่าจำนวนทด
begin
        c(0) <= '0';                               -- กำหนดให้ไม่มีการบวกจำนวนทดเริ่มแรก
        G1 : for i in 0 to n generate              --เป็นการต่อวงจรบวกทีละบิทให้บวกได้หลายๆๆบิท
                stage0 : fulladder port map(c(i),x(i),y(i),o(i),c(i+1));
                o(n) <= x(n) xor y(n) xor c(n);
        end generate;
end Behavioral;
```

FULLADDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fulladder is                                  --วงจรบวกค่าบิท
   Port ( cin,x,y : in std_logic;                    --x,y คือ input cin คือค่าทด
          s,cout : out std_logic);                   --s คือ output  cout คือ ค่าทด
end fulladder;

architecture Behavioral of fulladder is

begin
        s <= x xor y xor cin;
        cout <= (x and y) or (cin and x) or (cin and y);
end Behavioral;
```

Fq

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fqqq3 is                                      --วงจรfq
        generic(m :integer := 4);                    --จำนวนบิทของค่านับ
   Port ( fq : out std_logic;                        --เอาท์พุท fq
        fq2 : out std_logic;                         --เอาท์พุท fqที่shiftไป1คาบfs
        q_f : out std_logic;                         --เอาท์พุทfqที่shiftไปครึ่งคาบfs
        rin_reg : in std_logic_vector(4 downto 0);   --data ค่า q
        clk_reg : in std_logic;                      --clk อ่านค่า q
        reset_reg : in std_logic;                    --reset reg
        load : in std_logic;                         --load ค่า q เข้า edge
        fref : in std_logic;                         --ความถี่ที่ต้องการหาร
        reset_edge : in std_logic;                   --resetวงจรedge
        fs : in std_logic);                          --ความถี่ระบบ
end fqqq3;

architecture Behavioral of fqqq3 is
        COMPONENT edge_fq
        PORT(
                d : IN std_logic;
                reset : IN std_logic;
                fs : IN std_logic;
```

```
            q_f : out std_logic;
            q3  : out std_logic;
            q : OUT std_logic
            );
    END COMPONENT;
    COMPONENT counter
    generic(m:integer);
    PORT(
            clk : IN std_logic;
            load : IN std_logic;
            reg : IN std_logic_vector(m downto 0);
            y : OUT std_logic
            );
    END COMPONENT;

    COMPONENT reg
    generic(m:integer);
    PORT(
            rin : IN std_logic_vector(m downto 0);
            clk : IN std_logic;
            reset : IN std_logic;
            rout : OUT std_logic_vector(m downto 0)
            );
    END COMPONENT;

    signal counter_edge : std_logic;                    --สัญญาณเชื่อมต่อ
    signal reg_counter : std_logic_vector(m downto 0);  --สัญญาณเชื่อมต่อ
begin
    satge1: edge_fq                                     --เชื่อมต่อวงจร edge
            PORT MAP
            (fs =>fs ,reset =>reset_edge ,d =>counter_edge ,q =>fq,q3 =>fq2,q_f
=> q_f);
    stage2: counter                                     --เชื่อมวงจร counter
            generic map (m)
            PORT MAP(y =>counter_edge ,clk =>fref ,load =>load ,reg
=>reg_counter );
    stage3: reg                                         --เชื่อมวงจรregister
            generic map (m)
            PORT MAP(rin =>rin_reg ,clk =>clk_reg ,reset =>reset_reg ,rout
=>reg_counter );
end Behavioral;
```

EDGE_Fq

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity edge_fq is                          --วงจรสร้างedge
   Port ( d : in std_logic;                --ค่าที่จะทำedgeในที่นี่คือfref
          reset : in std_logic;            --resetวงจร
          fs : in std_logic;               --คาบที่ต้องการให้pulsenกว้างเท่านั้นในที่นี้คือfs
          q : out std_logic;               --ค่าoutputของedge
          q_f : out std_logic;             --ค่าoutputของedgeที่shiftจากqไปครึ่งคาบfs
          q3  : out std_logic);            --ค่าoutputของedgeที่shiftจากqไปหนึ่งคาบfs
end edge_fq;

architecture Behavioral of edge_fq is
          signal d2 : std_logic;
          signal q2 : std_logic;
begin
process (fs, RESET)                        --ถ้าทีresetจะให้d2 = 0 ถ้าไม่มีแล้วเจอขาขึ้นของfs
begin                                      --จะให้ d2 = d
  if RESET='1' then   --asynchronous RESET active High
    d2 <= '0';
  elsif (fs'event and fs='1') then  --CLK rising edge
    d2 <= d;
  end if;
end process;

process (fs, RESET)                        --ถ้าทีresetจะให้q2 = 0 ถ้าไม่มีแล้วเจอขาขึ้นของfs
begin                                      --จะให้ q2 = d2  ดังนั้น q2 จะshift ไป 1 คาบ fs
  if RESET='1' then   --asynchronous RESET active High
    q2 <= '0';
  elsif (fs'event and fs='1') then  --CLK rising edge
    q2 <= d2;
  end if;
end process;

          q <= d2 xor (q2 and d2);

process (fs, RESET)                        --ถ้าทีresetจะให้q2 = 0 ถ้าไม่มีแล้วเจอขาขึ้นของfs
begin                                      --จะให้ q2 = d2  ดังนั้น q2 จะshift ไป ครึ่ง คาบ
fs

  if RESET='1' then   --asynchronous RESET active High
    q_f <= '0';
  elsif (fs'event and fs='0') then  --CLK rising edge
```

```
    q_f <= d2;
  end if;
end process;

 q3 <= q2;                                    --output q3 = สัญญาณ q2

end Behavioral;
```

COUNTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
        generic (m : integer := 4);                                  --จำนวนบิทที่ใช้
        Port (
                y : out std_logic:='0';                              --output
                clk : in std_logic:='0';                             --clkที่ใช้นับ
                load : in std_logic := '1';                          --จังหวะload ค่า
                reg : in std_logic_vector(m downto 0):="00000"       --ค่าที่จะนับ
                );
end counter;

architecture Behavioral of counter is
        signal bufq : std_logic_vector(m downto 0) := (others => '0');--รองรับค่าลบ
        signal bufy : std_logic := '0';                              --รองรับoutput
        constant zero : std_logic_vector(m downto 0) := (others => '0');--ค่าเปรียบเทียบ0
begin
  y <= bufy ;                                                        --output = bufy
process (clk,reg,bufy,load)
begin
        if load ='1' then                                    --ถ้ามีการโหลดค่าให้
                bufq <= reg-1;                               -- bufq <= reg-1
                bufy <= '0';                                 -- bufy <= '0' เพื่อreset ค่า
        elsif clk='1' and clk'event then                     --ถ้าไม่โหลดแล้วเจอclk ให้ลด
                if  bufq = zero then                         --ค่าที่นับ
                        bufq <= reg-1;
                        bufy <= '1';
                else
                        bufq <= bufq - 1;
                        bufy <= '0';
                end if;
        end if;
```

```
end process;
end Behavioral;
```

REGISTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg is                                          --วงจรregister
        generic(m : integer := 4);
        Port (
                rin : in std_logic_vector(m downto 0) := (others => '0');
                clk,reset : in std_logic := '0';
                rout : out std_logic_vector(m downto 0) :=  (others => '0')
                );
end reg;

architecture Behavioral of reg is
begin
process (CLK, RESET)
begin
  if RESET='1' then   --asynchronous RESET active High
    rOUT <= (others => '0');
  elsif (CLK'event and CLK='1') then  --CLK rising edge
    rOUT <= rIN;
  end if;
end process;

end Behavioral;
```

PHASEWINDOWCOMPARATER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity p_f_compa2 is                                   --วงจรเปรียบเทียบความถี่และเฟส
        generic(n : integer := 3;nn : integer:=2;m : integer:= 4);
                                --n =จำนวนบิท dco  nn คือจำนวนบิท phase  m คือจำนวนบิท p
    Port ( fdco : in std_logic_vector(n downto 0);     --fdco
        fq : in std_logic;                             --fq
        reg : in std_logic_vector(m downto 0);         --p register
        lead : out std_logic;                          --lead
        lag : out std_logic;                           --lag
        high : out std_logic;                          --high
        low : out std_logic;                           --low
        resetzero : in std_logic;                      --reset zero      คือ fq3
        reset : in std_logic;                          --reset วงจร
        resetreg : in std_logic;                       --reset register
```

```
        clkreg : in std_logic                          --clk อ่านค่า p
                );
end p_f_compa2;
architecture Behavioral of p_f_compa2 is
        COMPONENT phase_compa
        generic(n : integer;nn : integer);
        PORT(
                fq : IN std_logic;
                in_n : IN std_logic_vector(n downto 0);
                lead : OUT std_logic;
                lag : OUT std_logic);
        END COMPONENT;
        COMPONENT freq_compa2
        generic(m : integer);
        PORT(
                fdco : IN std_logic;
                load : IN std_logic;
                reset : in std_logic;
                resetreg : IN std_logic;
                clkreg : IN std_logic;
                dataregin : IN std_logic_vector(m downto 0);
                resetzero : in std_logic;
                high : OUT std_logic;
                low : OUT std_logic);
        END COMPONENT;
begin
        stage1 : freq_compa2 generic map(m)            --เชื่อมต่อวงจรล็อกความถี่
                                port map (fdco(n),fq,reset
                                        ,resetreg,clkreg,reg,resetzero,high,low);
        stage2 : phase_compa generic map(n,nn)
                                port map (fq,fdco,lead,lag);   --เชื่อมต่อวงจรเฟส
end Behavioral;
```

PHASE_COMPARATER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity phase_compa is                                  --วงจรเปรียบเทียบ phase
        generic (n :integer := 3;nn : integer :=2);    --ขนาดของdco n บิท และphase nn บิท
    Port ( fq : in std_logic;                          --จังหวะการอ่านค่า ในที่นี่คือ fq
        in_n : in std_logic_vector(n downto 0):= "1111";   --fdco
        lead : out std_logic;                          --lead
        lag : out std_logic);                          --lag
end phase_compa;
architecture Behavioral of phase_compa is
        COMPONENT and_3
```

```
        PORT(
                x1 : IN std_logic;
                x2 : IN std_logic;
                x3 : IN std_logic;
                y : OUT std_logic
                );
        END COMPONENT;
signal x : std_logic_vector(nn downto 0) ;          --สัญญาณรับค่า and
signal y : std_logic_vector(nn downto 0) ;          --สัญญาณรับค่า or
signal z : std_logic;                               --สัญญาณรับค่า x กับ y
signal inn :std_logic_vector(n downto 0);           --สัญญาณรองรับค่า fdco
begin
        x(0) <= in_n(n-nn);                         --บิทต่ำสุดที่ได้เลือกของphase
        y(0) <= in_n(n-nn);                         --บิทต่ำสุดที่ได้เลือกของphase
process(x,y,in_n)                                   --นำบิทที่ได้เลือกมาandและorกันทั้งหมด
begin
        for i in 0 to nn-1 loop
        x(i+1) <= x(i) and in_n((n-nn)+i+1);
        y(i+1) <= y(i) or  in_n((n-nn)+i+1);
        end loop;
end process;

        z <= x(nn) nor (not y(nn));                 --z จะไม่มีค่าออกเมื่อ x=1ทั้งหมด และ
        inn(n) <= not in_n(n);                      --y= 0 ทั้งหมด
        stage_lead: and_3 PORT MAP(fq,inn(n),z,lead);   --เชื่อมต่อวงจรlead lag
        stage_lag: and_3 PORT MAP(fq,in_n(n),z,lag);

end Behavioral;
```

AND3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity and_3 is                                     --and 3 input
   Port ( x1 : in std_logic;                        --input1
        x2 : in std_logic;                          --input2
        x3 : in std_logic;                          --input3
        y : out std_logic);                         --output
end and_3;

architecture Behavioral of and_3 is
signal z : std_logic;                               --รับค่าการand
begin
```

```
        z <= x1 and x2;                                  --นำinputมาandกัน
        y <= z and x3;
end Behavioral;
```

FREQUENCY_COMPA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity freq_compa2 is                                    --วงจรล็อกความถี่
        generic(m :integer := 4);                        --จำนวนบิทที่นับของ p
        Port ( fdco : in std_logic;                      --ความถี่ dco
               load : in std_logic;                      --จังหวะloadค่า p ในที่นี้คือ fq
               high : out std_logic;                     --ค่าบอกว่าความถี่ที่ได้สูงกว่าที่ต้องการ
               low : out std_logic;                      --ค่าบอกว่าความถี่ที่ได้ต่ำกว่าที่ต้องการ
               resetzero : in std_logic;                 --reset วงจร zero ในที่นี้คือ fq3
               reset : in std_logic;                     --reset วงจรล็อก
               resetreg,clkreg : in std_logic;           --resetและclkอ่านค่าของ register p
               dataregin : IN std_logic_vector(m downto 0));        --ค่า p
end freq_compa2;

architecture Behavioral of freq_compa2 is

        COMPONENT zero_compa2
        generic(m :integer );
        PORT(
               zero : IN std_logic_vector(m downto 0);
               fq : IN std_logic;
               reset : in std_logic;
               high : OUT std_logic;
               low : OUT std_logic
               );
        END COMPONENT;
        COMPONENT pcounter2
        generic(m :integer);
        PORT(
               clk : IN std_logic;
               RESET: in STD_LOGIC;
               load : IN std_logic;
               din : IN std_logic_vector(m downto 0);
               count : INOUT std_logic_vector(m downto 0)
               );
        END COMPONENT;
        COMPONENT reg
```

```
        generic(m :integer );
        PORT(
                rin : IN std_logic_vector(m downto 0);
                clk : IN std_logic;
                reset : IN std_logic;
                rout : OUT std_logic_vector(m downto 0)
                );
        END COMPONENT;

        signal datareg : std_logic_vector(m downto 0);          --สัญญาณเชื่อมต่อ
        signal count   : std_logic_vector(m downto 0);          --สัญญาณเชื่อมต่อ
begin
        stage1 : reg     generic map (m => 4)                   --เชื่อมต่อ register p
                         port map(dataregin,clkreg,resetreg,datareg);
        stage2 : pcounter2       generic map (m => 4)           --เชื่อมต่อ p counter
                         port map(fdco,reset
                         ,load,datareg,count);
        stage3 : zero_compa2 generic map (m => 4)               --เชื่อมต่อวงจร zero
                         port map(count,load,resetzero,high,low);
end Behavioral;
```

P_COUNTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity pcounter2 is                                     --วงจร down counter
        generic (m : integer := 4);                     --ขนาดค่า p  = n บิท
        Port (
                CLK: in STD_LOGIC;                      --จังหวะการลดค่า
                RESET: in STD_LOGIC;                    --reset
                load : in std_logic;                    --load ค่า
                DIN: in STD_LOGIC_VECTOR(m downto 0);   --ค่าโหลด
                COUNT: inout STD_LOGIC_VECTOR(m downto 0) := (others => '0')
                                                        --output
                );
end pcounter2;
architecture Behavioral of pcounter2 is
begin
process (CLK,load,din,reset)
begin
  if (LOAD='1') or (reset = '1') then                   --load หรือ reset = 1 จะโหลดค่า
```

```
      COUNT <= DIN;
  elsif CLK='1' and CLK'event then                    --ถ้าเจอขอบขาขึ้นclk ให้ลดค่า
          COUNT <= COUNT - 1;
  end if;
end process;
end Behavioral;
```

ZERO_COMPARATER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity zero_compa2 is                                 --วงจรตรวจสอบค่า 0
        generic (m : integer := 4);                   --จำนวนบิทส่งค่า
        Port ( zero : in std_logic_vector(m downto 0);  --ค่าที่ส่งมา
               high : out std_logic;                  --high
               fq   : in std_logic;                   --จังหวะการอ่านค่า
               reset : in std_logic;                  --จังหวะreset ในที่นี้คือ fq3
               low  : out std_logic);                 --low
end zero_compa2;
architecture Behavioral of zero_compa2 is
        COMPONENT anddff
        PORT(
               fq : IN std_logic;
               zero : IN std_logic;
               reset : IN std_logic;
               d_and : OUT std_logic
               );
        END COMPONENT;

signal y : std_logic_vector(m downto 0) := (others => '0');  --รองรับค่าที่ตรวจว่า = 0
signal buf,buf2 : std_logic := '0';                  --รองรับค่า แทน lead lagที่ยังไม่and
begin
        y(0) <= zero(0);
process(zero,y)
begin
               for i in 0 to m-1 loop
                  y(i+1) <= y(i) or zero(i+1);       --ตรวจสอบว่า = 0 ไหม
               end loop;
end process;
        buf <= y(m) and zero(m);                      --มากกว่า 0
        buf2 <= y(m) and (not zero(m));               --น้อยกว่า 0
        stage1: anddff PORT MAP(                      --วงจรand
               fq => fq,
               zero => buf,
```

```
            reset => reset,
            d_and => high
    );
    stage2: anddff PORT MAP(                    --วงจรand
            fq => fq,
            zero => buf2,
            reset => reset,
            d_and => low
    );

end Behavioral;
```

AND_DFF

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity anddff is                            --วงจรandแบบขอบขาขึ้นของfq
   Port ( fq : in std_logic;                --inputที่นำมาandที่ 1
          zero : in std_logic;              --inputที่นำมาandที่ 2
          reset : in std_logic;             --input ที่นำมารีเซตค่า โดยในที่นี้
                                            --ออกแบบให้shiftไป 1 คาบfs จาก fq
          d_and : out std_logic);           --output
end anddff;

architecture Behavioral of anddff is
signal d : std_logic;
begin
process (fq, RESET)
begin
  if reset='1' then   --asynchronous RESET active High    --ถ้ามีreset d =0
    d <= '0';
  elsif (fq'event and fq='1') then  --CLK rising edge      --ถ้าไม่มีresetแล้วเช็คขอบขาของ
    d <= zero;                                            --fq ถ้ามี = zero
  end if;
end process;

d_and <= d and fq;                                        -- d and fq

end Behavioral;
```

K_CONTROLER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity k_control3 is                                    --วงจรควบคุมค่า k
        generic(n: integer := 3);                       --จำนวนบิทของ k
    Port ( reset : in std_logic;                        --resetวงจร
        lead : in std_logic;                            --input lead
        lag : in std_logic;                             --input lag
        high : in std_logic;                            --input high
        low : in std_logic;                             --input low
        clk : in std_logic;                             --input จังหวะการอ่านค่า ในที่นี้คือ fq_q
        nbit : out std_logic_vector(n downto 0));       --outputค่า k
end k_control3;
architecture Behavioral of k_control3 is
        COMPONENT highlow_to_updw
        PORT(
                high : IN std_logic;
                low : IN std_logic;
                lead : IN std_logic;
                lag : IN std_logic;
                up : OUT std_logic;
                dw : OUT std_logic
                );
        END COMPONENT;
        COMPONENT kmysar2
        generic(n:integer);
        PORT(
                reset : IN std_logic;
                up : IN std_logic;
                dw : IN std_logic;
                clk : in std_logic;
                kout : OUT std_logic_vector(n downto 0)
                );
        END COMPONENT;
        signal up ,dw: std_logic;                       --สัญญาณเชื่อมต่อ
begin
        stage1 : highlow_to_updw PORT MAP(              --เชื่อมต่อวงจร highlow
                high => high,
                low => low,
                lead => lead,
                lag => lag,
                up => up,
                dw => dw
        );
                Inst_ksar: kmysar2 generic map(n)       --เชื่อมต่อวงจร sar
                PORT MAP(reset ,up ,dw,clk,nbit);
```

```
end Behavioral;
```

HIGHLOW_TO_UPDW

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity highlow_to_updw is                          --วงจรแปลง highlow_to_updw
   Port ( high : in std_logic;
          low : in std_logic;
          lead : in std_logic;
          lag : in std_logic;
          up : out std_logic;
          dw : out std_logic);
end highlow_to_updw;

architecture Behavioral of highlow_to_updw is

begin
          up <= low or(lag  and not high);
          dw <= high or (lead and not low);
end Behavioral;
```

SAR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Kmysar2 is                                  --วงจร sar
   generic(n: integer := 3);                       --ค่าบิทของsar
   Port ( reset : in std_logic;                    --reset วงจร
          up : in std_logic;                       --ค่าที่บอกเพิ่ม ค่า k
          dw : in std_logic;                       --ค่าที่บอกเพิ่ม ค่า k
          clk : in std_logic;                      --จังหวะการอ่านค่าในนี้คือfq_q
          kout : out std_logic_vector(n downto 0));--k output
```

```
end Kmysar2;

architecture Behavioral of Kmysar2 is
        signal k            : std_logic_vector(n downto 0);     --รองรับค่าk
        signal count,cnt : integer range 0 to n+1 := 0;         --ตัวนับค่าที่จะset
                                                                --count ของ dw
                                                                --cnt    ของ up
begin
        process(clk,up,dw,count,reset,cnt)
        begin
                        if reset = '1' then                     --reset ค่า
                                kout <= (others =>'0');
                                count <= 0;
                                cnt   <= 0;
                        elsif rising_edge(clk) then             --ตรวจสอบกสนอ่านค่า
                         if up ='1' or dw = '1' then            --ตรวจสอบจังหวะอ่านค่ามีdata
                                                                --โดยdata = 1 ถ้า up=0 ถ้าdw
                                 if     count /= n+1 then --ตรวจสอบไม่ใช่บิท msn
                                        if dw = '0' then  --ตรวจสอบdata
                                                kout(count) <= '1';--เซตค่าเมื่อเป็น1
                                                count <= count+1;--เลื่อนไปบิทต่อไป
                                                cnt <= 0;
                                        elsif dw = '1' then     --เมื่อ data เป็น 0
                                                if      cnt /= n+1 then --ตรวจสอบ
                                                                --ไม่ใช่บิท msn
                                                kout(cnt) <= '0';เซตค่า
                                                cnt <= cnt + 1 ;
                                                count <= 0;
                                        elsif cnt = n+1 then   --เซตค่าเมื่อเป็น
                                                                --บิทmsb
                                                cnt <= 0 ;
                                                kout(0) <= '0';

                                                end if;
                                        end if;
                                elsif count = n+1 then          --เซตค่าเมื่อเป็นบิทmsb
                                        count <= 0;
                                        if dw = '0' then        --เซตค่าเมื่อเจอdata=1
                                                kout(0) <= '1';
                                        else                    --เซตเมื่อ เป็น 0
```

```
                                        kout(0) <= '0';
                                    end if;
                                end if;
                            end if;
                            end if;
        end process;
end Behavioral;
```