# การประยุกต์ใช้ FPGA ในการออกแบบอินเตอร์เฟสของฮาร์ดดิสคอนโทรเลอร์สำหรับเครื่องเอ็กซ์เรย์คอมพิวเตอร์

# INTERFACE DESIGN-BASED FPGA FOR HARD DISK CONTROLLER OF A X-RAY COMPUTER



นายสีหชาติ สดับธรรมารักษ์

MR.SEEHACHAT SADUBTHUMMARAK

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2539

ISBN 974-621-749-6

# INTERFACE DESIGN-BASED FPGA FOR HARD DISK CONTROLLER OF A X-RAY COMPUTER

SEEHACHAT SADUBTHUMMARAK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHCOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1996

ISBN 974-621-749-6

| | |
|---|---|
| **หัวข้อวิทยานิพนธ์** | การประยุกต์ใช้ FPGA ในการออกแบบอินเตอร์เฟสของฮาร์ดดิสคอนโทรเลอร์สำหรับเครื่องเอ็กซ์เรย์คอมพิวเตอร์ |
| **นักศึกษา** | นายสีหชาติ สดับธรรมารักษ์ |
| **อาจารย์ผู้ควบคุมวิทยานิพนธ์** | รศ.ดร. กอบชัย เดชหาญ |
| **ระดับการศึกษา** | วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า |
| **ภาควิชา** | วิศวกรรมโทรคมนาคม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง |
| **พ.ศ.** | 2539 |

## บทคัดย่อ

การออกแบบทางด้านดิจิตอลในปัจจุบัน อุปกรณ์ประเภท FPGA กำลังได้รับความสนใจศึกษาจากนักออกแบบและวิจัยทั่วไป เนื่องจากประสิทธิภาพ และความสะดวกต่อการออกแบบ อีกทั้งสามารถประยุกต์ใช้ได้กับงานในหลายรูปแบบ วิทยานิพนธ์ฉบับนี้ได้เสนอรูปแบบหนึ่งของการออกแบบใช้งานอุปกรณ์ FPGA เพื่อใช้สร้างเป็นองค์ประกอบหลักของวงจร โดยทำงานเสมือนกับเป็นตัวควบคุมฮาร์ดดิส สำหรับเครื่องเอกซ์เรย์คอมพิวเตอร์ ซึ่งมีลักษณะการทำงานเฉพาะ ไม่เหมือนกับมาตราฐานที่มีใช้ทั่วไปในปัจจุบัน และหาไม่ได้ในท้องตลาด เนื่องจากถูกยกเลิกการผลิตไป อีกทั้งการจะให้ยุติการใช้เครื่องเอกซ์เรย์ยังไม่ได้ เพราะการลงทุนในมูลค่าของเครื่องมือยังไม่คุ้มค่า การทำงานจะใช้โปรแกรมควบคุมการทำงาน โดยไม่ต้องแก้ไขหรือมีการดัดแปลงที่ตัวเครื่องหลักแต่ประการใด

| | |
|---|---|
| **Thesis Title** | Interface design-based FPGA for hard disk controller of a X-ray computer |
| **Student** | Mr. Seehachat Sadubthummarak |
| **Thesis Advisor** | Assoc.Prof.Dr. Kobchai Dejhan |
| **Level of Study** | Master of Engineering |
| **Department** | Telecommunication Engineering<br>King Mongkut's Institute of technology Ladkrabang |
| **Year** | 1996 |


## Abstract

Field Programmable Gate Array (FPGA) devices are very interesting for circuit designers and researcher according to the high efficientcy . This thesis concerns with an application of FPGA acting as a hard disk controller for x-ray computer. This hard disk controller of X-ray computer is specified and not the same as the general hard disk controller in the market. The company does not produce this type of controller and cannot support the spare part. The best solution is to lay the new X-ray computer, but it is expensive. Therefore, an application of FPGA design is suitable to replace the old type of hard disk controller without any modification of the principle X-ray computer.

# กิตติกรรมประกาศ

การจัดทำวิทยานิพนธ์ในครั้งนี้สำเร็จลุล่วงไปด้วยดี เพราะได้รับความเมตรากรุณาจากท่านอาจารย์ รศ.ดร.กอบชัย เดชหาญ ซึ่งได้ให้คำปรึกษาและแนะนำผู้วิจัยตลอดมา ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่าน และกราบขอบพระคุณเป็นอย่างสูง

ผู้วิจัยขอขอบคุณ บริษัท เอที. รีเซิช จำกัด ที่ได้ให้การสนับสนุนทั้งในด้านอุปกรณ์เครื่องมือและข้อมูลที่จำเป็นในการศึกษาวิจัยและทดสอบเป็นอย่างสูง

ขอขอบพระคุณ ผู้เชี่ยวชาญ และผู้ทรงคุณวุฒิ ที่ได้กรุณาชี้แนะแนวทางในการทำวิทยานิพนธ์ ตลอดจนขอขอบคุณเพื่อน ๆ ทุกท่าน ที่ได้ช่วยเหลือและให้กำลังใจต่อผู้วิจัยตลอดมา

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

สีหชาติ สดับธรรมารักษ์

# สารบัญ

หน้า

# สารบัญตาราง

# สารบัญภาพ

หน้า

## สารบัญภาพ (ต่อ)

หน้า

**สารบัญภาพ (ต่อ)**

หน้า

## สารบัญภาพ (ต่อ)

หน้า

# บทที่ 1
# บทนำ

ประเทศไทยได้มีการใช้เครื่องเอ็กซ์เรย์คอมพิวเตอร์ หรือเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง (Computerized Tomography : CT) มานานพอสมควร แต่เนื่องจากราคาของเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวางมีราคาสูงมาก (ประมาณ 15-30 ล้านบาท) ผู้ให้บริการคอมพิวเตอร์ถ่ายภาพตัดขวางจึงได้พยายามบำรุงรักษา เครื่องคอมพิวเตอร์ถ่ายภาพตัดขวางให้ใช้งานได้ยาวนานที่สุด แต่เครื่องคอมพิวเตอร์ถ่ายภาพตัดขวางบางรุ่นที่ได้ใช้งานมานานมาก ๆ จะประสบปัญหาด้านการจัดหาอุปกรณ์ทดแทนชิ้นส่วนที่เสื่อมสภาพไป เนื่องจากโรงงานผู้ผลิตได้ยกเลิกการผลิตอุปกรณ์ดังกล่าว และเครื่องก็ได้สิ้นสุดระยะรับประกันจากตัวแทนจำหน่ายไปแล้ว

เครื่องคอมพิวเตอร์ถ่ายภาพตัดขวางยี่ห้อ ELSCINT รุ่น EXEL 2002 ได้นำเข้ามายังประเทศไทยเป็นเวลาไม่ต่ำกว่า 15 ปี และขณะนี้กำลังประสบปัญหาด้านการบำรุงรักษาเนื่องจากไม่สามารถจัดหาอุปกรณ์ทดแทนได้ดังกล่าวมาข้างต้น รวมทั้งอุปกรณ์ด้านการจัดเก็บข้อมูลแบบฟิกดิส (fixed disk) ที่ใช้ในการจัดเก็บข้อมูลการเอ็กซ์เรย์คนไข้ก่อนที่จะนำไปถ่ายลงฟิล์ม ซึ่งได้หมดอายุการใช้งานไปแล้ว ทำให้การเก็บข้อมูลคนไข้ต้องกระทำผ่านอุปกรณ์จัดเก็บข้อมูลแบบฟลอปพี้ดิส (floppy disk) ซึ่งมีความเร็วในการจัดเก็บข้อมูลช้ากว่าฟิกดิสมาก เป็นผลให้เกิดความล่าช้าในการให้บริการ เนื่องจากการให้บริการคนไข้แต่ละรายจะต้องทำการเอ็กซ์เรย์ไม่ต่ำกว่า 12-20 ภาพ ขึ้นอยู่กับบริเวณที่ทำการเอ็กซ์เรย์ ซึ่งจะใช้เวลาทั้งสิ้นประมาณ 10-20 นาที สำหรับฟิกดิส และ 40-60 นาที สำหรับฟลอปพี้ดิส

จากปัญหาความล่าช้าของการใช้เครื่องคอมพิวเตอร์ถ่ายภาพตัดขวางดังกล่าว ทำให้ศูนย์บริการที่มีคนไข้มาใช้บริการจำนวนมากๆ ต้องประสบปัญหาเกี่ยวกับการให้บริการ คือ คนไข้ต้องรอคิวนาน (คนไข้รู้สึกเสียเวลา) และอาจย้ายไปใช้บริการศูนย์อื่นที่อยู่บริเวณใกล้เคียง ดังนั้นจึงได้มีแนวความคิดที่จะประยุกต์ใช้อุปกรณ์จัดเก็บข้อมูลแบบฮาร์ดดิส (hard disk) ในปัจจุบัน มาทดแทนอุปกรณ์จัดเก็บข้อมูลแบบฟิกดิส ยี่ห้อ SHUGART ที่ใช้อยู่ในเครื่อง CT เพื่อให้เครื่อง CT สามารถทำงานได้อย่างเต็มประสิทธิภาพดังเดิม

จากการศึกษาการทำงานของตัวควบคุมฟิกดิส (fixed disk controller) ยี่ห้อ SHUGART รุ่น SA4600 ซึ่งใช้ในเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวางยี่ห้อ ELSCINT รุ่น EXEL 2002 พบว่า

สามารถนำเครื่องไมโครคอมพิวเตอร์มาทำงานทดแทนได้ โดยการออกแบบแผงวงจรอินเตอร์เฟส (interface card) เพิ่มเติมเพื่อใช้ในการติดต่อกับเครื่อง CT แทน SA4600 ตัวเดิม แต่เนื่องจากข้อจำกัดทางด้านพื้นที่การติดตั้งต้องอยู่บนตู้ขนาด 19 นิ้วเดิม ทำให้ไม่เหมาะที่จะใช้เครื่องไมโครคอมพิวเตอร์ทั่วไปที่มีขนาดใหญ่ จึงได้ทำการเลือกใช้คอมพิวเตอร์อุตสาหกรรม (industrail computer) ที่มีขนาดเล็กและสามารถติดตั้งบนตู้เดิมได้มาใช้งานแทน

ในการออกแบบแผงวงจรอินเตอร์เฟสระหว่างเครื่องไมโครคอมพิวเตอร์และเครื่อง CT นั้น จำเป็นต้องใช้อุปกรณ์ไอซีจำนวนมาก จนไม่สามารถจัดวางลงบนแผ่นวงจรพิมพ์ขนาด 4.5" x 6.5" ตามมาตรฐานของคอมพิวเตอร์อุตสาหกรรมที่กำหนด และไม่สามารถจัดหาอุปกรณ์แบบ SMT (Surface Mount Technology) ซึ่งมีขนาดเล็กมาใช้ได้ ดังนั้นจึงได้ทำการออกแบบแผงวงจรอินเตอร์เฟส โดยใช้อุปกรณ์ FPGA (Field-Programmable Gate Array) ซึ่งนอกจากจะแก้ปัญหาด้านขนาดของแผ่นวงจรพิมพ์แล้ว ยังช่วยให้การพัฒนารวดเร็วขึ้น รวมทั้งสะดวกต่อการบำรุงรักษาในภายหลัง

วิทยานิพนธ์นี้ได้กล่าวถึงการพัฒนาชุดจำลองระบบจัดเก็บข้อมูลแบบฟิกดิส รุ่น SA4600 ที่ใช้บนเครื่อง CT (SA6400 emulator) โดยใช้เครื่องคอมพิวเตอร์อุตสาหกรรมที่มี สถาปัตยกรรมแบบบัสมาตรฐาน (Standard Bus) เป็นตัวควบคุมการทำงาน และใช้อุปกรณ์ FPGA ของ XILINX ตระกูล XC3000 ในการออกแบบแผงวงจรอินเตอร์เฟสเพื่อให้เครื่อง CT สามารถทำการจัดเก็บข้อมูลได้อย่างเต็มประสิทธิภาพ โดยไม่ต้องแก้ไขหรือดัดแปลงในส่วนของเครื่อง CT

เนื่องจากในการพัฒนาชุดอุปกรณ์จัดเก็บข้อมูลนี้ จะต้องทำการทดลองกับเครื่อง CT ที่ใช้งานจริง ณ.จังหวัดนครราชสีมา ซึ่งนอกจากอาจทำความเสียหายให้กับส่วนควบคุมหลักของเครื่อง CT แล้ว ยังมีความไม่สะดวกในด้านของพื้นที่ที่จัดวางอุปกรณ์พัฒนา รวมทั้งไม่สามารถทำการทดลองได้อย่างต่อเนื่องเมื่อมีคนไข้มาใช้บริการ ดังนั้นจึงได้มีการศึกษาและสร้างเครื่อง CT จำลองในส่วนที่ติดต่อกับอุปกรณ์จัดเก็บข้อมูลแบบฟิกดิสอย่างง่ายๆ เพื่อช่วยในการพัฒนาชุดอุปกรณ์จัดเก็บข้อมูลก่อนที่จะนำไปทดลองกับเครื่อง CT จริงในภายหลัง

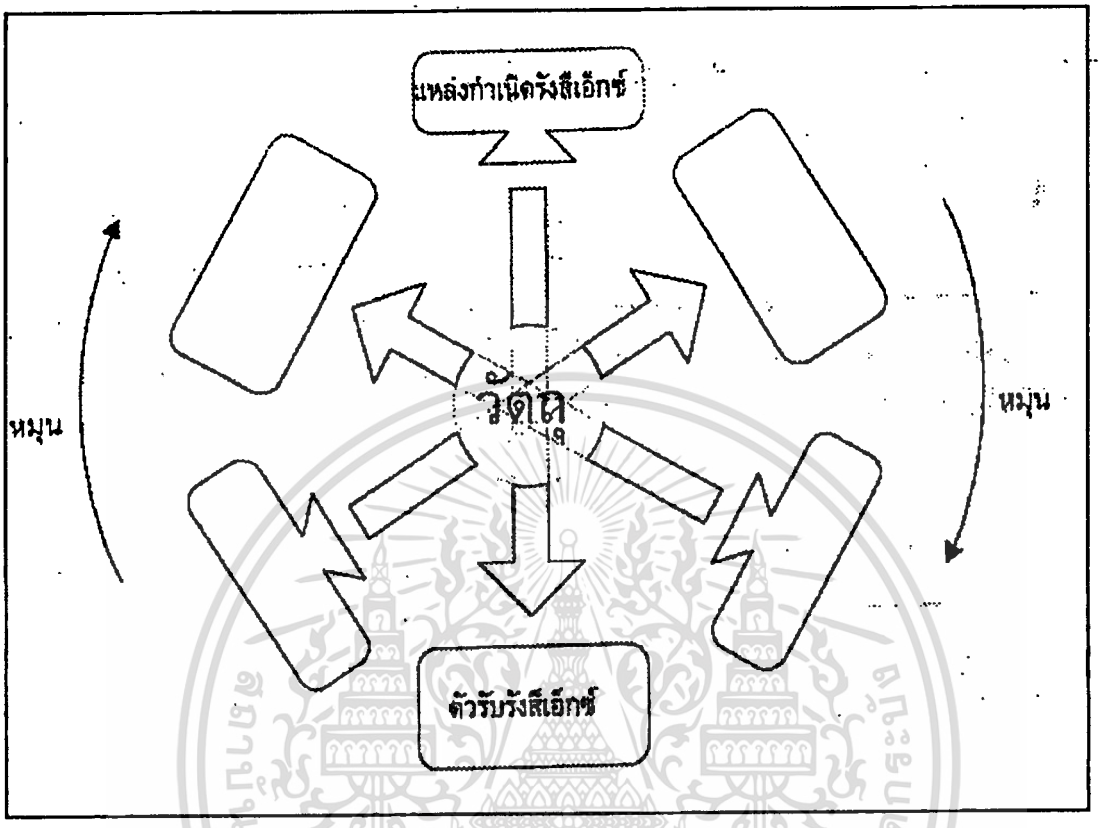
# บทที่ 2
# หลักการทั่วไปของเครื่องเอกซ์เรย์คอมพิวเตอร์

อุปกรณ์เครื่องเอ็กซ์เรย์คอมพิวเตอร์ คือเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง (Computer Tomography : CT ) จัดเป็นเครื่องมือทางการแพทย์ ที่สามารถช่วยให้แพทย์ทำการวินิจฉัยโรคได้ถูกต้องแม่นยำยิ่งขึ้น ภาพที่ได้จากเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวางจะเป็นภาพที่มองโดยเปรียบเสมือนกับ การนำมีดไปตัดขวางวัตถุหรืออวัยวะออกมาดู (แต่แท้จริงแล้วไม่ได้นำมีดไปตัดจริง) ทำให้สามารถมองเห็นองค์ประกอบหรือชิ้นส่วนอวัยวะที่อยู่ภายในได้อย่างชัดเจน

หลักการทำงานโดยทั่วไปของเครื่องเอ็กซ์เรย์คอมพิวเตอร์ หรือเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง จะใช้หลักการที่ใช้สื่อหรือตัวกลางที่สามารถเก็บข้อมูลของสิ่งที่อยู่ภายในของวัตถุที่ไม่สามารถมองเห็นได้จากภายนอกโดยใช้ตามนุษย์ธรรมดาได้ โดยในทางการแพทย์ก็ได้มีการใช้รังสีเอ็กซ์ซึ่งสามารถทะลุคน (หรือวัตถุ) ได้ออกมา แล้วทำการรับข้อมูลที่ออกมาทางด้านตรงข้าม

แหล่งกำเนิดรังสีเอ็กซ์จะฉายรังสีเอ็กซ์ ผ่านเข้าไปยังวัตถุที่ต้องการสร้างภาพตัดขวาง เช่น ผู้ป่วยหรือคนไข้ในโรงพยาบาล รังสีเอ็กซ์จะทะลุวัตถุออกมาทางด้านตรงข้าม โดยที่ความเข้มของรังสีเอ็กซ์ที่ผ่านทะลุวัตถุออกมา จะมีค่าเปลี่ยนไปตามคุณสมบัติการดูดซับรังสีเอ็กซ์ของวัตถุนั้น และถูกตรวจรับโดยตัวรับรังสีที่อยู่ในด้านตรงข้ามกับแหล่งกำเนิดรังสีเอ็กซ์ ตัวรับรังสีจะทำการแปลงค่าความเข้มของรังสีเอ็กซ์ ให้เป็นค่าที่สามารถนำไปประมวลผลโดยคอมพิวเตอร์ได้ต่อไป ชุดของข้อมูลที่รับได้นี้เรียกว่าโปรเจคชั่น (projection) หรือโปรไฟล์ (profile) ซึ่งหลังจากรับค่าโปรเจคชั่นแล้วจะหมุนแหล่งกำเนิดรังสีเอ็กซ์ และตัวรับรังสีเอ็กซ์ไปยังตำแหน่งถัดไป เพื่อทำการฉายรังสีเอ็กซ์และรับข้อมูลโปรเจคชั่นต่อไป กระทำเช่นนี้ไปจนครบรอบวัตถุก็จะได้ข้อมูลเป็นโปรเจคชั่นของวัตถุ โดยอาจทำการแบ่งรอบวัตถุซึ่งมีขนาด 360 องศา ออกเป็น 60, 120, 180 หรือ 360 โปรเจคชั่น (หรือมากกว่า) ซึ่งแต่ละโปรเจคชั่นก็จะห่างกันเป็นมุม 6, 3, 2 และ 1 องศา หรือน้อยกว่าตามลำดับ เมื่อได้โปรเจคชั่นของข้อมูลครบตามความต้องการแล้วจะนำข้อมูลเหล่านั้นไปทำการคำนวณเพื่อสร้างภาพตัดขวางของวัตถุนั้น โดยใช้คอมพิวเตอร์เป็นตัวช่วยในการคิดคำนวณเพื่อให้ผลออกมามีความถูกต้องและรวดเร็วยิ่งขึ้น เสร็จแล้วจึงนำภาพตัดขวางที่คำนวณสร้างมาได้นั้น นำออกแสดงผลทางจอภาพหรือพิมพ์ออกมาตามลำดับต่อไป

ภาพที่ 1




แสดงหลักการทำงานโดยทั่วไปของการสร้างภาพตัดขวางโดยใช้รังสีเอ็กซ์

<u>เครื่องเอ็กซ์เรย์คอมพิวเตอร์ถ่ายภาพตัดขวางที่ใช้งานอยู่ทั่วไป</u>

การติดตั้งเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง เพื่อใช้งานโดยทั่วๆ ไป มักจะมีสถานที่จัดเป็นห้องสำหรับติดตั้งเครื่องโดยเฉพาะ ซึ่งประกอบด้วยห้องที่เป็นส่วนสำคัญหลักๆ อยู่ 2 ห้องคือ

1. ห้องสแกน หรือห้องตรวจสอบคนไข้ (scan or examination room) เป็นห้องที่ให้คนไข้นอนเพื่อตรวจและสแกนภาพตัดขวาง ภายในห้องขณะที่ทำการเก็บข้อมูลจะมีรังสีเอ็กซ์ฉายออกมา ดังนั้นห้องสแกนนี้จะต้องมีการป้องกันอย่างดีเพื่อไม่ให้รังสีเอ็กซ์รั่วไหลออกมาภายนอกได้ และคนไข้จะนอนให้เครื่องตรวจสอบตามลำพัง ภายในห้องนี้จะมีเครื่องสำหรับสแกนเก็บข้อมูลภาพที่เรียกว่าแกนทรี (gantry) เตียงสำหรับให้คนไข้นอนตรวจ และอาจมีส่วนอื่นๆ อีกก็ได้ เช่น เครื่องกำเนิดแรงดันไฟฟ้าแรงสูงสำหรับหลอดรังสีเอ็กซ์ (X-ray high voltate generator) เป็นต้น

2. ห้องควบคุมหรือห้องปฏิบัติงาน (operator and control room) เป็นบริเวณที่ใช้ปฏิบัติงานของเจ้าหน้าที่หรือแพทย์ ในการควบคุมให้เครื่องทำการถ่ายภาพตัดขวางของคนไข้ให้เป็นไป

ตามที่ต้องการ ภายในห้องจะประกอบด้วย แผงควบคุมการทำงานของเครื่อง (console) จอแสดงภาพตัดขวางที่สร้างได้ กล้องมัลติฟอร์แมต ระบบคอมพิวเตอร์ และอุปกรณ์ประกอบอื่นๆ เช่น แหล่งจ่ายไฟ เป็นต้น โดยคอมพิวเตอร์และอุปกรณ์อื่นๆ อาจทำการกั้นเป็นห้องย่อยแยกต่างหากออกจากกันก็ได้ เพื่อความสะดวกเรียบร้อยสวยงาม

ทั้งห้องสแกนและห้องควบคุม จะสามารถมองทะลุเห็นถึงกันได้ โดยมองผ่านกระจกตะกั่ว ทำให้เจ้าหน้าที่ควบคุมเครื่องสามารถปฏิบัติงานได้อย่างสะดวกและปลอดภัย

<u>ส่วนประกอบหลักสำคัญของเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง</u>

สำหรับส่วนประกอบต่างๆ ที่สำคัญของเครื่องเอ็กซ์เรย์คอมพิวเตอร์ หรือเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง สามารถแยกตามลักษณะหน้าที่การทำงานหลักๆ ได้ดังนี้

1. แกนทรี (Gantry) : เป็นส่วนที่หมุนแหล่งกำเนิดรังสีเอ็กซ์และ/หรือตัวรับข้อมูลรังสีเอ็กซ์ที่ออกจากวัตถุ หมุนไปรอบวัตถุ เพื่อทำการเก็บข้อมูลนำไปใช้ในการสร้างภาพตัดขวางต่อไป

2. ระบบรังสีเอ็กซ์และส่วนควบคุมการฉายรังสีเอ็กซ์ : ทำหน้าที่กำเนิดรังสีเอ็กซ์ตามที่ต้องการฉายผ่านเข้าไปยังวัตถุหรือคนไข้ เพื่อให้ได้ข้อมูลที่อยู่ภายในวัตถุหรือคนไข้ไปสร้างภาพตัดขวางต่อไป สำหรับหลอดรังสีเอ็กซ์ที่ใช้กับเครื่อง CT จะต้องมีความสามารถเปิดฉายรังสีเอ็กซ์ได้โดยต่อเนื่องได้นานกว่าเครื่องฉายรังสีเอ็กซ์ธรรมดาทั่วไปมาก ส่วนการควบคุมการฉายรังสีเอ็กซ์ ได้แก่การควบคุมกระแส (mA) และแรงดัน (KV) ของหลอดรังสีเอ็กซ์ การเลือกเวลาในการฉายรังสี(เวลาในการรับข้อมูล) ซึ่งการควบคุมจะกระทำผ่านทางแผงควบคุม (console) โดยใช้วงจรอิเล็คทรอนิคส์และคอมพิวเตอร์ช่วยในการควบคุมให้มีค่าตรงตามความต้องการ

3. ระบบรับข้อมูลจากรังสีเอ็กซ์เข้าคอมพิวเตอร์ : ทำหน้าที่ในการรับข้อมูลจากรังสีเอ็กซ์ที่ผ่านวัตถุหรือผ่านคนไข้ออกมา แล้วแปลงข้อมูลนั้นให้อยู่ในรูปแบบที่คอมพิวเตอร์สามารถนำไปใช้ประมวลผลเพื่อการสร้างภาพตัดขวางได้ต่อไป ส่วนนี้ยังประกอบด้วยส่วนย่อยๆ ที่เป็นองค์ประกอบหลักอีกดังนี้

- ตัวรับรังสีเอ็กซ์
- วงจรขยายสัญญาณที่ได้จากตัวรับรังสีเอ็กซ์
- วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล
- วงจรควบคุม

4. เตียงคนไข้และส่วนควบคุมเตียงคนไข้ : ส่วนนี้จะเป็นที่สำหรับคนไข้ได้นอน เพื่อทำการถ่ายภาพตัดขวาง ก่อนถ่ายภาพตัดขวางจะต้องทำการจัดตำแหน่งของคนไข้ให้อยู่ในตำแหน่งที่ถูกต้องในบริเวณที่ต้องการถ่ายภาพ หลังจากนั้นจะทำการเลื่อนเตียงคนไข้เข้าสู่แกนทรี เพื่อทำการถ่ายภาพตัดขวางคราวละภาพ (slice) หลังจากเสร็จหนึ่งภาพ ก็จะเลื่อนเตียงคนไข้ให้อยู่ในตำแหน่งถ่ายภาพถัดไป ซึ่งปกติจะกระทำการเลื่อนครั้งละ 5 หรือ 10 มิลลิเมตร ขึ้นกับความต้องการ ในระหว่างการถ่ายแต่ละภาพ เตียงและคนไข้จะต้องอยู่ในสภาพนิ่งให้มากที่สุดเพื่อจะได้ภาพที่คมชัด

5. ระบบคอมพิวเตอร์และโปรแกรมซอฟแวร์ : ทำหน้าเกี่ยวกับการประมวลผลข้อมูลการสร้างภาพตัดขวางเพื่อใช้ในการสร้างภาพตัดขวางที่ถูกต้องและรวดเร็ว

6. ส่วนแสดงผลภาพ : ทำหน้าที่นำภาพตัดขวางอวัยวะที่คำนวณสร้างได้ออกมาแสดงทางจอภาพ โดยผู้ควบคุมหรือแพทย์จะสามารถควบคุมการแสดงผลภาพ เพื่อให้แสดงในส่วนที่ต้องการความชัดเจนที่สุดได้

7. กล้องมัลติฟอร์แมต (ตัวเลือก) : ทำหน้าที่นำภาพตัดขวางที่ได้ ถ่ายลงบนฟิล์ม ในลักษณะเดียวกับฟิล์มรังสีเอ็กซ์เรย์ เพื่อนำไปใช้ภายหลังต่อไป โดยไม่ต้องใช้จอภาพในการดูภาพตัดขวาง โดยภาพที่นำลงบนฟิล์มจะสามารถควบคุมจำนวนภาพที่มีต่อฟิล์มเดียวกันได้ เช่น 1, 3, 6, 9, หรือ 12 ฟิล์ม ต่อหนึ่งฟิล์ม ทำให้สามารถมองเห็นภาพส่วนต่างๆ ในฟิล์มเดียวกัน ส่งผลให้แพทย์สามารถวินิจฉัยได้สะดวกขึ้น แต่ละภาพที่ปรากฏบนฟิล์มจะเป็นแบบเดียวกับที่ปรากฏบนจอภาพของส่วนแสดงผลภาพตัดขวาง

# บทที่ 3
# คุณลักษณะโดยทั่วไปของอุปกรณ์ประเภท FPGA

ในบทนี้จะกล่าวถึงคุณสมบัติทั่วไปที่มีอยู่ในตัวอุปกรณ์ FPGA ซึ่งได้อ้างอิงกับอุปกรณ์ที่เป็นผลิตภัณฑ์ของ Xilinx ตระกูล XC3000 เนื่องจากในการวิจัยนี้อาศัยอุปกรณ์ในตระกูลนี้มาใช้งาน จัดเป็นตระกูลในประเภท LCA (Logic Cell Array) ที่มีประสิทธิภาพและความจุสูง สถาปัตยกรรมโครงสร้างภายในอุปกรณ์ประกอบด้วยส่วนโครงสร้างสำคัญ 3 ส่วนด้วยกันคือ I/O Bocks (IOBs), Configurable Logic Bocks (CLBs) และ resources สำหรับการเชื่อมต่อระหว่างกัน ลักษณะโครงสร้างโดยทั่วไปของอุปกรณ์ LCA แสดงดังภาพที่ 2

การใช้งานหน้าที่ทางตรรกและการเชื่อมต่อระหว่างกันของ LCA ถูกกำหนดด้วยข้อมูลโปรแกรมโครงแบบ (Configuration program data) ที่เก็บอยู่ในเซลล์หน่วยความจำแบบสถิตภายใน (static memory cells) ซึ่งโปรแกรมนี้สามารถสั่งให้ทำงานในลักษณะต่าง ๆ ตามความเหมาะสมของแต่ละระบบที่ต้องการ ข้อมูลโปรแกรมอาจอยู่ในหน่วยความจำภายนอก เช่น EEPROM, EPROM หรือ ROM ที่อยู่บนแผงวงจร หรือเก็บอยู่ในฟลอปพี้ดิสหรือฮาร์ดดิสก์ได้ เมื่อทำการเปิดเครื่องหรือป้อนแหล่งจ่ายไฟเข้าไป ข้อมูลโปรแกรมจะถูกดึงลงมายังอุปกรณ์ LCA ในทันที

คุณสมบัติเด่นของอุปกรณ์ตระกูล XC3000 คือมีความสามารถและความจุทางตรรกที่มากมาย รูปร่างภายนอกของชิ้นอุปกรณ์หลากหลาย อีกทั้งช่วงอุณหภูมิการทำงานและความเร็วสูง

## สถาปัตยกรรมโครงสร้าง

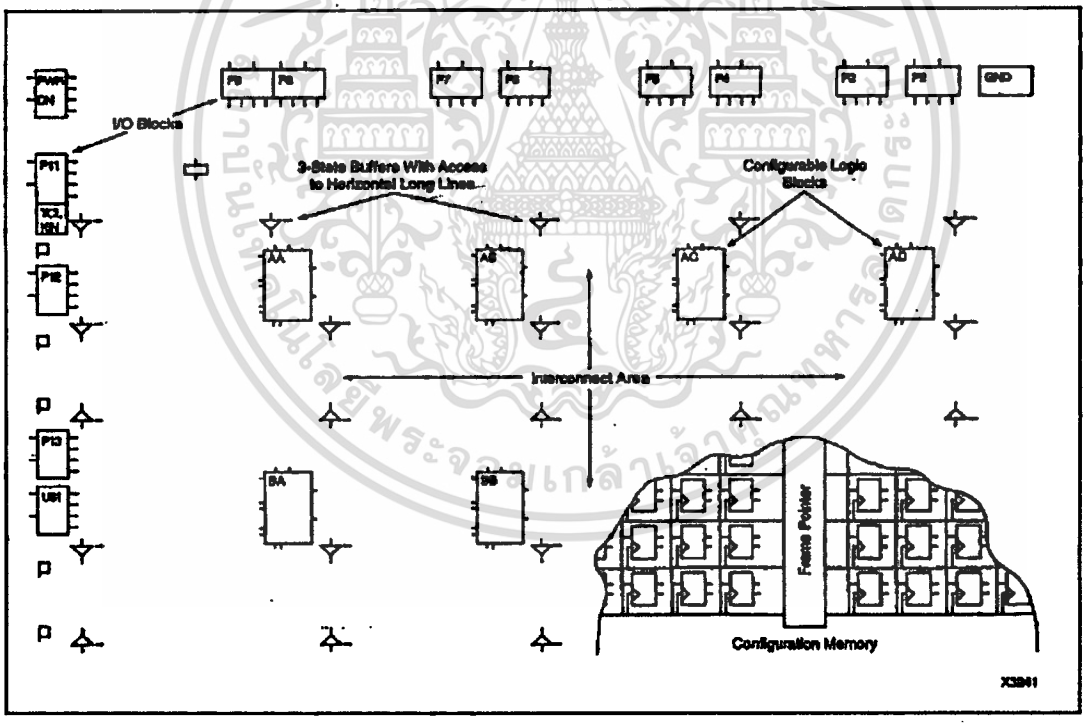
บริเวณโดยรอบกรอบสี่เหลี่ยมของ configurable IOBs จัดการเกี่ยวกับการอินเตอร์เฟสระหว่างแถวลำดับตรรก (logic array) ภายในกับขาต่าง ๆ ของอุปกรณ์ ส่วนแถวของ CLBs ให้การทำงานทางตรรกตามที่ผู้ใช้งานกำหนด และส่วน resources การเชื่อมต่อระหว่างกันถูกใช้ในการสร้างโครงข่าย และนำพาสัญญาณตรรกระหว่างบล็อก คล้ายกับลายเส้นทองแดงในแผงวงจร

บล็อกการทำงานทางตรรกสามารถทำงานให้เป็นผลจริงได้โดยลักษณะการโปรแกรมที่เป็นแบบตารางเปิดดู ส่วนการทำงานอื่นเพิ่มเติมทำได้โดยโปรแกรมควบคุมตัวมัลติเพลกเซอร์ และการเชื่อมต่อระหว่างกันในโครงข่ายระหว่างบล็อกต่าง ๆ ทำได้โดยการเชื่อมต่อส่วนที่เป็นโลหะด้วยโปรแกรมที่ควบคุมผ่านทางตัวทรานซิสเตอร์

การทำงานของอุปกรณ์ LCA จะมีความถูกต้องเที่ยงตรง เมื่อดึงเอาโปรแกรมโครงแบบเข้ามาไว้ภายใน และกระจายออกไปตามเซลล์หน่วยความจำโครงแบบต่าง ๆ การดึงหรือโหลดเอาโปรแกรมโครงแบบเข้ามานี้จะเกิดขึ้นทันทีที่ป้อนแหล่งจ่ายไฟเข้าไป หรือจากคำสั่งให้โหลดข้อมูลโดยตรงใน LCA จะประกอบด้วยสัญญาณตรรกและสัญญาณควบคุมที่เป็นแบบธรรมดาและแบบอัตโนมัติ

ข้อมูลโปรแกรมอาจอยู่ในรูปแบบขนานหรืออนุกรมต่อ ๆ กันก็ได้ ระบบพัฒนาที่มีชื่อเรียกว่า XACT จะช่วยในการสร้างหรือกำเนิดขบวนบิตข้อมูลของโปรแกรมที่ใช้สำหรับเป็นโครงแบบหรือกำหนดการทำงานให้กับอุปกรณ์ LCA ซึ่งขบวนการทำงานดึงหรือเก็บเกี่ยวกับหน่วยความจำจะขึ้นอยู่กับการใช้งานในหน้าที่ต่าง ๆ ไม่เกี่ยวข้องกัน

ภาพที่ 2




แสดงโครงสร้าง โดยทั่วไปของอุปกรณ์ FPGA

- หน่วยความจำโครงแบบ

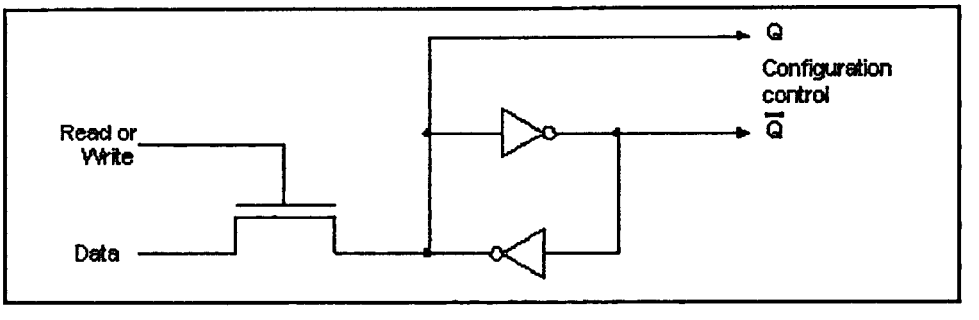
ใน LCA ส่วนของ หน่วยความจำโครงแบบ จะใช้เซลล์หน่วยความจำแบบสถิต ซึ่งได้รับการออกแบบขึ้นมาให้มีความเชื่อถือได้สูง อีกทั้งปราศจากสัญญาณรบกวนหรือแปลกปลอมเข้า

มาแทรกแซง ความแน่นอนของหน่วยความจำโครงแบบของอุปกรณ์ LCA บนพื้นฐานการออกแบบนี้ทำให้สามารถรับรองได้ว่าจะไม่เกิดเหตุการณ์ที่อยู่นอกเหนือจากเงื่อนไขที่ต้องการ เปรียบเทียบกับอุปกรณ์ในประเภทที่โปรแกรมได้อย่างอื่น พบว่าหน่วยความจำแบบสถิตนี้ให้ผลดีที่สุดทั้งในด้านความจุ ประสิทธิภาพ และมีความเชื่อถือได้สูง ดังแสดงในภาพที่ 3 เซลล์หน่วยความจำพื้นฐานประกอบด้วย CMOS อินเวอร์เตอร์ 2 ตัว รวมกับทรานซิสเตอร์ที่ใช้สำหรับการเขียน-อ่านเซลล์ข้อมูล ซึ่งเซลล์จะสามารถถูกเขียนได้อย่างเดียวในระหว่างทำการสร้างและอ่านอย่างเดียวในระหว่างทำการดึงกลับออกมา ในภาวะการทำงานปกติ เซลล์จะให้การควบคุมเป็นไปอย่างต่อเนื่องและตัวทรานซิสเตอร์จะไม่ทำงาน และไม่มีผลกระทบกับความเสถียรภาพของเซลล์แต่อย่างใด ซึ่งค่อนข้างแตกต่างไปจากอุปกรณ์อื่นโดยทั่วไปในด้านที่เซลล์มักจะมีการอ่านและเขียนเข้าอยู่เป็นประจำ

ขาออกของเซลล์หน่วยความจำ Q และ $\overline{Q}$ จะใช้ระดับของ กราวด์ และ Vcc ซึ่งให้เป็นแบบต่อเนื่องและควบคุมโดยตรง ความสามารถเพิ่มเติมเกี่ยวกับโหลดพร้อมกับที่ไม่มีการถอดรหัสตำแหน่งและความรวดเร็วของตัวขยายช่วยให้เกิดความเสถียรแก่เซลล์สูง และจากโครงแบบของเซลล์หน่วยความจำโครงแบบ มันไม่มีผลกระทบที่เนื่องมาจากการเปลี่ยนแปลงอย่างกะทันหันของแหล่งจ่ายไฟหรือพลังงานรังสี ในการทดสอบ พบว่าไม่มีข้อผิดพลาดใดเกิดขึ้นมาให้เห็นในขณะที่ทำงานภายใต้บริเวณที่มีพลังงานรังสีสูง ๆ

กรรมวิธีการโหลดข้อมูลโครงแบบมีหลายวิธีให้เลือกใช้ โดยปกติแล้วจะใช้ระบบพัฒนา XACT ช่วยในข้อมูลโปรแกรมเพื่อส่งตรงไปยังเซลล์หน่วยความจำ โครงแบบข้อมูลที่เป็นแบบอนุกรมและนับตามความยาว ช่วยให้เกิดความเข้ากันได้สำหรับการผสมใช้งานอุปกรณ์ LCA หลาย ๆ ตัวในแบบซิงโครนัส, อนุกรม หรือแบบลูกโซ่

ภาพที่ 3

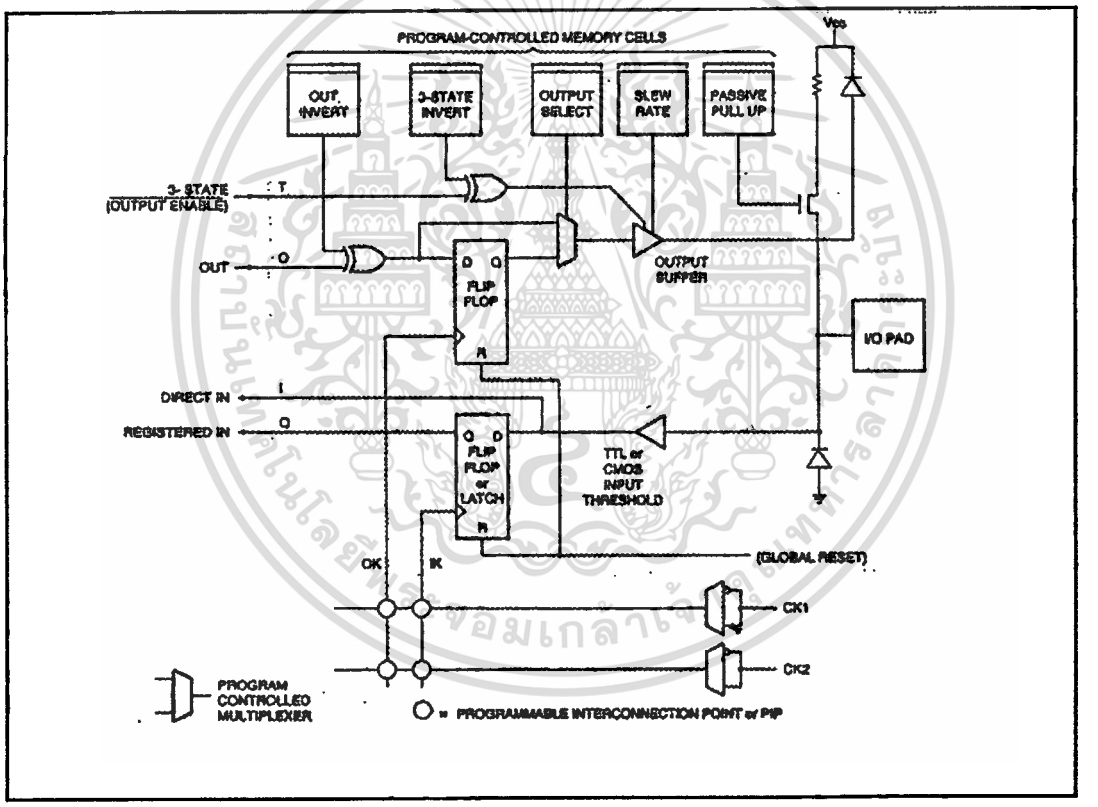



เซลล์หน่วยความจำโครงแบบแบบสถิต

- I/O Block

ในโครงสร้างของ IOB แสดงดังภาพที่ 4 ทำหน้าที่เชื่อมต่อระหว่างขาภายนอกของอุปกรณ์เข้ากับการใช้ตรรกที่อยู่ภายในตัวอุปกรณ์ ในแต่ละ IOB จะมีทั้งการรีจิสเตอร์และเส้นทางตรงอินพุต และให้ 3-สเตต เอาท์พุตบัฟเฟอร์ ที่อาจถูกขับโดยการรีสเตอร์หรือสัญญาณเอาท์พุตโดยตรง ในโครงแบบยอมให้มีการกลับ (inversion) ในแต่ละ IOB ควบคุมอัตราการสลูว์ (slew rate) และ อิมพีแดนซ์สูง และในแต่ละทางเข้าของวงจรจะมีตัว แคมป์ปิ้งไดโอด เพื่อป้องกันเกี่ยวกับไฟฟ้าสถิต และวงจรป้องกันการเกิด latch-up เนื่องจากกระแสที่เข้ามา

ภาพที่ 4




บล็อกอินพุต-เอาต์พุท (IOB)

ส่วนของอินพุต-บัฟเฟอร์ ในแต่ละ IOB จะให้ขอบเขตหรือจัดระดับของการตรวจจับเกี่ยวกับการแปลงหรือพาสัญญาณจากภายนอกที่ขาของอุปกรณ์ไปยังระดับตรรกภายใน ค่าระดับหรือขอบเขตที่ตัว อินพุต-บัฟเฟอร์ สามารถโปรแกรมได้ให้อยู่ในระดับ TTL หรือ CMOS สัญญาณที่เข้ามาอยู่ในบัฟเฟอร์ ทำหน้าที่ส่งหรือขับข้อมูลที่เข้ามาของส่วนที่เก็บข้อมูล ซึ่งอาจจะอยู่ในรูปของฟลิปฟลอปหรือแลตช์ ในส่วนของขั้วสัญญาณนาฬิกาที่ใช้สามารถทำการโปรแกรมได้ไม่ว่าจะ

ต้องการใช้ขอบด้านใดของสัญญาณนาฬิกาก็ตามในการเป็นสัญญาณกระตุ้นให้ส่วนอื่นเริ่มทำงาน เมื่อมีการใช้เส้นสัญญาณนาฬิกาไปกระตุ้นให้กับฟลิปฟลอปหรือแลตช์ ต้องมีการชดเชยสำหรับความแตกต่างของสัญญาณนาฬิกาด้วยเสมอ ชิ้นส่วนเก็บข้อมูล I/O จะถูกรีเซ็ตในขณะที่ทำโครงแบบหรือโดยสัญญาณสั่งให้รีเซ็ตสัญญาณอินพุต ทั้งคู่ (ขา I และขา Q ของ IOB) สามารถเชื่อมต่อระหว่างกันได้

เพื่อความเชื่อถือได้ในขณะทำงาน อินพุตที่เข้ามาจะต้องมีค่าเวลาในการเปลี่ยนแปลงระดับต่ำกว่า 100 nSec. และต้องไม่มีการเหลือตกค้างอยู่ ซึ่งอาจทำให้เกิดการผิดพลาดเป็นสัญญาณรบกวนในระบบได้ ในแต่ละ IOB ที่ใช้จะมีตัวต้านทานอิมพีแดนซ์สูง ซึ่งเลือกโปรแกรมได้เพื่อใช้สำหรับกรณีที่ไม่มีการขับไปยังขาของอุปกรณ์

เวลาประวิงในลูปสำหรับ IOB และฟลิปฟลอปจะมีค่าประมาณ 3 nSec. ซึ่งถือว่าสั้นมาก ทำให้เกิดประสิทธิภาพสูงในการทำงานภายใต้เงื่อนไขของข้อมูลและสัญญาณนาฬิกาที่เป็นแบบอะซิงโครนัส และจากค่าเวลาประวิงที่ต่ำนี้ทำให้อุปกรณ์ LCA สามารถนำสัญญาณซิงโครนัสจากภายนอกมาใช้งานได้

เซลล์หน่วยความจำที่ถูกโปรแกรมควบคุมในภาพที่ 5 ใช้เลือกควบคุมหน้าที่ต่าง ๆ ดังนี้

- การผกผันตรรกของ อินพุต จะถูกควบคุมโดยโปรแกรมโครงแบบเพียง 1 บิตต่อ IOB

- ตรรกควบคุม 3-สเตต ของแต่ละ IOB เอาท์พุตบัฟเฟอร์ จะถูกกำหนดโดยสถานะของบิตโปรแกรมโครงแบบที่จะให้บัฟเฟอร์ปิดหรือเปิด หรือเลือกเป็น 3-สเตต (ที่ขา T ของ IOB) เมื่อสัญญาณควบคุม เอาท์พุต ของ IOB เป็นระดับสูงหรือตรรก '1' ตัวบัฟเฟอร์จะอยู่ในสภาวะใช้งานไม่ได้ และที่ขาภายนอกจะเป็น อิมพีแดนซ์สูง แต่ถ้าสัญญาณควบคุมนี้เป็นระดับต่ำหรือตรรก '0' ก็หมายถึงบัฟเฟอร์ที่ใช้งานได้และมีสัญญาณออกไปที่ขาอุปกรณ์

- Direct หรือ registered เอาท์พุต สามารถเลือกได้ในแต่ละ IOB ตัวรีจิสเตอร์จะใช้ขอบที่เป็นบวกของสัญญาณนาฬิกา ซึ่งสัญญาณนาฬิกานี้อาจมาจากเส้นใดเส้นหนึ่งในจำนวน 2 เส้นที่มี

- การเพิ่มความเร็วของการเปลี่ยนระดับเอาท์พุตสามารถทำการเลือกความเร็วนี้ให้เหมาะสมกับค่าเวลาที่จำเป็นต้องใช้ ถ้าเพิ่มความเร็วนี้ขึ้นมากก็จะเกิดสัญญาณรบกวนมากตาม

- ตัวต้านทานที่เป็น อิมพีแดนซ์สูง ภายในใช้ป้องกันกรณีที่ไม่มีการใช้งานหรือไม่ได้เชื่อมต่อ อินพุต ไม่ให้เกิดการรบกวนเข้าไปยังส่วนอื่นของวงจร

ดังนั้นการเลือกในส่วนของ I/O สามารถสรุปได้ดังนี้

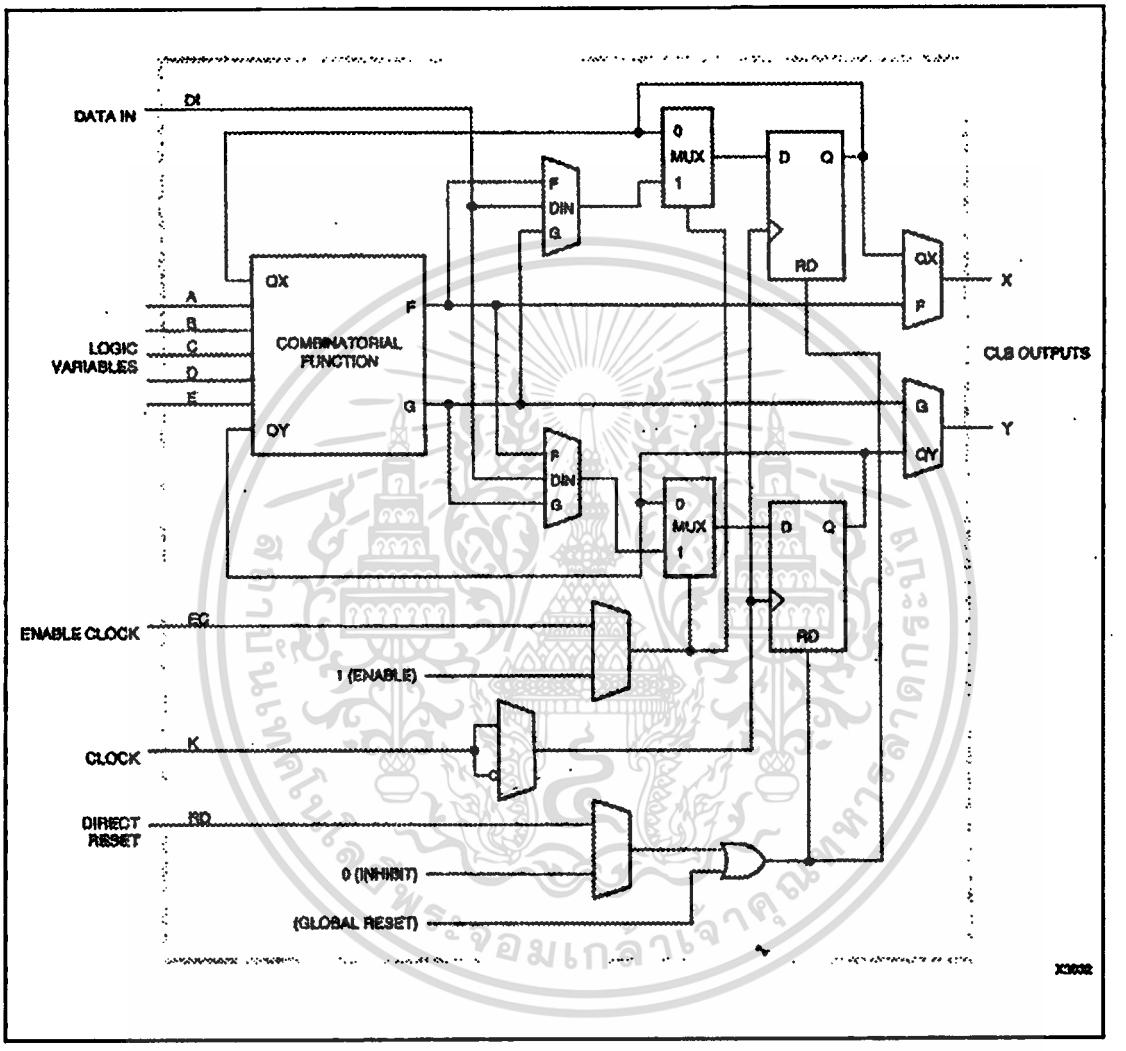
- ในด้าน อินพุต
    - Direct
    - Flip-flop หรือ latch
    - CMOS หรือ TTL threshold (chip อินพุตs)
    - Pull-up resistor หรือ open circuit
- ในด้าน เอาท์พุต
    - Direct หรือ registered
    - inverted หรือ not
    - 3-state หรือ on หรือ off
    - Full speed หรือ slow limited
    - 3-state หรือ เอาท์พุต enable (inverse)

- Configurable Logic Block (CLB)

แถวลำดับของ CLBs ให้การทำงานตามที่ผู้ใช้กำหนดหรือสร้างขึ้นมา โดยบล็อกตรรก เหล่านี้ถูกจัดเรียงในลักษณะของเมตริกซ์อยู่ภายในกรอบของ IOBs ตัวอย่างเช่นตัว XC3020 จะมีทั้งหมด 64 บล็อก จัดเรียงเป็น 8 แถว และ 8 ช่อง และใช้ระบบพัฒนา XACT ในการคอมไพล์ข้อมูลโครงแบบที่จะถูกนำไปสู่หน่วยความจำโครงแบบภายใน เพื่อใช้กำหนดหน้าที่การทำงาน และการเชื่อมต่อระหว่างกันของแต่ละบล็อก ผู้ใช้หรือผู้ออกแบบสามารถกำหนดเกี่ยวกับ CLBs และการเชื่อมต่อในโครงข่ายในแบบที่แปลงจากรูปวงจรแบบอัตโนมัติหรือเลือกใช้จากคลังข้อมูลที่ติดตั้งอยู่ หรือโดยผู้ใช้กำหนดเองก็ได้

แต่ละ CLB จะมีทั้งส่วนที่รวมตรรก, ฟลิปฟลอป 2 ตัว และส่วนควบคุมภายในดังแสดงในภาพที่ 5 คือ มี 5 ตรรกที่เป็นอินพุต (A, B, C, D และ E), สัญญาณนาฬิการ่วมที่เข้ามา (K), อะซิงโครนัสรีเซ็ตที่เข้ามาโดยตรง (RD) และ สัญญาณนาฬิกาให้ทำงาน (EC) ซึ่งทั้งหมดจะถูกขับจาก resources การเชื่อมต่อที่อยู่ใกล้ ๆ กับบล็อก แต่ละ CLB จะมีเอาท์พุต อยู่ 2 เส้นคือ X กับ Y ซึ่งอาจใช้ส่งไปยังโครงข่ายที่มีการเชื่อมต่อถึงกัน

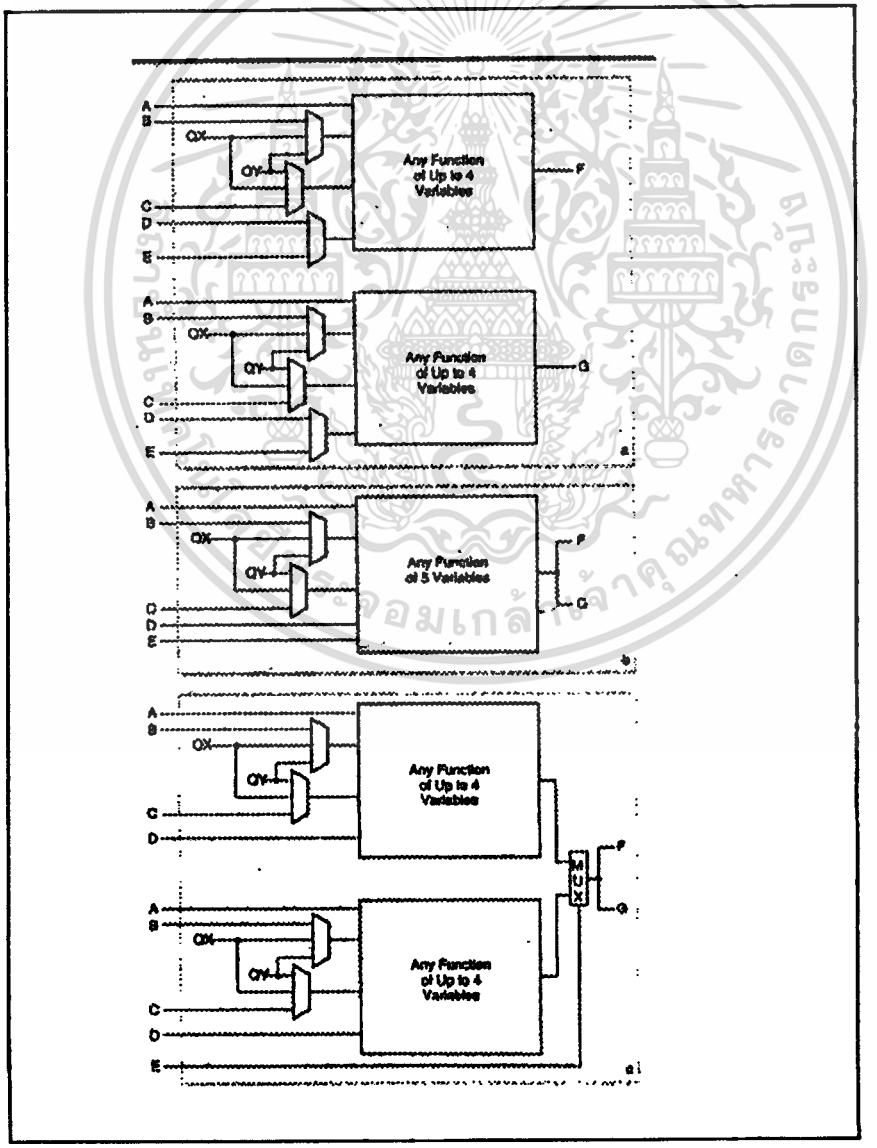
ภาพที่ 5




โครงสร้างของบล็อกตรรก

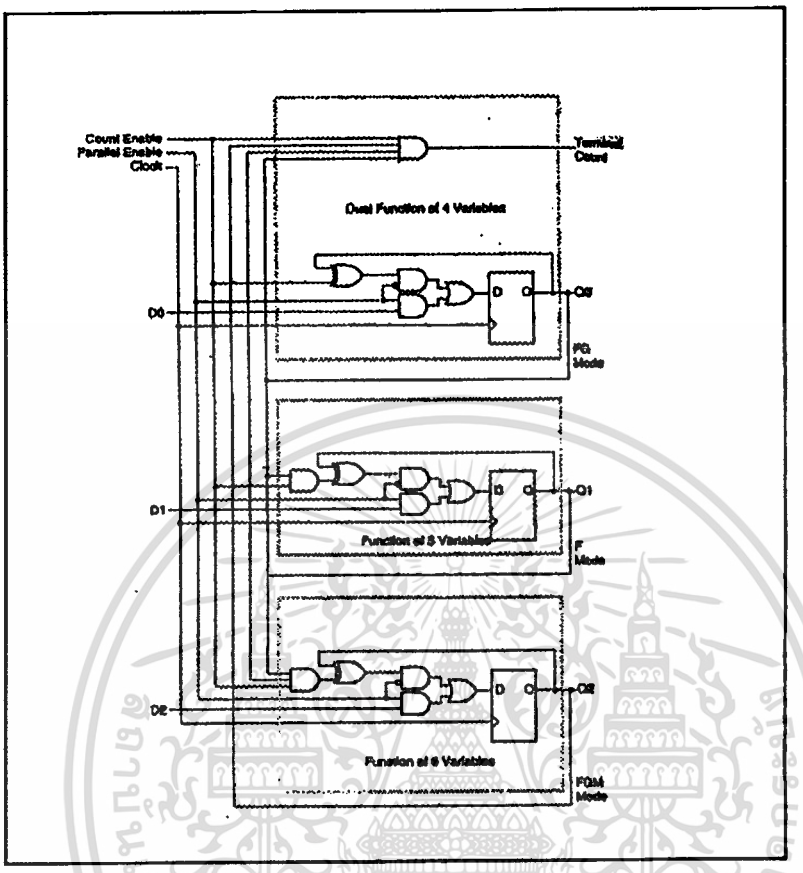
ส่วนของตรรกรวมใน CLB จะใช้ตารางเปิดดู แบบ 32 ต่อ 1 ในการแทนหน้าที่การทำงานตามสมการแบบบูล การเลือกที่เปลี่ยนแปลงได้จากทั้ง 5 อินพุต และฟลิปฟลอปทั้ง 2 ตัวภายในจะถูกใช้เป็นตำแหน่งของตาราง ค่าเวลาประวิงโดยรวมที่เกิดขึ้นในตลอดโครงข่ายจะขึ้นอยู่กับหน้าที่การทำงานที่สร้างขึ้นมาแต่ละหน้าที่ และแยกอิสระจากกันสำหรับอินพุตที่เปลี่ยนไปแต่ละตัว ทำให้สามารถสร้างหน้าที่การทำงานทางตรรกที่เป็นอิสระต่อกัน 2 อย่างแบบมี 4 ตัวแปรดังภาพที่ 6(a) หรือหน้าที่เดียวแต่มี 6 ตัวแปรดังภาพที่ 6(b) หรือหน้าที่บางอย่างที่มี 7 ตัวแปร ดังภาพที่ 6(c) ส่วนในภาพที่ 7 แสดงถึงตัวนับแบบ Modulo-8 ซึ่งใช้ตรรกรวมเพียงบล็อกเดียวในแต่ละส่วน

ภาพที่ 6




ลักษณะการจัดรวมตรรกให้มีหน้าที่การทำงานสำหรับจำนวนตัวแปรที่ต่างกัน

ภาพที่ 7




ตัวนับแทน modulo-8 ที่ใช้บล็อกตรรกรวมเดียวในแต่ละส่วน

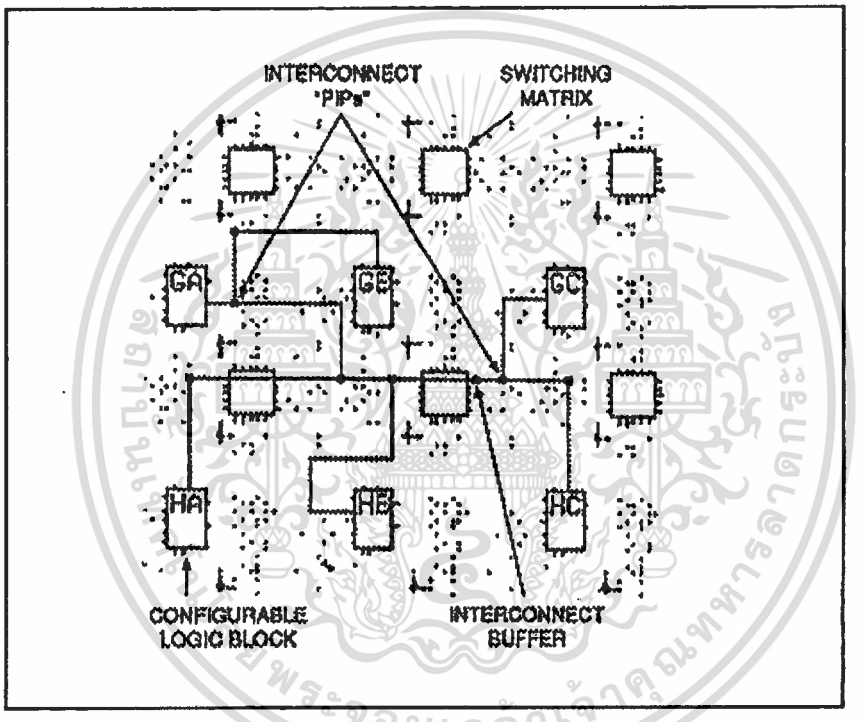
- การเชื่อมต่อระหว่างกันแบบโปรแกรมได้ (Programmable Interconnect)

ทรัพยากรสำหรับใช้เชื่อมต่อระหว่างกันที่อยู่ใน LCA เป็นเส้นทางการเชื่อมต่อถึงกันของ IOBs และ CLBs เข้ากันเป็นโครงข่ายผ่านทางเส้นที่เป็นโลหะ ออกแบบกำหนดผ่านทางตัวทรานซิสเตอร์ ซึ่งแต่ละตัวถูกควบคุมโดยบิตของโปรแกรมโครงแบบ จากจุดเชื่อมต่อระหว่างกันที่โปรแกรมเลือกได้ (PIPs) และตัวสวิตช์เมตริกซ์ จะใช้ในกรณีที่จำเป็นสำหรับการเชื่อมต่อระหว่างเส้นโลหะที่เลือกใช้กับขาของบล็อก ในภาพที่ 8 แสดงตัวอย่างของเส้นทางเชื่อมต่อโครงข่าย สำหรับระบบพัฒนา XACT จะถูกใช้เมื่อต้องการให้เส้นทางการเชื่อมต่อและตำแหน่งการวางเป็นไปแบบอัตโนมัติ อินพุตของ IOBs และ CLBs เป็นตัวมัลติเพลกเซอร์ ซึ่งสามารถถูกโปรแกรมให้เลือกโครงข่ายอินพุตจากส่วนการเชื่อมต่อที่อยู่ใกล้กัน และเนื่องจากการเชื่อมต่อไปยังบล็อกอินพุต สามารถเลือกได้ทั้งสองทิศทางเหมือนกับทางบล็อกเอาต์พุต จะใช้เพียงเฉพาะที่เป็นการต่อบล็อกอินพุตเท่านั้น และไม่เกี่ยวกับการทำเส้นทาง ดังภาพที่ 9 เป็นตัวอย่างแสดงการทำเส้นทาง

ของบล็อกตรรกอินพุต, ส่วนควบคุมและบล็อกเอาต์พุต และส่วนของโลหะ 3 ชนิด ที่จัดเตรียมหรือมีไว้ให้กับความต้องการของการเชื่อมต่อระหว่างกันในโครงข่ายแบบต่าง ๆ คือ

- การเชื่อมต่อระหว่างกันตามวัตถุประสงค์ทั่วไป
- การต่อตรง
- เส้นยาว (longlines)

ภาพที่ 8

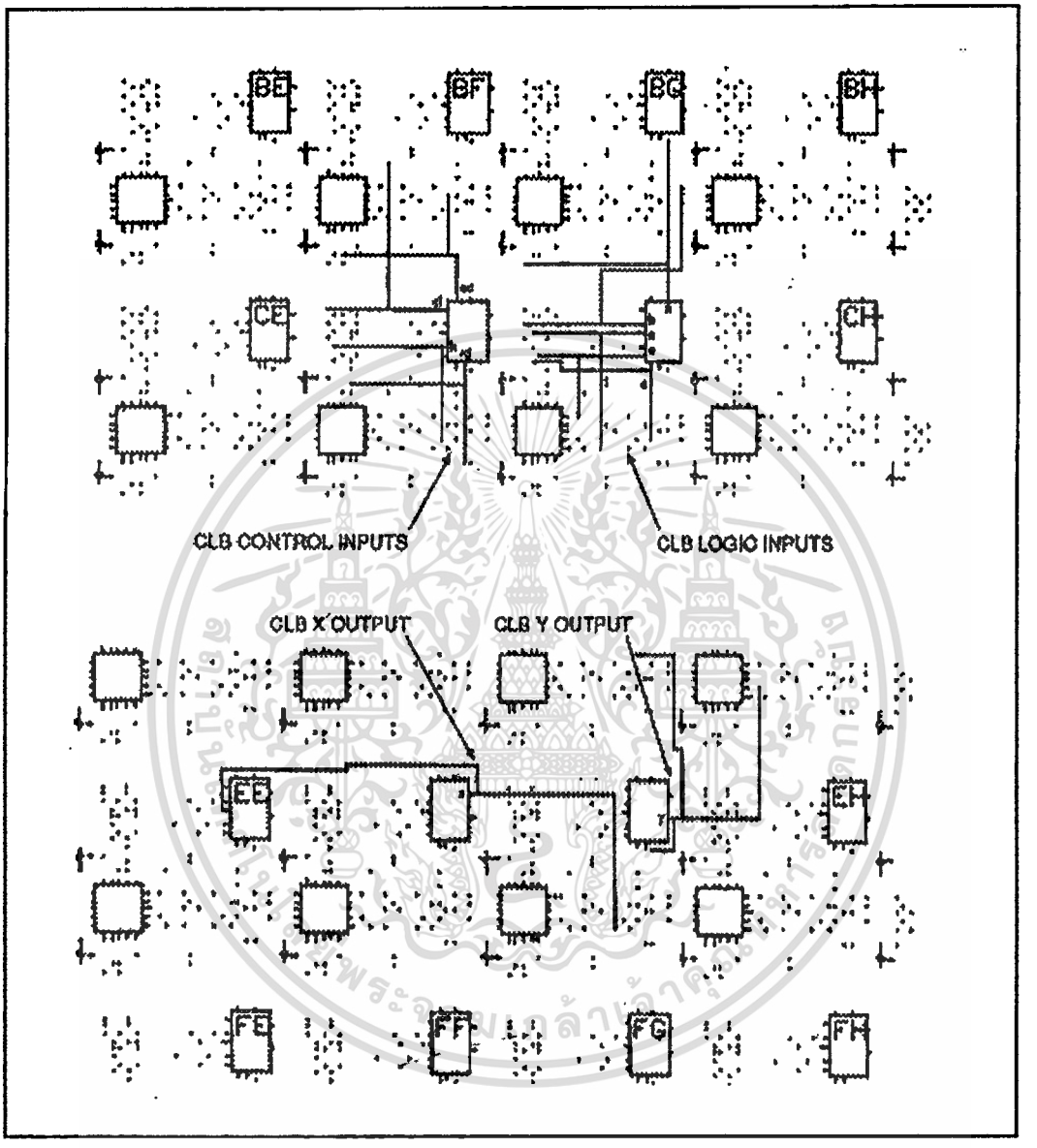



ภาพที่ได้จากระบบพัฒนา XACT ในส่วนการใช้ routing resources

- การเชื่อมต่อระหว่างกันตามวัตถุประสงค์ทั่วไป

ดังแสดงในภาพที่ 10 ประกอบด้วยตะแกรงโลหะ 5 แถวทางแนวนอน และ 5 แถวทางแนวตั้ง วางไว้ในตำแหน่งระหว่างช่องของตรรกและ IOBs แต่ละส่วนจะมีความกว้างและสูงตามบล็อกตรรก ตัวสวิตช์เมตริกซ์จะเป็นตัวเชื่อมตรงปลายของส่วนต่าง ๆ และยอมให้เลือกการเชื่อมต่อระหว่างส่วนตะแกรงโลหะทางแนวนอนและแนวตั้งที่อยู่ติดกัน ถ้าสวิตช์ตัวไหนไม่ถูกโปรแกรมเลือกก็จะไม่มีการเชื่อมต่อ ซึ่งการเลือกเส้นทางเชื่อมต่อนี้อาจกระทำได้โดยใช้โปรแกรม

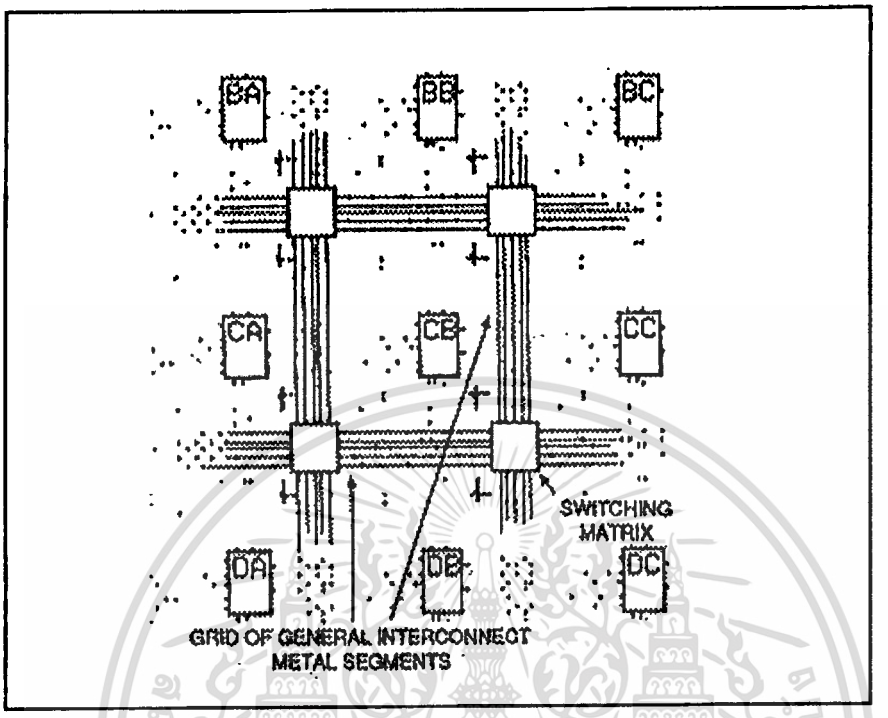
ภาพที่ 9




ภาพที่ได้จากระบบพัฒนา XACT ในส่วนของตำแหน่งและการเชื่อมต่อระหว่างกัน

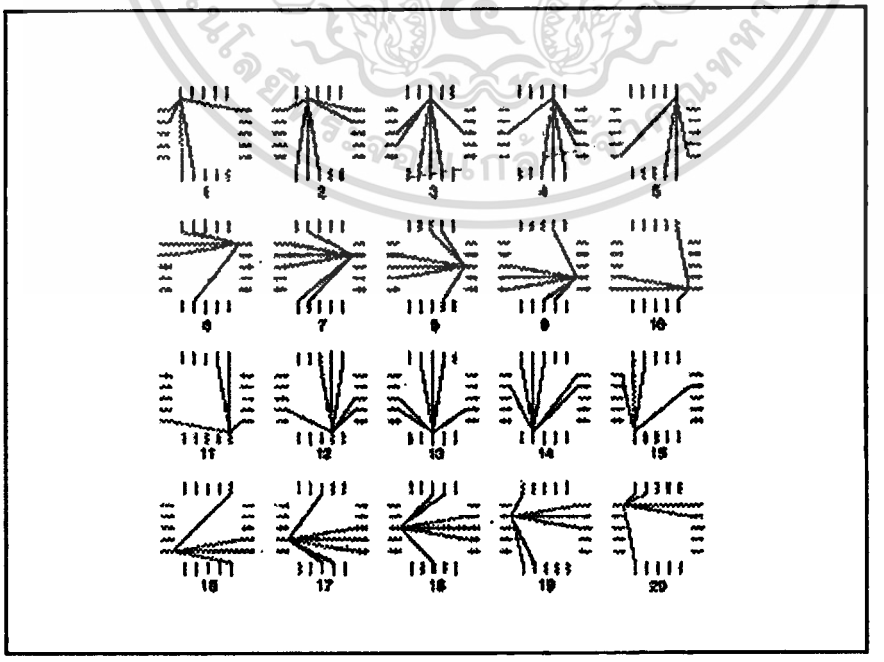
ซอฟต์แวร์เข้าช่วยในการเลือกขาของเมตริกซ์ที่จะถูกต่อหรือไม่ต่อ ในภาพที่ 11 แสดงถึงตัวอย่างสวิตช์เมตริกซ์สำหรับแต่ละขาที่ถูกต้องและอาจใช้คำสั่ง Show-Matrix ที่มีอยู่ในระบบ XACT เพื่อแสดงภาพให้เห็นชัดเจนยิ่งขึ้นได้

ภาพที่ 10




การเชื่อมต่อถึงกันภายในอุปกรณ์ FPGA ตามแบบทั่วไป

ภาพที่ 11

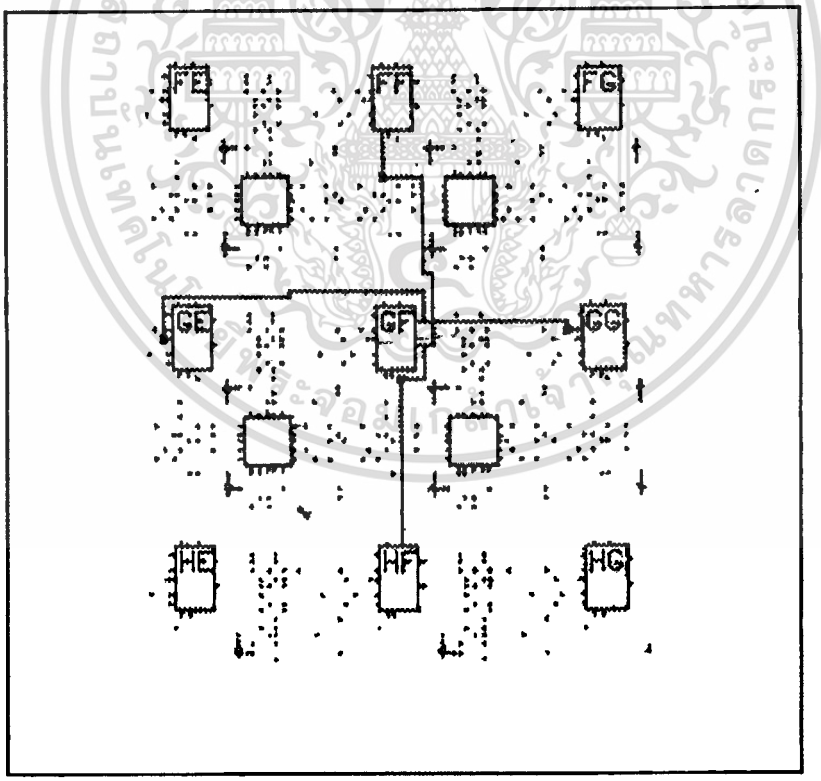


ทางเลือกของการเชื่อมต่อถึงกันของสวิตช์เมตริกซ์สำหรับแต่ละขา

- การต่อตรงระหว่างกัน

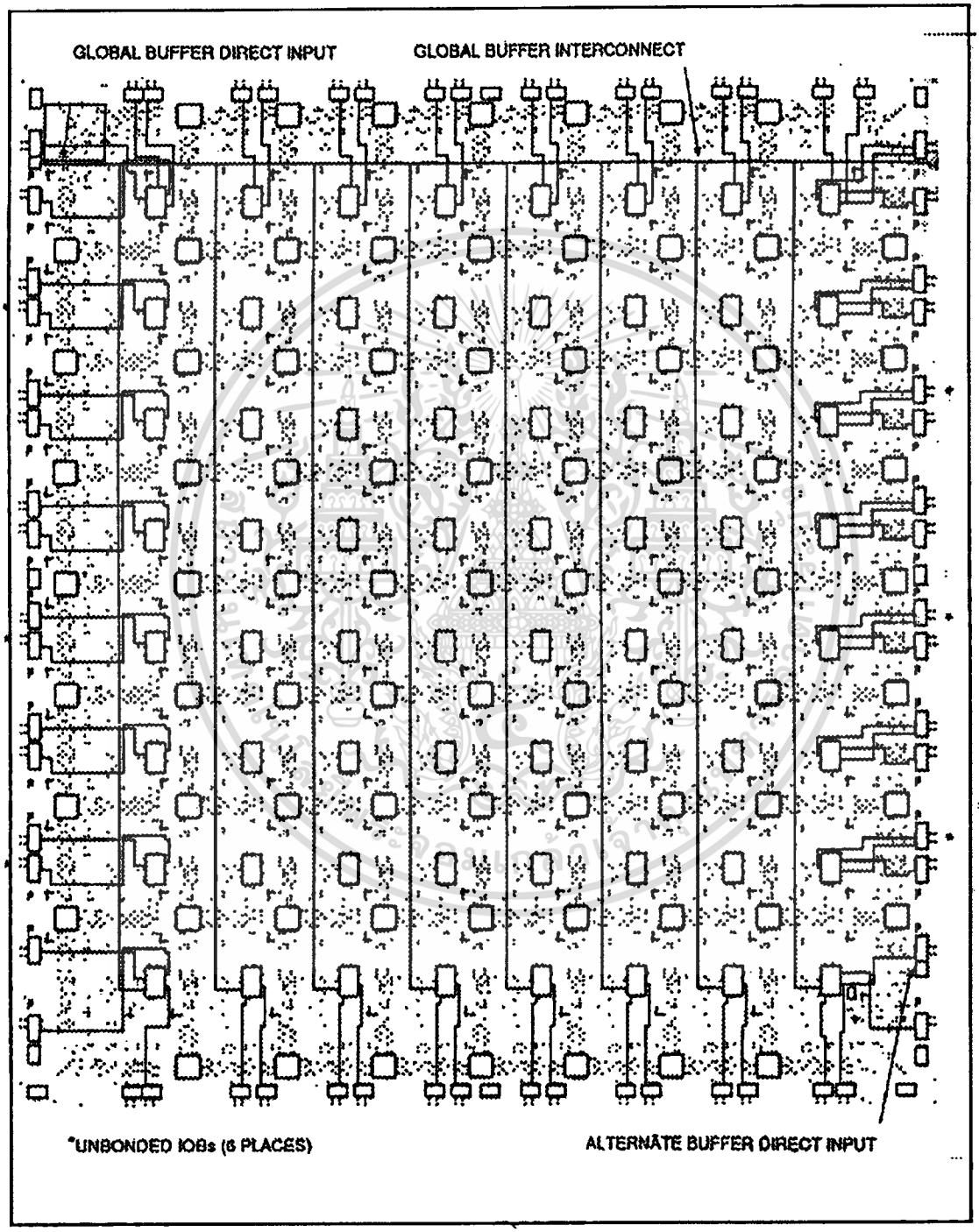
ดังแสดงในภาพที่ 12 ซึ่งเป็นการแทนโครงข่ายที่มีประสิทธิภาพสูงในการเชื่อมต่อระหว่างบล็อกของ IOBs กับ CLBs ที่อยู่ใกล้กัน สัญญาณจะถูกส่งผ่านจากบล็อกถึงบล็อกได้โดยตรง ทำให้ไม่ต้องใช้เส้นสำหรับการเชื่อมต่อทั่วไปที่มีอยู่ สำหรับในแต่ละ CLB จุดเอาต์พุต X อาจต่อตรงถึงอินพุต B ของ CLB ทางด้านขวา และต่อไปยังจุดอินพุต C ของ CLB ทางด้านซ้าย ส่วนจุดเอาต์พุต Y สามารถต่อตรงไปยังจุดอินพุต D ของบล็อกที่อยู่ข้างบน และจุดอินพุต A ของบล็อกที่อยู่ด้านล่าง การต่อตรงในลักษณะนี้จะถูกนำมาใช้เมื่อมีความต้องการความเร็วที่สูงสุดในส่วนของตรรก ตำแหน่งสถานที่บล็อกตรรกที่อยู่ใกล้กับ IOBs การต่อตรงจะเลือกได้ทั้ง 4 มุม มุมทางด้านขวาจะให้การต่อตรงจากเอาต์พุต ของ CLBs ไปยัง IOBs ที่อยู่ใกล้กัน ดังตัวอย่างที่แสดงในภาพที่ 13

ภาพที่ 12




แสดงการต่อถึงกันโดยตรงระหว่างเอาต์พุท X และ Y ของ CLB

ภาพที่ 13



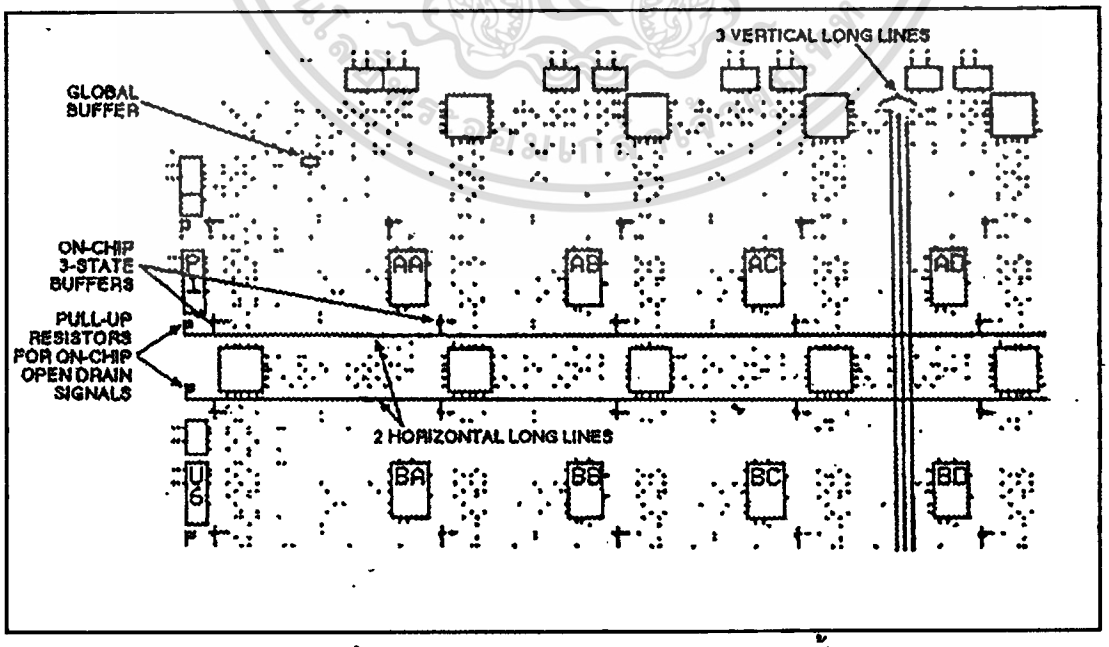

ตัวอย่างการต่อถึงกันโดยตรงระหว่าง CLB ที่อยู่ใกล้กันภายในอุปกรณ์ XC3020

- เส้นยาว

ใช้สำหรับเป็นถนนให้กับสวิตช์เมตริกซ์ และสำหรับการเดินทางของสัญญาณที่ต้องใช้ระยะมาก หรือต้องมีจำนวนการผ่านน้อยสุดในระหว่างจุดปลายทาง เส้นยาวแสดงให้เห็นดังภาพที่ 14 มีเส้นวิ่งทั้งทางแนวนอนแนวตั้งตามขนาดความสูงและความกว้างของบริเวณการเชื่อมต่อ ในแต่ละช่องจะมีเส้นแนวตั้ง 3 เส้น และระหว่างแถวจะมีเส้นทางแนวนอนอยู่ 2 เส้นที่เป็นเส้นยาว และมีอีก 2 เส้นอยู่ภายนอกของกลุ่มสวิตช์เมตริกซ์

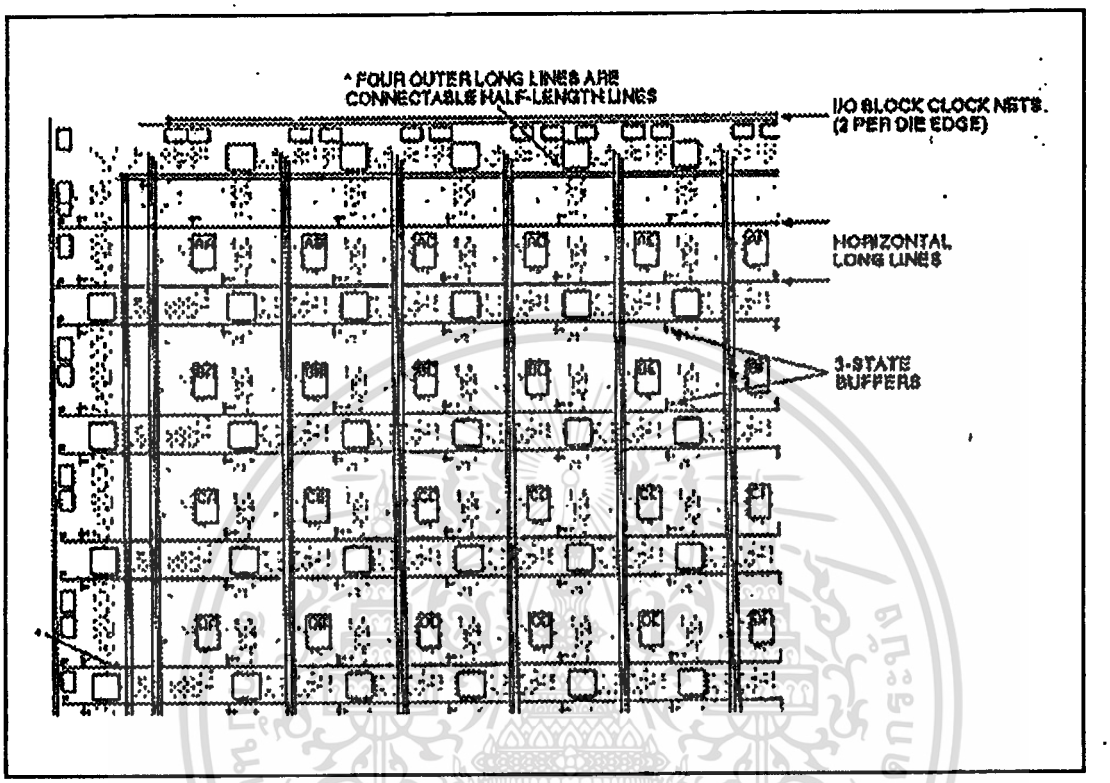
เส้นยาว สามารถใช้งานโดยบล็อกตรรก หรือเอาต์พุต ของ IOB แบบช่องต่อช่อง การใช้งานด้วยเส้นยาวแบบนี้ทำให้ลดจำนวนเส้นของสัญญาณนาฬิกาหรือสัญญาณควบคุมภายในแต่ละช่องของบล็อกตรรกลง ลักษณะการเชื่อมต่อดังแสดงในภาพที่ 15 โดยแยกบัฟเฟอร์ออกที่แต่ละอินพุตไปยังเส้นยาว ตัวบัฟเฟอร์ทางด้านมุมซ้ายของชิพ LCA จะส่งไปตลอดทั่วทั้งหมดทุกอินพุต K ของบล็อกตรรก และด้วยการใช้บัฟเฟอร์ทั่วไปนี้สำหรับเป็นสัญญาณนาฬิกาจะทำให้สามารถมีค่าแฟนเอาต์ (fan-out) สูง และใช้สัญญาณนาฬิกาเข้าจังหวะพร้อมกันทั้งหมดได้ทุก ๆ อันทั้งหมดใน IOBs และ CLBs ส่วนบัฟเฟอร์ที่อยู่มุมด้านขวาจะขับเส้นยาวทางแนวนอน ซึ่งสามารถส่งการเชื่อมต่อไปได้ยังเส้นทางแนวตั้งได้ในแต่ละช่องของการเชื่อมต่อระหว่างกัน

ภาพที่ 14




แสดงเส้นยาว (longlines) ทางแนวนอนและแนวตั้ง

ภาพที่ 15

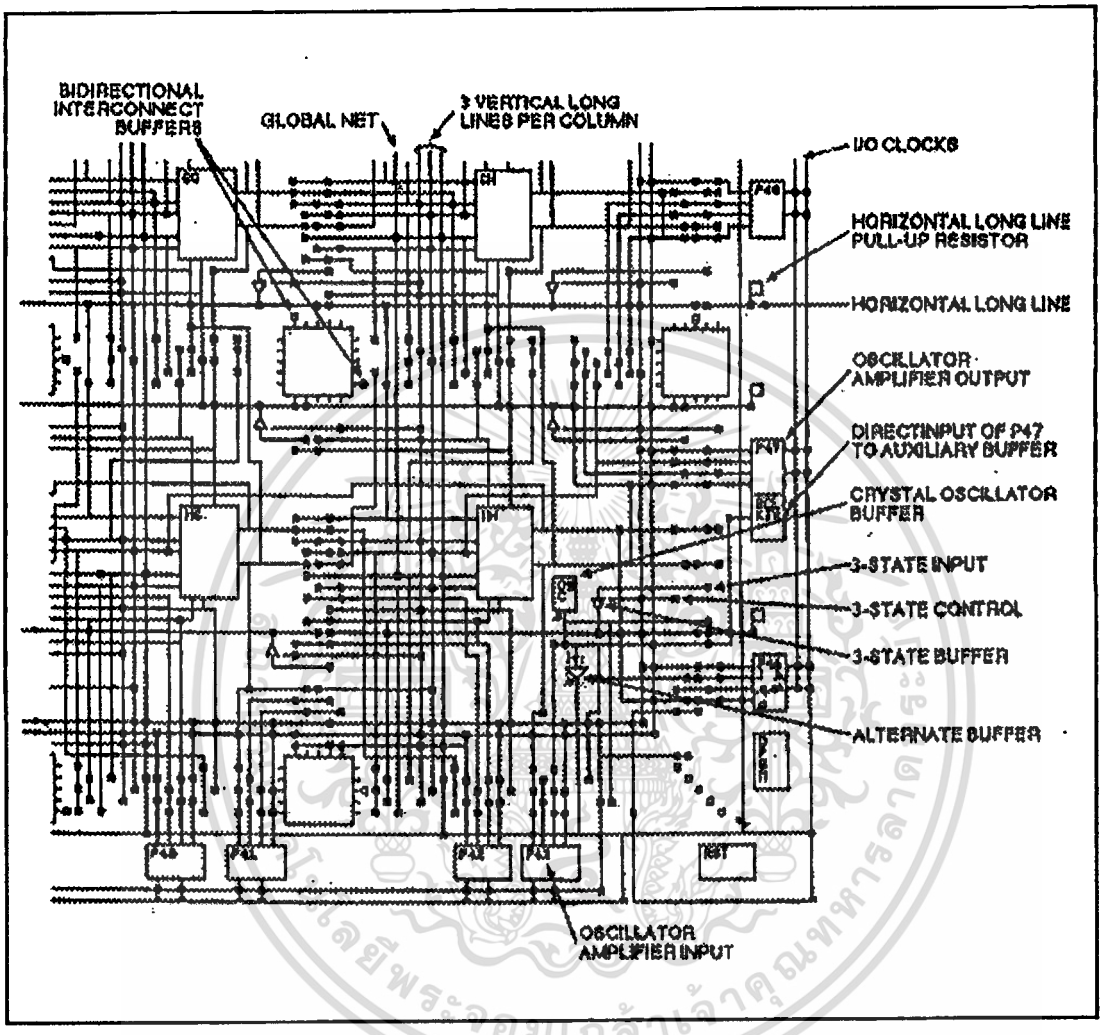



แสดงตัวอย่างการเลือกจุดต่อถึงกันของเส้นยาว

- บัสภายใน

คู่ของบัฟเฟอร์แบบ 3-สเตต ที่วางไว้ในตำแหน่งใกล้กับ แต่ละ CLB จะให้ตรรกที่ใช้ขับเส้นยาวทางแนวนอน การทำงานตรรกของบัฟเฟอร์แบบ 3-สภาวะนี้ ควบคุมให้ทำหน้าที่เป็นการผสมสัญญาณทั่วไป ซึ่งทุกๆ อินพุต สามารถเลือกได้โดยใช้ระดับสัญญาณตรรกต่ำบนเส้นควบคุมของ 3-สเตต สำหรับการส่งขับไปยังเส้นยาวทางแนวนอน เมื่อผู้ใช้ตัดการหลีกเลี่ยงการขัดแย้งซึ่งสามารถเป็นผลมาจากการที่มีตัวขับหลายตัว และมีระดับของตรรกที่ไม่เหมือนกัน ต้องควบคุมให้อินพุต ที่จะออกไปขับตัวบัฟเฟอร์เป็นระดับสัญญาณเดียวกันหมด ถ้าเป็นระดับสูงหมดก็จะอยู่ในสภาวะอิมพีแดนซ์สูง คือไม่ได้ใช้งาน แต่ถ้าเป็นระดับตรรกต่ำหมด ก็คือใช้บัฟเฟอร์ขับไปยังเส้นทางที่ต่ออยู่ ส่วนตัวต้านทานแบบพูลอัพ (pull-up) ที่อยู่ตรงปลายของเส้นยาวจะให้เอาต์พุตเป็นตรรกสูง เมื่อไม่ได้ใช้ตัวบัฟเฟอร์ ภาพที่ 17 แสดงให้เห็นถึงการเชื่อมต่อระหว่างกันของตัวบัฟเฟอร์แบบ 3-สเตต, เส้นยาวและตัวต้านทานแบบพูลอัพ ที่เป็นไปได้โดยนำมาเฉพาะส่วนที่อยู่มุมขวาด้านล่างของตัว XC3020

ภาพที่ 16



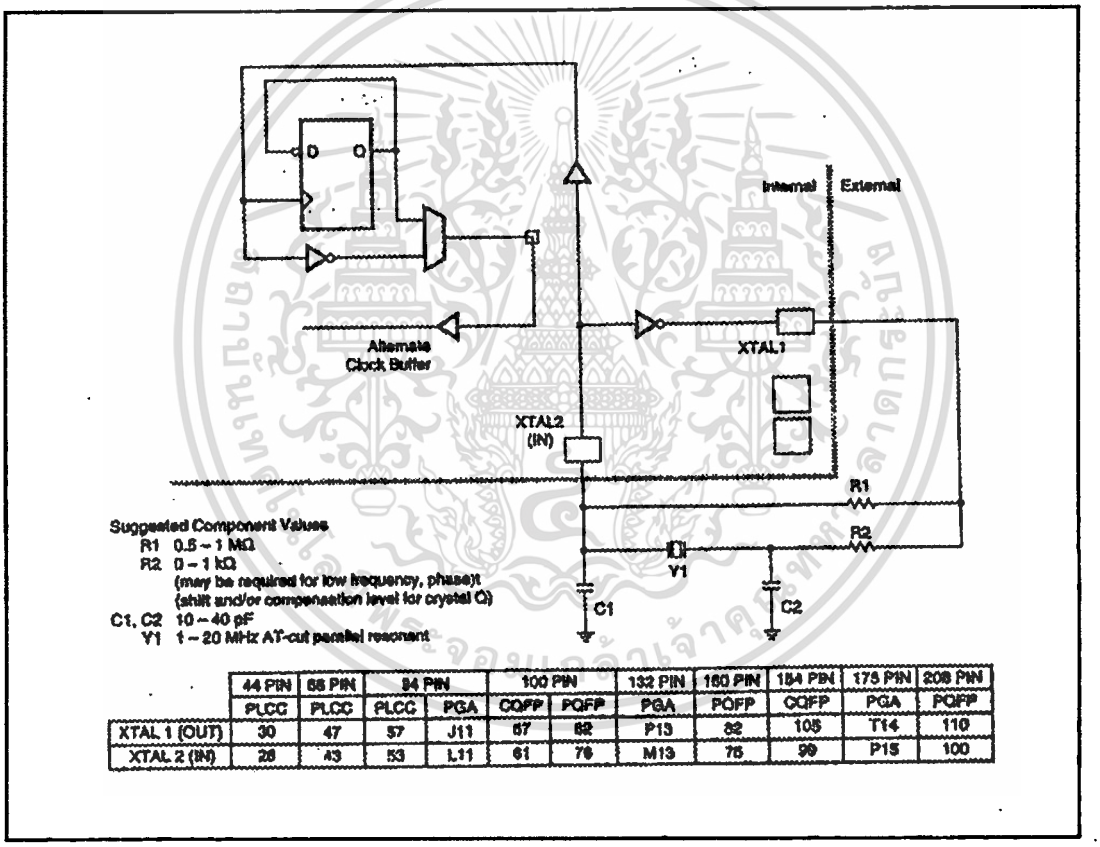

ตัวอย่างการเชื่อมต่อระหว่างกันภายใน (เฉพาะมุมขวาล่าง) ของอุปกรณ์ XC3020 ที่ได้จากระบบพัฒนา XACT

- คริสตอลออสซิลเลเตอร์ (Crystal Oscillator)

ภาพที่ 17 แสดงถึงตัวขยายแบบผกผันความเร็วสูงที่มีอยู่ภายในอุปกรณ์ ซึ่งอาจใช้เป็นตัวคริสตอลออสซิลเลเตอร์บนชิพได้ ซึ่งใช้บัฟเฟอร์ช่วยในมุมขวาด้านล่าง เมื่อมีการสร้างตัวออสซิลเลเตอร์ขึ้นมาโดยโปรแกรม Make Bits และการเชื่อมต่อใช้เป็นแหล่งจ่ายสัญญาณ จะมีการใช้ IOBs พิเศษ 2 ตัว สำหรับเชื่อมต่อตัวขยายออสซิลเลเตอร์ กับส่วนประกอบที่เป็นคริสตอลออสซิล-เลเตอร์ภายนอก ดังในภาพที่ 18 ซึ่งต้องแบ่งทั้ง 2 ส่วนให้สมมาตรกัน วงจรออสซิลเลเตอร์จะเริ่มต้นทำงานก่อนขบวนการอื่น เพื่อความเสถียรภาพของระบบการเชื่อมต่อจริงภายในจะถูกประวิง

เอาไว้จนกว่าขบวนการสร้างจะเสร็จสมบูรณ์ ในภาพที่ 17 ตัวต้านทานป้อนกลับ R1 ที่อยู่ระหว่างอินพุตและเอาต์พุต จะเป็นไบอัสให้กับตัวขยาย; การผกผันของตัวขยายร่วมกับวงจร R-C จะให้การเลื่อนเฟสไป 360 องศา ส่วนตัวต้านทานที่อยู่อนุกรมอยู่ R2 อาจนำมาต่อใช้เมื่อต้องการควบคุมการเลื่อนเฟส หรือควบคุมขนาดของสัญญาณที่ออกมาจากตัวขยาย ระดับแรงดันป้อนกลับที่สูงไปอาจทำให้ถูกต้องได้ โดยใช้อัตราส่วนของ C2 ต่อ C1 ตัวขยายถูกออกแบบให้ใช้งานตั้งแต่ความถี่ 1 MHz ไปจนถึงประมาณครึ่งหนึ่งของความถี่ท็อกเกิล (toggle) ของตัว CLBs

ภาพที่ 17




| | 44 PIN | 68 PIN | 84 PIN | | 100 PIN | | 132 PIN | 160 PIN | 164 PIN | 175 PIN | 208 PIN |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | PLCC | PLCC | PLCC | PGA | CQFP | PQFP | PGA | PQFP | CQFP | PGA | PQFP |
| XTAL 1 (OUT) | 30 | 47 | 57 | J11 | 67 | 82 | P13 | 82 | 105 | T14 | 110 |
| XTAL 2 (IN) | 28 | 43 | 53 | L11 | 61 | 76 | M13 | 76 | 99 | P15 | 100 |

แสดงลักษณะการใช้คริสตอลออสซิลเลเตอร์

<u>การโปรแกรม</u>

เมื่อมีแรงไฟ Vcc มาถึงส่วนของอุปกรณ์ LCA จะเริ่มต้นทำงาน (ปกติใช้ 2.5 ถึง 3 V) ค่าเวลาประวิงจะมีขึ้นจนกว่าแรงไฟของแหล่งจ่ายจะคงที่อยู่ในสภาพที่เสถียร และมีช่วงเวลา time-out ของการเริ่มต้นปกติประมาณ 11 ถึง 33 mSec. ในตารางที่ 1 แสดงให้เห็นรูปแบบโหมดการ

ทำงาน 5 อย่างให้เลือกใช้โดยกำหนดที่ระดับอินพุต ของขาโหมดทั้ง 3 คือ M0, M1 และ M2 โดยแบ่งโหมดการทำงานออกเป็น 3 โหมดใหญ่ ๆ คือ master, peripheral และแบบ slave

ตารางที่ 1

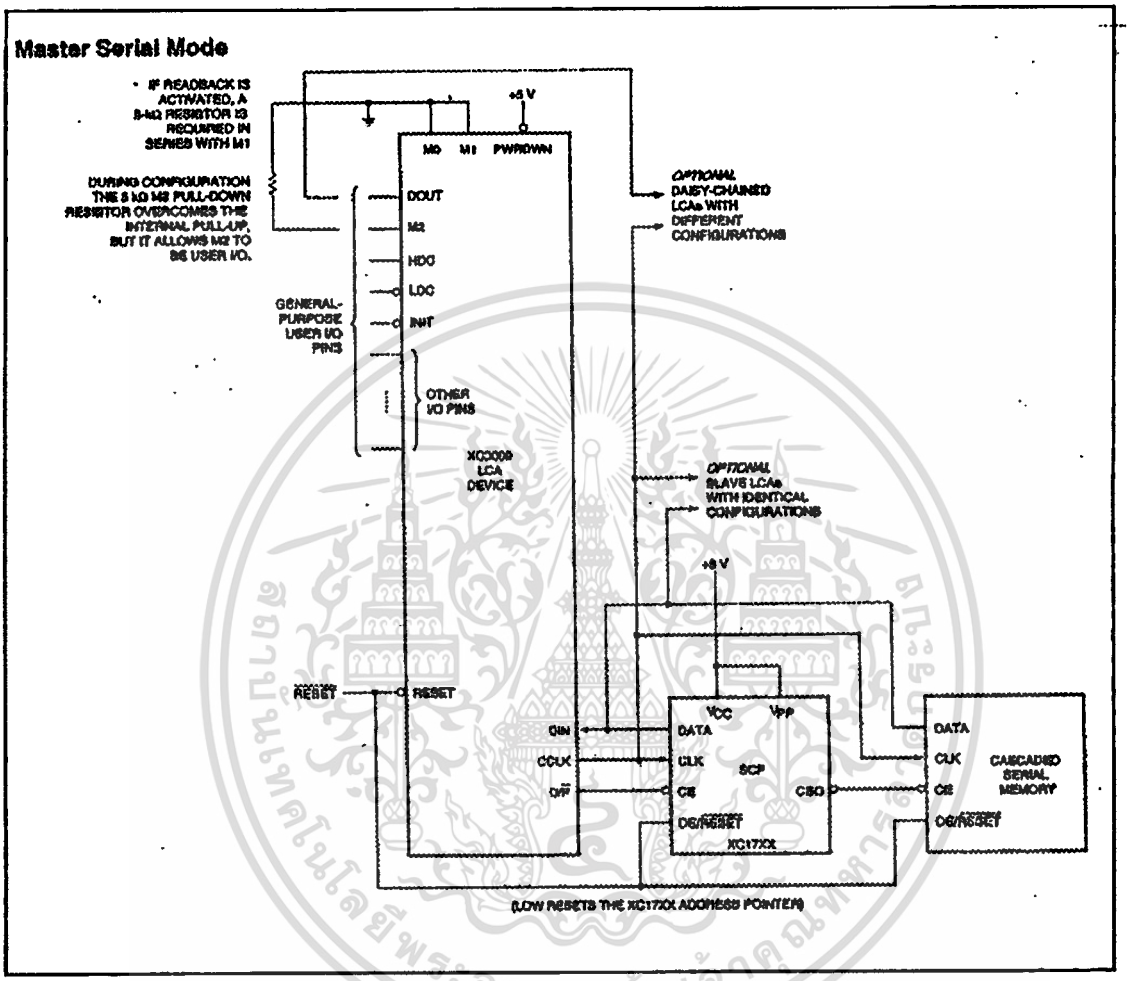
การเลือกใช้โหมดการทำงานของอุปกรณ์ FPGA ที่สามารถกระทำได้

| MO | M1 | M2 | CCLK | Mode | Data |
|---|---|---|---|---|---|
| 0 | 0 | 0 | เอาต์พุต | Master | Bit Serial |
| 0 | 0 | 1 | เอาต์พุต | Master | Byte Wide Addr. = 0000 up |
| 0 | 1 | 0 | - | reserved | - |
| 0 | 1 | 1 | เอาต์พุต | Master | Byte Wide Addr. = FFFF down |
| 1 | 0 | 0 | - | reserved | - |
| 1 | 0 | 1 | เอาต์พุต | Peripheral | Byte Wide |
| 1 | 1 | 0 | - | reserved | - |
| 1 | 1 | 1 | อินพุต | Slave | Bit Serial |

ในโครงสร้างแบบ master นั้น อุปกรณ์ LCA จะกลายเป็นแหล่งจ่ายของ Configulation Clock (CCLK) ส่วนการเริ่มต้นของอุปกรณ์ในโครงสร้างที่เป็น peripheral หรือ slave นั้นจะถูกประวิงเวลาเอาไว้นานเท่าที่จำเป็นให้เพียงพอต่อการเริ่มต้นที่สมบูรณ์ ในอุปกรณ์ LCA ที่ถูกเลือกใช้เป็นโหมดแบบ Master นั้น สภาวะเริ่มต้นจะถูกเพิ่มขึ้นเป็น 4 เท่าตัว คือประมาณ 43 ถึง 130 mSec เพื่อให้แน่ใจว่าอุปกรณ์ตัวลูกที่ต่อพ่วงอยู่ทั้งหมด อยู่ในสภาพพร้อมที่จะทำงานได้แล้ว และหลังจากที่เป็นข้อมูลโครงสร้างถูกโหลดเข้าไปแล้ว และเปรียบเทียบจำนวนความยาวเรียบร้อย สภาพของขา I/O ก็พร้อมใช้งานได้

สำหรับในการนำอุปกรณ์ LCA หรือ FPGA มาใช้งานในการทดลองวิจัยนี้ได้เลือกใช้โครงสร้างเป็นแบบ Master Serial Made ซึ่งเป็นรูปแบบหนึ่งใน Master Mode ดังแสดงตัวอย่างในภาพที่ 18 โดยลักษณะการทำงานแบบนี้ อุปกรณ์ LCA หรือ FPGA จะทำการโหลดข้อมูลโครงแบบจากอุปกรณ์หน่วยความจำภายนอกอย่างอัตโนมัติ เมื่อมีการป้อนแหล่งจ่ายไฟเข้าไปหรือจากคำสั่งโหลดข้อมูลโดยตรง ซึ่งในลักษณะของ Master Serial Mode นี้จะใช้ข้อมูลโครงสร้างแบบอนุกรม เป็น Data-in (DIN) มาจากแหล่งจ่ายอนุกรมแบบซิงโครนัส เช่น อุปกรณ์ Xilinx Serial Configuration PROM หรืออุปกรณ์หน่วยความจำประเภทอื่นที่มีลักษณะในทำนองเดียวกันนี้

ภาพที่ 18




ลักษณะการใช้งานอุปกรณ์ FPGA ให้มีการทำงานแบบ Master Serial Mode

# บทที่ 4
# แนวทางการออกแบบ

## โปรแกรมซอฟต์แวร์ช่วยในการออกแบบ (Design Software)

จากสถาปัตยกรรมโครงสร้างภายในของอุปกรณ์ FPGA ซึ่งประกอบด้วยแถวลำดับทางตรรกของเซลล์จำนวนมากมาย ผู้ออกแบบต้องมีความรู้ความชำนาญในการกำหนดและเชื่อมต่อเซลล์เหล่านี้ให้ถูกต้องและแม่นยำเพื่อให้มีการทำงานตรงตามที่ต้องการ แต่ทว่ากรณีที่วงจรมีความยุ่งยากซับซ้อนหรือมีขนาดใหญ่ การกำหนดจุดเชื่อมต่อต่าง ๆ ย่อมยุ่งยากขึ้น และก่อให้เกิดความเบื่อหน่ายต่อผู้ออกแบบ ดังนั้น จึงมีความต้องการซอฟต์แวร์ที่สามารถช่วยทำหน้าที่ในการแปลงวงจรที่ออกแบบขึ้นมาเข้าไปสู่อุปกรณ์ FPGA อย่างอัตโนมัติ

มีหน้าที่เบื้องต้น 2 ประการที่ซอฟต์แวร์ต้องสามารถกระทำได้คือ 1 ต้องสามารถเปลี่ยนหน้าที่การทำงานต่าง ๆ ในส่วนที่ถูกออกแบบขึ้นมา ให้อยู่ในรูปการทำงานของเซลล์ที่อยู่ภายในตัวอุปกรณ์ FPGA ได้ และ 2 ต้องสามารถทำการตรวจสอบได้ด้วยว่าสิ่งที่กระทำลงไปนั้นถูกต้องตรงตามที่ต้องการหรือไม่

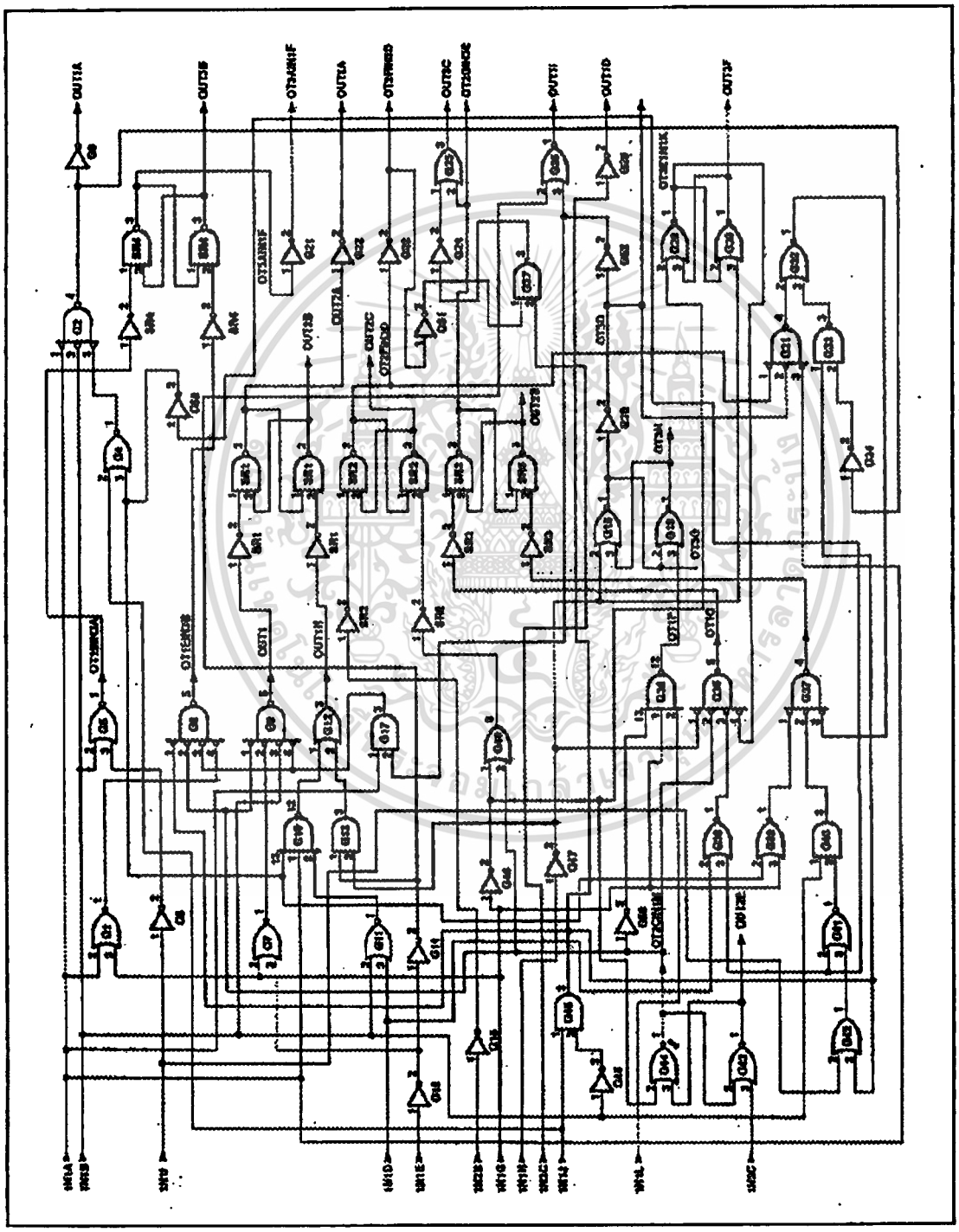
หน้าที่แรกเรียกว่า การเปลี่ยนรูปแบบ (translation) หรือบางครั้งเรียกว่า การแปลโปรแกรม (compliling) หรือ ฟิตติ้ง (fitting) หน้าที่ที่สองเรียกว่า การทวนสอบ (verification) ซึ่งสามารถกระทำได้โดยการใช้ design verification software และ การจำลองทางตรรก (logic simmulation) ทั้งสองหน้าที่ของซอฟต์แวร์นี้จะขึ้นอยู่กับส่วนที่เรียกว่า เน็ตลีส (netlist) ซึ่งจะอธิบายถึงการทำงานและการเชื่อมต่อของวงจรที่ออกแบบขึ้นมา

### ปัญหาของการเปลี่ยนรูปแบบ

ภาพที่ 19 แสดงวงจรที่สร้างขึ้นมาจากเกตแบบแนนด์ โดยละเอียดทั้งทางด้านอินพุตและเอาต์พุต แต่ยังไม่ทราบว่าวงจรทำงานอย่างไรในขั้นต้นนี้ ถ้าสมมติว่าเราต้องการแปลงวงจรในภาพที่ 19 เข้าไปสู่โครงสร้างของอุปกรณ์ FPGA แบบที่แสดงในภาพที่ 20 ซึ่งจำเป็นต้องมีซอฟต์แวร์ที่ใช้ในการเปลี่ยนรูปแบบ ในหลักการคือ ขั้นแรก ให้ทำการนับจำนวนขาของอินพุตและเอาต์พุต และทำการเปรียบเทียบกับจำนวนขาที่มีเอาไว้ให้อยู่ในโครงสร้างภายในของ FPGA

ตามภาพที่ 20 ถ้าถูกต้องตรงกันก็สามารถกระทำได้เลย แต่ถ้าไม่ตรงกัน เช่น มีบางส่วนเกินหรือขาดไป เช่นในการใส่ขาทางอินพุต ต้องพิจารณาดูว่ามีจำนวนของตรรกเซลล์ที่อยู่ภายในภาพ 20 เพียงพอ

ภาพที่ 19



ตัวอย่างของการออกแบบทางดิจิตอล

ต่อการใช้เพื่อให้เกิดการทำงานตรงกับในภาพที่ 19 หรือไม่ ซึ่งต้องพิจารณาให้ดีเพราะอาจทำให้การเปลี่ยนรูปแบบ หรือ แมปปิ้ง (mapping) ไม่สมบูรณ์

เมื่อพิจารณาเข้าไปในส่วนของการแมปปิ้ง โดยปกติแล้วจะพยายามใช้แต่ละเซลล์ของมักซ์ (mux) ที่มีอยู่ในภาพ 20 ให้เต็มที่มากที่สุดเท่าที่จะเป็นไปได้ และเนื่องจากว่าในแต่ละมักซ์เซลล์ (mux cell) นั้นสามารถใช้สร้างเป็นหน้าที่ทางตรรกในแบบที่มี 3 อินพุต ทุก ๆ หน้าที่ ให้ทำการพิจารณาดูว่าส่วนประกอบในภาพ 19 ทั้งหมด จัดแบ่งกลุ่มของเกตที่มี 3 อินพุต และมีเอาต์พุตอันเดียว โดยยังไม่ต้องสนใจในส่วนที่เป็นฟลิปฟลอป ในการพิจารณาให้แยกกลุ่มของเกต ที่มี 3 อินพุตและ 1 เอาต์พุตกลุ่มใหญ่สุดออกมา ส่วนกลุ่มที่เล็กลงมาจะมีความสำคัญน้อยลงไปตามลำดับ ซึ่งการแยกออกเป็นกลุ่มนี้จะทำให้สามารถลดจำนวนของมักซ์เซลล์ที่จะใช้ลงไปได้ ยกตัวอย่างประกอบเพื่อความเข้าใจยิ่งขึ้น การแมปปิ้งนี้จะทำการวางตำแหน่งของเกต เข้าไปยังมักซ์เซลล์ในตัวอุปกรณ์ FPGA ซึ่งจะกระทำโดยทั่วตลอดทั้งหมดดังภาพที่ 20 อย่างรวดเร็ว การยุบกลุ่มของเกตให้เข้าไปสู่มักซ์เซลล์เพียงตัวเดียวนั้น อาจเป็นดังนี้คือ

Gates G45 (INVERTER) และ G46 (AND) ;

Gates G41 (NOR) และ G40 (AND) ;

Gates G51 (interter), G27 (NAND) และ G24 (inverter) ;

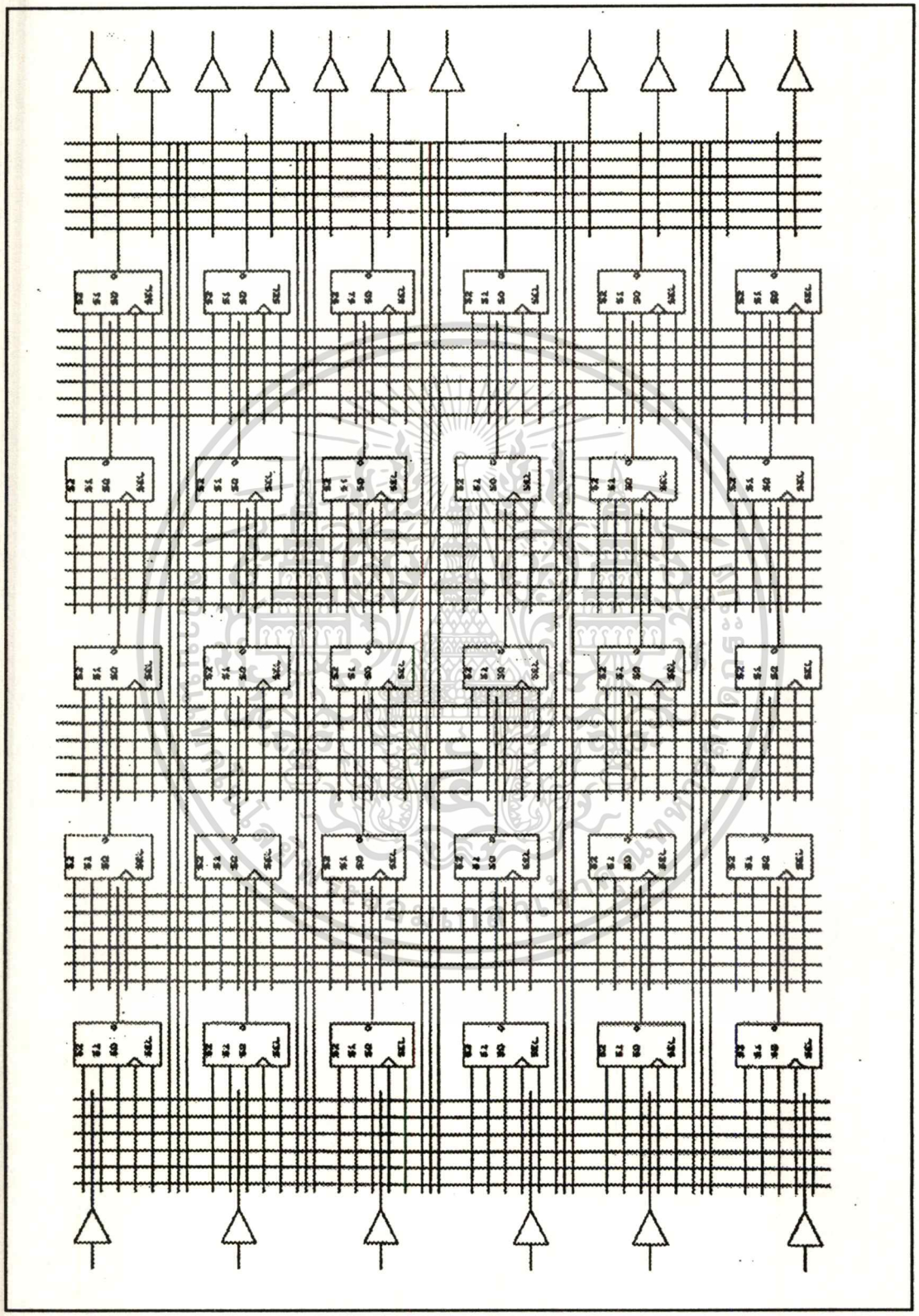
Gates G33 (NAND) และ G32 (NOR)

เกตที่ใช้มักซ์เซลล์อันเดียว ที่มี 3 อินพุต ประกอบด้วยเกต G37, G35 G10 และ G2 เกตที่ต้องใช้มักซ์เซลล์อย่างน้อยสองตัว ประกอบด้วยเกต G36, G8 และ G9 เป็นตัวอย่าง

สำหรับเกตที่มีอินพุตมากกว่า 3 จะต้องการใช้จำนวนมักซ์เซลล์ อย่างน้อย 2 เซลล์ขึ้นไป เพื่อเหตุผลในทางด้านความเร็วของการทำงาน จะทำการจัดกลุ่มให้เกตมี 3 อินพุตและ 1 เอาต์พุตเป็นกลุ่มใหญ่ที่สุด แล้วจึงนำเอาต์พุตที่ได้ออกมาจากเซลล์นี้ ป้อนเป็นอินพุตเข้ากับมักซ์เซลล์ตัวต่อไป เป็นการลดจำนวนชั้นที่จำเป็นต้องใช้ในการสร้างหน้าที่การทำงานของเกตลงส่งผลให้ความเร็วเพิ่มขึ้น

เมื่อวิเคราะห์ในส่วนของวงจร รูปแบบการทำงานของเกตบางตัวก็เหมาะสม บางตัวก็ไม่เหมาะสมหรือพอดีต่อการใช้มักซ์เซลล์ ดังนั้นเพื่อให้เกิดประสิทธิภาพที่ดีขึ้น ต้องทำการปรับปรุงใหม่ให้ทุก ๆ เกตสามารถแปลงเป็นมักซ์เซลล์ ได้อย่างเหมาะสมทั้งหมด ด้วยการจัดกลุ่มของเกตให้มีรูปแบบเป็น 3 อินพุตทั้งหมด แต่ต้องพิจารณาถึงจำนวนมักซ์เซลล์ ที่มีบรรจุอยู่ภายในอุปกรณ์ FPGA ประกอบด้วย ซึ่งหน้าที่นี้เป็นสิ่งสำคัญที่ซอฟต์แวร์ต้องกระทำได้ กล่าวคือ ซอฟต์แวร์มีหน้าที่ในการแปลงวงจรที่ออกแบบเอาไว้ให้เข้าไปสู่อุปกรณ์ FPGA ให้ได้อย่างสมบูรณ์

ภาพที่ 20



แสดงโครงสร้างของอุปกรณ์ FPGA ที่มีการเชื่อมต่อเซลล์ด้วยแซนเนลการเชื่อมต่อระหว่างกัน

ซอฟต์แวร์ ซึ่งทำหน้าที่แปลงวงจรที่ถูกออกแบบขึ้นมาให้เข้าไปสู่อุปกรณ์ FPGA เป็นไปตามกฎเกณฑ์ของการแทนที่กลุ่มการทำงานของวงจรเข้าไปสู่กลุ่มของมักซ์เซลล์ที่อยู่ในตัวอุปกรณ์ ซึ่งมีความจำเป็นที่จะต้องมีรูปแบบหรือแผนการแทนที่ที่ดีที่สุด

ขบวนการเปลี่ยนรูปแบบ อาจเรียกว่าการแปลโปรแกรม (compiling) ตัวแปลโปรแกรมจะทำการเปลี่ยนรูปของความต้องการเบื้องต้น เช่น สมการทางคณิตศาสตร์ ให้อยู่ในรูปของวงจรการทำงานหรือการเชื่อมต่อต่าง ๆ ที่ให้ได้คุณสมบัติตรงกัน การทำงานส่วนอื่นที่อยู่ในขบวนการเปลี่ยนรูปแบบนี้รวมไปถึงเทคโนโลยีของการแมปปิ้ง, การจัดตำแหน่ง (placement) และการจัดเส้นทาง (routing) เทคโนโลยีของการแมปปิ้งรวมไปถึงขบวนการจัดรูปแบบเพื่อให้ได้ผลดีที่สุด การแมปปิ้ง จะทำการโอนถ่ายวงจรหรือความต้องการที่ถูกออกแบบขึ้นมาลงไปยังเซลล์ที่อยู่ใน FPGA หลังจากการแมปปิ้งแล้ว เซลล์ต่าง ๆ ที่มีอยู่ในอุปกรณ์จะถูกกำหนดตำแหน่งเอาไว้อย่างแน่นอนภายใน ซึ่งเรียกว่าการจัดตำแหน่ง ในขณะที่เซลล์ถูกกำหนดตำแหน่งเอาไว้นั้น สัญญาณก็จะถูกกำหนดให้เดินทางอยู่ในเส้นทางเฉพาะตามลักษณะการเชื่อมต่อภายใน เมื่อรวมแผนการในการกำหนดตำแหน่งและเส้นทางนี้อาจเรียกว่า 'filter' ซึ่งเป็นคำใหม่ที่มีการเรียกขานกันในกลุ่มผู้ใช้งานกับอุปกรณ์ FPGA แต่ก็อาจมีชื่อเรียกแตกต่างไปจากนี้ได้ตามแต่กลุ่มผู้ขายอุปกรณ์จะเรียกขานกัน

ในขณะที่ซอฟต์แวร์ทำการตรวจสอบหรือพิจารณาวงจรต้นแบบเพื่อที่จะทำการกำหนดตำแหน่งหรือกำหนดการเชื่อมต่ออยู่นั้น มันจะมีขบวนการกำจัดหรือตัดเอาส่วนของวงจรที่ไม่มีความจำเป็นออกไปได้ด้วย ซึ่งเป็นไปได้เพราะว่าวงจรที่ออกแบบขึ้นมาอาจไม่สามารถทำการถ่ายโอนเข้าไปสู่อุปกรณ์ FPGA ได้ทั้งหมด จึงต้องมีการบีบหรือยุบวงจรลงให้มีขนาดที่เล็กลงให้ได้มากที่สุดเท่าที่เป็นไปได้ ก่อนที่จะทำการแปลงลงสู่อุปกรณ์ FPGA

**การทำเน็ตลิสให้เหมาะที่สุด (Netlist Optimization)**

ปกติเทคนิคของการออกแบบวงจรตรรกทั่วไป นิยมใช้หลักการของคาร์นอแมป และพีชคณิตแบบบูล ซึ่งซอฟต์แวร์ในปัจจุบันก็ยังคงใช้เทคนิคนี้อยู่ และยังใช้ netlist optimization ร่วมด้วย ซึ่งเป็นการยุบหรือทำให้วงจรที่ออกมามีขนาดเล็กที่สุด หลังจากการแปลงรูปเพื่อสร้างเป็นเน็ตลิสนี้เป็นไฟล์ข้อความ (text file) ที่แสดงถึงหน้าที่การทำงานทางตรรก พร้อมทั้งการเชื่อมต่อทั้งหมดทางอินพุตและเอาต์พุต

เน็ตลิสสามารถใช้แสดงให้เห็นถึงหน้าที่การทำงานขนาดใหญ่ เช่น วงจรบวก, มัลติเพล็กเซอร์, วงจรนับ หรือไมโครโพรเซสเซอร์ และในทางกลับกันมันสามารถใช้แสดงถึงหน้าที่การ

ทำงานเล็ก ๆ ได้ เช่น ฟลิปฟลอป, เกต, อินเวอร์เตอร์, สวิตช์ หรือ ทานซิสเตอร์ ซึ่งรูปแบบการอธิบายของมันจะเหมือนกันหมดในทุก ๆ ระดับ ดังนั้นจึงมีความยืดหยุ่นในการใช้งานมาก

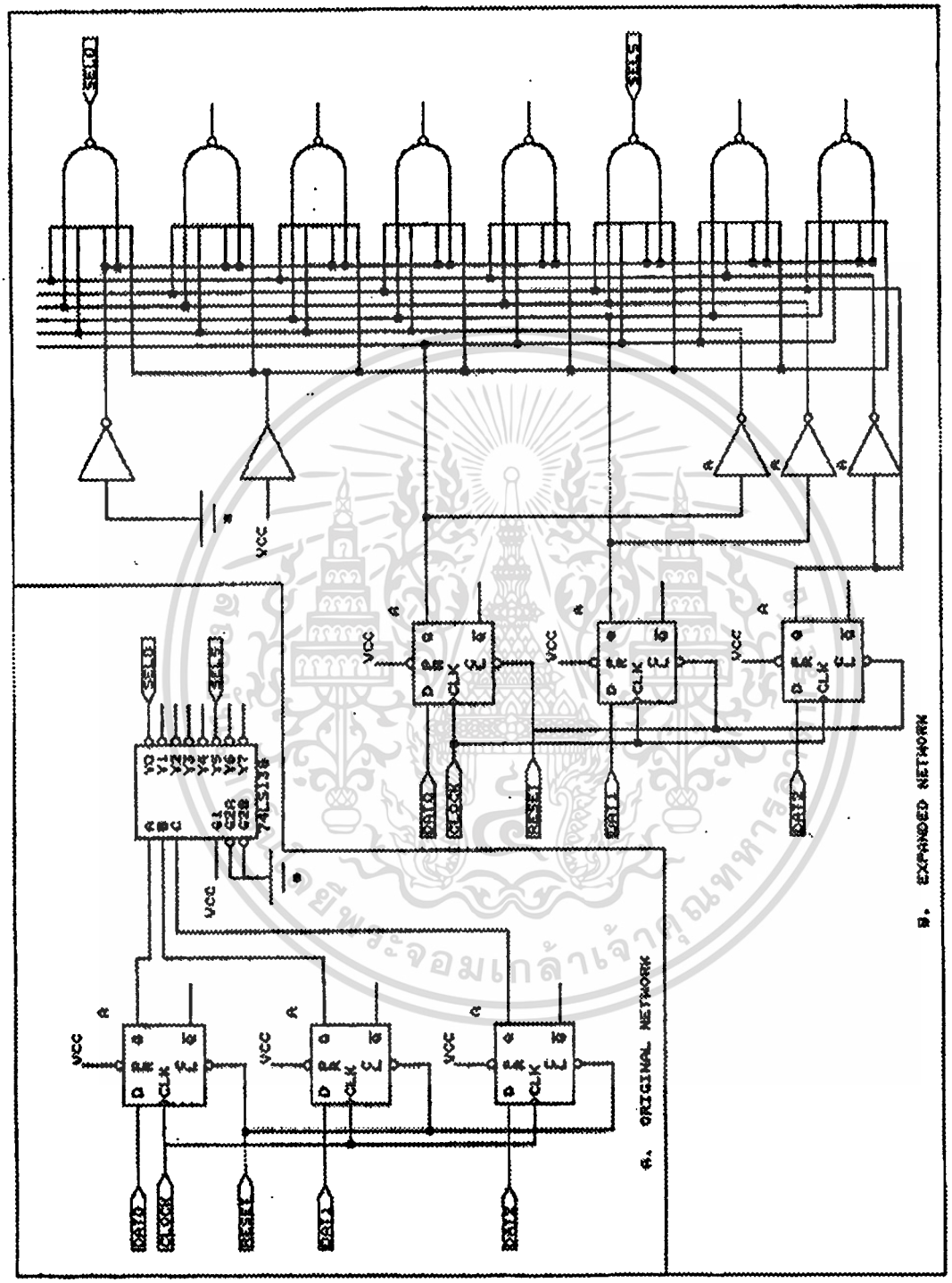
เน็ตลีสที่เป็นตัวนับ สามารถเขียนใหม่ให้อยู่ในรูปของส่วนประกอบภายใน เช่น เป็นตัวฟลิปฟลอปและเกตก็ได้ คล้ายกับเป็นการเปลี่ยนรูป เราเรียกว่า เน็ตลีสเอกซ์แพนชัน (netlist expansion)

เนื่องจากสามารถเปลี่ยนรูปให้อยู่ในแบบต่าง ๆ ทดแทนกันได้ ดังนั้นจึงเป็นไปได้ว่าเราสามารถใช้กฎของการทดแทนหรือแทนที่ต่าง ๆ เข้ามาช่วย เพื่อจุดมุ่งหมายในการกำจัดหรือยุบหน้าที่การทำงานทางตรรกบางอย่างลงไป โดยที่การทำงานรวมทั้งหมดยังคงเดิม ซึ่งนับว่าเป็นหัวใจสำคัญของซอฟต์แวร์ที่ต้องมีอยู่ในปัจจุบัน โดยปกติแล้วภายในของซอฟต์แวร์จะประกอบด้วยกฎของการแทนที่หรือทดแทนในรูปแบบต่าง ๆ อยู่เป็นจำนวนนับร้อยรูปแบบ

ตัวอย่างของการยุบวงจรคือ เมื่อต้องการรักษาหน้าที่การทำงานของตัวถอดรหัสในภาพที่ 21 ซึ่งตัวถอดรหัสนี้ใช้งานขาอินพุตทั้งหมด แต่ใช้ขาทางเอาต์พุตเพียง 2 ขาเท่านั้นจากทั้งหมด 8 ขา ลักษณะการยุบให้เล็กลงนั้น ขั้นแรกคือทำการหาขาทางเอาต์พุตที่ไม่ได้ถูกใช้งาน (ภาพที่ 22) แล้วทำการตัดเกตที่ทำหน้าที่ขับไปยังขาที่ไม่ได้ถูกนำไปใช้งานก็จะทำการกำจัดทิ้งไป กระทำในลักษณะนี้ไปจนกว่าจะไม่สามารถพบส่วนที่ไม่จำเป็นหรือกระทำไปจนกระทั่งไม่สามารถที่จะทำการกำจัดหรือตัดส่วนหนึ่งส่วนใดของวงจรออกไปได้อีก

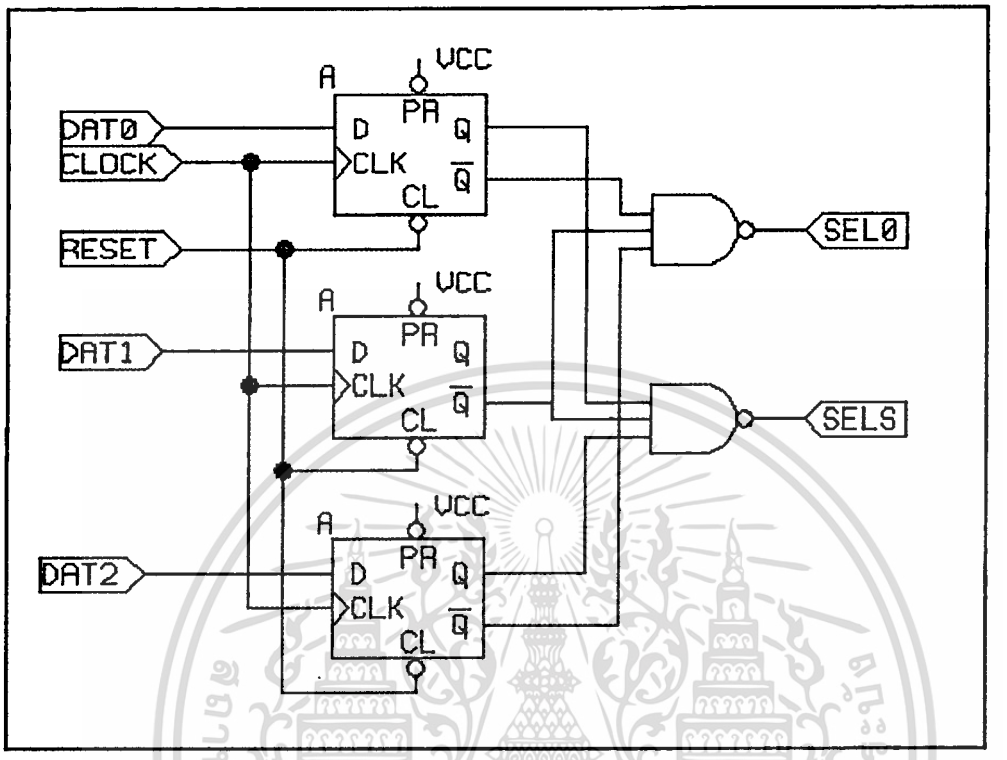
โปรแกรมสมัยใหม่ที่ใช้ในการทำให้เน็ตลีส เล็กลงนั้นสามารถที่จะกำจัดหรือตัดเกตที่ไม่ได้ใช้งานทิ้งไป, ทำการรวมหรือผสมเกตหลาย ๆ ตัวเข้าด้วยกันเพื่อให้มีการทำงานตามที่ต้องการ, ทำการกำจัดฟลิปฟลอปทิ้งไปถ้าไม่จำเป็น หรือทำการรวมเอาหน้าที่ทางตรรกเข้าไปในฟลิปฟลอปตามต้องการ ในความเป็นจริงลักษณะการทำงานของโปรแกรมจะมีความสลับซับซ้อนมากกว่าในตัวอย่างที่กล่าวมา แต่ทว่ามีหลักการและความมุ่งหมายเหมือนกัน

ภาพที่ 21




การทำงานของเน็ตลีส

ภาพที่ 22




ตัวอย่างการทำงานของเน็ตลีส (ต่อ)

**การจัดตำแหน่งเซลล์**

เมื่อมีการแปลงและลดขนาดของวงจรที่ออกแบบ มีความจำเป็นต้องกำหนดหน้าที่การทำงานทางตรรกลงไปยังตำแหน่งที่อยู่ภายในอุปกรณ์ FPGA จัดเป็นปัญหาหนึ่งของการจัดตำแหน่งซึ่งไม่ใช่เรื่องเล็กน้อย ถ้าเป็นไปได้ควรจะมีโปรแกรมคอมพิวเตอร์มาช่วยในการกระทำหน้าที่นี้เนื่องจากว่าขบวนการนี้จะต้องกระทำซ้ำ ๆ กันหลายรอบ ถ้าต้องการทำด้วยมือหรือคนก็กระทำได้ แต่จะทำให้เกิดความเบื่อหน่ายและใช้เวลานาน และบ่อยครั้งจำเป็นที่จะต้องกระทำใหม่เพราะได้ผลออกมาไม่ดีพอ ปกติแล้วจะต้องทำจนกว่าจะได้ผลลัพธ์ออกมาดีที่สุด ซึ่งความยากง่ายขึ้นอยู่กับขนาดและความซับซ้อนของวงจร หรือหน้าที่ทางตรรกต่าง ๆ ประกอบกัน ยิ่งในโครงสร้างที่ประกอบด้วยเซลล์แบบ 2 มิติ เป็นการยากที่จะกระทำด้วยมือให้ได้ผลลัพธ์ออกมาดีถึงขนาดที่ต้องการ

หลักการสำคัญในการจัดตำแหน่งเซลล์ คือต้องการเชื่อมต่อเซลล์ต่าง ๆ เข้าด้วยกัน เพื่อให้ได้ผลการทำงานที่ถูกต้อง รวมถึงต้องใช้เวลาหรือเส้นทางในการเชื่อมต่อน้อยที่สุดและมีค่าเวลาประวิงต่ำสุดเท่าที่จะเป็นไปได้ ดังนั้นเพื่อให้ได้ผลลัพธ์ออกมาดีที่สุดต้องอาศัยความพยายาม

มากพอสมควร และจะเป็นการง่ายยิ่งขึ้นถ้ามีโปรแกรมที่สามารถช่วยในการกระทำหน้าที่นี้แทน

**การเชื่อมต่อระหว่างเซลล์**

สมมติว่าสามารถทำการแทนที่การทำงานของวงจร ลงไปยังเซลล์ได้อย่างดีแล้ว ขั้นตอนต่อมาก็คือการเชื่อมต่อทุก ๆ ส่วนเข้าด้วยกัน เรียกว่าการจัดเส้นทาง (routing) เมื่อการจัดเส้นทางเริ่มต้นขึ้น ส่วนของเน็ตลีสจะถูกทำการตรวจสอบเพื่อใช้เป็นข้อมูลของการเชื่อมต่อภายใน รวมถึงการตรวจสอบตำแหน่งการวางของเซลล์ต่าง ๆ ด้วย มีซอฟต์แวร์ที่ใช้สำหรับการเชื่อมต่อภายในซึ่งจะทำการกำหนดการวิ่งของสัญญาณจากตำแหน่งเอาต์พุตของเซลล์ตัวหนึ่งเข้าไปยังอินพุตของเซลล์อีกตัวหนึ่ง ยกตัวอย่างเช่น จากภาพที่ 20 จะมีการเชื่อมต่อเส้นทางเกิดขึ้นในระหว่างมักซ์เซลล์ ซึ่งเป็นรูปแบบ FPGA แบบง่าย ๆ เมื่อเส้นโลหะมีสัญญาณเอาต์พุตออกมาและพยายามที่จะส่งป้อนเข้าไปเป็นอินพุตของมักซ์เซลล์ตัวอื่น ถือว่ามีการใช้เส้นโลหะนั้น ขบวนการเชื่อมต่อจะดำเนินต่อไปเรื่อย ๆ ซึ่งก็คือมีการใช้งานเส้นโลหะที่มีอยู่ภายในโครงสร้างซ้ำไปซ้ำมาจนเกิดความแออัดหรือยุ่งเหยิงขึ้นมา เมื่อเกิดความแออัดขึ้นมาการเชื่อมต่ออันถัดมาอาจไม่สามารถกระทำได้ เพราะหาเส้นทางเชื่อมต่อไปยังส่วนที่ต้องการไม่ได้แล้ว โปรแกรมก็ไม่สามารถทำงานต่อไปได้ ถ้าเป็นอย่างนี้ ซอฟต์แวร์จะต้องทำการวางหรือจัดเรียงตำแหน่งของเซลล์ภายในใหม่แล้วทำการหาเส้นทางการเชื่อมต่ออีกครั้งหนึ่ง

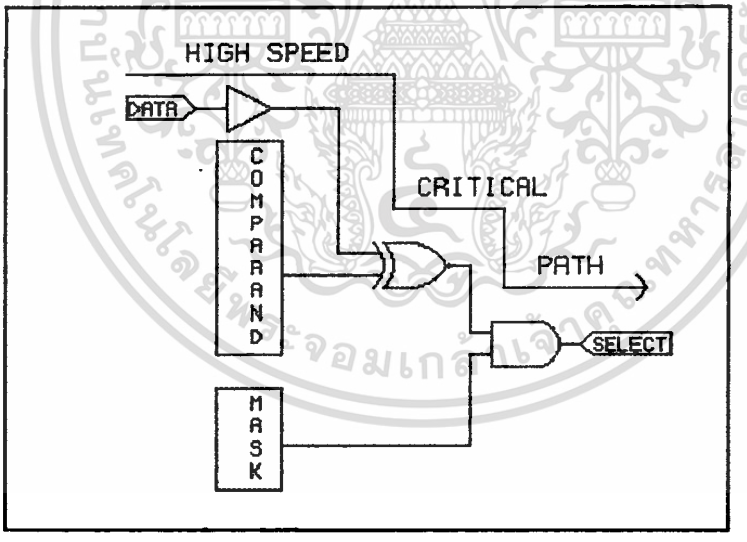
ผลลัพธ์ที่ได้จากการแทนที่และการจัดเส้นทาง ก็คือ design file ที่อธิบายถึงวงจรที่ถูกออกแบบขึ้นมาในรูปของเซลล์ภายใน FPGA ในไฟล์นี้จะอธิบายถึงตำแหน่งที่กำหนดเอาไว้ของเซลล์และการเชื่อมต่อภายในระหว่างเซลล์ต่าง ๆ สำหรับแซนเนล หรือ foldback network design file นี้เป็นขั้นสุดท้ายสำหรับการสมนัยบิต (bitmap) ซึ่งสามารถส่งเข้าไปยังอุปกรณ์ที่ทำหน้าที่โปรแกรมลงไปสู่อุปกรณ์ FPGA

จากตรงนี้ สามารถมองเห็นถึงลักษณะเชื่อมต่อภายในอุปกรณ์ FPGA ซึ่งสำคัญมากต่อการประเมินหรือกำหนดการเชื่อมต่อของวงจรที่ออกแบบให้ได้ผลดีที่สุด โปรแกรมซอฟต์แวร์ที่ฉลาดช่วยในการทำหน้าที่นี้ให้ได้ผลลัพธ์ที่ดีได้ แต่ถ้าวงจรที่ออกแบบขึ้นมาตอนแรกมีการกำหนดการเชื่อมต่อต่าง ๆ ไม่ดี ซอฟต์แวร์ก็ไม่สามารถช่วยอะไรได้มากนัก แต่ถ้าใช้ซอฟต์แวร์ที่ไม่ดีคือมีความสามารถไม่สูงพอ แม้ว่าวงจรที่ออกแบบขึ้นมานั้นดีแล้ว ผลที่ได้จากซอฟต์แวร์ก็อาจจะไม่ดีพอตามความต้องการก็เป็นไปได้ ดังนั้นจึงจำเป็นต้องใช้ซอฟต์แวร์ที่ช่วยในการออกแบบ FPGA ที่มีประสิทธิภาพสูงมาช่วยในการออกแบบเสมอ

**วิถีวิกฤต (Critical paths)**

ถ้าหากว่าทุก ๆ เซลล์ มีการวางตำแหน่งและการเชื่อมต่อไปยังทุก ๆ จุด การทำงานของซอฟต์แวร์ก็จะง่ายขึ้น แต่ในความเป็นจริงมักไม่เป็นเช่นนี้ เนื่องจากวงจรที่ออกแบบขึ้นมามักจะมีความยุ่งยากซับซ้อนในการกำหนดตำแหน่ง และการเชื่อมต่อของส่วนต่าง ๆ รวมถึงยังต้องพิจารณาในเรื่องของเวลาประกอบด้วย อีกทั้งสัญญาณบางอย่างต้องการมาจากภายนอกป้อนเข้ามายังอุปกรณ์ FPGA เพื่อใช้สร้างหน้าที่การทำงานทางตรรก และส่งไปยังส่วนอื่นอีก ลักษณะเช่นนี้มีความจำเป็นต้องใช้ตัวไมโครโพรเซสเซอร์ มาช่วยในการถอดรหัสและเปรียบเทียบให้ได้ในเวลาอันสั้น ดังแสดงในภาพที่ 23 ค่าเวลาต่าง ๆ ทั้งหมดของตัวฟลิปฟลอปที่อยู่ในส่วนรีจิสเตอร์เลื่อนข้อมูล หรือตัววงจรนับ จะคงอยู่และต่อเนื่องไปยังเซลล์ตัวถัดไปที่อยู่ภายในตัว FPGA เพื่อเป็นการรับประกันว่าวงจรทั้งหมดทำงานภายใต้การควบคุมของสัญญาณนาฬิกาพร้อมเพรียงกัน จึงเกิดข้อจำกัดทางด้านของเวลาขึ้น ซึ่งเรียกว่า วิถีวิกฤต หรือ critical nets

ภาพที่ 23




แสดงวิถีวิกฤต

ข้อจำกัดทางด้านของเวลา หรือวิถีวิกฤต มีผลทำให้เกิดความยุ่งยากเพิ่มมากขึ้นในการจัดตำแหน่งเซลล์และเส้นทางการเชื่อมต่อภายใน ขณะที่ทำการออกแบบ เนื่องจากสิ่งเหล่านี้มีผลต่อการเพิ่มขึ้นทางด้านเวลาที่สัญญาณต้องใช้ในการเดินทางทั้งสิ้น โดยหลักการแล้วเวลาออกแบบจะทำการวางเซลล์ไว้ในตำแหน่งที่เป็นวิถีวิกฤตก่อนเป็นอันดับแรก หลังจากนั้นจึงค่อยวางใน

ตำแหน่งที่เหลือต่อมา ทำการเชื่อมต่อเซลล์ที่อยู่ในตำแหน่งวิถีวิกฤต ให้ได้ค่าเวลาตามที่ต้องการ แล้วจึงทำการเชื่อมต่อในส่วนที่เหลือภายหลัง ด้วยการกระทำการเชื่อมต่อและวางตำแหน่งเซลล์ลงในวิถีวิกฤตเป็นอันดับแรกนี้ จะทำให้เงื่อนไขในส่วนที่เหลือน้อยลง กล่าวคือทำให้การวางตำแหน่งหรือเชื่อมต่อส่วนอื่น ๆ ที่เหลือของวงจรง่ายขึ้น แต่ถ้าหากว่ามีการกำหนดหรือต้องการวิถีวิกฤตเป็นจำนวนมาก ย่อมเป็นที่คาดหมายได้ว่าการกำหนดตำแหน่งและการเชื่อมต่อต้องใช้เวลานานมากขึ้นเป็นสัดส่วนกัน ดังนั้นเพื่อให้ได้ผลดีที่สุดจึงควรมี วิถีวิกฤต น้อยที่สุดเท่าที่เป็นไปได้

**การทวนสอบ (Design verification)**

ถ้าหากการออกแบบทุกอย่างถูกต้องเหมาะสมหมด ไม่ว่าจะเป็นการแปลงวงจร การลดขนาดวงจร การกำหนดหรือวางตำแหน่งของเซลล์และการเชื่อมต่อ ก็ไม่มีความจำเป็นต้องมีการตรวจสอบหรือทดสอบเกิดขึ้น แต่ทว่าในความเป็นจริง มักจะเกิดข้อติดขัดหรือยุ่งเหยิงสับสนเกิดขึ้นเสมอในทุก ๆ ขบวนการ หรือเกิดข้อผิดพลาดเกิดขึ้น design verification จึงมีไว้เพื่อช่วยแก้ไขปัญหาเหล่านี้

โดยปกติเครื่องมือที่ใช้เป็น design verification มักจะเป็นโปรแกรมซอฟต์แวร์ที่สามารถทำการแปลงเน็ตลีสุออกมา แล้วทำการตรวจสอบ รวมถึงทำการวิเคราะห์คุณสมบัติโดยทั่วไปในขั้นตอนสุดท้ายของการออกแบบ ยกตัวอย่างเช่น หลักการตรวจสอบวงจรที่ออกแบบสามารถที่จะแยกเอาต์พุตของเซลล์ออกมา และสามารถนับจำนวนอินพุตของเซลล์ที่ได้รับการขับจากมัน ในแต่ละเซลล์ที่ถูกขับจะเป็นการเพิ่มพูนค่าเวลาประวิงเข้าไปยังเอาต์พุตของเซลล์ที่ขับ ถ้าหากมีจำนวนของเซลล์ที่ถูกขับมีจำนวนมากเกินไป อาจทำให้ระบบหรือวงจรที่ออกแบบขึ้นมานั้นไม่สามารถใช้งานได้ เนื่องจากมีค่าของเวลาประวิงมากเกินไป อาจทำให้ระบบหรือวงจรที่ออกแบบขึ้นมานั้นใช้งานไม่ได้เพราะมีค่าของเวลาประวิงมากเกินไป ดังนั้นผู้ออกแบบจะต้องทำการแบ่งหรือทำให้มีจำนวนของเซลล์ที่ต้องขับนั้นน้อยลง มีมาตรฐานหรือรูปแบบของการตรวจสอบบางประเภทสามารถหลีกเลี่ยงปัญหาของค่าเวลาที่เกิดจากจำนวนเซลล์อินพุต แต่จะมีปัญหาของสัญญาณรบกวนแทนที่เข้ามา บางรูปแบบจะทำการกำหนดหรือแบ่งแยกเซลล์เอาต์พุต โดยพยายามกระทำให้อยู่ในรูปแบบต่อตรงถึงกัน ซึ่งไม่มีความสามารถทาง 3-สเตต ทุก ๆ การตรวจสอบเหล่านี้ต่างถูกจัดรวมเข้าไว้ในโปรแกรมสำเร็จรูปในการเปลี่ยนรูปแบบ (translation packages) ในขณะที่ซอฟต์แวร์อื่นจะรวมเอากฎของการตรวจสอบเข้าไว้ในส่วนของการจำลองเลียนแบบ

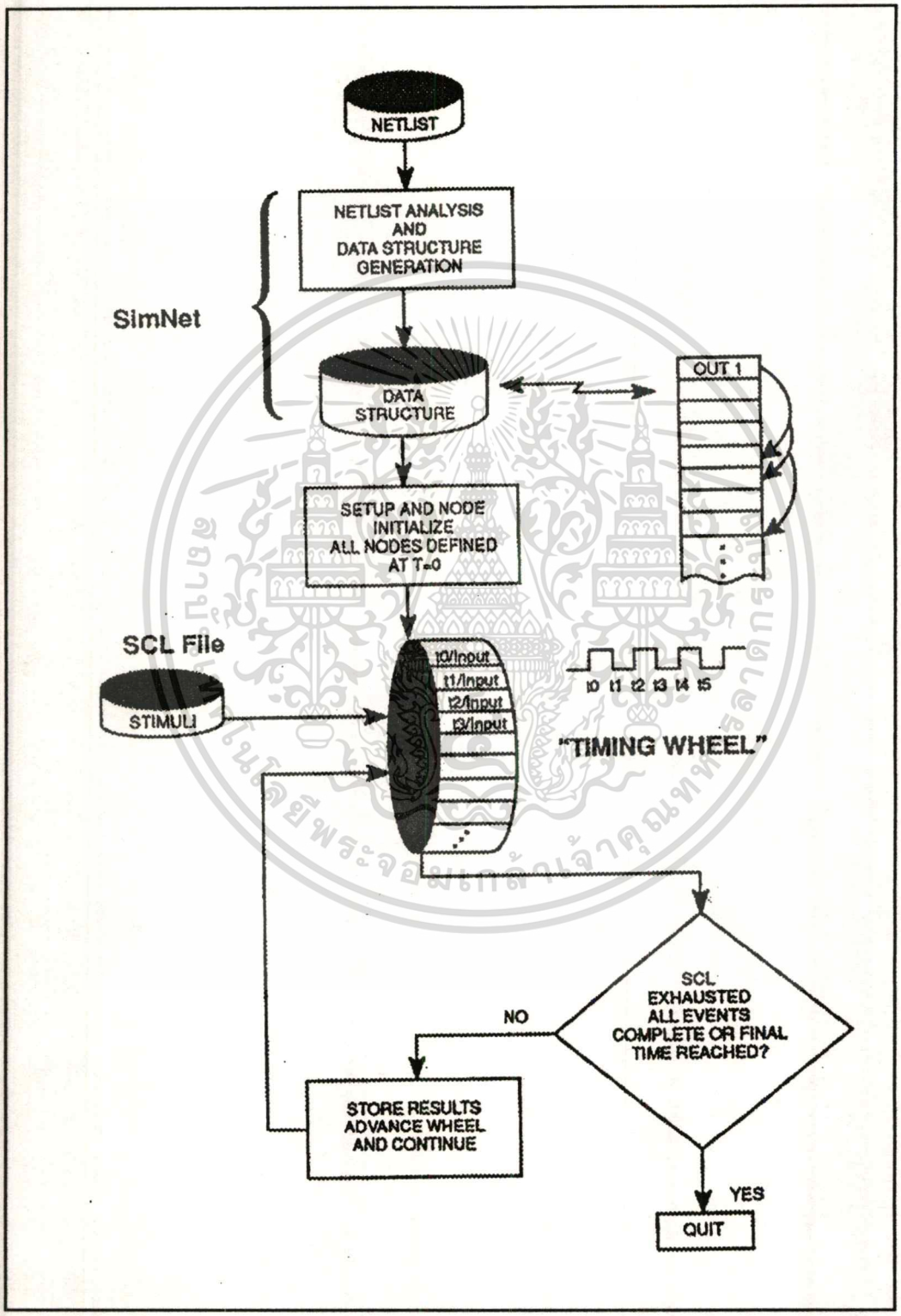
**การจำลองทางตรรก (Logic simulation)**

การจำลองการทำงานทางตรรกนี้เป็นเครื่องมือที่ใช้สำหรับกำหนดหรือประเมินคุณลักษณะของการทำงาน และค่าช่วงเวลาต่าง ๆ ในการออกแบบ FPGA ในทุก ๆ รูปแบบของการจำลองเลียนแบบ จะมีการสร้างรูปแบบของวงจรตรรกขึ้นมา และถูกขับโดยรูปแบบของสัญญาณอินพุต ที่เรียกว่า 'กระตุ้น' เพื่อสร้างรูปแบบของสัญญาณเอาต์พุตขึ้นมาเรียกว่า 'ตอบสนอง' คุณสมบัติที่สำคัญของการจำลองเลียนแบบที่ต้องมีคือ สามารถที่จะสังเกตเห็นผลหรือการตอบสนองทางตรรกภายในได้ในขณะที่ไม่สามารถสังเกตเห็นได้จริงที่ขาของอุปกรณ์ การแก้ไขหรือปรับปรุงวงจรที่ไม่มีการจำลองเลียนแบบการทำงานขึ้นมาก่อนนั้น คล้ายกับการแก้ไขวงจรดิจิตอลขนาดใหญ่ โดยการมองหรือพิจารณาแค่เพียงจุดเชื่อมต่อที่ขาต่าง ๆ ของอุปกรณ์จากแผงวงจรเท่านั้น ซึ่งก็สามารถแก้ไขวงจรได้แต่ยากมาก

การจำลองเลียนแบบมีหลายประเภทด้วยกัน ในแต่ละประเภทต่างก็มีข้อดีแตกต่างกันออกไป ประเภทแรกคือ การจำลองฟังก์ชัน (function simulation) ซึ่งทำการจัดรูปแบบของเซลล์ตรรกที่อยู่ในวงจรออกแบบและรวมเข้าด้วยกันโดยใช้รูปแบบของแรงดันอินพุต ที่เป็นแบบไบนารี่เพื่อใช้สร้างหรือกำเนิดรูปแบบการตอบสนองของสัญญาณเอาต์พุตที่สัมพันธ์กันขึ้นมา (อยู่ในรูปของแรงดันไฟฟ้า) ข้อดีของแบบนี้ก็คือ ทำได้ง่ายและสามารถทราบผลลัพธ์ออกมาได้อย่างรวดเร็ว แต่จะขาดความถูกต้องแม่นยำ และรายละเอียดเกี่ยวกับค่าเวลาที่เกี่ยวข้องในระหว่างกลุ่มของสัญญาณ ดังที่ทราบกันว่า ในการกำหนดตำแหน่งและเส้นทางการเชื่อมต่อภายใน FPGA ต้องให้ความสนใจในเรื่องของค่าเวลาที่ใช้ในเส้นทางต่าง ๆ ด้วย แต่การจำลองเลียนแบบประเภทนี้ไม่สามารถช่วยในเรื่องของค่าเวลาเหล่านี้ได้

ประเภทที่สอง คือ การจำลองทางเวลาเชิงเลข (digital timing simulation) ซึ่งเป็นแบบที่นิยมใช้มากที่สุดในแบบนี้ วงจรที่ถูกออกแบบขึ้นมาทั้งหมดจะถูกทำการจำลองสร้างขึ้นมาในรูปแบบการเชื่อมต่อของเซลล์ขึ้นมา พร้อมกับรูปแบบเซลล์เอาต์พุตแต่ละอัน ค่าเวลาประวิงของเซลล์แต่ละเซลล์ ซึ่งโดยปกติรูปแบบที่ออกแบบของจริงจะประกอบด้วยรูปแบบของเซลล์เป็นจำนวนร้อยหรือจำนวนพัน ๆ เซลล์ แต่ละเซลล์จะมีค่าของเวลาประวิงมากหรือน้อยนั้น ขึ้นอยู่กับตัวแปร 3 ตัว คือ 1 กรณีที่เซลล์นั้นปราศจากการเชื่อมต่อกับภายนอก ค่าเวลาประวิงจะมีอยู่ 2 ค่าด้วยกันคือ ขณะที่เปลี่ยนสภาวะจากโลจิก '1' เป็น '0' และในขณะที่เปลี่ยนสภาวะจากโลจิก '0' เป็น '1' ซึ่งค่าของเวลาประวิงนี้ปกติแล้วจะมีบอกเอาไว้ในคู่มือของอุปกรณ์ FPGA อย่างที่ 2 คือ ค่าเวลาประวิงที่เกิดในเส้นทางการเชื่อมต่อระหว่างเซลล์ที่ทำหน้าที่เป็นตัวขับ กับเซลล์ที่ทำหน้าที่เป็นตัวรับ อย่างที่ 3 คือ ผลรวมของค่าอินพุตอิมพีแดนซ์ทั้งหมดของเซลล์ที่ต้องขับ การจำลอง-

ภาพที่ 24




ลำดับขั้นการจำลองเลียนแบบการทำงาน

เลียนแบบบางประเภทจะไม่สนใจส่วนนี้ แต่บางประเภทจะทำการรวมเอาค่าเวลาประวิงสองอย่างหลังเข้าเป็นค่าเดียว ในระหว่างที่ทำการจำลองเลียนแบบการทำงาน เซลล์แต่ละตัวจะถูกกำหนดค่าของเวลาประวิงเข้าไปด้วยเสมอ ดังนั้นการจำลองเลียนแบบประเภทนี้จึงมีความยุ่งยากซับซ้อนมากกว่าในประเภทแรก

ประเภทที่ 3 คือ การจำลองความผิดพร่อง (fault simulator) ในประเภทนี้ต้องใช้เทคนิคพิเศษเพื่อใช้สร้าง score (เรียกว่า fault grade) สำหรับการออกแบบและการสร้างสัญญาณกระตุ้น จุดมุ่งหมายของการจำลองเลียนแบบประเภทนี้ก็คือ เพื่อช่วยให้ผู้ออกแบบสามารถทำการทดสอบวงจรที่ออกแบบขึ้นมาในทุก ๆ ลักษณะตามที่ต้องการ ด้วยการสร้างหรือจำลองปัญหาขึ้นมาเอง (เรียกว่า faults) และทำการดูผลลัพธ์ที่เกิดขึ้นว่าเป็นไปตามที่เราออกแบบเอาไว้หรือไม่ การทำงานพื้นฐานของการจำลองเหตุเสียแบบนี้คือ ทำให้สัญญาณเอาต์พุตของเซลล์เป็น '0' หรือเป็น '1' ในขณะที่ป้อนสัญญาณกระตุ้นหรืออินพุตเข้าไป แล้วดูผลลัพธ์ ขบวนการนี้ไม่สามารถกระทำพร้อมกันทุกเซลล์ในครั้งเดียว แต่สามารถทำได้ทุกเวลาถ้าหากว่าเซลล์หรือตำแหน่งที่ทำการทดสอบสามารถแยกอิสระออกมาจากส่วนอื่น ๆ ได้

การจำลองความผิดพร่อง ต้องมีการคำนวณเหตุเสียซ้ำไปมาจำนวนมากมายหลายรอบ ซึ่งอาจใช้เวลานานกว่าจะได้ผลลัพธ์ที่ตรงตามความต้องการ ด้วยเหตุนี้จึงจำเป็นต้องมีการจำลองความผิดพร่อง ที่มีความเร็วสูงและเปี่ยมประสิทธิภาพมาช่วยในการทำงาน

ปกติในการออกแบบ FPGA จะใช้การจำลองทางเวลาเชิงเลข ซึ่งสามารถตรวจสอบหรือสำรวจลึกเข้าไปในรายละเอียดได้มากกว่า และง่ายต่อการทำงาน แต่ถ้าเป็นการออกแบบของอุปกรณ์ PLD ที่เล็กกว่านี้มักจะใช้การจำลองฟังก์ชัน โดยไม่สนใจความถูกต้องในเรื่องของค่าเวลาประวิง ส่วนในการออกแบบอุปกรณ์ขนาดใหญ่แบบ ASIC (Application Specific Integrated Circuit) จำเป็นต้องใช้การจำลองความผิดพร่อง เพื่อรับประกันในเรื่องของสัญญาณกระตุ้นอินพุต มีมากเพียงพอสำหรับการทดสอบและออกแบบ การเรียนรู้เกี่ยวกับการจำลองความผิดพร่อง มีความจำเป็นมากในกรณีที่ผู้ออกแบบต้องการที่จะโอนย้ายวงจรที่ออกแบบอยู่ในตัว FPGA ให้เข้าไปสู่ตัว ASIC ต่อไปจะเป็นการอธิบายในรายละเอียดเพิ่มเติมเกี่ยวกับการจำลองทางเวลาเชิงเลข

การจำลองทางเวลาเชิงเลข : ภาพที่ 24 แสดงขั้นตอนของโปรแกรมที่ใช้จำลองเลียนแบบการทำงานทางดิจิตอล ตัวจำลองนี้จะใช้รูปแบบต่าง ๆ ในการแก้ไขหรือเปลี่ยนแปลงลักษณะของข้อมูลอินพุตที่จะทำให้เป็นข้อมูลเอาต์พุต เช่นเดียวกับโปรแกรมโดยทั่วไป ข้อมูลอินพุตจะประกอบด้วย 2 ส่วน คือ 1. รูปแบบจำลองของการออกแบบ ที่เปลี่ยนแปลงหรือขึ้นอยู่กับ เน็ตลีส

แต่ละอันทำการปรับปรุงให้มีความเหมาะสมกับความต้องการของตัวจำลอง และ 2. รูปแบบของสัญญาณกระตุ้นอินพุต ในลักษณะของแรงดันไฟฟ้าที่เกิดขึ้นในแต่ละช่วงเวลา

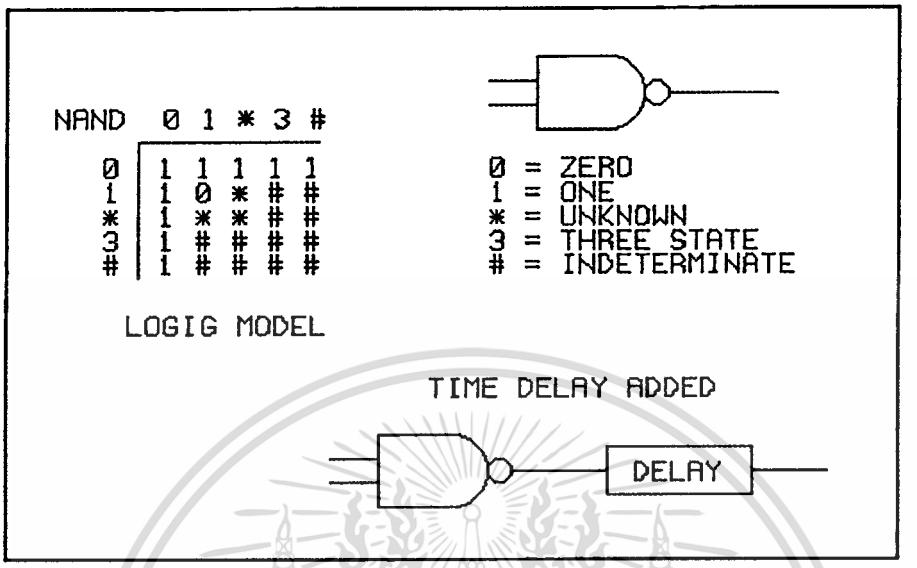
ตัวจำลองจะทำการประยุกต์สัญญาณกระตุ้นให้เข้ามาตามรูปแบบที่ต้องการสำหรับวงจร และสร้างรูปแบบของสัญญาณเอาต์พุตขึ้นมา เนื่องจากว่าสัญญาณกระตุ้นอินพุตที่เข้ามาเป็นลักษณะของสัญญาณไบนารี่ ดังนั้นจึงเรียกว่า 'อินพุตเวกเตอร์' ในทำนองเดียวกันสัญญาณทางเอาต์พุต ก็เรียกว่า 'เอาต์พุตเวกเตอร์' ภายใต้เงื่อนไขที่แน่นอน ทั้งอินพุตและเอาต์พุตเวกเตอร์ ที่จำลองขึ้นมาสามารถใช้ได้โดยตัวทดสอบอิเล็กทรอนิกส์ เพื่อทำการทดสอบคุณสมบัติการทำงานของอุปกรณ์ FPGA ที่จำลองขึ้น ในกรณีนี้ทั้งอินพุตและเอาต์พุตเวกเตอร์ จะรวมกันเรียกว่า 'เวกเตอร์ทดสอบ'

การทำงานของตัวจำลองจะเป็นไปในลักษณะที่ตรงไปตรงมา แต่อย่างไรก็ตามในการจำลองเลียนแบบการทำงานมีความต้องการตัวแปรหรือเงื่อนไขมากกว่านี้ ในส่วนแรกของตัวจำลอง จะรักษารูปแบบของเวลาและรายการต่าง ๆ ที่เกิดขึ้นในขณะที่มีการจำลองเลียนแบบการทำงานขั้นต่อไปเอาไว้ ซึ่งมีชื่อเรียกว่า 'โปรแกรมกำหนดงาน (scheduler)' ในการจำลองเลียนแบบการทำงานจะสามารถกระทำได้ง่ายมากถ้าโหนดของเน็ตลิสเป็นลักษณะไบนารี่ คือมีแค่ 2 ค่าเท่านั้น โหนดในที่นี้หมายถึงเอาต์พุตของเกตที่อยู่ภายในของฟลิปฟลอป อินพุตหรือเอาต์พุตเบื้องต้น การเปลี่ยนของโหนดก็จะมีแค่เปลี่ยนจากโลจิก '1' เป็น '0' หรือเปลี่ยนจาก '0' เป็น '1' เท่านั้น แต่ในความเป็นจริงของวงจรอิเล็กทรอนิกส์มักจะยุ่งยากซับซ้อนเกินกว่าที่จะแทนค่าด้วยสัญญาณไบนารี่ ดังนั้นโปรแกรมการจำลองเลียนแบบที่ดีต้องมีรูปแบบของสัญญาณที่โหนดมากกว่าแบบไบนารี่ธรรมดา เพื่อการจำลองเลียนแบบที่เหมือนจริงมากยิ่งขึ้น

Scheduling : ตัวโปรแกรมกำหนดงานจะทำการเก็บรักษาค่าของเวลาและเหตุการณ์ที่เกิดขึ้นเอาไว้ส่งข้อมูลของเหตุการณ์ตามติดกันไปเพื่อใช้สำหรับการคำนวณหาค่าต่าง ๆ เมื่อถึงเวลา ตัวที่ทำหน้าที่รับข้อมูลเหตุการณ์นี้ก็คือ รูปแบบการหาค่าที่ใช้ประเมินหรือกำหนดเงื่อนไขทางตรรกสำหรับแต่ละเหตุการณ์ที่ถูกเลือกไว้ รูปแบบการหาค่านี้จะสร้างค่าขึ้นมาใหม่ สำหรับโหนด (ตามค่าที่อยู่ในตารางความจริง) พร้อมทั้งกำหนดค่าเวลาประวิงสำหรับเหตุการณ์ตัวใหม่ที่เกิดขึ้นมา เหตุการณ์ตัวใหม่จะย้อนกลับไปยังตัวโปรแกรมกำหนดงาน และสอดแทรกเข้าไปในรายการของเหตุการณ์ ที่ตัวโปรแกรมกำหนดงาน ก็จะทำการคัดเลือกลำดับเหตุการณ์ของมันและทำการหาเหตุการณ์อันใหม่เพื่อส่งออกไป ขบวนการนี้จะกระทำซ้ำ ๆ ไปจนกระทั่งใช้อินพุตเวกเตอร์จนหมด หรือหยุดตามเวลาที่ผู้ออกแบบกำหนด

ตัวโปรแกรมกำหนดงาน ประเภทที่มีรูปแบบการรักษาค่าของเวลาในแต่ละเหตุการณ์ตาม

ภาพที่ 25




ตัวอย่างแสดงการจำลองเลียนแบบการทำงานของเกตแบบแนนด์

รายการที่กำหนดเรียกว่า 'วงล้อเวลา (time wheel)' ในแบบนี้จะมีการสอดแทรกหรือดึงเอาเหตุการณ์ออกมาได้ตามความต้องการของผู้ออกแบบ ซึ่งจัดว่ามีประสิทธิภาพสูงพอสมควร

ในอีกทางหนึ่งสำหรับการักษาค่าเวลาไว้ คือ เพิ่มการเปลี่ยนของเวลาขึ้นและกำหนดรูปแบบทุก ๆ อย่างที่สามารถเกิดขึ้นได้ ถ้าหากว่าไม่มีอะไรเกิดขึ้น ค่าเวลาจะเพิ่มขึ้นและมีการส่งคำถามออกไปอีกครั้งหนึ่ง ลักษณะคล้ายกับการ polling ซึ่งมีชื่อเรียกว่า 'time driven simulation'

ในแบบวงล้อเวลา จะให้การจำลองเลียนแบบเป็น 'event driven' ซึ่งมีความรวดเร็วกว่าแบบ time driven เนื่องจากว่า event หรือเหตุการณ์จะเกิดขึ้นไม่บ่อยนัก ดังนั้นจึงประหยัดเวลาในการคำนวณมากขึ้น ทำให้มีประสิทธิภาพสูงกว่าในการหาค่าการทำงานทางตรรก และการกำหนดค่าให้ในขณะที่เหตุการณ์หนึ่งไปกระตุ้นให้กับอีกเหตุการณ์หนึ่ง ขบวนการเลือกค่าเวลาของวงล้อเวลา มีผลต่อประสิทธิภาพที่จะเกิดขึ้น ใช้กำหนดว่าจะเกิดอะไรขึ้นในเวลาต่อมา โครงสร้างข้อมูลควรจะมีความยืดหยุ่นต่อการเก็บค่าเหตุการณ์เอาไว้ตามลำดับของเวลาด้วย

Evaluation : ในรูปแบบการหาค่าจะเป็นไปในลักษณะที่ตรงไปตรงมา ตัวจำลองจะมีคลังเก็บข้อมูลหน้าที่การทำงานมากมาย และแต่ละข้อมูลที่เข้ามาสู่คลังนี้จะมีค่าประจำที่คำนวณหาค่าตามตารางความจริง ซึ่งไม่เหมือนกับตารางความจริงโดยทั่วไป ข้อมูลที่เข้ามายังตารางความจริงที่จำลองเอาไว้นี้จะได้ผลลัพธ์ที่เป็นจริงตามเงื่อนไขทางด้านไฟฟ้ามากกว่า ตัวอย่างเงื่อนไข

เหล่านี้คือ 3-สเตต เอาต์พุต, สภาวะที่ไม่รู้ค่า, และค่าเวลาที่เดินไปเรื่อย ๆ ในตัวจำลองบางตัวยอมให้แต่ละโหนดแสดงเงื่อนไขต่าง ๆ ได้มากกว่า 12 เงื่อนไข มีการแสดงสภาวะมากขึ้น พร้อมทั้งยังมีความถูกต้องแม่นยำสูงกว่า ปกติแล้วยิ่งมีเงื่อนไขที่เป็นไปได้มากเท่าไร การจำลองเลียนแบบก็ต้องใช้เวลานานมากขึ้นตาม

ในภาพที่ 25 แสดงตัวอย่างเล็กน้อย (แต่ก็เพียงพอแล้ว) ของตารางความจริง สำหรับรูปแบบหน้าที่การทำงานของแนนด์ ที่มีเปลี่ยนสภาวะได้ 5 อย่าง โดยปกติตารางความจริงของเกตประเภทแนนด์นี้จะมีเพียง 4 ค่าเท่านั้น กรณีมี 2 อินพุต แต่ตารางความจริงนี้จะมีถึง 25 ค่า เนื่องจากว่าอินพุตที่เข้ามายังเกตจะสามารถเปลี่ยนแปลงได้ถึง 5 ค่าในแต่ละอินพุต ดังในภาพที่ 25 ตำแหน่งอินพุต ปกติอาจจะเป็นค่าไบนารี่ '1' หรือ '0' แต่ทว่ามันอาจจะเป็นสภาวะที่ไม่รู้ค่า 3-สเตต หรือไม่ได้มีการกำหนดเอาไว้ก็ได้ เงื่อนไขที่เกิดสภาวะที่ไม่รู้ค่านั้นมีเกิดขึ้นในสถานการณ์จริงถ้าหากว่า อินพุตของเกตนั้นมาจากตัวฟลิปฟลอปที่ไม่ทราบค่าเอาต์พุต เช่นในขณะที่เปิดไฟเลี้ยงเข้าไป ส่วนกรณีของ 3-สเตต นั้นเกิดขึ้นได้ในกรณีที่เกตนี้ได้รับการขับมาจากเอาต์พุตของเกตตัวก่อนหน้ามันซึ่งมีค่าอิมพีแดนซ์สูง ๆ การกำหนดเงื่อนไขของอินพุต สามารถทำได้ถ้าหาก 3-สเตต บัฟเฟอร์ 2 ตัว ต่างก็ขับไปยังโหนดเดียวกับด้านของโหนดที่เป็นแอนด์ และบัฟเฟอร์ตัวหนึ่งหยุดการขับของมันลง ในขณะที่อีกตัวหนึ่งยังคงทำการขับอยู่ต่อไป

เพียงแค่ 5 สภาวะสำหรับ 2 อินพุตของเกต ก็ทำให้เกิดความยุ่งยากเกิดขึ้นแล้ว แต่ทว่ายิ่งมีจำนวนของสภาวะมากเท่าไร ก็ยิ่งทำให้วงจรมีความถูกต้องแม่นยำเพิ่มขึ้นเท่านั้น แม้จะต้องใช้เวลาในการคำนวณหาค่านานมากขึ้นก็ตาม

การสร้างแบบจำลอง (Modelling) : จากแนวคิดของการจำลองเลียนแบบได้มีการใช้ Behavioral Language Models (BLM) ซึ่งมีประสิทธิภาพสูง ตัว BLM เป็นรูปแบบการคำนวณหาค่าแบบง่าย ๆ โดยทำการสร้างหรือกำเนิดผลตอบสนองเอาต์พุตที่ถูกต้องตามอินพุตที่เข้ามา ขบวนการทำงานของ BLM ถูกเขียนขึ้นมาเพื่อใช้แสดงอาการตอบสนองหรือเปลี่ยนแปลงที่ถูกต้องได้ แต่ทว่ามันปราศจากรูปแบบของเกตในทางปฏิบัติ ตัวอย่างการพิจารณาที่ง่ายสุดของ BLM คือดูที่ตัว D-ฟลิปฟลอป ดังรายการที่แสดงในภาพที่ 26 ซึ่งรายการรหัสนี้เป็นรูปแบบของ D-ฟลิปฟลอป ที่ถูกเขียนขึ้นมาจาก VHDL (VSIC Hardware Definition Language) ซึ่งได้รับการปรับปรุงแก้ไขให้ได้ผลดีในการส่งผ่านเข้าไปยังทุก ๆ ระบบการออกแบบด้วยคอมพิวเตอร์

สำหรับรายละเอียดเกี่ยวกับ VHDL ไม่ขอกล่าวถึง เพียงให้ทราบว่า มันเป็นรายการรหัสที่มีการรับรู้ของสัญญาณนาฬิกา, clear, เอาต์พุต Q และ อินพุต D ที่ออกแบบเอาไว้ และให้ดูตัวอย่างตามคำอธิบายของภาพที่ 26

ภาพที่ 27 แสดงถึงเกต 6 ตัวในฟลิปฟลอป โดยมีคุณสมบัติเหมือนกันหมด ภายในตัวจำลองคำนวณค่าในรูปแบบการทำงานตามตารางความจริงของแนนด์

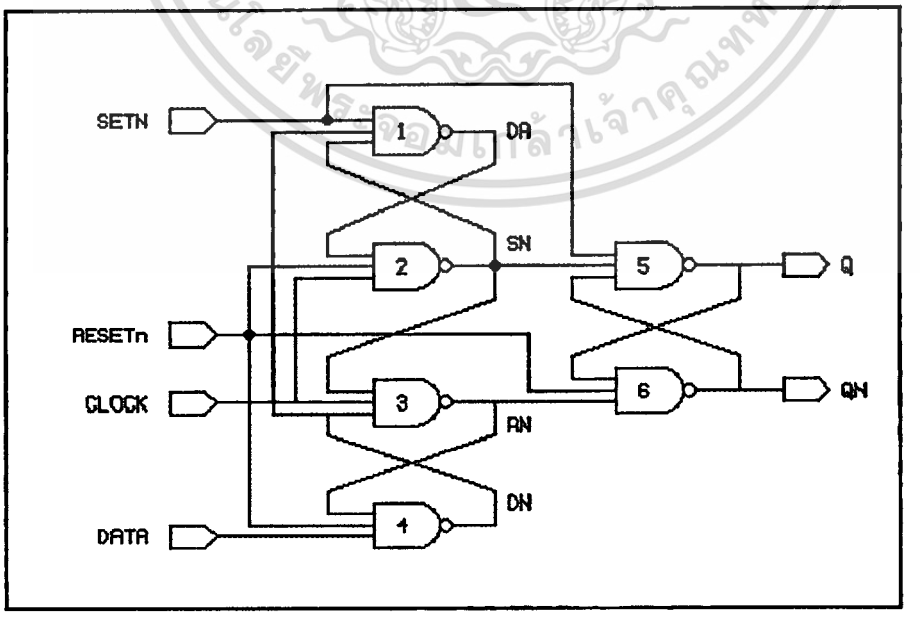
ภาพที่ 28 แสดงตัวอย่างของส่วนที่เป็น BLM สำหรับ D-ฟลิปฟลอป ที่ถูกเขียนขึ้นมาด้วยภาษาปาสกาล (PASCAL) โดยรวมความสามารถในการตรวจสอบและสร้างเงื่อนไขหรือตัวแปรต่าง ๆ เข้าไปด้วย

ภาพที่ 26

```
EDGE-TRIGGERED-D : block (CLK = '1' and not CLK 'STABLE' or CLR = '1')
begin
Q < = guarded '0' when CLR = 1 ; dse
D when CLK = '1' and not CLK'STABLE else
Q;
end block EDGE-TRIGGERED-D;
```

รูปแบบฟลิปฟลอบ VHDL

ภาพที่ 27




รูปแบบฟลิปฟลอบระดับเกต

ภาพที่ 28

```
PROCEDURE d_ff_cl;
VAR clock_val,d_val : INTERGER16;
    rec_ptr          : ^rec_t;
BEGIN
    rec_ptr := gsim_instance_ptr^.user_data_area;
    WITH qsim_instance_ptr^ : I, rec_ptr^ : r DO
    BEGIN
        clock_val := qsim_con_value[i.d_ff_i_ck^^.bits[0]];
        d_val := qsim_con_value[i.d_ff_i_dd^^.bit[0]];
        IF clock_val = qsim_unknown THEN
        BEGIN
                qsim_drive_delay_output(i.d_ff_o_q,CHR(qsim_unknown));
                qsim_drive_delay_output(i.d_ff_o_q_,CHR(qsim_unknown));
        END;
        IF (clock_val = qsim_one) AND (r.old_clock = qsim_zero) THEN
        BEGIN
                qsim_drive_delay_output(i.d_ff_o_q,CHR(d_val));
                qsim_drive_delay_output(i.d_ff_o_q_,CHR(4-d_val));
        END;
        r.old_clock := clock_val;
    end;
end;
```

ส่วนหนึ่งในรูปแบบฟลิปฟลอบ BLM

แต่ละรูปแบบวิธีการต่างก็มีทั้งข้อดีและข้อเสียรวมกัน ใน BLM จะมีการตรวจสอบภายใน สำหรับการจัดและรักษาค่าเวลาภายในเส้นทางเชื่อมต่อเพื่อวิเคราะห์การทำงานของฟลิปฟลอบ

ส่วนตัวเกตนั้นต้องการตัวช่วยหรือสัญญาณจากภายนอกเข้ามาเพื่อใช้ในสำหรับเงื่อนไขทางด้านเวลาตามที่ต้องการ ปกติแล้วการคำนวณหาค่าหรือการทำงานของ BLM จะทำได้อย่างรวดเร็ว แต่จะสามารถแสดงให้เห็นถึงการทำงานของฟลิปฟลอปภายในได้ คล้ายกับเป็นกล่องดำที่มืดมน ในส่วนของตัวเกตจะสามารถจำแนกให้อยู่ในรูปแบบการทำงานที่สมบูรณ์แบบได้นั้นต้องอาศัยเวลาของการจำลองเลียนแบบเพิ่มมากขึ้น

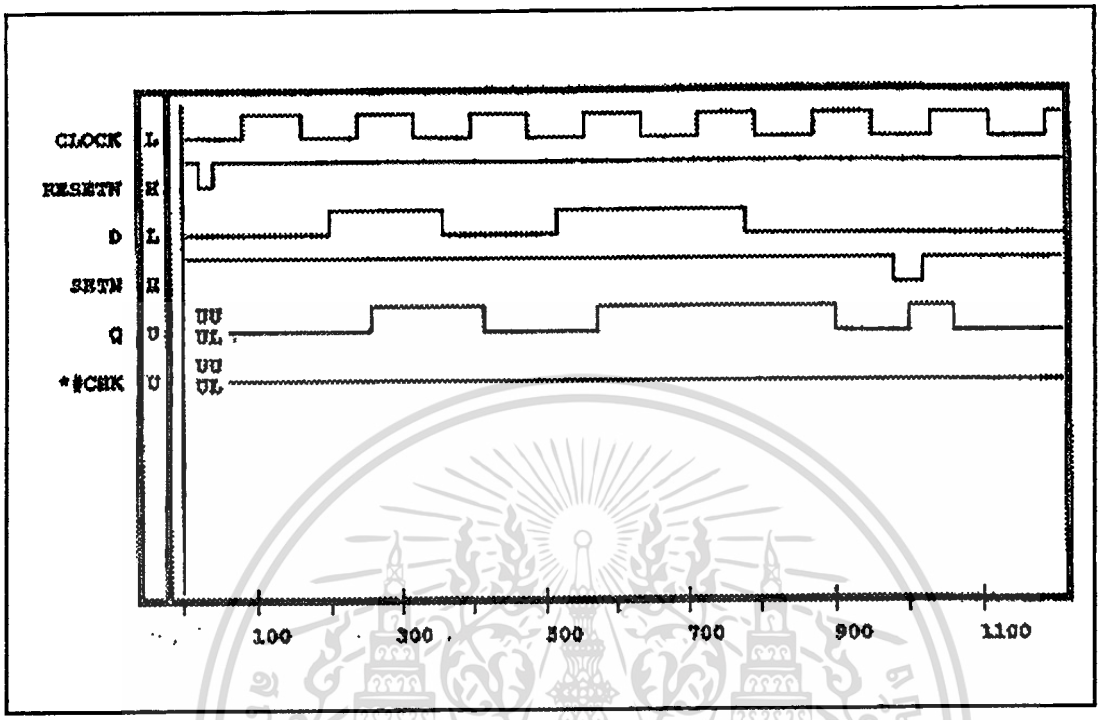
ในทางปฏิบัติ รูปแบบใน BLM จะใช้กรณีที่เป็นหน้าที่การทำงานขนาดใหญ่ ถ้าเป็นการทำงานย่อยลงมาจะใช้ลักษณะโครงสร้างของเกตในการทำงาน

Libraries และ Convenience Features : เพื่อให้การตอบสนองกับผู้ใช้งานขณะที่มีการจำลองเลียนแบบการทำงาน และการลดขนาดวงจรให้มากที่สุด กลุ่มผู้ผลิตตัว simulator จึงมีการสร้างรูปแบบจำลองต่าง ๆ รวมเก็บไว้เป็นคลังข้อมูลเป็นกลุ่ม ๆ เอาไว้เพื่อรองรับสำหรับทุก ๆ การออกแบบ โดยปกติแล้วในคลังข้อมูลจะประกอบด้วยเกตพื้นฐานจำนวนมาก, ฟลิปฟลอป, อินพุต, เอาต์พุต, transceivers และหน้าที่เพิ่มเติมอื่น ๆ

เพื่อเพิ่มความสะดวกและง่ายต่อการใช้งาน มักจะมีความสามารถในการเปลี่ยนแปลงหรือแก้ไขค่าของเวลาประวิงที่โหนดภายในด้วย เพื่อใช้สร้างหรือช่วยเหลือตามเงื่อนไขในรูปแบบต่าง ๆ ส่งผลให้ผู้ออกแบบสามารถทำงานได้อย่างมีประสิทธิภาพสูงขึ้นและสามารถพิจารณาเฉพาะลงไปในรายละเอียดภายในวงจรพร้อมทั้งแก้ไขปรับปรุง

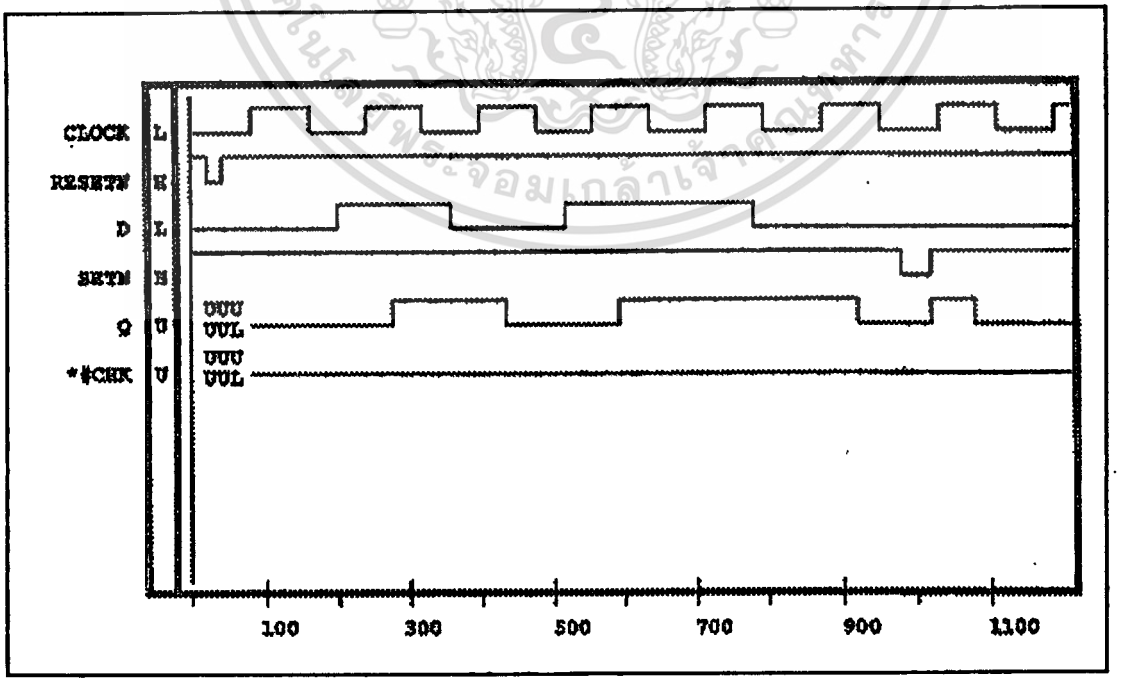
แบล็คแอนโนเทชัน (Back Annotation) : ขบวนการเปลี่ยนแปลงค่าของเวลาประวิงที่อยู่ในโหนดภายใน เรียกว่าแบล็คแอนโนเทชัน ซึ่งขบวนการนี้ทำให้เกิดความยุ่งยากเพิ่มขึ้นในการทำงาน แต่ก็เป็นส่วนสำคัญซึ่งไม่อาจมองข้ามไปได้ ในด้านเริ่มต้นทำการจำลองเลียนแบบจะทำเน็ตลีสที่ง่าย ๆ ขึ้นมาจากวงจรที่ออกแบบ ก่อนที่จะวางหรือกำหนดตำแหน่งและเส้นทางการเชื่อมต่อของการออกแบบลงไป ในขณะนี้ค่าเวลาประวิงจากตำแหน่งหนึ่งที่อยู่ใน FPGA ไปยังตำแหน่งอื่นยังไม่สามารถทราบค่าได้ แต่เมื่อได้มีการวางตำแหน่งและเส้นทางการเชื่อมต่อเสร็จแล้ว ก็จะทราบถึงค่าที่แน่นอนของเวลาประวิงที่เกิดขึ้น โปรแกรมซอฟต์แวร์จะทำการคำนวณค่าเวลาประวิงของโหนดภายใน โดยการใช้หลักเกณฑ์การทำงานของวงจรความยาวของเส้นทางที่เป็นโลหะ ค่าคงที่ของสารที่ไม่นำไฟฟ้าภายใน รวมทั้งตัวแปรอื่น ๆ ที่เกี่ยวข้อง ตามขบวนการจัดการในเรื่องหาเวลาประวิง

ภาพที่ 29




ตัวอย่างการจำลองเลียนแบบการทำงานของฟลิปฟลอบ

ภาพที่ 30




ตัวอย่างการจำลองเลียนแบบการทำงานของฟลิปฟลอบ (ต่อ)

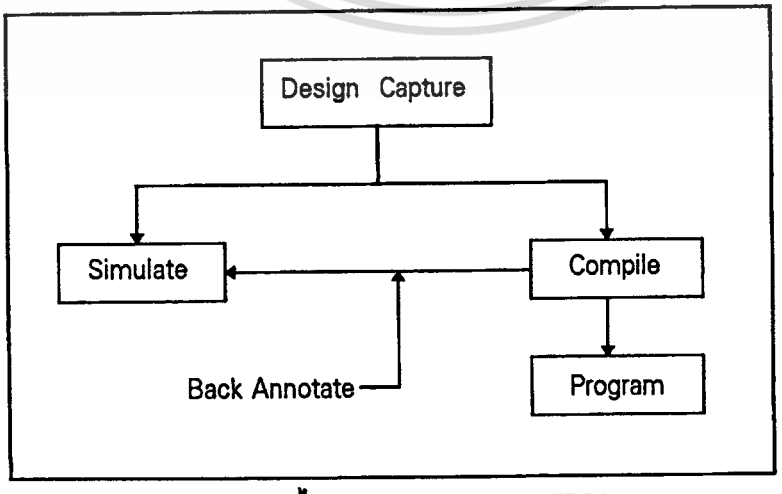
ด้วยการใช้การคำนวณค่าเวลาประวิง โปรแกรมซอฟต์แวร์จะทำการสร้างคำอธิบายประกอบเน็ตลีส ตามค่าเวลาประวิงของโหนดขึ้นมาโดยการแก้ไขจากเน็ตลีสต้นแบบ ซึ่งก็คือ แบล็คแอนโนเทชัน แล้วต่อมาการจำลองเลียนแบบก็จะใช้เน็ตลีส ที่ถูกแก้ไขนี้มาใช้ในการทำงานเพื่อเพิ่มความถูกต้องแม่นยำมากขึ้น การจำลองเลียนแบบจะสามารถแสดงให้เห็นถึงจุดที่มีปัญหาที่อยู่ในวงจรออกแบบ ซึ่งไม่สามารถสังเกตเห็นได้จากภายนอก การทำการจำลองเลียนแบบที่ใช้แบล็คแอนโนเทชันร่วมด้วย เปรียบเสมือนกับช่วยให้ผู้ออกแบบสามารถเจาะลึกเข้าไปในรายละเอียดแต่ละชั้น ๆ ของวงจรได้โดยแท้จริง

ตัวอย่างการจำลองเลียนแบบและแบล็คแอนโนเทชัน ดังในภาพที่ 29, 30 ทั้ง 2 ภาพ ซึ่งเป็นเกตแบบแนนด์ 6 ตัว ของ D-ฟลิปฟลอป ภาพแรก 29 แสดงถึงรูปแบบของแนนด์ที่มีค่าเวลาประวิง 1 nSec. ส่วนภาพต่อมา 30 แสดงถึงรูปแบบเดียวกัน แต่มีการทำแบล็คแอนโนเทชันกับเวลาประวิงร่วมด้วย จากภาพทั้ง 2 อาจไม่สามารถเห็นความแตกต่างได้ชัดเจนมากนัก และทั้ง 2 แบบอาจมีความเหมาะสมกับการใช้งานตามความต้องการ ขอให้พิจารณาความแตกต่างของช่วงเวลาระหว่างสัญญาณที่เป็น SETN กับ Q ที่เกิดขึ้น

An FPGA flow

ภาพที่ 31 แสดงถึงส่วนประกอบในขบวนการทำงานของ FPGA ขั้นตอนสำคัญก็คือส่วนของวงจรที่ออกแบบ การจำลองเลียนแบบและการแปลโปรแกรม ลำดับขั้นการทำงานโดยทั่วไปก็คือ ทำการแปลงวงจรต้นแบบ ซึ่งอาจอยู่ในรูปของสมการทางคณิตศาสตร์ให้อยู่ในรูปของเน็ตลีส

ภาพที่ 31




แสดงการไหลของการออกแบบ FPGA

แล้วส่งต่อไปยังตัวจำลอง เพื่อทำการตรวจสอบและทดสอบการทำงานทั้งหมดให้ได้ผลลัพธ์ถูกต้องตามต้องการ พร้อมทั้งทำการแปลโปรแกรมไปด้วยในเวลาเดียวกัน ในส่วนของการคอมไพล์ประกอบด้วยการวางตำแหน่ง และการเชื่อมต่อของเซลล์ เพื่อเป็นโครงสร้างของอุปกรณ์ FPGA ต่อไป ในขณะที่ทำการคอมไพล์ซอฟต์แวร์จะทำการกำจัดหรือลดค่าของเวลาประวิงที่เกิดขึ้นในวงจรพร้อมทั้งการทำแบล็คแอนโนเทชัน เมื่อได้ผลจากการจำลองเลียนแบบดีถึงขนาดที่ต้องการแล้ว ผลสุดท้ายก็จะกลายมาเป็น design file ซึ่งจะถูกโปรแกรมเข้าไปสู่อุปกรณ์ FPGA ต่อไป

ดังที่ทราบแล้วว่า วงจรตรรกที่ออกแบบขึ้นมาจะต้องได้รับการตรวจสอบดูว่าสามารถที่จะนำลงไปสู่ภายในอุปกรณ์ FPGA ได้อย่างสมบูรณ์หรือไม่ โดยทั่วไปจะแยกพิจารณาเป็นส่วนๆ ของวงจรตามจำนวนขาและค่าต่างๆตามที่กำหนด ถ้าหากว่าสามารถนำลงไปยังอุปกรณ์ FPGA ได้พอดี ที่ตำแหน่งขาของอุปกรณ์ FPGA ทั้งที่เป็นอินพุตและเอาต์พุต ก็จะสามารถตอบสนองความต้องการตามวงจรที่ออกแบบได้โดยง่าย

การออกแบบด้วยอุปกรณ์ FPGA เพียงตัวเดียว มีข้อดีคือ ง่ายต่อการใช้งานและการทดสอบ เพราะใช้ชิพเพียงตัวเดียว แต่สามารถตอบสนองการทำงานทางตรรกของวงจรที่ออกแบบได้ทั้งหมด อุปกรณ์ตระกูล FPGA ที่มีอยู่ทั่วไปต่างก็มีคุณสมบัติ หรือข้อกำหนดต่างๆแตกต่างกันออกไป สามารถเลือกใช้ได้ตามความเหมาะสม หรือความต้องการของผู้ออกแบบ และการคาดคะเนถึงสิ่งที่ต้องใช้หรือต้องมีอยู่ภายในอุปกรณ์ FPGA โดยปกติแล้วมักจะใช้ตารางเสมือนของการใช้จำนวนเซลล์ที่ระดับ MSI (Medium Scale Integration) ดังแสดงในตารางที่ 2 ซึ่งแทนหน้าที่การทำงานทางตรรกโดยทั่วไป เช่น เกต, ฟลิปฟลอป, วงจรนับ, รีจิสเตอร์เลื่อนข้อมูล และ วงจรถอดรหัส โดยมักจะต้องมีการนำมาใช้งานกันอยู่เสมอ แต่ละโรงงานผู้ผลิตจะให้ตารางที่แสดงถึงการใช้จำนวนเซลล์ เพื่อทำงานตามหน้าที่ทางตรรกต่างๆที่มีอยู่ สิ่งที่ควรคำนึงถึงเมื่อใช้ตารางนี้ คือ 1. ตารางนี้จะไม่ได้มีข้อมูลทั้งหมดโดยสมบูรณ์เกี่ยวกับหน้าที่ทางตรรก ต้องพิจารณาให้ถูกต้องว่าข้อมูลที่บอกไว้ในตารางนั้นหมายถึงอะไร เช่นถ้าบอกว่าเป็น วงจรนับแบบ 4 บิต อาจจะหมายถึงตัวนับแบบนับขึ้น หรือเป็นแบบนับลงก็ได้ บางโรงงานอาจให้ข้อมูลของตัวนับทั้งแบบนับขึ้นและนับลง แต่ทว่าบางโรงงานอาจบอกเพียงแค่แบบนับขึ้นอย่างเดียวเท่านั้น ในทำนองเดียวกันกับตัวถอดรหัส และ มัลติเพลกเซอร์ ซึ่งบางครั้งอาจจะเป็นชนิดที่มีการกลับโลจิกตรงตำแหน่งที่เป็นเอาต์พุต หรือไม่มีก็ได้ ถ้าหากสามารถทำความเข้าใจเกี่ยวกับตารางได้อย่างชัดเจน การออกแบบเพื่อให้ได้ตรงตามความต้องการเกี่ยวกับจำนวนเซลล์ที่จำเป็นต้องใช้จะกระทำได้อย่างถูกต้องแม่นยำมากขึ้น วิธีการนี้ไม่ยุ่งยากนัก แต่ต้องการความแม่นยำสูง สิ่งที่มีอยู่ในตารางอาจไม่สามารถตอบสนองตามความต้องการของผู้ออกแบบได้ทั้งหมด แต่ก็ช่วยให้การออกแบบเป็นไป

ตารางที่ 2

แสดงตัวอย่างของการใช้จำนวนเซลล์

| | DIGITAL FUNCTION | จำนวน MUX CELL |
|---|---|---|
| 1. | Logic Gates (1-3 input) | 1 cell |
| 2. | Logic Gates (4-5 input) | 2 cells |
| 3. | Logic Gates (6-7 input) | 3 cells |
| 4. | D Flip Flop | 1 cell |
| 5. | D Latch | 2 cells |
| 6. | JK Flip Flop | 2 cells |
| 7. | Decoder | |
| | 2:4 | 4 cells |
| | 3:8 | 8 cells |
| 8. | Multiplexers | |
| | 2:1 | 1 cell |
| | 4:1 | 2 cells |
| 9. | Comparator (equality) | |
| | 2 bit | 1 cell |
| | 4 bit | 2 cells |
| 10. | Registers | |
| | 4 bit | 4 cell |
| | 8 bit | 8 cells |
| 11. | Counters | |
| | 2 bit | 4 cell |
| | 4 bit | 7 cells |

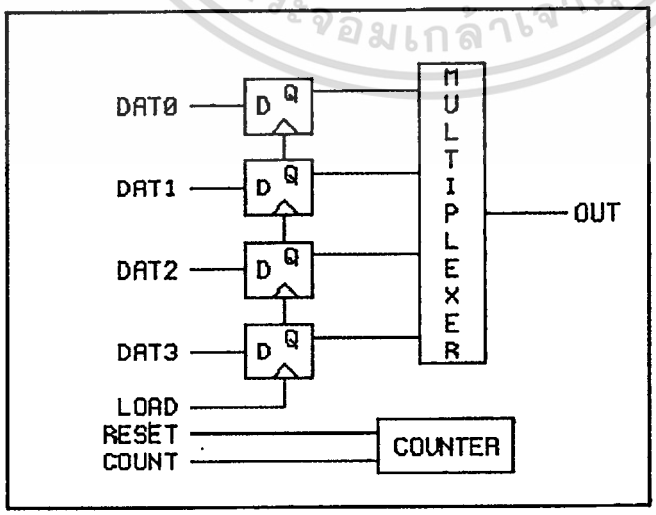
ได้อย่างรวดเร็วขึ้น ตัวอย่างเช่น หากต้องการใช้ตัววงจรนับขนาด 14 บิตแบบนับลง แต่ในตารางมีเพียงตัวนับขนาด 2 และ 4 บิตแบบนับขึ้น ผู้ออกแบบก็สามารถทำการประมาณหรือคาดหมายได้ว่าจะต้องใช้จำนวนเซลล์เท่าไร โดยการใช้ตัวนับขนาด 4 บิต 3 ชุดต่อพ่วงกัน รวมกับตัวนับขนาด 2 บิตอีก 1 ชุด ก็สามารถทำหน้าที่เป็นตัวนับขนาด 14 บิตแบบนับลงตามที่ต้องการได้โดย

การกลับ (invert) ที่จุดเอาต์พุต หรือถ้าต้องการตัวนับขนาด 13 บิต ก็กระทำได้โดยใช้ตัวนับขนาด 4 บิตต่อพ่วงกัน ร่วมกับการใช้เกตจำนวนหนึ่ง เหล่านี้เป็นต้น

ในบางกรณี การประมาณหรือวิเคราะห์จะให้ผลดีที่สุด ด้วยการใช้โปรแกรมซอฟต์แวร์ คือใช้ซอฟต์แวร์ช่วยในการวิเคราะห์วงจร และตัดเอาส่วนที่ไม่จำเป็นทิ้งไป แต่โดยทั่วไปแล้ว ผู้ออกแบบส่วนใหญ่มักนิยมทำการคำนวนหาจำนวนเซลล์ที่จำเป็นต้องใช้ด้วยตนเอง มากกว่าที่จะใช้ซอฟต์แวร์ช่วย

เมื่อทราบขนาดปริมาณของเซลล์และจำนวนขาอินพุต และเอาต์พุต การพิจารณาขั้นต่อมามักจะเป็นเรื่องเกี่ยวกับความเร็วหรือราคา ให้ได้ผลลัพธ์ที่ดีที่สุด เพราะโดยปกติแล้วในการออกแบบมักจะมีความจำเป็นต้องใช้ความจุของเซลล์มากกว่าความเป็นจริงเสมอ เนื่องจากการเกิดความแออัดยุ่งเหยิงขึ้นในการออกแบบ จนไม่สามารถกำหนดเส้นทางการเชื่อมต่อที่มีประสิทธิภาพสูงสุดตามความต้องการของผู้ออกแบบได้ ดังนั้นการศึกษาถึงส่วนประกอบของเซลล์และการเชื่อมต่อภายในของอุปกรณ์ FPGA จึงมีความจำเป็น โดยปกติแต่ละโรงงานผู้ผลิตให้การแนะนำว่าควรใช้จำนวนเซลล์ไม่เกิน 80% จากทั้งหมดในการวิเคราะห์หรือออกแบบ รูปแบบโครงสร้างขนาดใหญ่บางอย่างที่ถูกออกแบบขึ้นมา เช่น ในลักษณะของรีจิสเตอร์เลื่อนข้อมูลขนาดใหญ่ มักจะไม่ค่อยยุ่งยากซับซ้อน ดังนั้นอาจใช้ประโยชน์จากเซลล์ภายในอุปกรณ์ได้เกือบ 100% ทั้งนี้ขึ้นอยู่กับความสามารถและประสบการณ์ของผู้ออกแบบแต่ละคน

ภาพที่ 32



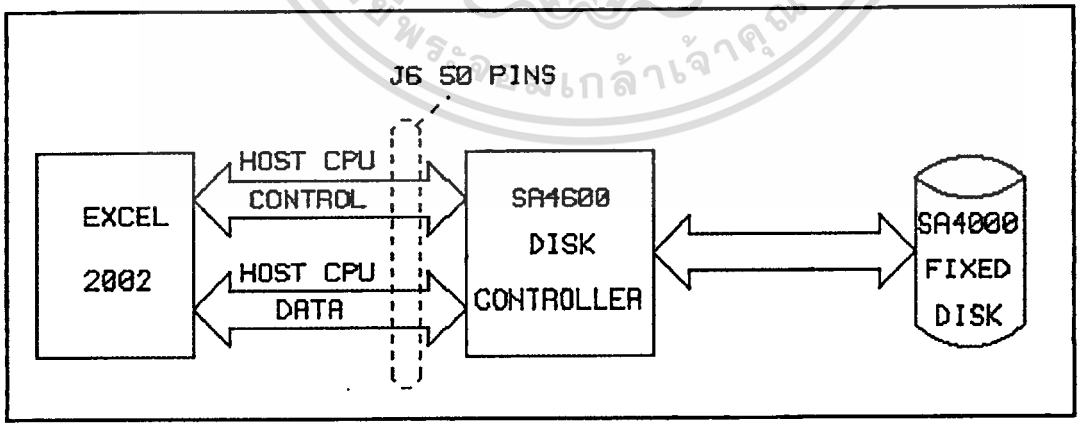

แสดงตัวอย่างรูปแบบง่ายๆ ของการออกแบบ

## การออกแบบสร้างตัวจำลอง SA4600 ในส่วนของฮาร์ดแวร์

SA 4600 เป็นตัวควบคุมฟิกดิสที่ใช้กับ ฟิกดิสของ SHUGART รุ่น SA 4000 ซึ่งเป็นฟิกดิสแบบ 8 นิ้ว ขนาด 29 เมกะไบต์, 202 ไซลินเดอร์, 32 เซกเตอร์/แทร็ก, 8 หัวเคลื่อนที่ (move heads), 512 ไบต์/เซกเตอร์ โดยใช้งานร่วมกับเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง (Computer Tomography : CT ) หรืออีกชื่อหนึ่งว่าเครื่องเอ็กซ์เรย์คอมพิวเตอร์ รุ่น EXCEL 2002 ซึ่งมีติดตั้งอยู่ 7 ศูนย์ (หรือมากกว่า) ทั่วประเทศ

เนื่องจากเครื่อง CT รุ่นนี้ได้ถูกใช้งานมาแล้วหลายปี ดังนั้นสภาพของอุปกรณ์บางส่วนเช่น ฟิกดิสที่ใช้เก็บภาพที่ได้จากการสแกนคนไข้ จึงได้มีการเสื่อมสภาพลง ส่งผลให้การเก็บข้อมูลเกิดความไม่แน่นอน เช่น เกิดข้อมูลสูญหายไปบางส่วน หรือข้อมูลสูญหายทั้งหมด ทำให้ต้องทำการสแกนคนไข้ใหม่เป็นประจำ สร้างความยุ่งยากให้กับผู้ปฏิบัติงานเป็นอย่างยิ่ง แต่เนื่องจากฟิกดิสของเดิมนั้นเป็นแบบเก่า และในปัจจุบันได้เลิกการผลิตไปแล้ว ดังนั้นแนวทางในการแก้ปัญหานี้ก็คือ การทดแทนระบบฟิกดิสรุ่นเก่านี้ ด้วยระบบฮาร์ดดิสของปัจจุบัน โดยการสร้างวงจรอินเตอร์เฟสขึ้นมาทดแทนตัวควบคุมฟิกดิส SA 4600 และ ใช้ฮาร์ดดิสทั่วไปแทนที่ฟิกดิสของ SHUGART รุ่น SA 4000 ซึ่งควบคุมการทำงานด้วยระบบคอมพิวเตอร์

ภาพที่ 33

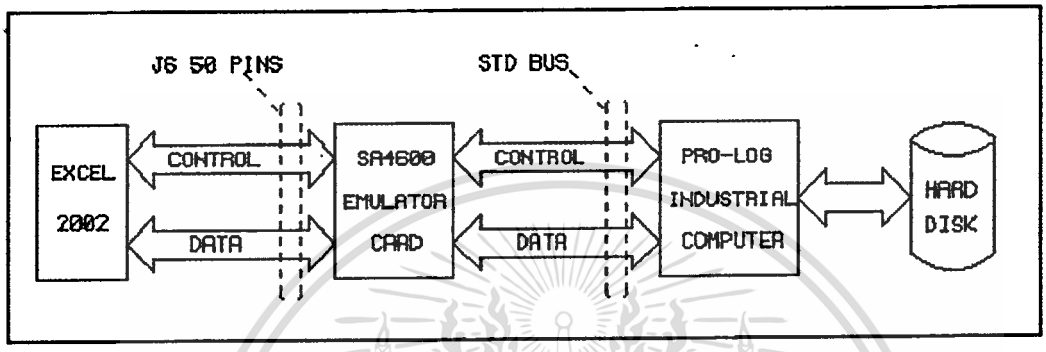



แสดงระบบการเชื่อมต่อของฟิกดิสกับเครื่อง CT แบบเดิม

## การออกแบบจริง

ทำการนำชุดตัวเลียนแบบ SA 4600 (emulator) มาต่อแทนที่ชุดควบคุม SA 4600 โดยผ่านตัวเชื่อมต่อ (connector) ขนาด 50 ขา ดังภาพที่ 34

ภาพที่ 34




แสดงระบบการเชื่อมต่อ ที่สร้างขึ้นใหม่

สิ่งที่ต้องทำการศึกษาคือ การสร้างการ์ดหรือแผงวงจรเลียนแบบ SA 4600 ที่ใช้ต่อเชื่อมระหว่างเครื่อง EXCEL 2002 และเครื่องคอมพิวเตอร์ ในการทำงานนี้ใช้เครื่องคอมพิวเตอร์อุตสาหกรรม ที่มีระบบบัสแบบมาตราฐาน (standard bus) เพราะมีขนาดเล็กและมีความน่าเชื่อถือสูงกว่าเครื่องคอมพิวเตอร์ทั่วไป

การออกแบบนี้เน้นเฉพาะการติดต่อระหว่างเครื่อง EXCEL 2002 กับการ์ดหรือแผงวงจรเลียนแบบ SA 4600 เป็นหลัก เนื่องจากส่วนการติดต่อกับคอมพิวเตอร์นั้นสามารถศึกษาได้จากคู่มือของบัสแบบมาตราฐานทั่วไป

## สัญญาณของตัวเชื่อมต่อขนาด 50 ขา จากเครื่อง CT

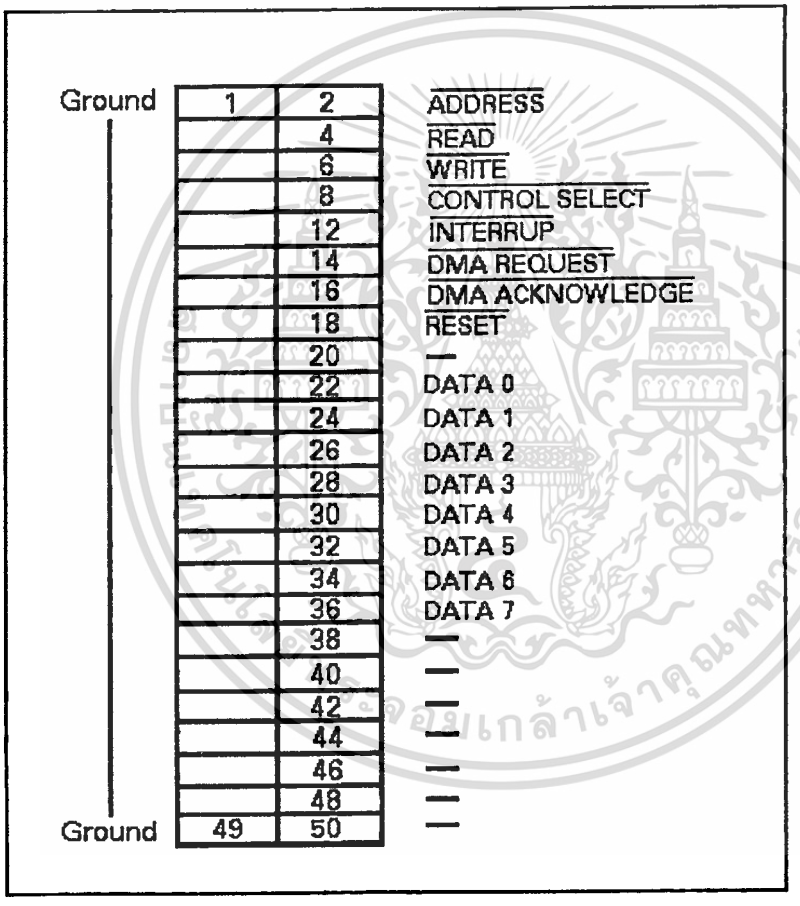
ประกอบด้วยสัญญาณต่างๆ ดังนี้

1. $\overline{\text{RESET}}$ (อินพุต) สั่งรีเซ็ต รีจิสเตอร์ต่างๆของ SA 4600
2. $\overline{\text{READ}}$ (อินพุต) เครื่อง CT ขออ่านผลลัพธ์ในรีจิสเตอร์ของ SA 4600
3. $\overline{\text{WRITE}}$ (อินพุต) เครื่อง CT ขอเขียนคำสั่งลงในรีจิสเตอร์ของ SA 4600
4. $\overline{\text{CONTROL SELECT}}$ (อินพุต) สัญญาณเลือกติดต่อจาก CT

5. $\overline{\text{INTERRUPT}}$ (เอาต์พุต) สัญญาณบอกเครื่อง CT ว่าได้ทำตามคำสั่งเรียบร้อยแล้ว และพร้อมให้ CT อ่านผลลัพธ์ไปตรวจสอบว่าสำเร็จหรือไม่

6. $\overline{\text{DMA REQUEST}}$ (เอาต์พุต) เมื่อ SA 4600 ได้รับคำสั่งเกี่ยวกับการโอนถ่าย (transfer) ข้อมูล จะทำการส่งสัญญาณตัวนี้ไปบอก CT ให้ทำการรับ-ส่งข้อมูลทุกๆไบต์ จนครบ

ภาพที่ 35




สัญญาณของตัวเชื่อมต่อขนาด 50 ขา

7. $\overline{\text{DMA ACKNOWLEDGE}}$ (อินพุต) เครื่อง CT ใช้สัญญาณนี้ในการตอบรับ กับ สัญญาณ $\overline{\text{DMA REQUEST}}$ เพื่อทำการรับ - ส่งข้อมูล 1 ไบต์จากการ์ด SA 4600

8. $\overline{\text{ADDRESS}}$ (อินพุต) ใช้คู่กับสัญญาณ $\overline{\text{READ}}$ และ $\overline{\text{WRITE}}$ เพื่อทำการอ่าน - เขียนข้อมูลกับรีจิสเตอร์ ของ SA 4600 ตามตารางดังนี้

ตารางที่ 3

แมปของอินเตอร์เฟสรีจิสเตอร์

| ADDRESS | READ REGISTER | WRITE REGISTER |
|---|---|---|
| 1 | STATUS | COMMAND |
| 0 | RESULT | PARAMETER |

9. Data bus 0-7 เป็นแบบ 2-ทิศทาง 3-สเตต

## อินเตอร์เฟสรีจิสเตอร์

เนื่องจากการติดต่อระหว่างเครื่อง CT และตัวควบคุมฟิกดิส จะติดต่อผ่านรีจิสเตอร์ขนาด 8 บิต จำนวน 4 ตัว ตามตารางที่ 3 ดังนั้น จึงต้องศึกษาความหมายของรีจิสเตอร์ต่างๆ กัน ดังนี้

1. รีจิสเตอร์คำสั่ง (command register) (8 บิต เขียนอย่างเดียว) : เป็นตัวรับคำสั่งจาก CT ซึ่งรีจิสเตอร์นี้จะไม่มีการเขียนทับจนกว่า CT จะมาอ่านผลลัพธ์ของคำสั่งก่อนหน้าผ่านทางรีจิสเตอร์ผลลัพธ์

2. พารามิเตอร์รีจิสเตอร์ (parameter register) (8 บิต เขียนอย่างเดียว) : ใช้ในการรับค่าพารามิเตอร์ของคำสั่งต่าง ๆ จาก CT ซึ่งชุด SA4600 จะยังไม่ทำงานจนกว่าจะรับค่าพารามิเตอร์ต่าง ๆ ของคำสั่งจนครบ

3. รีจิสเตอร์ผลลัพธ์ (result register) : ใช้เก็บผลลัพธ์การปฏิบัติงานตามคำสั่งของ CT ซึ่ง SA4600 จะส่งสัญญาณ $\overline{\text{INTERRUPT}}$ ไปบอกให้ CT มาอ่านผลลัพธ์นี้ไป

4. รีจิสเตอร์สถานะ (status register) : ใช้ในการควบคุมการรับ-ส่งคำสั่ง, พารามิเตอร์ และผลลัพธ์ของคำสั่งระหว่าง CT และตัวควบคุมดิส (แต่จะไม่มีผลต่อการรับ-ส่งข้อมูล) โดยประกอบด้วยแฟลก ต่าง ๆ ดังนี้

4.1 Command busy (บิต0) : ใช้แสดงว่า SA4600 กำลังปฏิบัติงานอยู่และไม่ว่าง

เซ็ต เมื่อ CT สั่งเขียนคำสั่ง

รีเซ็ต เมื่อ CT อ่านผลลัพธ์ของการปฏิบัติงาน

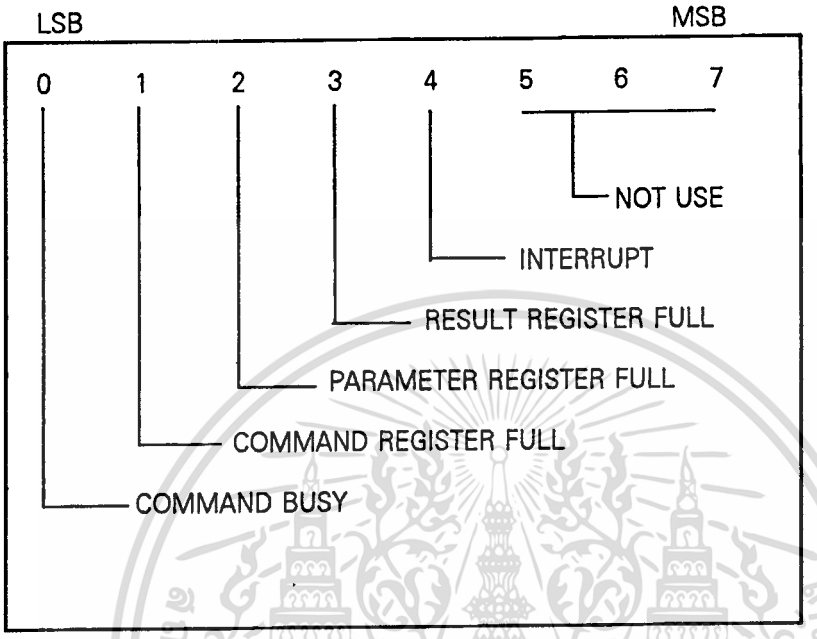
4.2 Command reg. full (บิต1) : แสดงสภาวะ รีจิสเตอร์คำสั่ง ของ SA4600

เซ็ต เมื่อ CT สั่งเขียนคำสั่ง

รีเซ็ต เมื่อ SA4600 อ่านคำสั่งจากรีจิสเตอร์คำสั่ง

ตารางที่ 4

แมปของรีจิสเตอร์สถานะ




4.3 Parameter reg. full (บิต2) : แสดงสภาวะพารามิเตอร์รีจิสเตอร์ของ SA4600

เซ็ต เมื่อ CT สั่งเขียนพารามิเตอร์

รีเซ็ต เมื่อ SA4600 อ่านพารามิเตอร์จากพารามิเตอร์รีจิสเตอร์ และ CT สามารถที่จะเขียนพารามิเตอร์ตัวต่อไป

4.4 Result reg. full (บิต3) : เป็นการแสดงว่า SA4600 ได้ทำงานตามคำสั่ง และเก็บผลลัพธ์ไว้ใน รีจิสเตอร์ผลลัพธ์ เรียบร้อยแล้ว

เซ็ต เมื่อ SA4600 เก็บผลลัพธ์ของคำสั่งลงใน รีจิสเตอร์ผลลัพธ์

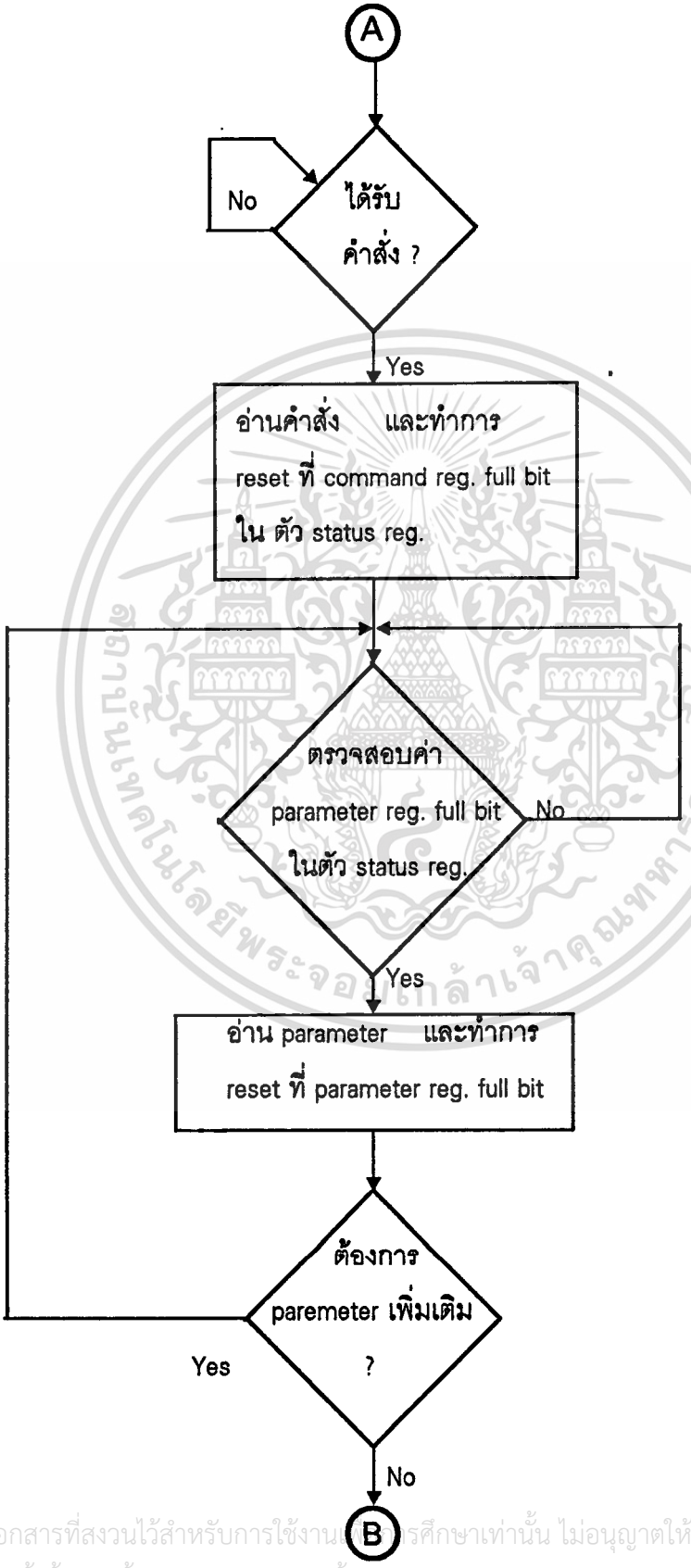
รีเซ็ต เมื่อ CT มาอ่านผลลัพธ์ใน รีจิสเตอร์ผลลัพธ์ ไป

4.5 Interrupt (บิต4) : SA4600 จะทำการเซ็ตบิตนี้เมื่อต้องการติดต่อกับ CT

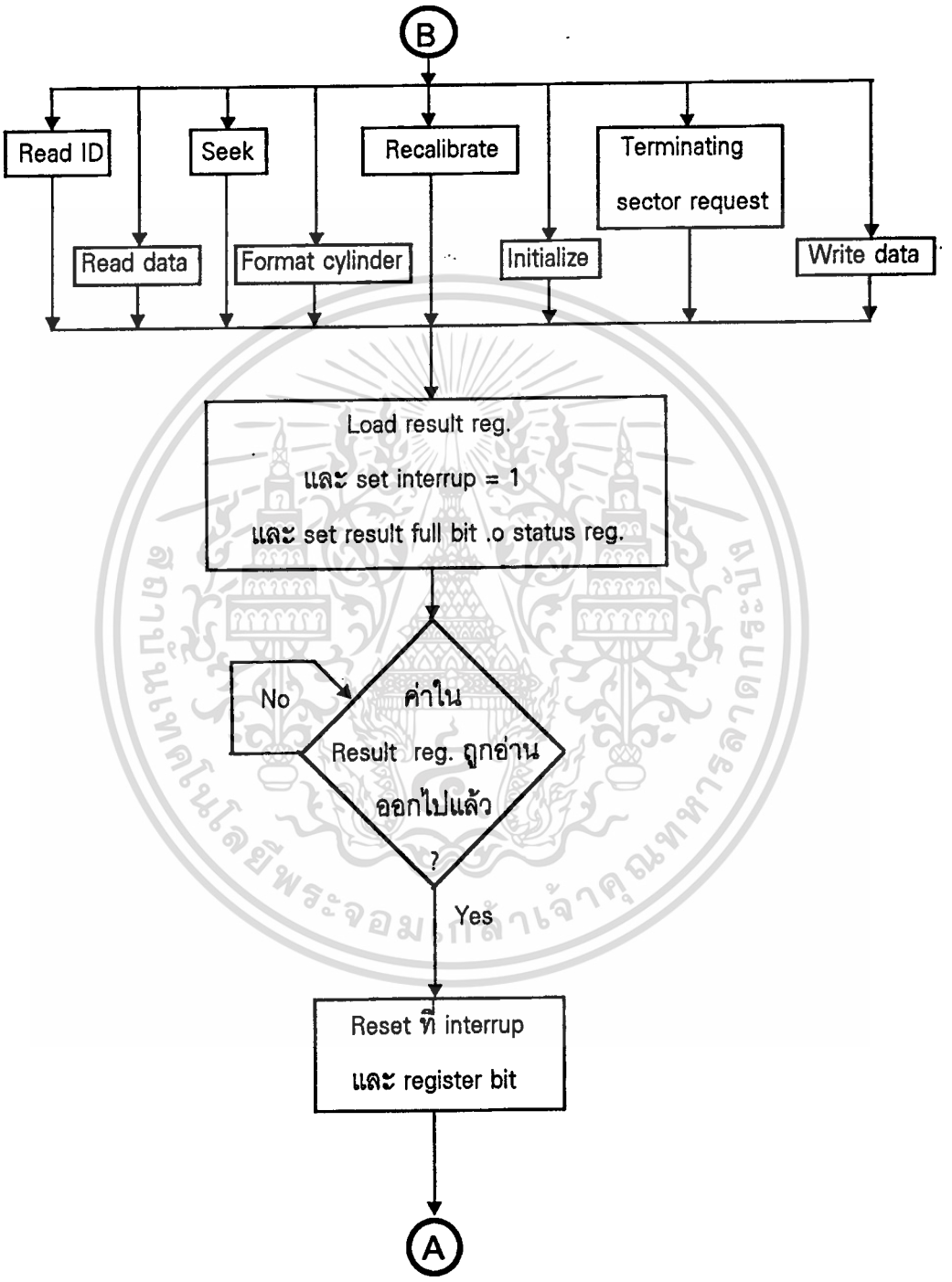
เซ็ต เมื่อ SA4600 ส่งสัญญาณขัดจังหวะไปบอกให้ CT มาอ่านผลลัพธ์ของการทำงานในรีจิสเตอร์ผลลัพธ์

รีเซ็ต เมื่อ CT มาอ่านผลลัพธ์ในรีจิสเตอร์ผลลัพธ์

ภาพที่ 36 : สายงานแสดงการรับ-ส่งข้อมูลของ SA4600



ภาพที่ 36 : สายงานแสดงการรับ-ส่งข้อมูลของ SA4600 (ต่อ)




ในการติดต่อกับเครื่อง CT เราสามารถแยกออกเป็น 2 ส่วนใหญ่ ๆ คือ

1. การรับ คำสั่ง/พารามิเตอร์ และการส่งผลลัพธ์ผ่านทางรีจิสเตอร์ ซึ่งควบคุมโดยรีจิสเตอร์สถานะ

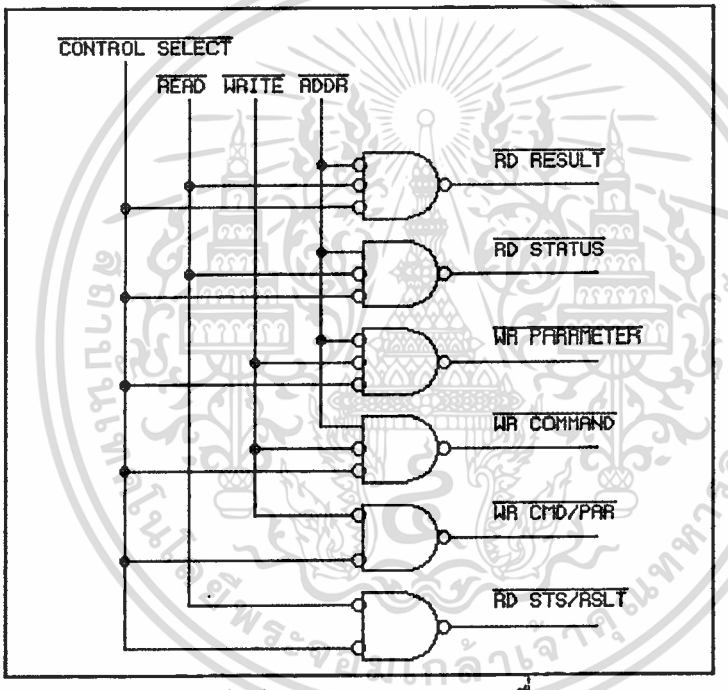
2. การถ่ายโอนข้อมูล ผ่านทาง DMA

ในการออกแบบ จะทำการแยกออกแบบเป็นส่วน ๆ ดังนี้

## วงจรส่วนแยกสัญญาณควบคุมของ CT

จากนิยามของสัญญาณต่าง ๆ ของตัวเชื่อมต่อขนาด 50 ขา สามารถออกแบบวงจรเพื่อแยกสัญญาณตามตารางที่ 3 และสัญญาณแสดงการ อ่าน/เขียน ได้ดังนี้

ภาพที่ 37




การถอดรหัสสัญญาณควบคุมจากเครื่อง CT

## การถอดรหัสสัญญาณควบคุมจากเครื่องคอมพิวเตอร์

ในการติดต่อระหว่างเครื่องคอมพิวเตอร์และการ์ดเลียนแบบ SA4600 ผ่านทางบัสมาตรฐานนั้น เราจะใช้แนวคิดของการติดต่อ อินพุต/เอาต์พุต เพื่อสร้างสัญญาณในการอ่านเขียนกับรีจิสเตอร์ ต่าง ๆ ใน SA4600 ดังนี้

I/O Read port # 1C0H = $\overline{\text{PCRD CMD/PAR}}$

I/O Read port # 1C2H = $\overline{\text{PCRD STATUS}}$

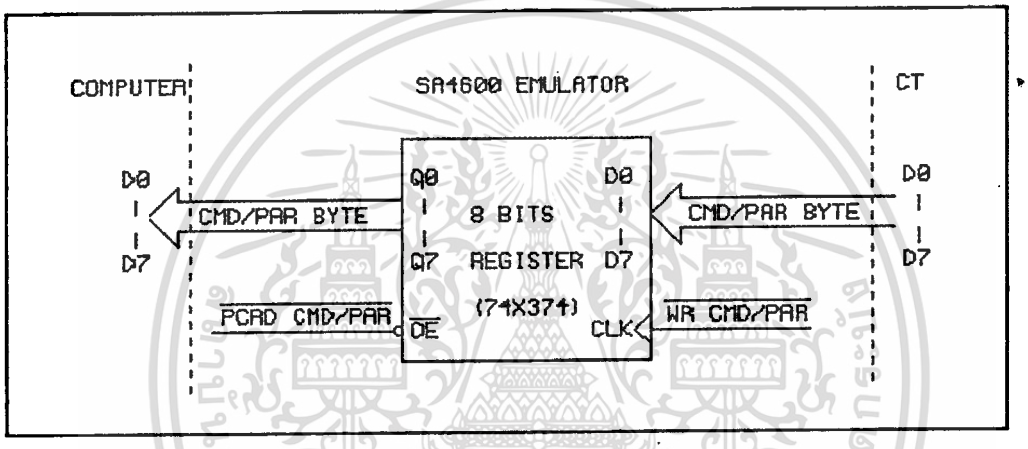
I/O Write port # 1C1H = $\overline{\text{PCWR RESULT}}$

I/O Write port # 1C3H = $\overline{\text{PCWR STATUS}}$

## วงจรของ คำสั่ง/พารามิเตอร์ รีจิสเตอร์

เนื่องจากในขณะทำงานจริง เครื่อง CT จะทำการเขียนคำสั่งและพารามิเตอร์ ไม่พร้อมกัน (CT จะเขียนพารามิเตอร์ตัวแรกก็ต่อเมื่อ SA4600 ได้อ่านคำสั่งไปเรียบร้อยแล้ว ซึ่งดูได้จาก command reg. full bit ของรีจิสเตอร์สถานะ) ดังนั้น เราจึงสามารถใช้ รีจิสเตอร์ ขนาด 8 บิต เพียง 1 ตัว มาทำหน้าที่เป็นทั้ง คำสั่งและพารามิเตอร์รีจิสเตอร์

ภาพที่ 38

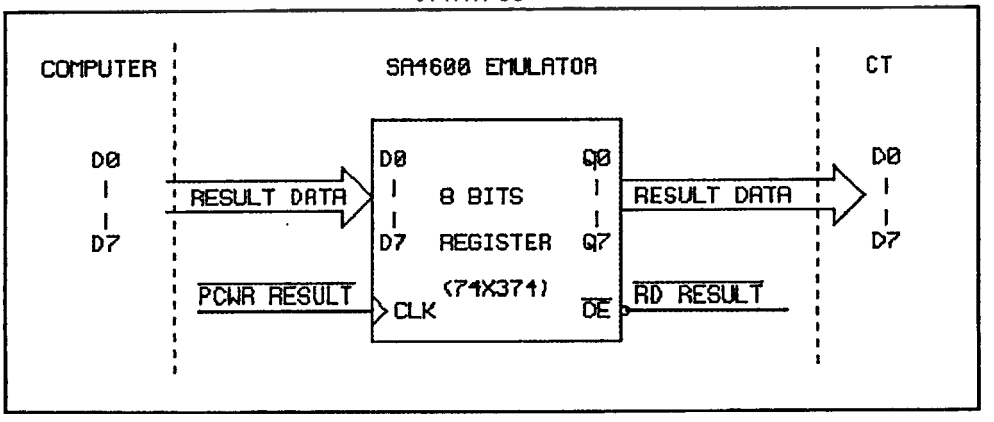



วงจร คำสั่ง / พารามิเตอร์ รีจิสเตอร์

## วงจรของ รีจิสเตอร์ผลลัพธ์

ใช้หลักการเดียวกับวงจรคำสั่ง/พารามิเตอร์ รีจิสเตอร์ เพียงแต่กลับทิศทางการ อ่าน-เขียน

ภาพที่ 39




วงจร รีจิสเตอร์ผลลัพธ์

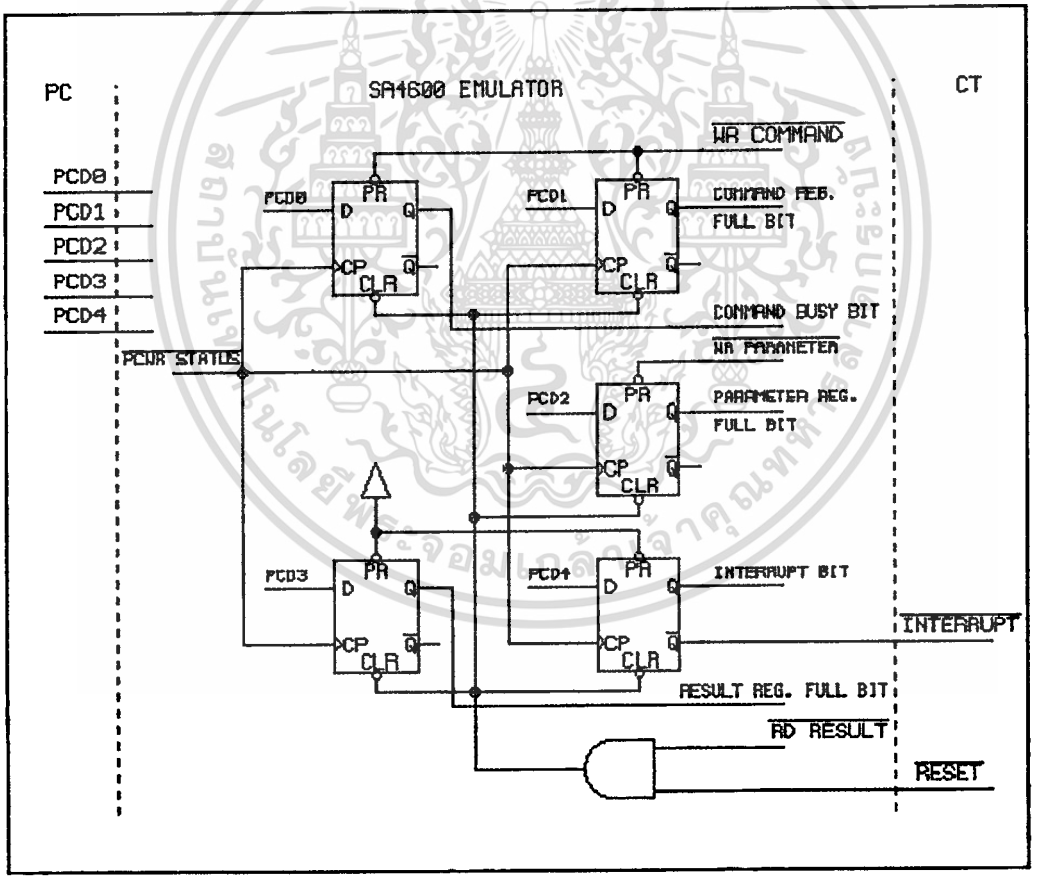
## วงจรส่วน รีจิสเตอร์สถานะ

เนื่องจาก รีจิสเตอร์สถานะนี้ใช้เป็นอาณัติสัญญาณในการ อ่าน-เขียน คำสั่ง/พารามิเตอร์ และ รีจิสเตอร์ผลลัพธ์ ซึ่งจะต้องมีคุณสมบัติ ดังนี้

- สามารถ เซ็ต สถานะต่าง ๆ ได้อย่างรวดเร็ว
- สามารถสั่ง เซ็ต ได้ทั้งจากเครื่องคอมพิวเตอร์และ CT
- สามารถอ่าน สถานะ ได้ทั้งจากเครื่องคอมพิวเตอร์และ CT

ดังนั้น ในการออกแบบจึงต้องให้สามารถ เซ็ต และเคลียร์แต่ละบิต ได้ทั้งแบบ ฮาร์ดแวร์ และซอฟต์แวร์ (ผ่านทางพอร์ต อินพุต/เอาต์พุต เหมือนกับรีจิสเตอร์ผลลัพธ์) ดังนี้

ภาพที่ 40




วงจร รีจิสเตอร์สถานะ

จากวงจรข้างบนจะเห็นว่า

- คอมพิวเตอร์ สามารถเซ็ต และเคลียร์บิตต่าง ๆ ของรีจิสเตอร์สถานะ ผ่านทางสัญญาณ PCWR STATUS (I/O port # 1C3H) (เช่น เคลียร์บิตให้เป็น 0 ตอนเริ่มต้น)

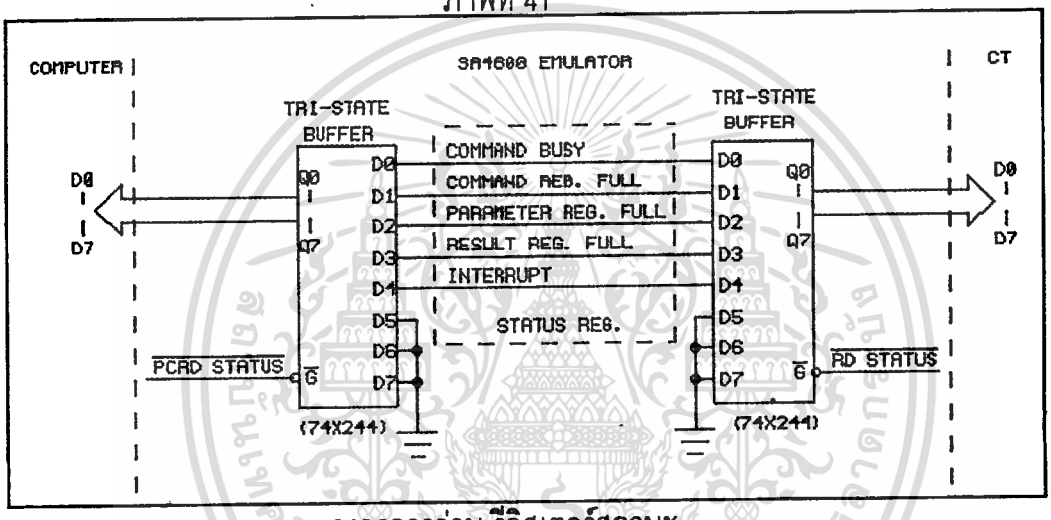
- มีการสร้างสัญญาณ $\overline{\text{INTERRUPT}}$ ไปบอกให้ CT มาอ่านรีจิสเตอร์ผลลัพธ์ โดยอัตโนมัติ เมื่อเราสั่ง เซ็ต บิตขัดจังหวะด้วยคอมพิวเตอร์

- Command busy, command reg. full และ parameter reg. full จะถูกเซ็ตโดยอัตโนมัติ เมื่อ CT สั่งเขียน คำสั่ง/พารามิเตอร์

- บิตต่าง ๆ จะถูกเคลียร์ เมื่อ CT อ่านรีจิสเตอร์ผลลัพธ์ หรือสั่งรีเซ็ต

ในส่วนของการ อ่านสถานะ เราจะได้

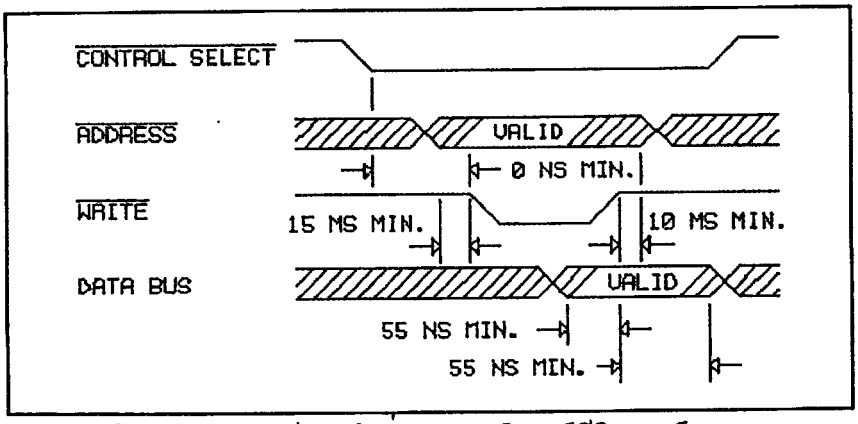
ภาพที่ 41




วงจรการอ่าน รีจิสเตอร์สถานะ

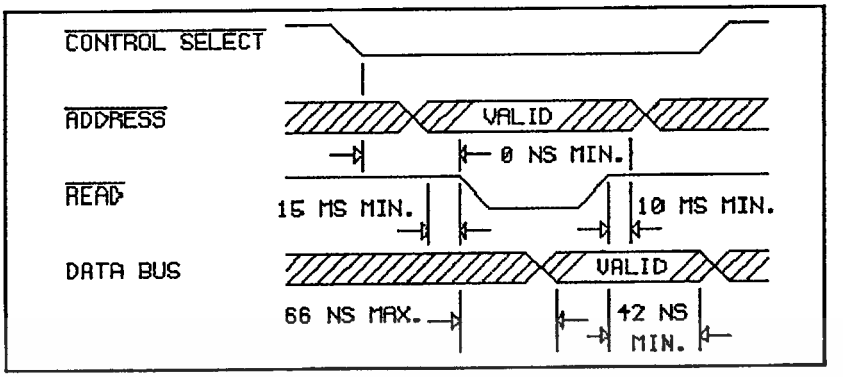
จากส่วนของการอ่านรีจิสเตอร์สถานะ เราจะเห็นว่าทั้งคอมพิวเตอร์ และ CT สามารถอ่านค่าในรีจิสเตอร์สถานะ ได้พร้อม ๆ กัน โดยที่ไม่มีปัญหาเรื่องสัญญาณชนกัน

ภาพที่ 42




ข้อกำหนดการเขียน คำสั่ง / พารามิเตอร์ รีจิสเตอร์

ภาพที่ 43




ข้อกำหนดการอ่าน ผลลัพธ์/สถานะ รีจิสเตอร์

**หลักการออกแบบของโปรแกรมบนคอมพิวเตอร์ ในการอ่าน-เขียนคำสั่งกับ CT**

```
Initial all status register bit
while true  // endless loop
Begin
        If command reg full bit setted  //  I/O read  #  1C2
        Begin
            Read command via command/parameter register;  //  I/O read part # 1C0
            Clear command register full bit;  //  remains set command busy bit (I/O WR # 1C3)
            If need any parameter
            Begin
                While need parameter
                Begin
                    If parameter full bit setted; //  I/O read # 1C2
                    Begin
                        Read 1 parameter via command/parameter register;
                        Clear parameter full bit;
                    End;
```

```
            End;
        End;
        Process corresponding command;
        Set result into result register; // I/O WR # 1C1
        Set result register full & interrupt bit in status register; // I/O write # 1C3
        While result register bit still set  // wait until CT read result register
        Begin
            Wait;                    // result register
        End:
    End:
End;
```

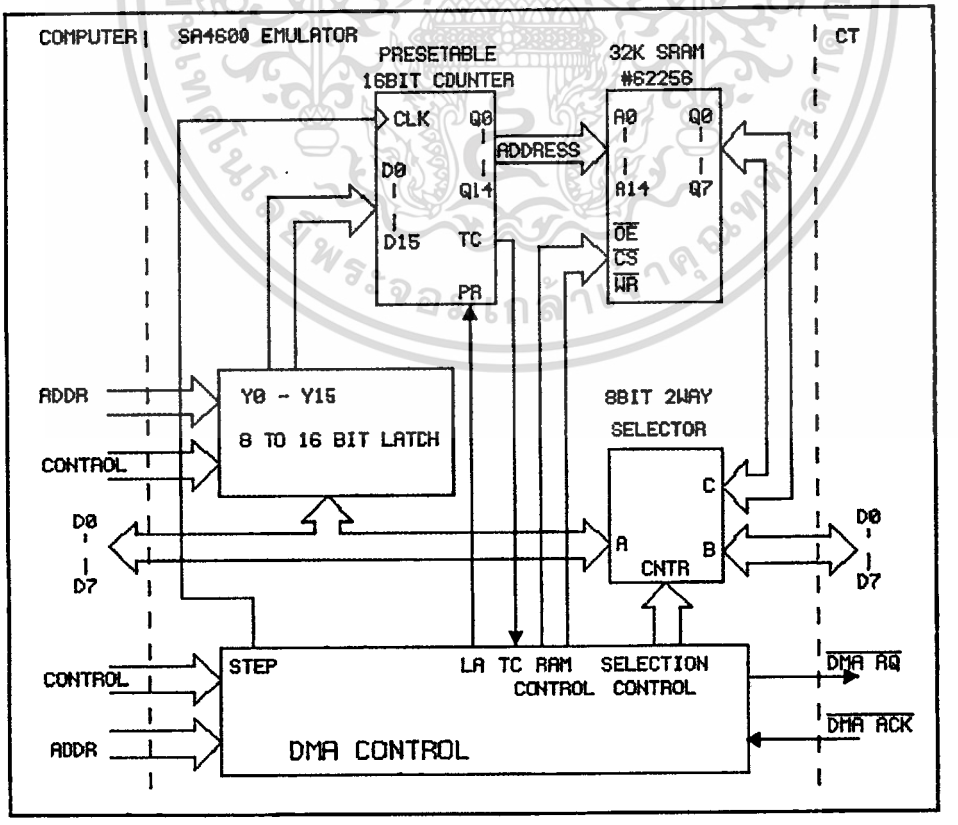
## วงจรถ่ายโอนข้อมูล

วงจรถ่ายโอนข้อมูลด้วยขบวนการ DMA ถือว่าเป็นหัวใจสำคัญของระบบทั้งหมด เนื่องจากจะเป็นตัวกำหนดประสิทธิภาพ และความน่าเชื่อถือของระบบโดยรวมและเนื่องจากอัตราเร็วของการถ่ายโอน มีค่าสูงกว่า 1 เมกะไบต์/วินาที ดังนั้นเราจึงต้องทำการถ่ายโอนข้อมูลกับเครื่อง CT โดยอาศัยฮาร์ดแวร์เข้าช่วย ซึ่งวงจรในส่วนนี้จะต้อง

- สามารถถ่ายโอนได้ตั้งแต่ 512 ไบต์ - 32 กิโลไบต์ ต่อรอบ (ในกรณีที่ต้องการถ่ายโอนข้อมูลมากกว่า 32 กิโลไบต์ จะแบ่งการถ่ายโอนออกเป็นหลาย ๆ รอบ)
- สามารถอ่าน-เขียนข้อมูล ได้ทั้งกับเครื่องคอมพิวเตอร์และ CT
- สามารถนับจำนวนไบต์ในการถ่ายโอน และกำหนดทิศทางการอ่าน-เขียนได้ เนื่องจากเครื่อง CT จะไม่มีสัญญาณเสดงการ อ่าน/เขียน และ terminate
- มีอัตราเร็วของการถ่ายโอน สูงกว่า 1 เมกะไบต์/วินาที

จากคุณสมบัติดังกล่าว ได้เลือกออกแบบโดยใช้ แรมแบบสถิต 2 ทาง ขนาด 32 กิโลไบต์ เป็นตัวกลางในการรับ-ส่งข้อมูลระหว่างเครื่อง CT และคอมพิวเตอร์ ดังภาพ

ภาพที่ 44

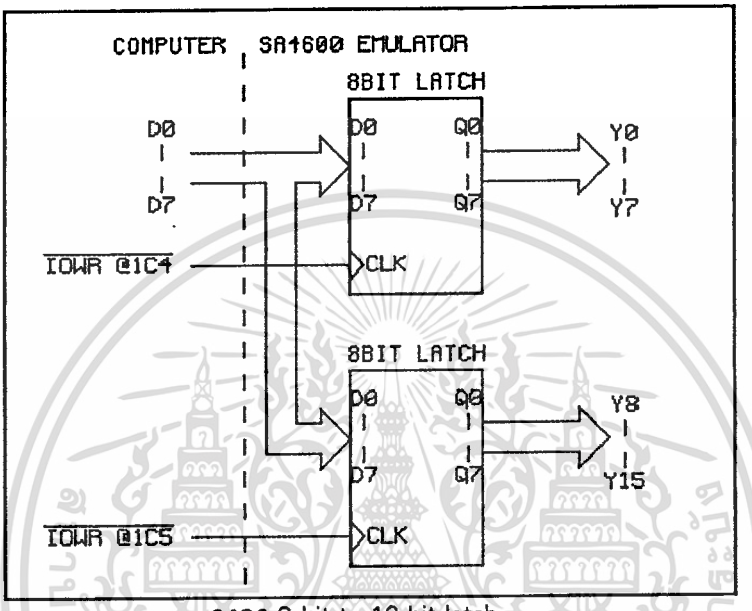



แสดงแผนภาพบล็อก ของวงจร DMA

จากแผนภาพบล็อก ของวงจร DMA จะแยกกล่าวถึงแต่ละบล็อก ได้ดังนี้

**8 bit to 16 bit lactch ;** ใช้ในการสร้างค่าเริ่มต้น (preset value) ให้กับวงจรนับ 16 บิต

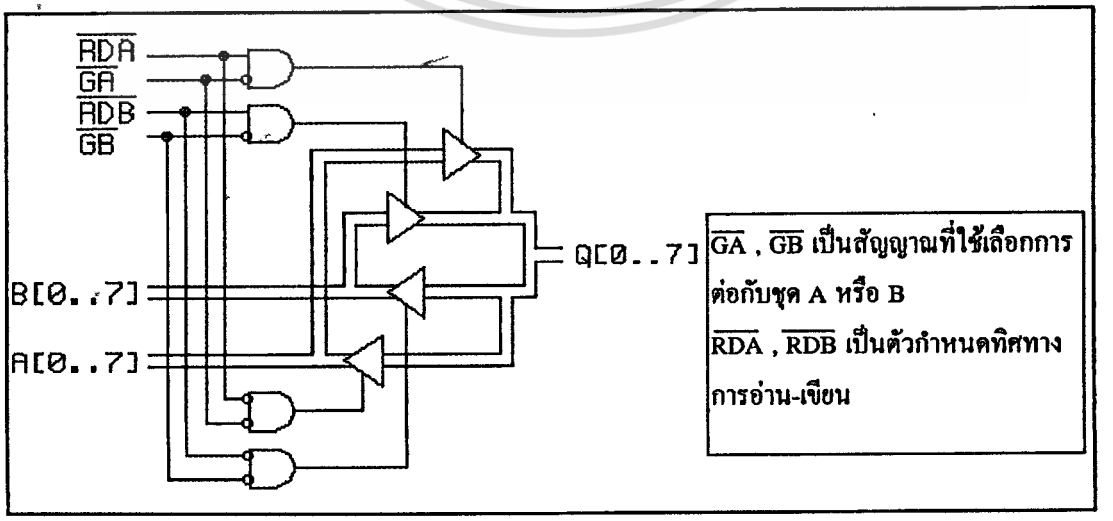
ภาพที่ 45




วงจร 8 bit to 16 bit latch

**8 bit 2 way selector :** เป็นตัวเลือกที่ใช้เลือกว่าจะให้เป็นการติดต่อระหว่างแรมแบบสถิต กับคอมพิวเตอร์หรือ CT และเป็นขบวนการอ่านหรือเขียน

ภาพที่ 46




วงจร 8 bits 2 way Selector

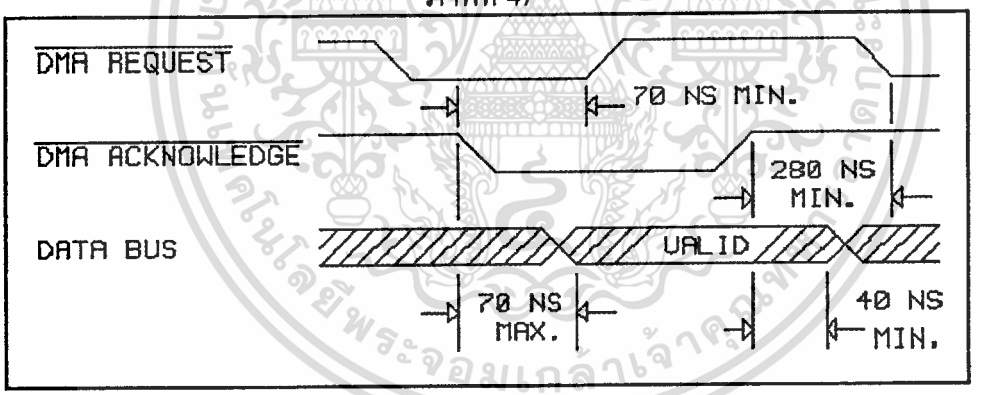
การที่เราออกแบบสัญญาณควบคุมเป็น 4 เส้น แทนที่จะใช้เพียง 2 เส้น (A/B และ R/W) เพราะว่าสัญญาณที่เราจะนำมาควบคุมชุดตัวเลือกนี้ มีที่มาจากสัญญาณ DMA IN/OUT ของ CT และสัญญาณ อ่าน/เขียน แรมจากคอมพิวเตอร์ที่มี 4 เส้น อยู่แล้ว

## วงจรควบคุม DMA

วงจรควบคุม DMA เป็นส่วนที่ควบคุมจังหวะการ DMA ระหว่าง CT กับแรมแบบสถิต กำหนดทิศทางการส่งถ่ายข้อมูล รวมทั้งนำข้อมูลเข้า-ออกระหว่างคอมพิวเตอร์กับแรมแบบสถิต

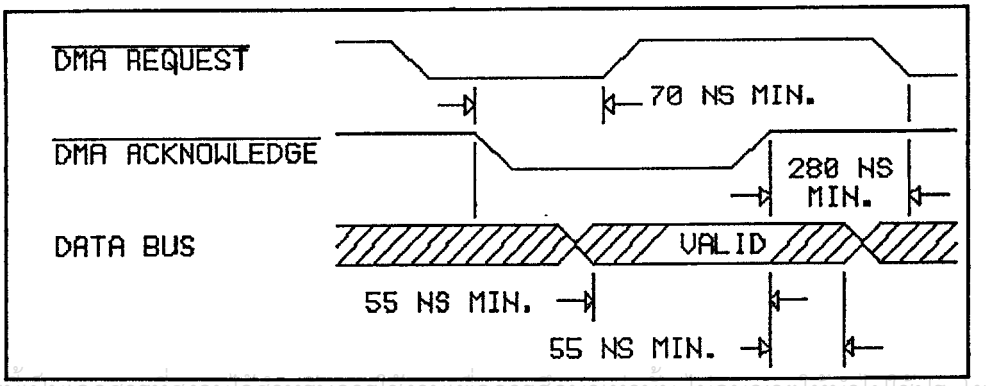
เนื่องจากสัญญาณ DMA ของเครื่อง CT ไม่มีการบอกทิศทางว่าเป็นการ DMA IN หรือ DMA OUT จึงต้องรู้ทิศทางเองจากคำสั่งที่ได้รับจาก CT ว่าขณะนี้ CT ต้องการเขียนหรืออ่านข้อมูลจากฟิกดิส และเมื่อเครื่อง SA4600 พร้อมที่จะรับ-ส่งข้อมูลกับ CT ก็จะทำการ active สัญญาณ DMA REQUEST เพื่อเริ่มขบวนการ DMA ซึ่งมีจังหวะเวลา (timing) ดังนี้

ภาพที่ 47




ข้อกำหนดการ DMA OUT

ภาพที่ 48




ข้อกำหนดการ DMA IN

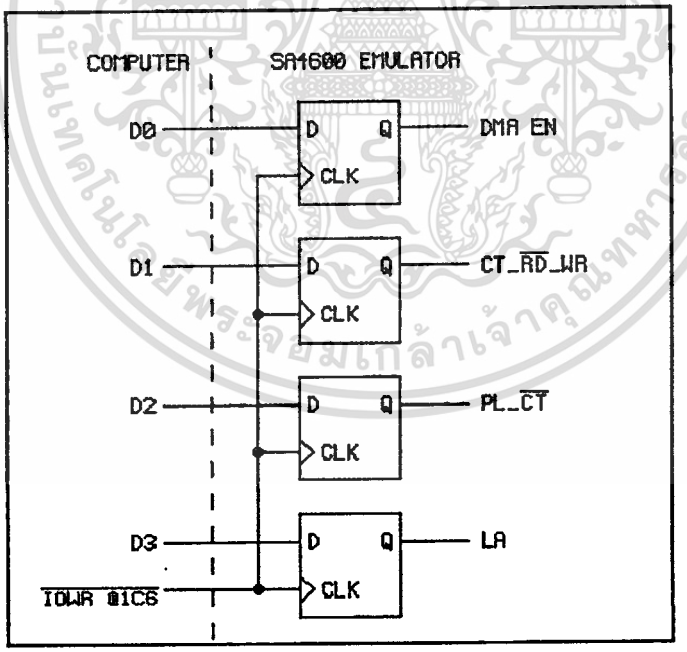
เพื่อให้การออกแบบเป็นไปได้ง่ายขึ้น จะสร้างสัญญาณที่เป็นตัวกำหนดการทิศทางการรับ-ส่งข้อมูลและเริ่มต้นการทำงาน ดังนี้

- สัญญาณ DMA_EN ใช้เริ่มต้นขบวนการ DMA ระหว่าง CT และแรมแบบสถิต
- สัญญาณ PL_$\overline{CT}$ เป็นตัวกำหนดว่าให้แรมแบบสถิต ทำกับรับ-ส่งข้อมูลกับเครื่อง CT หรือเครื่องคอมพิวเตอร์
- สัญญาณ CT_$\overline{RD}$_WR ใช้เป็นตัวบอกว่าจะทำการ DMA IN หรือ DMA OUT (ในกรณีที่ติดต่อกับคอมพิวเตอร์สามารถใช้สัญญาณ อ่าน/เขียน ของคอมพิวเตอร์มาเป็นตัวกำหนดทิศทางได้)

และเนื่องจากส่วนของการควบคุม DMA นี้ ยังเป็นตัวส่งสัญญาณไปตั้งค่าเริ่มต้นของวงจรนับ 16 บิต (หรือตำแหน่งของแรมแบบสถิต) และให้ชื่อสัญญาณนี้ว่า latch address (LA)

ในการออกแบบใช้ I/O port 1C6H เป็นตัวสร้างสัญญาณต่าง ๆ ดังนี้

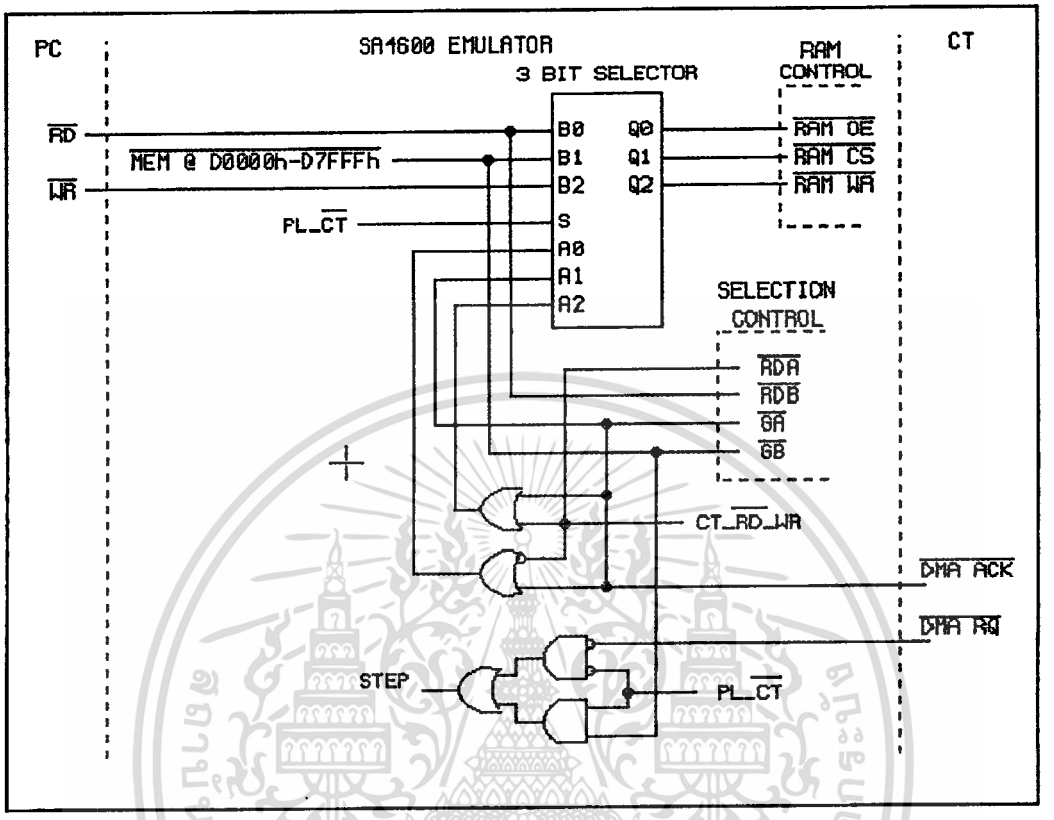
ภาพที่ 49




การสร้างสัญญาณควบคุมการ DMA

ในการรับ-ส่ง ข้อมูลระหว่างคอมพิวเตอร์และแรมแบบสถิตนั้น จะทำการออกแบบให้ แรมแบบสถิตอยู่ที่ address ตำแหน่งที่ D0000-D7FFF ซึ่งเป็นช่วงหน่วยความจำที่ว่างของคอมพิวเตอร์ ดังนั้นจะได้วงจรในส่วนของการควบคุม DMA ดังนี้

ภาพที่ 50




วงจรการควบคุม DMA

## การสร้างสัญญาณ DMA REQUEST

เนื่องจากสัญญาณ DMA REQUEST นี้ต้องเป็นแบบ FREE RUNNING ที่เริ่มทำงานเมื่อได้รับสัญญาณ DMA EN และหยุดการทำงานเมื่อได้รับสัญญาณ TC (Terminate Counted) ตามตารางดังนี้

ตารางที่ 5

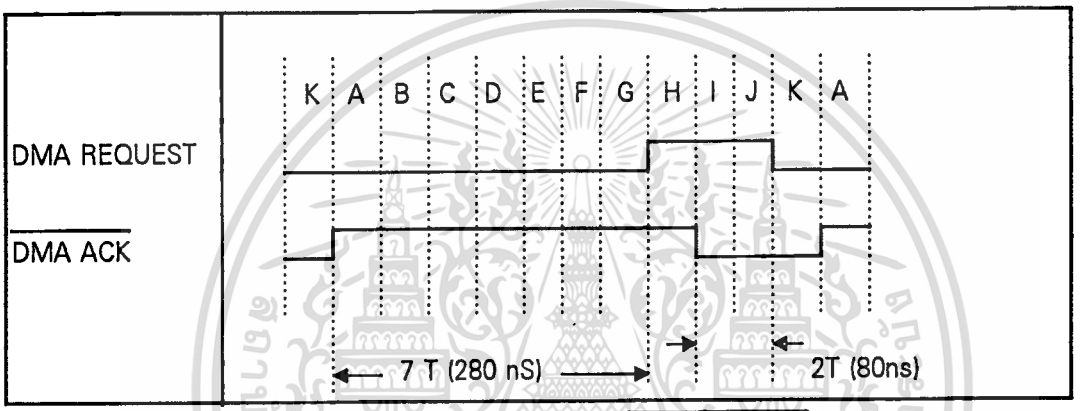
เงื่อนไขการทำงานของสัญญาณ DMA REQUEST

| DMA EN | TC | DMA REQUEST |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | FREE RUNNING |
| 1 | 1 | 0 |

ในการออกแบบจะใช้ JK-ฟลิปฟลอปที่มีขาเคลียร์ โดยนำสัญญาณ DMA EN และ TC มาควบคุมที่ขาเคลียร์ของฟลิปฟลอป

จากข้อกำหนดการ DMA ตามรูปที่ 47 และ 48 ถ้าเลือกใช้สัญญาณนาฬิกาความถี่ 25 MHz. หรือ 40 nSec. มาเป็นฐานในการทำงานของ DMA จะสามารถแบ่งสเตตของสัญญาณ DMA REQUEST ได้ดังนี้

ภาพที่ 51




สเตตของสัญญาณ $\overline{\text{DMA REQUEST}}$

จากรูปข้างบนแบ่งออกเป็น 11 สเตต จึงต้องใช้ฟลิปฟลอปจำนวน 4 ตัวในการอิมพีเมนท์ และเพื่อลดปัญหาเรื่องสัญญาณรบกวนที่อาจเกิดขึ้นในสัญญาณ $\overline{\text{DMA ACK}}$ ในช่วงของสเตต A ถึง C ถ้าพบว่าสัญญาณ $\overline{\text{DMA ACK}}$ เป็น '0' จะกลับมาเริ่มต้นที่สเตต K ใหม่ จากการอิมพีเมนท์จะได้

$J3 = \overline{Y0} \cdot \overline{Y1} \cdot Y2$

$K3 = Y1$

$J2 = \text{DMA ACK} \cdot Y0 \cdot Y1$

$K2 = Y1 \cdot Y3$

$J1 = \overline{\text{DMA ACK}} \cdot Y0 \cdot Y3 + \text{DMA ACK} \cdot Y0 \cdot \overline{Y2}$

$K1 = \overline{\text{DMA ACK}} \cdot Y2 + Y0 \cdot Y2$

$J0 = \text{DMA ACK} \cdot \overline{Y1} \cdot \overline{Y2} + Y1 \cdot Y2 + \overline{\text{DMA ACK}} \cdot Y3$

$K0 = \overline{\text{DMA ACK}} \cdot \overline{Y2} \cdot + Y1 \cdot Y3 + \overline{Y1} \cdot Y2 \cdot \overline{Y3} + Y1 \cdot \overline{Y2}$

$\text{DMA REQUEST} = Y2 \cdot Y3 (Y0 + \overline{Y1})$

# บทที่ 5
# การทดลองและผลการทดลอง

จากการที่ ชุดเลียนแบบ SA4600 (emulator) ประกอบไปด้วยส่วนที่เป็นฮาร์ดแวร์ และซอฟต์แวร์ การทดสอบจึงต้องแยกทำทีละส่วน โดยในขั้นแรก จะทำการทดสอบในส่วนของฮาร์ดแวร์เบื้องต้น คือ การถอดรหัส (decode) สัญญาณควบคุมต่างๆ ทั้งในส่วนที่ติดต่อกับเครื่องคอมพิวเตอร์ และส่วนที่ติดต่อกับเครื่อง CT

สัญญาณควบคุมเบื้องต้น ในส่วนคอมพิวเตอร์ประกอบไปด้วยสัญญาณ $\overline{\text{PCRD CMD/PAR}}$ , $\overline{\text{PCRD STATUS}}$ , $\overline{\text{PCWR RESULT}}$ และ $\overline{\text{PCWR STATUS}}$ ซึ่งสร้างขึ้นจากสัญญาณ $\overline{\text{IORD @1C0}}$ , $\overline{\text{IORD @1C2}}$ , $\overline{\text{IOWR @1C1}}$ และ $\overline{\text{IORD @1C3}}$ จะสามารถตรวจสอบได้โดยการนำวงจรส่วนการถอดรหัส ที่ออกแบบไว้ ถ่าย ลงไปบนอุปกรณ์ FPGA และต่อสัญญาณควบคุมที่ต้องการออกมาที่ขาว่างของอุปกรณ์ FPGA แล้วใช้โลจิกโพรบ จับสัญญาณสั่งเครื่องคอมพิวเตอร์ให้ IN/OUT port ที่ต้องการ (เช่น ใช้คำสั่ง I 1C0 , I 1C2 , O 1C1 , O 1C3 FF ในโปรแกรม debug.com)

ในส่วนของเครื่อง CT นั้น จำเป็นต้องสร้าง ชุดจำลองการทำงานของ CT ขึ้นมาช่วยในการทดสอบและพัฒนา เนื่องจากเหตุผลหลักๆ 3 ประการคือ

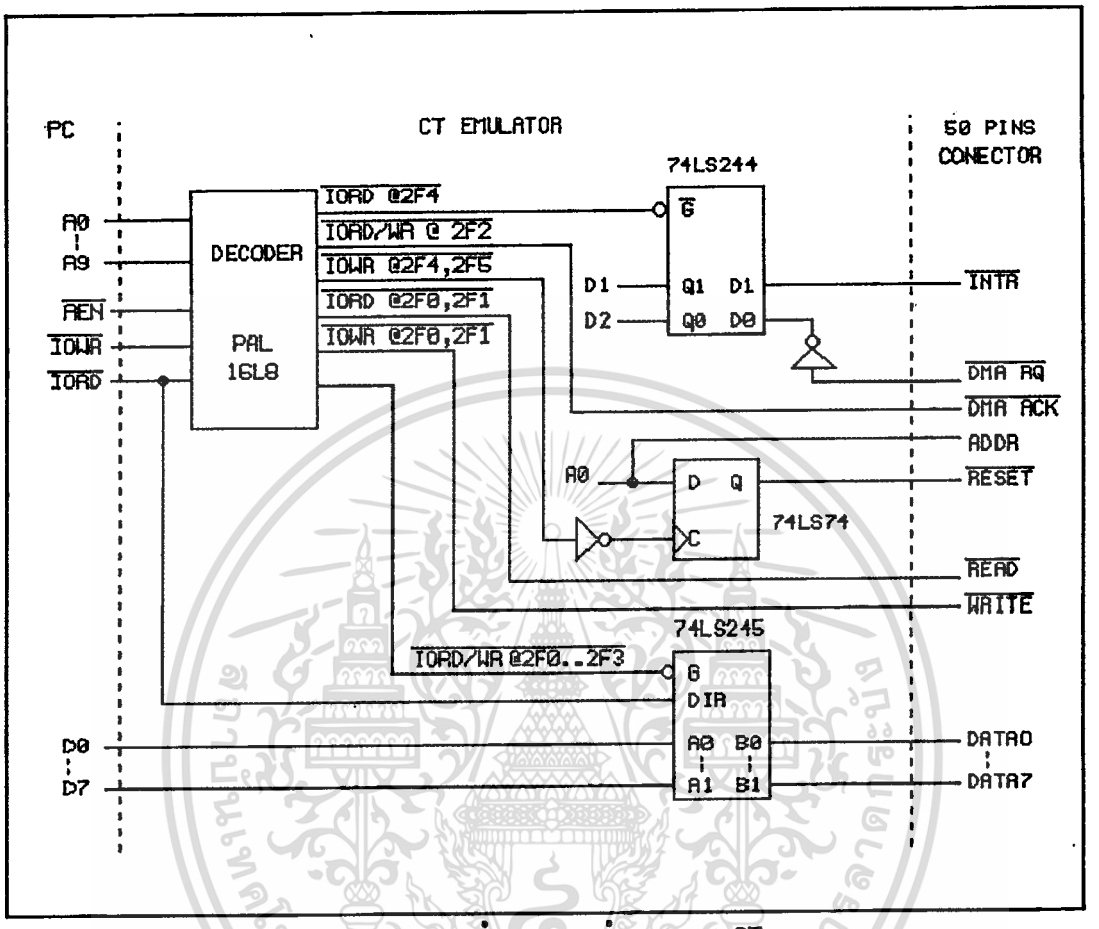
1. ไม่สามารถจัดหาเครื่อง CT สำรอง เพื่อใช้ในการทดสอบได้ จำเป็นต้องไปทดสอบ ณ. สถานที่จริง เป็นการไม่สะดวกและเสียค่าใช้จ่ายสูง

2. ไม่สามารถสั่งให้เครื่อง CT ทำการส่งสัญญาณต่างๆตามที่กำหนดหรือต้องการได้ ต้องปล่อยให้เครื่อง CT ทำการติดต่อกับ SA4600 ตามขั้นตอนการทำงานปกติ ซึ่งจะหยุดการติดต่อทันทีเมื่อเกิดข้อผิดพลาดขึ้น ทำให้การแก้ปัญหาในช่วงเริ่มต้นกระทำได้ลำบากและต้องใช้เวลานาน

3. ไม่สามารถทำการทดสอบกับเครื่อง CT ได้อย่างต่อเนื่อง จำเป็นต้องต่อระบบ SA4600 เดิม คืนให้เจ้าหน้าที่ทุกครั้งเมื่อมีผู้มาขอใช้บริการเอ็กซเรย์

ในที่นี้ใช้เครื่องคอมพิวเตอร์ทั่วไป (ISA BUS) ในการสร้างชุด CT จำลอง โดยมีวงจรช่วยสร้างสัญญาณดังนี้

ภาพที่ 52 .




แสดงวงจรจำลองการทำงานของ CT

ตารางที่ 6

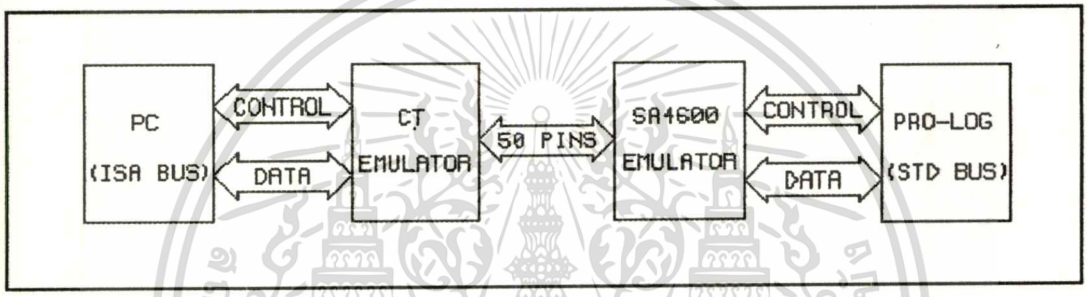
แสดง I/O map ของวงจรจำลองการทำงานของ CT

| Port | I/O Read | I/O Write |
|---|---|---|
| 2F0 | อ่านผลลัพธ์ | เขียนพารามิเตอร์ |
| 2F1 | อ่านสถานะ | เขียนคำสั่ง |
| 2F2 | DMA IN | DMA OUT |
| 2F4 | รับสถานะของสัญญาณ | เซ็ต สัญญาณ reset |
| 2F5 | $\overline{\text{DMA RQ}}$ และ $\overline{\text{INTR}}$ | เคลียร์ สัญญาณ reset |

## การทดสอบสัญญาณการ อ่าน-เขียน ระหว่าง CT และชุดเลียนแบบ SA4600

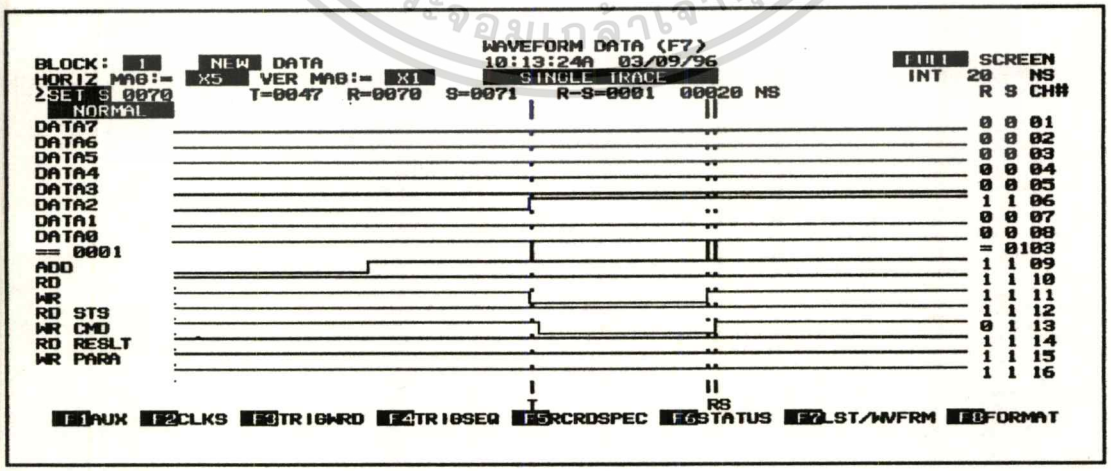
ในการตรวจสอบสัญญาณควบคุมในด้านของ CT จะทำการติดต่อชุดเลียนแบบ SA4600 เข้ากับชุด CT จำลอง ดังแสดงในภาพ จากนั้นสั่งชุด CT จำลอง ให้ทำการอ่าน ผลลัพธ์/สถานะ หรือ ทำการเขียนคำสั่ง/พารามิเตอร์ และใช้เครื่องมือวัด Logic analyzer จับสัญญาณที่สอดคล้องบนชุดเลียนแบบ SA4600

ภาพที่ 53



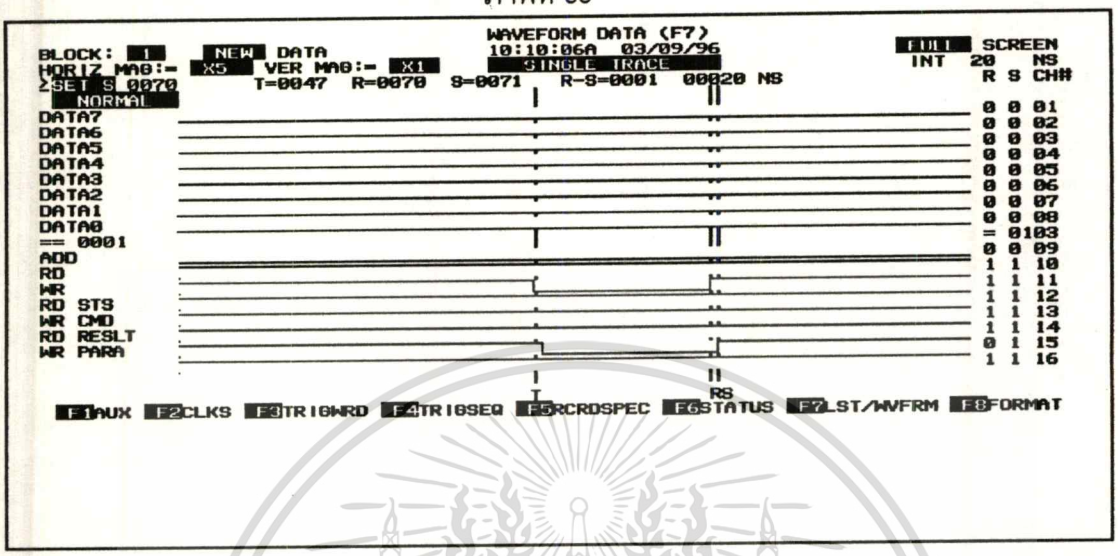

แสดงการเชื่อมต่อระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

ภาพที่ 54




แสดงสัญญาณ $\overline{\text{WR COMMAND}}$ ระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

ภาพที่ 55



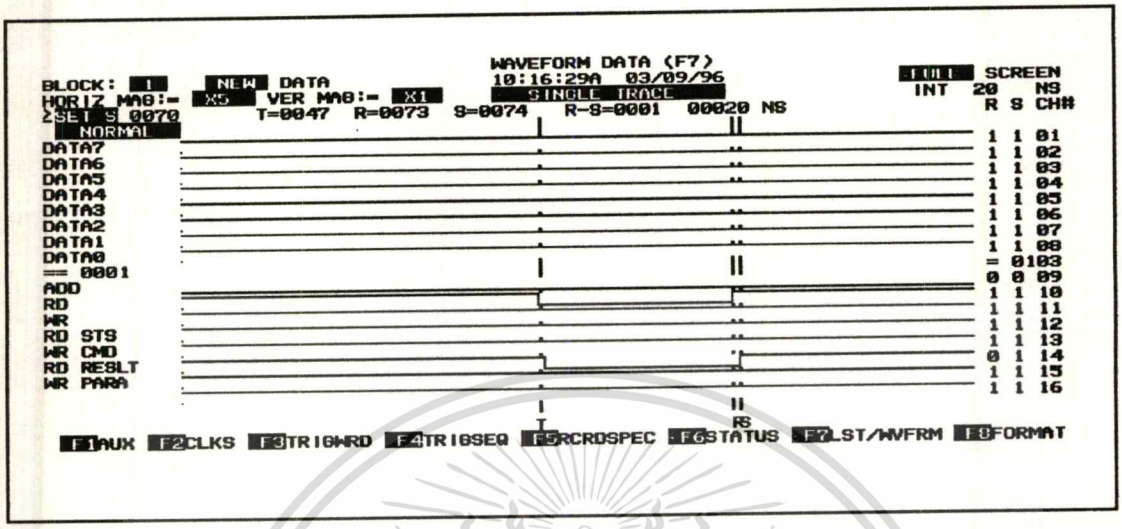

แสดงสัญญาณ $\overline{\text{WR PARAMATER}}$ ระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

จากการเปรียบเทียบสัญญาณ การเขียนคำสั่งและพารามิเตอร์ ในรูปที่ 54 และ 55 กับข้อกำหนดการเขียนคำสั่ง/พารามิเตอร์ ของชุด SA4600 ตามรูปที่ 42 ในจุดที่มีความสำคัญจะได้ว่า

- จากข้อกำหนด - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{WRITE}}$ ข้อมูลใน data bus จะคงอยู่ต่ออีกไม่น้อยกว่า 55 nSec.

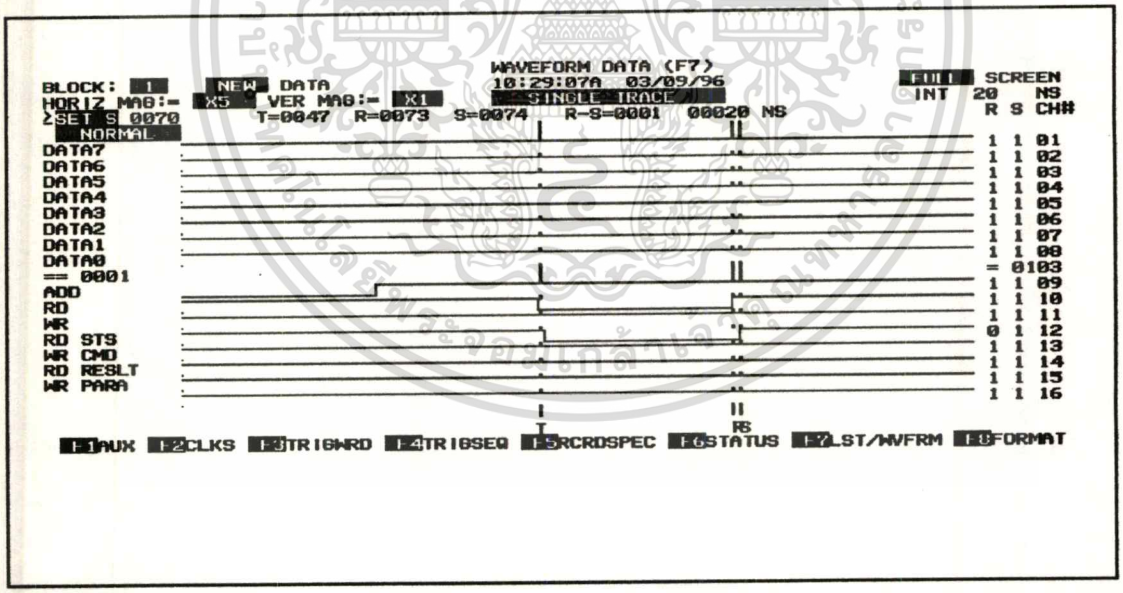
- ผลการทดลอง - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{WRITE}}$ สัญญาณ $\overline{\text{WR COMMAND}}$ และสัญญาณ $\overline{\text{WR PARAMETER}}$ จะขึ้นในเวลา 20 nSec. ทำให้สามารถแลตช์ข้อมูลของ data bus ลงในรีจิสเตอร์ได้ ก่อนที่ข้อมูลของ data bus จะหายไป

ภาพที่ 56




แสดงสัญญาณ $\overline{\text{RD RESULT}}$ ระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

ภาพที่ 57




แสดงสัญญาณ $\overline{\text{RD STATUS}}$ ระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

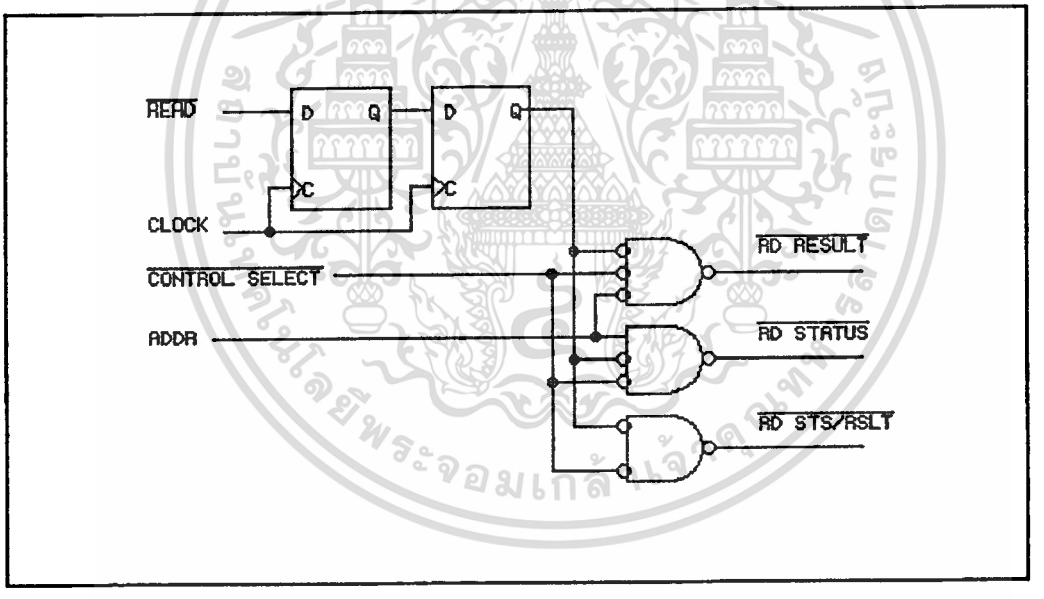
จากการเปรียบเทียบสัญญาณ การอ่านผลลัพธ์และอ่านสถานะ ในรูปที่ 56 และ 57 กับข้อกำหนดการอ่านผลลัพธ์/สถานะ ของชุด SA4600 ตามรูปที่ 43 ในจุดที่มีความสำคัญจะได้ว่า

- จากข้อกำหนด - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{READ}}$ ข้อมูลใน data bus จะต้องคงอยู่ต่ออีกไม่น้อยกว่า 42 nSec.

- ผลการทดลอง - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{READ}}$ สัญญาณ $\overline{\text{RD RESULT}}$ และสัญญาณ $\overline{\text{RD STATUS}}$ จะขึ้นตามในเวลาประมาณ 20 nSec. ซึ่งเมื่อนำสัญญาณเหล่านี้ไปเป็นตัวเปิดรีจิสเตอร์ผลลัพธ์/สถานะ ตามภาพที่ 39 และภาพที่ 40 จะทำให้ data bus คงอยู่เพียง 20 nSec. ซึ่งต่ำกว่าข้อกำหนด และอาจทำให้เครื่อง CT ไม่สามารถอ่านข้อมูลได้อย่างถูกต้อง

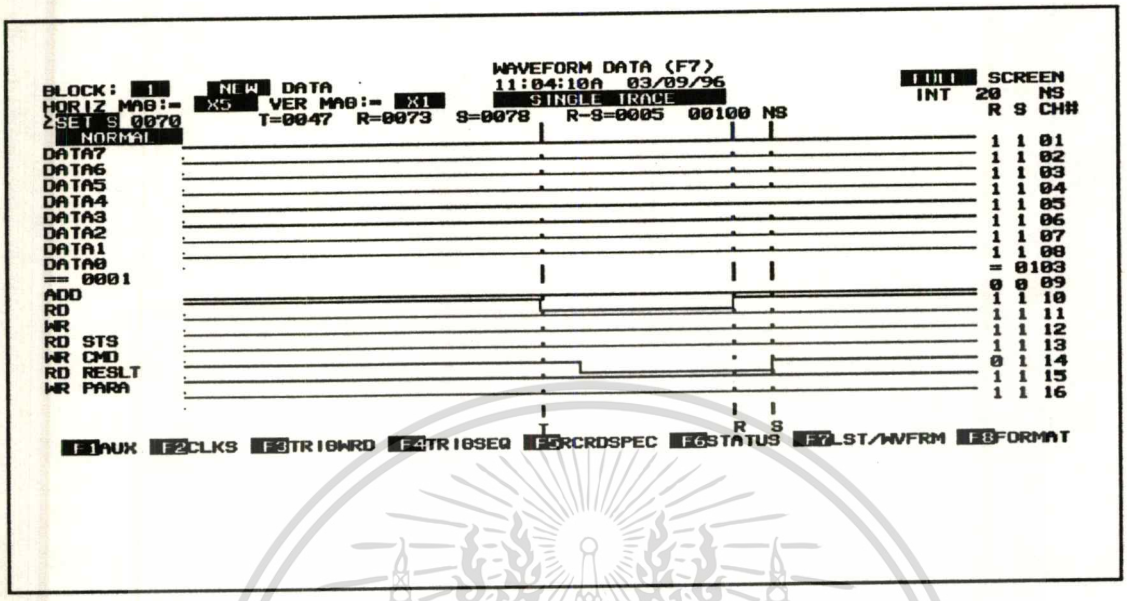
เพื่อให้สัญญาณอ่านผลลัพธ์และอ่านสถานะ เป็นไปตามข้อกำหนด จึงมีการประวิงสัญญาณ $\overline{\text{READ}}$ ที่นำมาถอดรหัสสัญญาณ $\overline{\text{RD RESULT/STATUS}}$ ออกไป 2 clock ซึ่งจะประวิงเวลาเพิ่มได้อีก 40-80 nSec. ตามที่แสดงดังภาพที่ 58

ภาพที่ 58



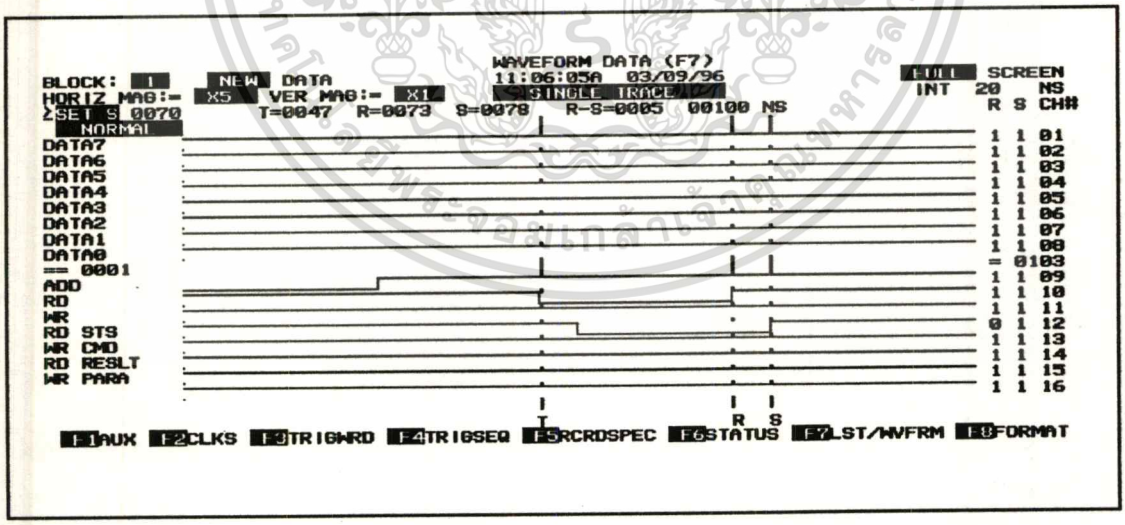

การถอดรหัสสัญญาณควบคุมจากเครื่อง CT ที่ทำการปรับปรุงในส่วนของ $\overline{\text{RD RESULT}}$ และ $\overline{\text{RD STATUS}}$

ภาพที่ 59




สัญญาณ $\overline{\text{RD RESULT}}$ ที่ได้รับการปรับปรุง

ภาพที่ 60




สัญญาณ $\overline{\text{RD STATUS}}$ ที่ได้รับการปรับปรุง

จากสัญญาณ $\overline{\text{RD RESULT}}$ และ $\overline{\text{RD STATUS}}$ ที่ได้รับการหน่วงเวลาออกไปตามภาพที่ 59 และภาพที่ 60 นั้น จะเห็นได้ว่าเป็นไปตามข้อกำหนดของการอ่านผลลัพธ์/สถานะ ตามภาพที่ 43

เมื่อได้ทำการตรวจสอบสัญญาณควบคุมเบื้องต้นเรียบร้อยแล้ว ขั้นต่อไปคือทำการแทนข้อมูล (map) วงจรในส่วนของรีจิสเตอร์คำสั่ง, พารามิเตอร์รีจิสเตอร์, รีจิสเตอร์ผลลัพธ์ และ รีจิสเตอร์สถานะ ลงในอุปกรณ์ FPGA (วงจรในภาพที่ 38 ถึง 41 ) เพื่อทำการทดสอบในส่วนของรีจิสเตอร์ ต่างๆว่าทำงานได้ถูกต้องตรงตามที่ต้องการหรือไม่ โดย

- รีจิสเตอร์คำสั่ง
  - CT สั่ง $\overline{\text{WR COMMAND}}$ ( O 2F1 5A )
  - SA4600 สั่ง $\overline{\text{PCRD CMD/PARA}}$ ( I 1C0 )

- พารามิเตอร์รีจิสเตอร์
  - CT สั่ง $\overline{\text{WR PARAMETER}}$ ( O 2F0 A5 )
  - SA4600 สั่ง $\overline{\text{PCRD CMD/PARA}}$ ( I 1C0 )

- รีจิสเตอร์ผลลัพธ์
  - SA4600 สั่ง $\overline{\text{PCWR RESULT}}$ ( O 1C1 5S )
  - CT สั่ง $\overline{\text{RD RESULT}}$ ( I 2F0 )

- รีจิสเตอร์สถานะ
  - CT สั่งเคลียร์สัญญาณรีเซ็ต ( O 2f5 00 ) เพื่อให้ รีจิสเตอร์สถานะ ทำได้โดยอิสระ
  - CT สั่ง $\overline{\text{WR COMMAND}}$ ( O 2F1 00 ) เพื่อเซ็ตบิต command reg. full และบิต command busy ของ รีจิสเตอร์สถานะ
  - CT สั่ง $\overline{\text{RD STATUS}}$ ( I 2F1 ) และ SA4600 สั่ง $\overline{\text{PCRD STATUS}}$ ( I 1C2 ) เพื่อตรวจสอบบิต 0 และ 1 ของรีจิสเตอร์สถานะ
  - CT สั่ง $\overline{\text{WR PARAMETER}}$ ( O 2F0 00 ) เพื่อเซ็ตบิต parameter full
  - CT และ SA4600 สั่งอ่านรีจิสเตอร์สถานะเพื่อตรวจสอบบิตที่2
  - SA4600 สั่ง $\overline{\text{PCWR STATUS}}$ ( O 1C3 1F ) เพื่อเซ็ตบิต result reg. full และบิต interrup ของ รีจิสเตอร์สถานะ
  - CT และ SA4600 สั่งอ่านรีจิสเตอร์สถานะ เพื่อตรวจสอบค่าของรีจิสเตอร์สถานะ
  - CT ส่ง I 2F4 เพื่อตรวจสอบสัญญาณ INTERRUP (บิต 1)

- CT และ SA4600 สั่งอ่านรีจิสเตอร์สถานะ เพื่อตรวจสอบค่าต่างๆของ รีจิสเตอร์สถานะ
- PC สั่ง $\overline{\text{RD RESULT}}$ เพื่อเคลียร์รีจิสเตอร์สถานะ ( I 2F0 )
- CT และ SA4600 สั่งอ่านรีจิสเตอร์สถานะ เพื่อตรวจสอบค่าของ รีจิสเตอร์สถานะ
- SA4600 ส่ง $\overline{\text{PCWR STATUS}}$ ( O 1C3 1F ) เพื่อตั้งค่าต่างๆใน รีจิสเตอร์สถานะ
- CT สร้างพัลส์ของสัญญาณ $\overline{\text{RESET}}$ เพื่อเคลียร์ค่าในรีจิสเตอร์สถานะ ( O 2F4 00 และ O 2F5 00 ตามลำดับ)
- CT และ SA4600 สั่งอ่าน รีจิสเตอร์สถานะ เพื่อตรวจสอบค่าของ รีจิสเตอร์สถานะ (จะอ่านได้ค่า = 00 )

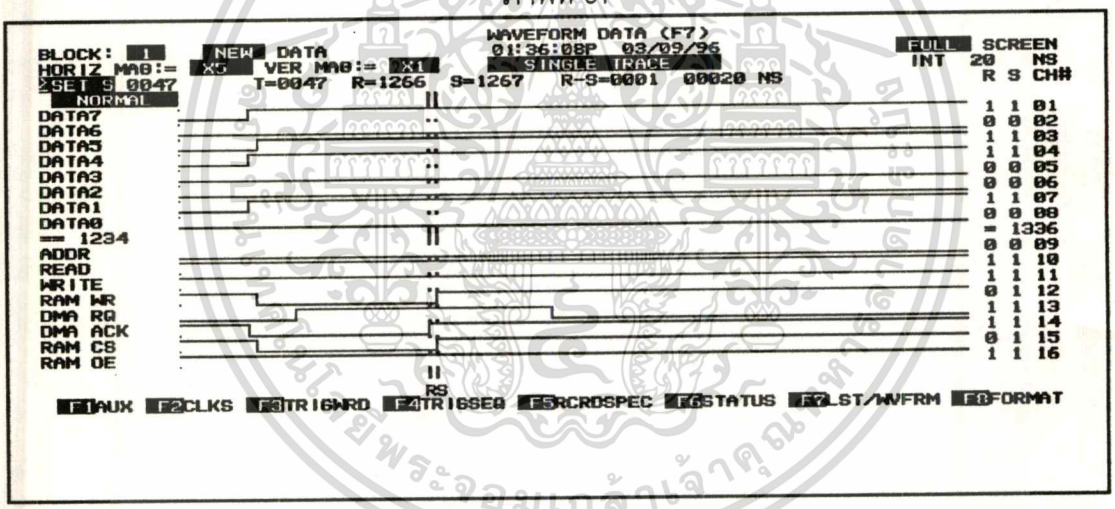
**การทดสอบสัญญาณในขบวนการ DMA ระหว่าง CT และชุดเลียนแบบ SA4600**

ขั้นต่อไปจะทำการทดสอบในส่วนของ DMA ตามวงจรในภาพที่ 44,45,46,49,50 และสมการของชุด DMA REQUEST

ในส่วนการอ่านเขียนข้อมูล ระหว่างเครื่องคอมพิวเตอร์และแรมแบบสถิตนั้น ทดสอบได้โดยการ เขียน-อ่าน หน่วยความจำที่ตำแหน่ง 0D0000h - 0D7FFFh

จากการเปรียบเทียบสัญญาณการ DMA IN ในภาพที่ 61 กับข้อกำหนดการ DMA IN ในภาพที่ 48 จะได้

ภาพที่ 61

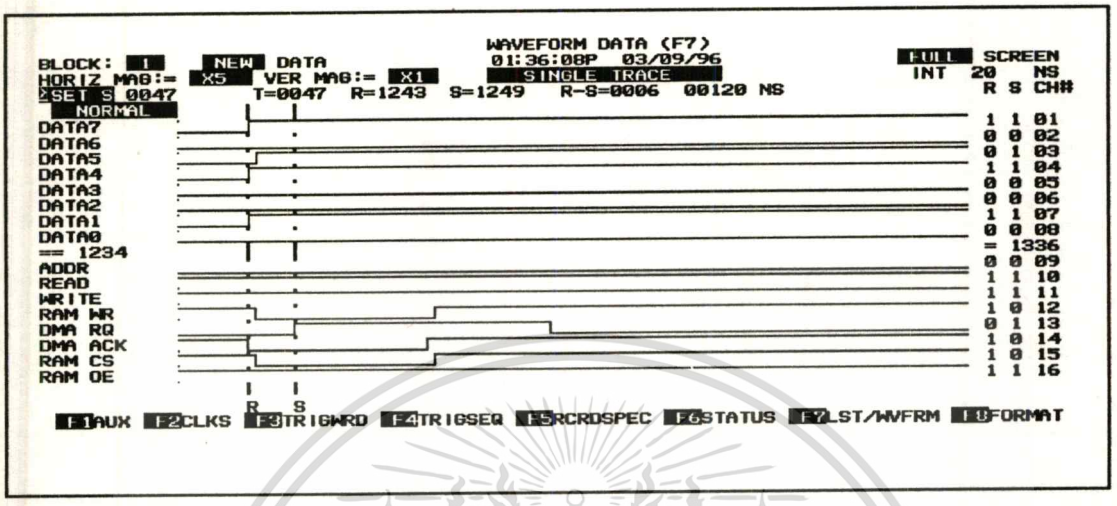



แสดงช่วงเวลาระหว่างขอบขาขึ้นของสัญญาณ $\overline{\text{DMA ACK}}$ และสัญญาณ $\overline{\text{RAM WR}}$ ระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

- จากข้อกำหนด - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{DMA ACK}}$ ข้อมูลใน data bus จะคงอยู่ต่ออีกไม่น้อยกว่า 55 nSec.

- ผลการทดลอง - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{DMA ACK}}$ สัญญาณ $\overline{\text{RAM WR}}$ ซึ่งใช้ในการเขียนข้อมูลลงในแรมแบบสถิต จะทำงานในเวลา 20 nSec. ทำให้สามารถนำข้อมูลการ DMA เก็บไว้ในแรมแบบสถิต ได้ก่อนที่ข้อมูลของ data bus จะหายไป

ภาพที่ 62

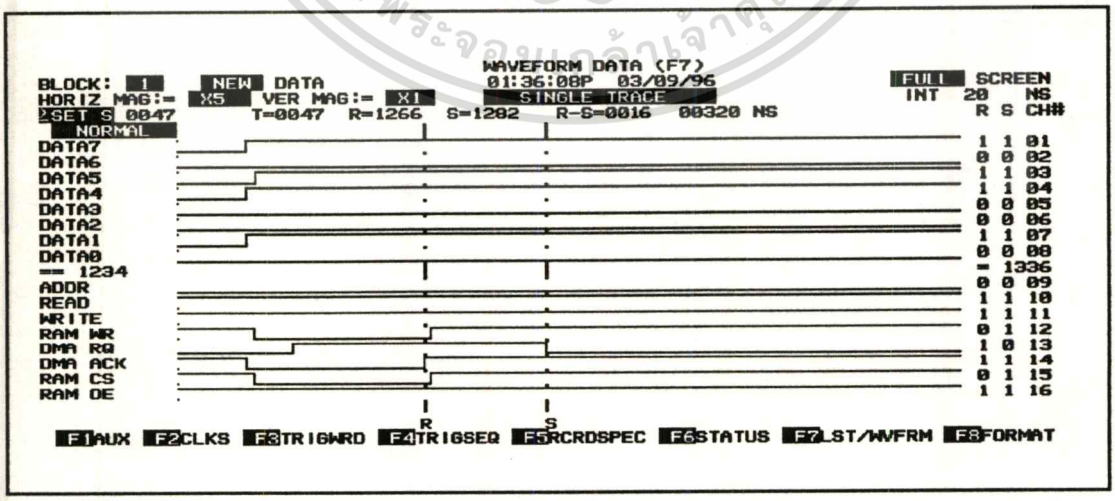



แสดงช่วงเวลา hold ของสัญญาณ $\overline{\text{DMA RQ}}$ ระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

- จากข้อกำหนด - หลังจากขอบขาลงของสัญญาณ $\overline{\text{DMA ACK}}$ , สัญญาณ $\overline{\text{DMA RQ}}$ จะต้องคงอยู่ต่ออีกไม่น้อยกว่า 70 nSec.

- ผลการทดลอง - สัญญาณ $\overline{\text{DMA RQ}}$ ยังคงอยู่หลังจากขอบขาลงของสัญญาณ $\overline{\text{DMA ACK}}$ อีก 120 nSec. ซึ่งเป็นไปตามข้อกำหนด

ภาพที่ 63

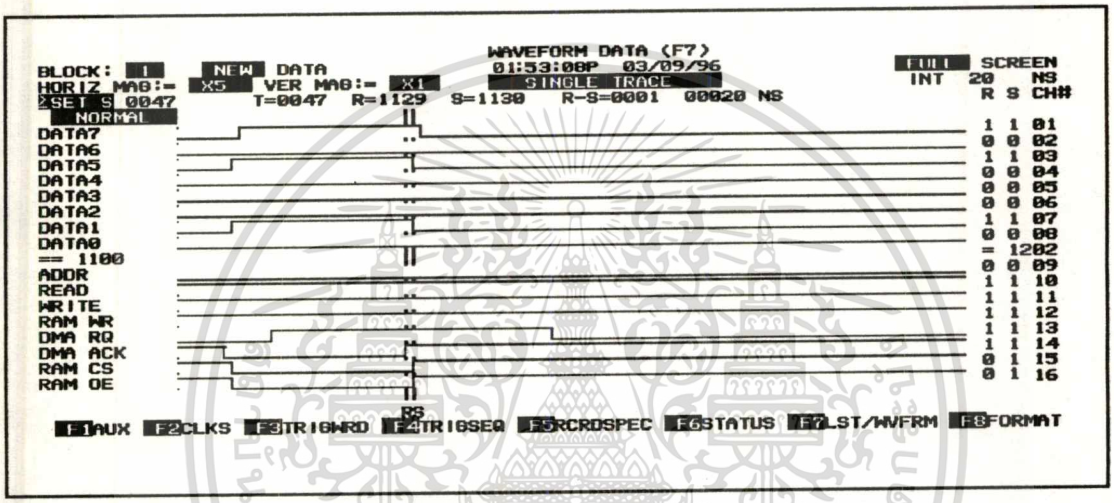



แสดงช่วงเวลาประวิงก่อนที่จะขอ DMA ข้อมูลไบต์ต่อไประหว่างชุด CT จำลอง และชุดเลียนแบบ SA4600

- จากข้อกำหนด - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{DMA ACK}}$ สัญญาณ $\overline{\text{DMA RQ}}$ จะเริ่ม active ใหม่ได้ เมื่อเวลาผ่านไปไม่น้อยกว่า 280 nSec.

- ผลการทดลอง - สัญญาณ $\overline{\text{DMA RQ}}$ จะเริ่มร้องขอข้อมูลไบต์ต่อไป เมื่อเวลาผ่านไป 320 nSec. ซึ่งเป็นไปตามข้อกำหนด

ภาพที่ 64




สัญญาณการ DMA OUT ระหว่างชุด CT จำลองและชุดเลียนแบบ SA4600

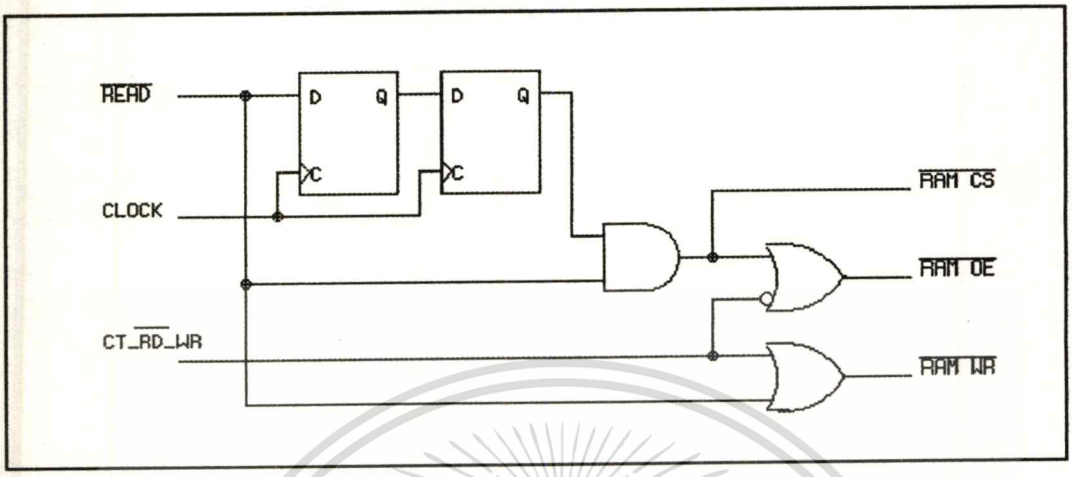
จากการเปรียบเทียบสัญญาณการ DMA OUT ในภาพที่ 64 กับข้อกำหนดการ DMA OUT ในภาพที่ 47 จะได้

- จากข้อกำหนด - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{DMA ACK}}$ ข้อมูลใน data bus จะคงอยู่ต่ออีกไม่น้อยกว่า 40 nSec.

- ผลการทดลอง - หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{DMA ACK}}$ สัญญาณ $\overline{\text{RAM CS}}$ และ $\overline{\text{RAM OE}}$ ซึ่งใช้ในการอ่านข้อมูลจากแรมแบบสถิต คงอยู่เป็นเวลา 20 nSec. ทำให้ข้อมูลการ DMA คงอยู่บน data bus เพียง 20 nSec. ซึ่งอาจทำให้การ DMA ได้ข้อมูลที่ไม่ถูกต้อง

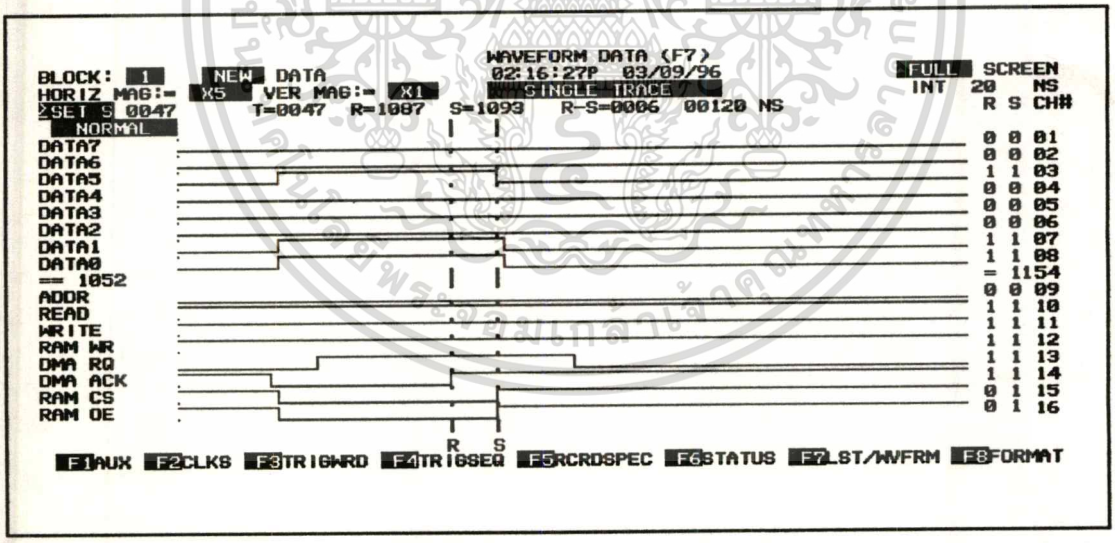
เพื่อให้สัญญาณ $\overline{\text{RAM CS}}$ และ $\overline{\text{RAM OE}}$ เป็นไปตามข้อกำหนด จึงต้องทำการขยายสัญญาณทั้งสองออกไปอีกอย่างน้อย 40 nSec. โดยใช้วงจรดังนี้

ภาพที่ 65




แสดงวงจรยืดสัญญาณ $\overline{\text{RAM CS}}$ และ $\overline{\text{RAM OE}}$

ภาพที่ 66




สัญญาณ $\overline{\text{DMA OUT}}$

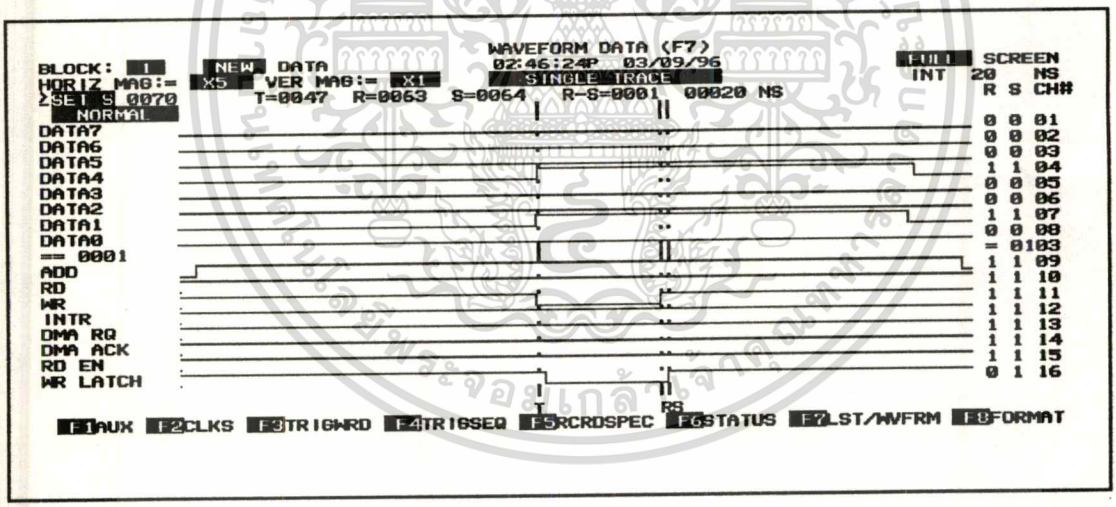
ภาพที่ 66 แสดงช่วงเวลาของสัญญาณ $\overline{\text{RAM CS}}$ และ $\overline{\text{RAM OE}}$ ที่ได้รับการขยายเวลาไปเป็น 120 nSec. เพื่อให้เป็นไปตามข้อกำหนดของการ DMA OUT

จากการทดลองส่งถ่ายข้อมูลระหว่างชุดชุดเลียนแบบ SA4600 กับเครื่อง CT จำลอง ปรากฏว่าความเร็วในการรับส่งข้อมูลต่ำกว่าเครื่อง CT ของจริงมาก เนื่องจากชุด CT จำลอง ใช้ซอฟต์แวร์ในการ polling สัญญาณ $\overline{\text{DMA RQ}}$ ทำให้การตอบสนอง DMA ช้ากว่าเครื่อง CT จริง ซึ่งใช้การทำงานจากฮาร์ดแวร์มาก

**ผลการตรวจวัดสัญญาณต่างๆ เมื่อทดสอบกับเครื่องจริง**

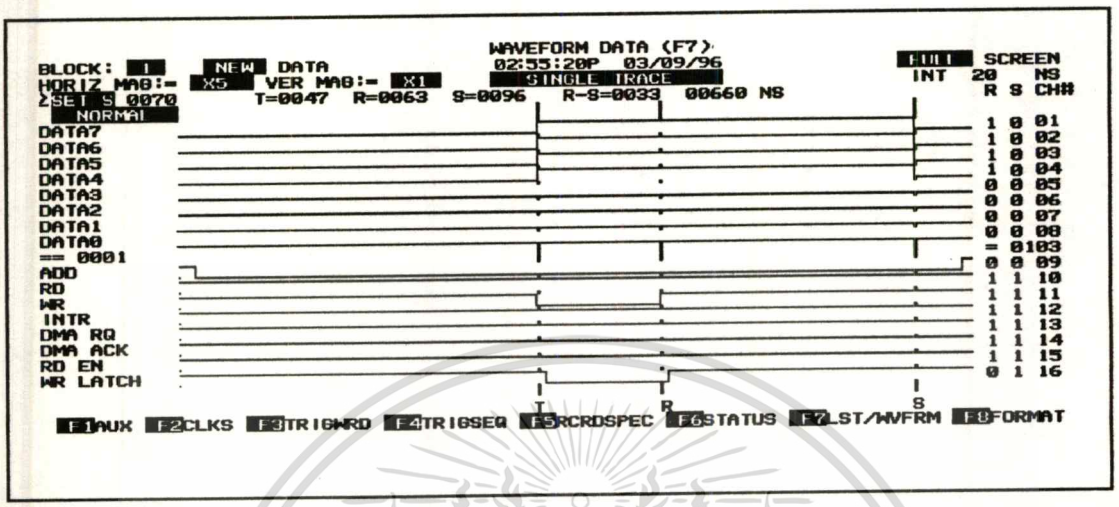
เมื่อนำชุดเลียนแบบ SA4600 ไปติดตั้งทดลองทำงานกับเครื่อง CT ณ.สถานที่จริง จะได้จังหวะเวลา (timing) ของการติดต่อต่างๆดังนี้

ภาพที่ 67




สัญญาณ $\overline{\text{WRITE COMMAND}}$

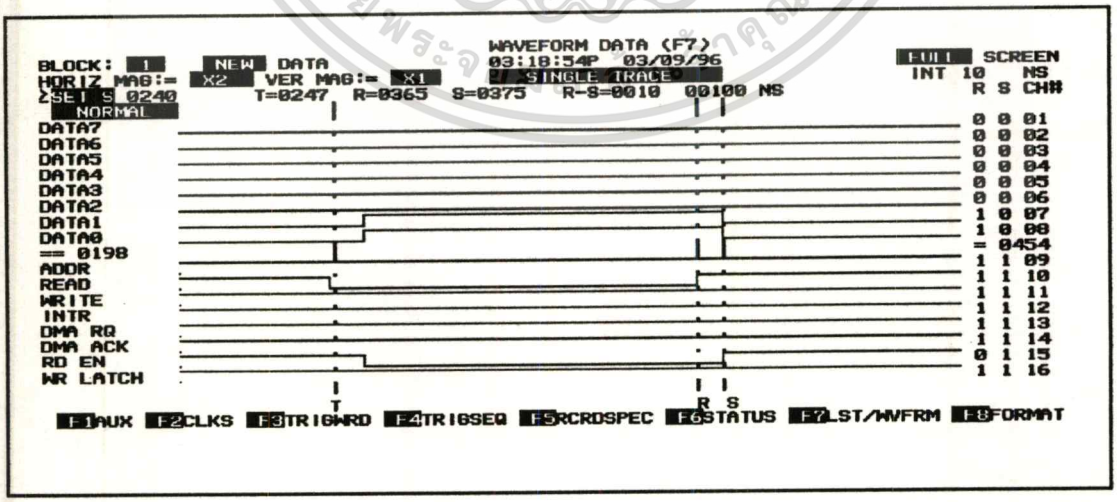
ภาพที่ 68




สัญญาณ $\overline{\text{WRITE}}$ PARAMETER

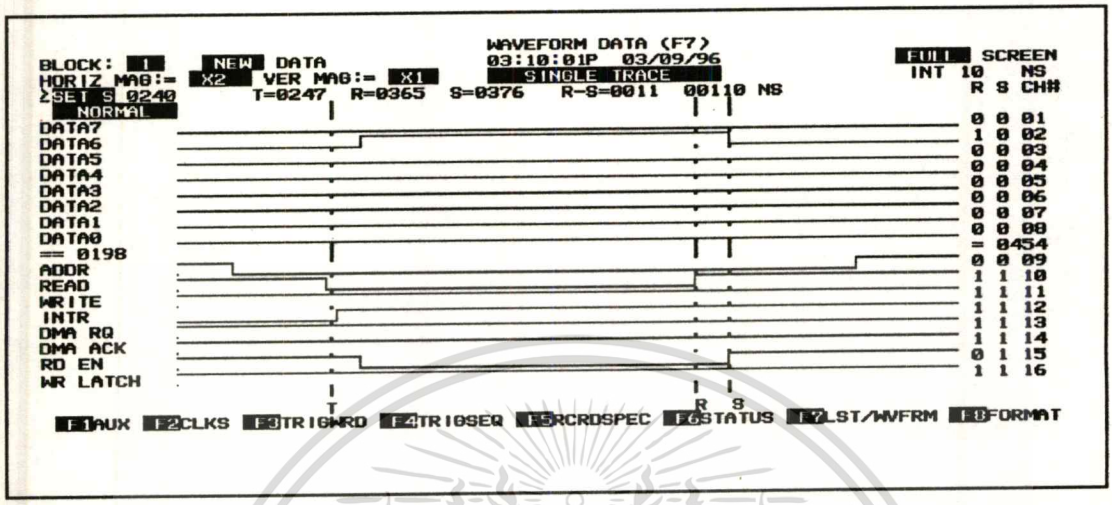
ในส่วนของการเขียนคำสั่ง และเขียนพารามิเตอร์ จะเห็นได้ว่าเครื่อง CT ได้ hold ข้อมูลใน data bus หลังจากขอบขาขึ้นของสัญญาณ $\overline{\text{WR}}$ ไว้ประมาณ 600 nSec. ทำให้ชุดเลียนแบบ SA4600 สามารถอ่านข้อมูลได้อย่างสะดวก

ภาพที่ 69




สัญญาณการ $\overline{\text{READ}}$ STATUS

ภาพที่ 70

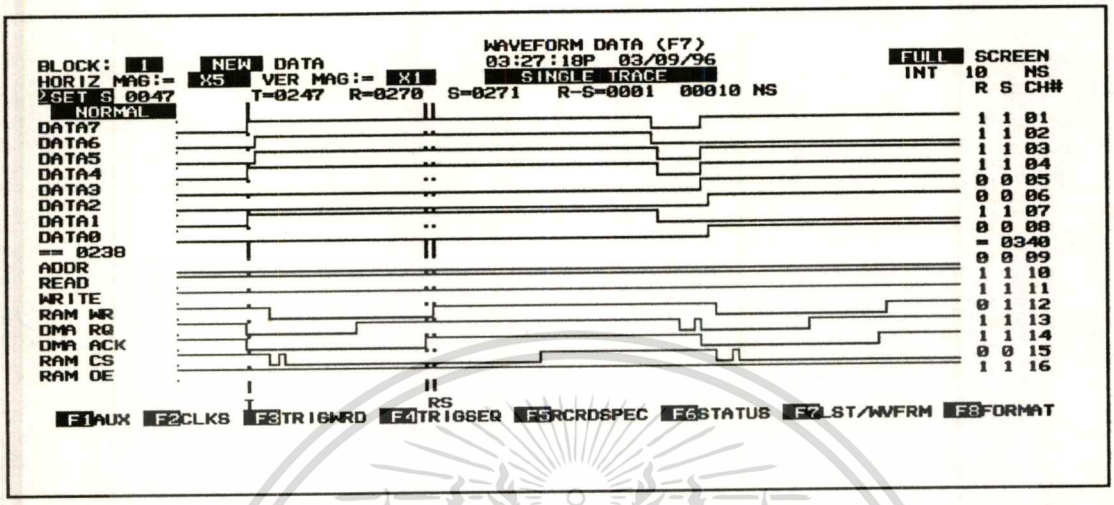



สัญญาณการ $\overline{\text{READ RESULT}}$

ในส่วนของการอ่านสถานะ และอ่านผลลัพธ์ ชุดเลียนแบบ SA4600 ได้ hold ข้อมูลใน data bus ไว้ประมาณ 100 nSec. ซึ่งมากพอสำหรับเครื่อง CT ในการแลตช์ข้อมูลเอาไว้ และจากที่มีการประวิงสัญญาณ $\overline{\text{READ}}$ ออกไปก่อนที่จะนำมาถอดรหัสสัญญาณ $\overline{\text{RD STATUS}}$ และสัญญาณ $\overline{\text{RD RESULT}}$ ตามวงจรดังภาพที่ 28 นั้น จะไม่กระทบต่อการทำงานของระบบ เนื่องจากสัญญาณ $\overline{\text{READ}}$ ที่มาจากเครื่อง CT มีความกว้างถึง 1,180 nSec.

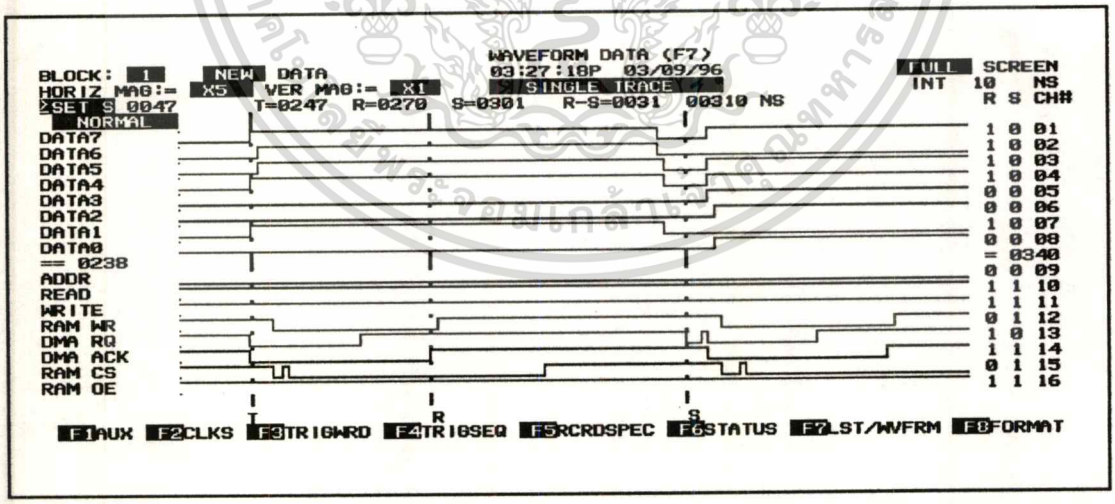
จากจังหวะเวลาของการอ่านผลลัพธ์ จะเห็นว่าสัญญาณ $\overline{\text{INTERRUP}}$ จะถูกเคลียร์ทันทีที่ได้รับสัญญาณการ $\overline{\text{READ RESULT}}$ (ตามภาพวงจรที่ 40) ซึ่งค่อนข้างเร็วเกินไป แต่จะไม่มีผลกระทบต่อการทำงาน เนื่องจากชุดเลียนแบบ SA4600 จะทำการตรวจสอบรีจิสเตอร์สถานะ เพื่อรอให้บิต interrup ถูกเคลียร์ และบิต command reg. full ถูกเซ็ตขึ้นเมื่อได้รับคำสั่งชุดต่อไป

ภาพที่ 71



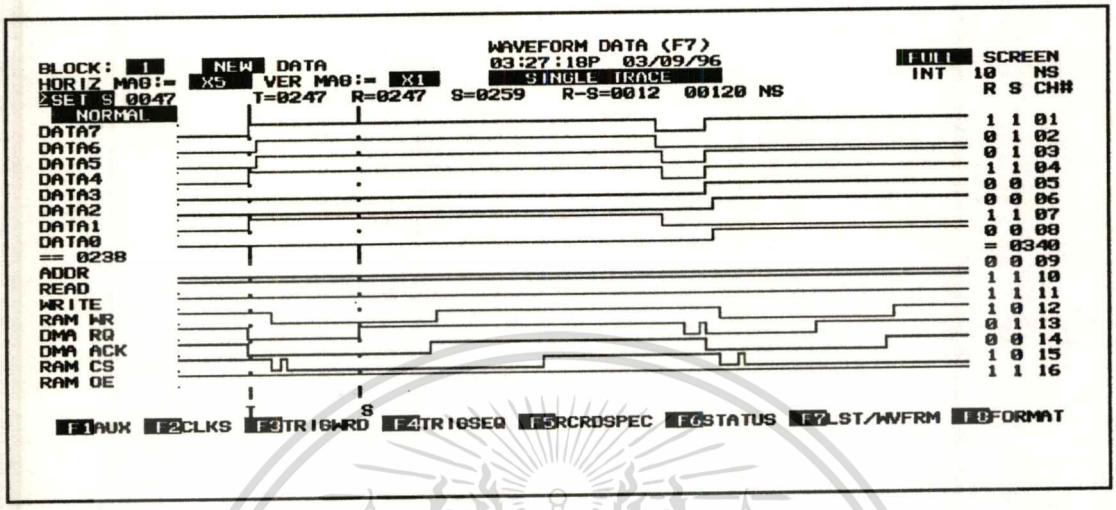

แสดงช่วงเวลาระหว่างขอบขาขึ้นของสัญญาณ $\overline{\text{RAM WR}}$ และสัญญาณ $\overline{\text{DMA ACK}}$ ในการ DMA IN ระหว่างชุด CT และชุดเลียนแบบ SA4600

ภาพที่ 72



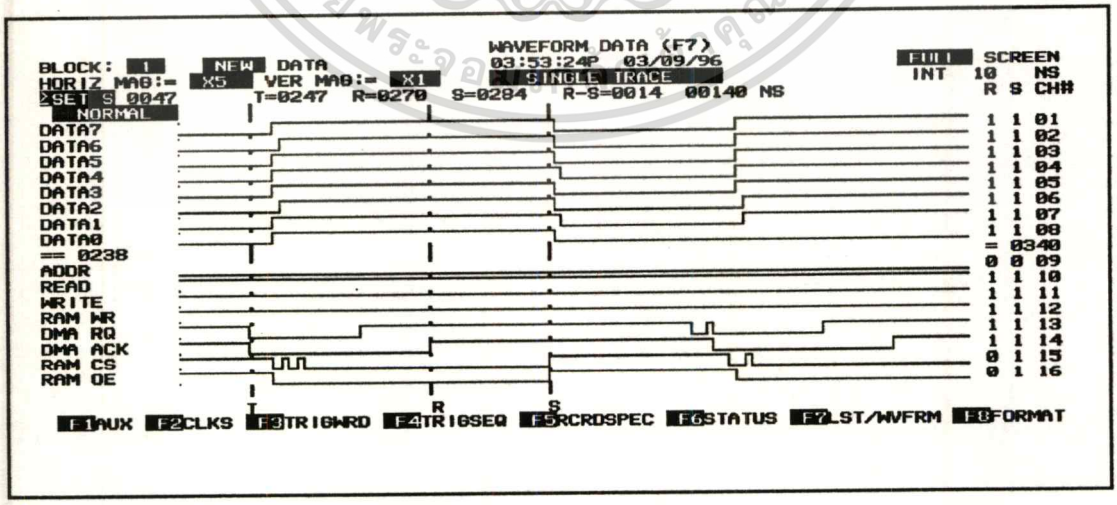

แสดงช่วงเวลาประวิงก่อนที่จะขอ DMA IN ข้อมูลไบต์ต่อไป ในการ DMA IN ระหว่างชุด CT และชุดเลียนแบบ SA4600

ภาพที่ 73



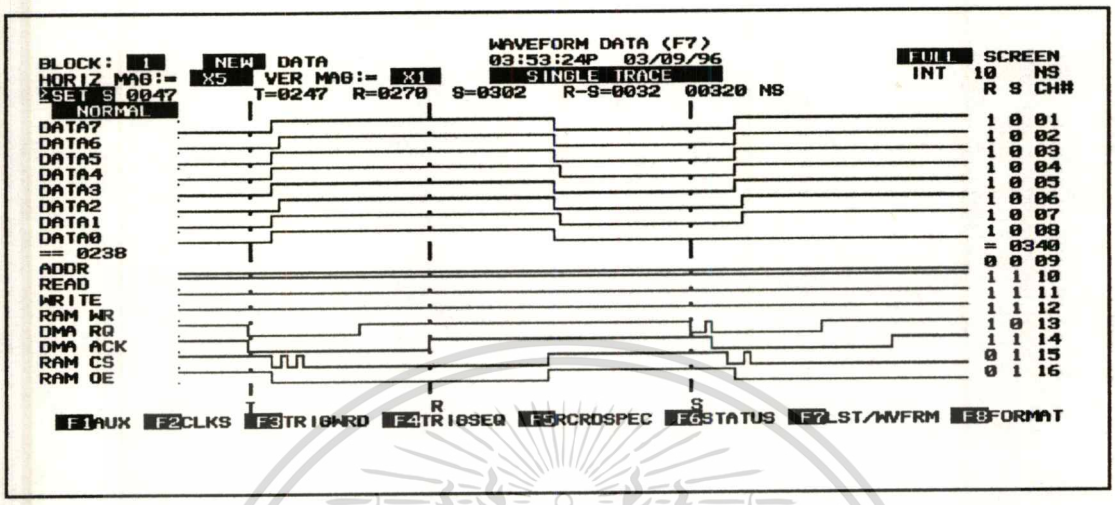

แสดงช่วงเวลา hold ของสัญญาณ $\overline{\text{DMA RQ}}$ หลังจากได้รับสัญญาณ $\overline{\text{DMA ACK}}$ ในการ DMA IN ระหว่างชุด CT และชุดเลียนแบบ SA4600

ภาพที่ 74




แสดงช่วงเวลาระหว่างขอบขาขึ้นของสัญญาณ $\overline{\text{DMA ACK}}$ และสัญญาณ $\overline{\text{RAM OE}}$ ในการ DMA OUT ระหว่างชุด CT และชุดเลียนแบบ SA4600

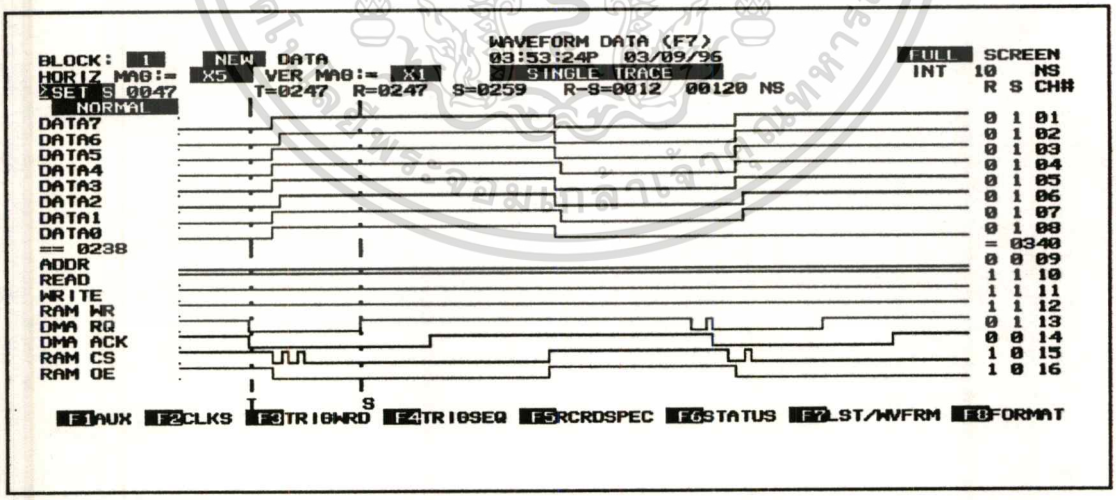
ภาพที่ 75




แสดงช่วงเวลาประวิงก่อนที่จะขอ DMA ข้อมูลไบต์ต่อไป

ในการ DMA OUT ระหว่างชุด CT และชุดเลียนแบบ SA4600

ภาพที่ 76




แสดงช่วงเวลา hold ของสัญญาณ $\overline{\text{DMA RQ}}$ หลังจากได้รับสัญญาณ $\overline{\text{DMA ACK}}$

ในการ DMA OUT ระหว่างชุด CT และชุดเลียนแบบ SA4600

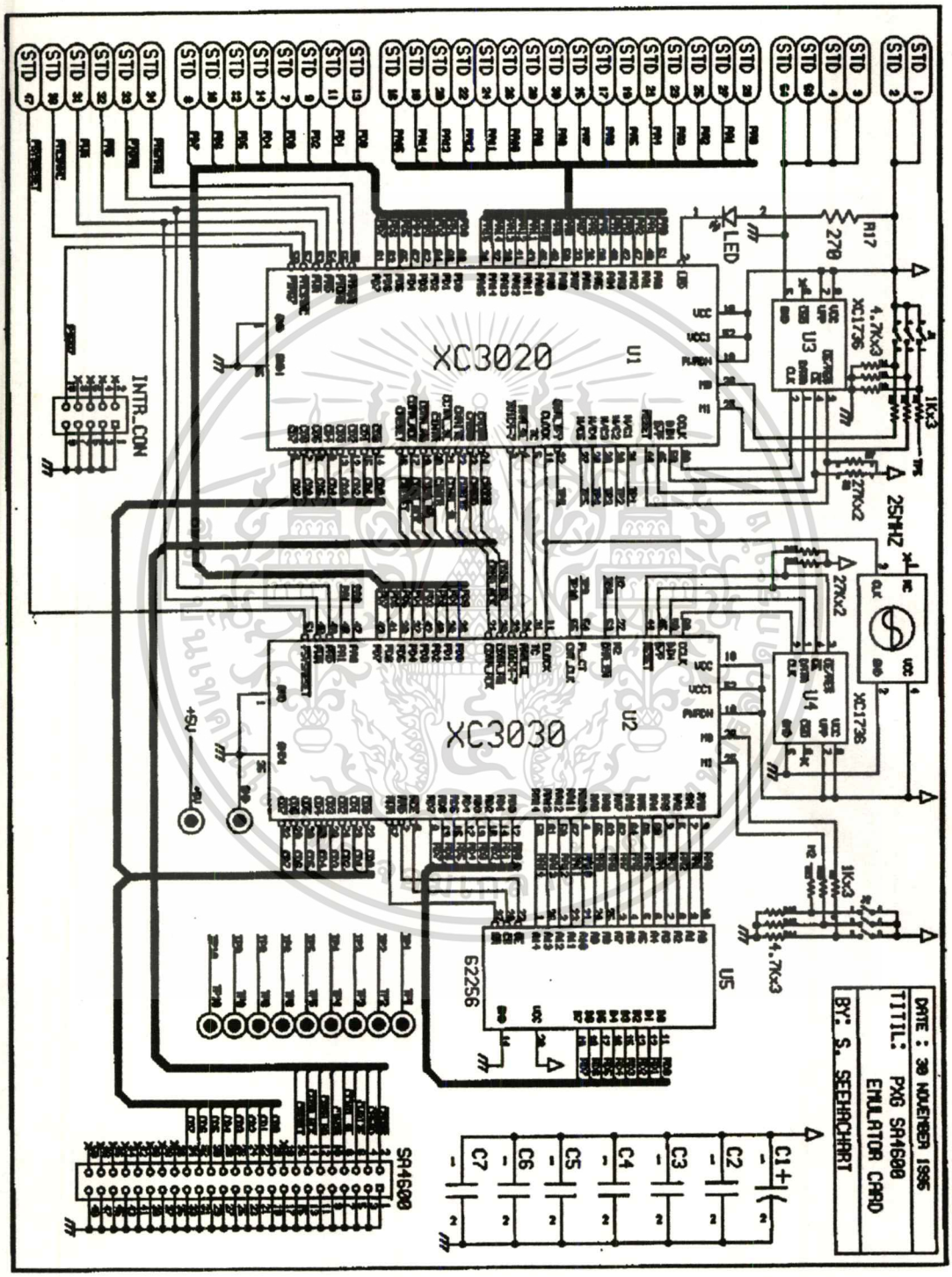
จากจังหวะเวลาของการ DMA ระหว่างเครื่อง CT และชุดเลียนแบบ SA4600 ดังแสดงใน ภาพที่ 72 และ 75 นั้น จะเห็นได้ว่า CT มีการตอบสนองต่อสัญญาณ $\overline{\text{DMA RQ}}$ เร็วมาก และ

การรับ-ส่งข้อมูล 1 ไบต์ ใช้เวลาเพียง 550 nSec. หรือ 1.8 เมกะไบต์ต่อวินาที แต่จากผลการทดลองเปลี่ยนแปลงช่วงเวลาการส่งสัญญาณ $\overline{\text{DMA RQ}}$ เพื่อขอ DMA ข้อมูลไบต์ต่อไปให้ช้าลงจาก 320 nSec. เป็น 520 nSec. ปรากฏว่าไม่ได้ทำให้ค่าเวลารวมของการทำงานเร็วขึ้นแต่อย่างไร ทั้งในการทำงานตามปกติ หรือ การทำ full initial format ซึ่งมีสาเหตุมาจาก

1. การรับ-ส่งข้อมูลของการ DMA จะกระทำเป็นช่วงๆ ซึ่งสังเกตุได้จากการตอบสนองต่อสัญญาณ $\overline{\text{DMA RQ}}$ ซึ่งบางครั้งจะนานมาก

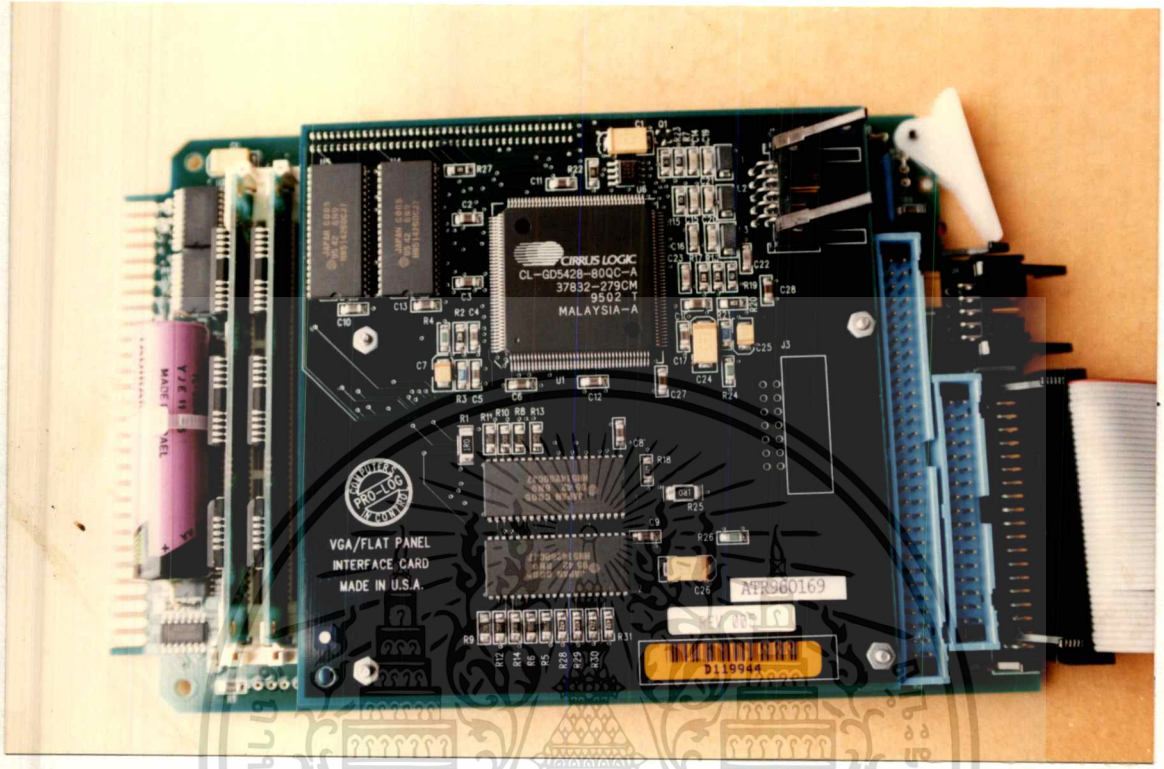
2. ข้อมูลที่ CT ทำการรับ-ส่งจากชุด SA4600 แต่ละครั้งจะอยู่ที่ประมาณ 150 ถึง 300 กิโลไบต์ (รูปสแกน 1 ภาพ) ซึ่งใช้เวลาในการ DMA น้อยมาก เมื่อเทียบกับการนำข้อมูลไปประมวลผลของเครื่อง CT ซึ่งใช้ CPU เบอร์ 8080

ภาพที่ 77

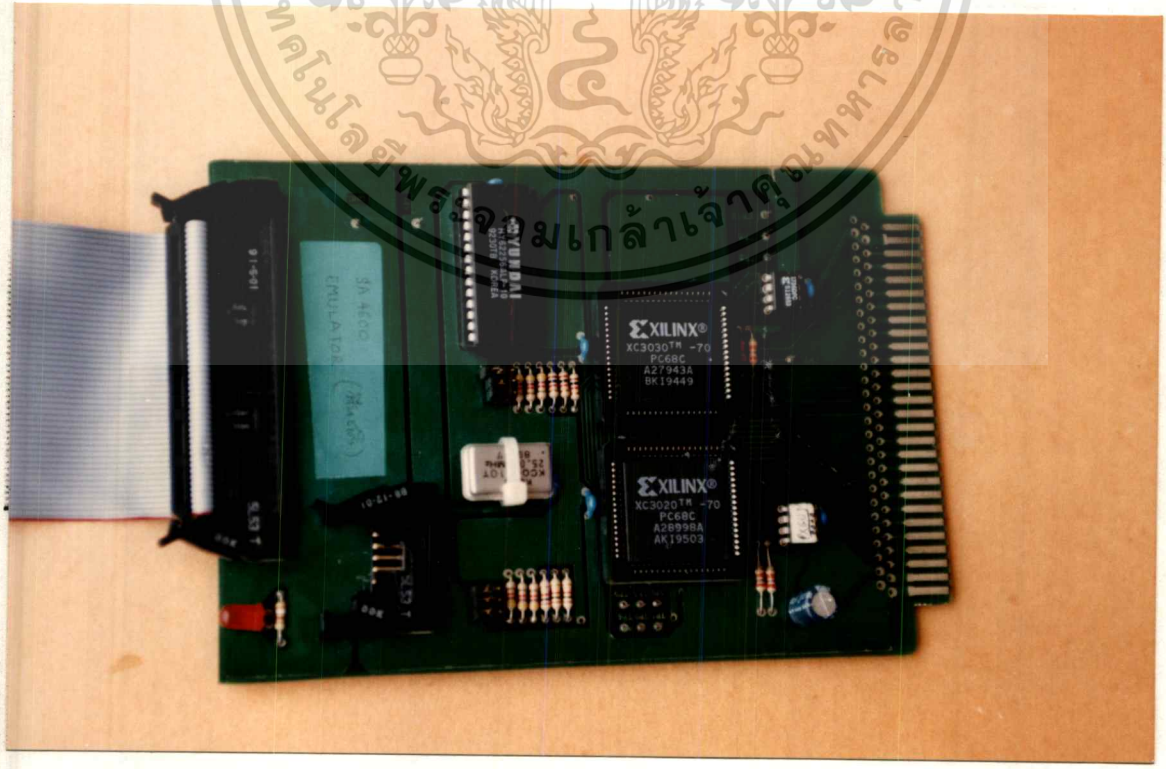



แสดงวงจรที่ออกแบบ

ภาพที่ 78 และ ภาพที่ 79

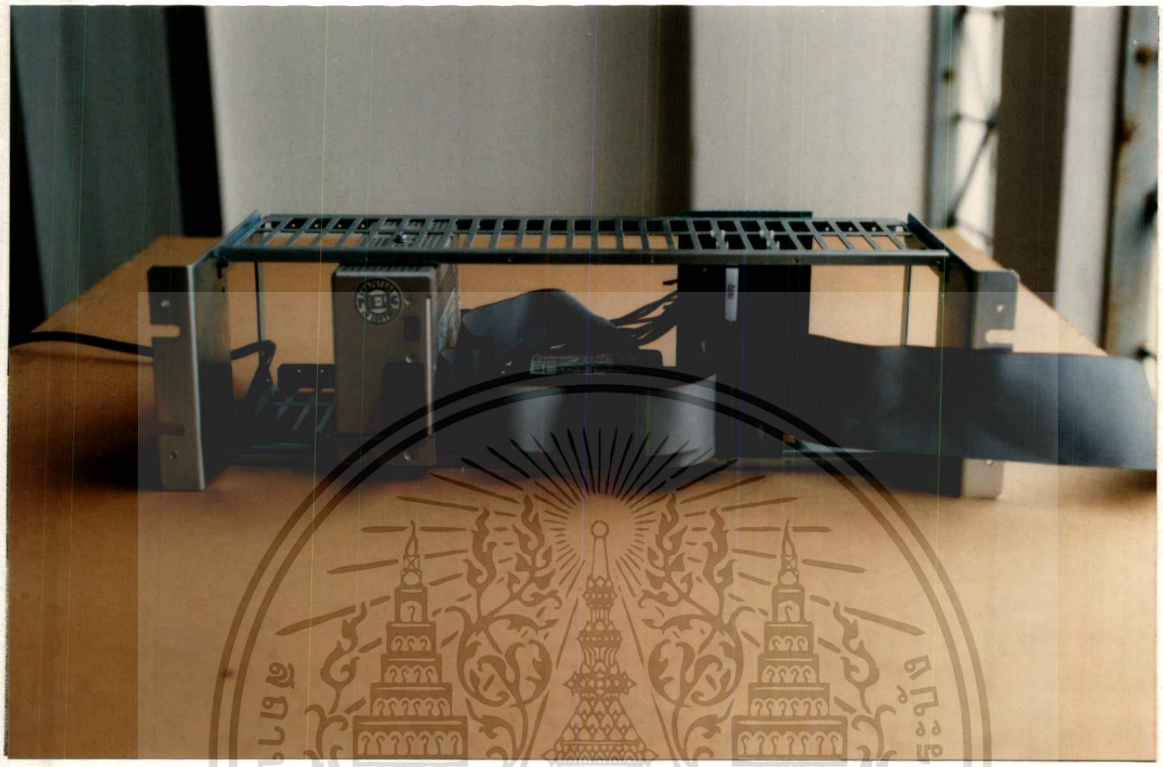



ภาพถ่ายบอร์ดคอมพิวเตอร์อุตสาหกรรมแบบบัสมาตรฐาน ยี่ห้อ PRO-LOG 7872 (386SX-25)

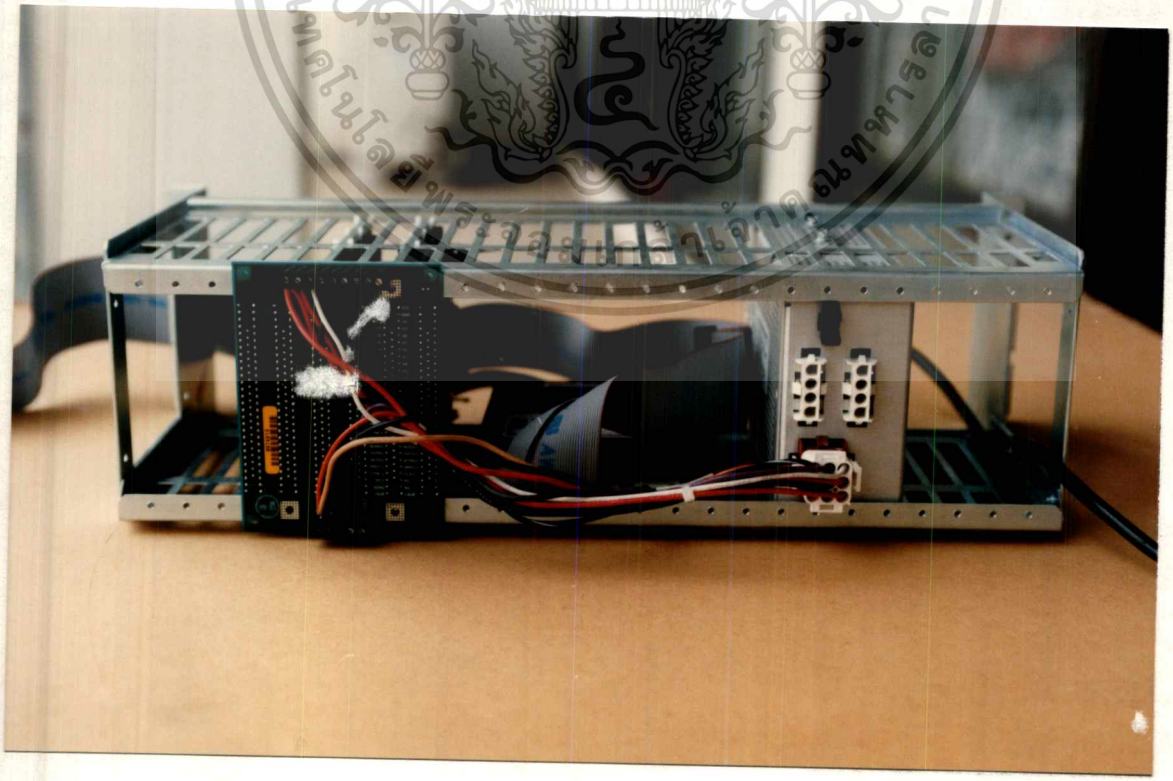



ภาพถ่ายบอร์ดเชื่อมต่อที่สร้างขึ้นโดยใช้อุปกรณ์ FPGA

ภาพที่ 80 และ ภาพที่ 81

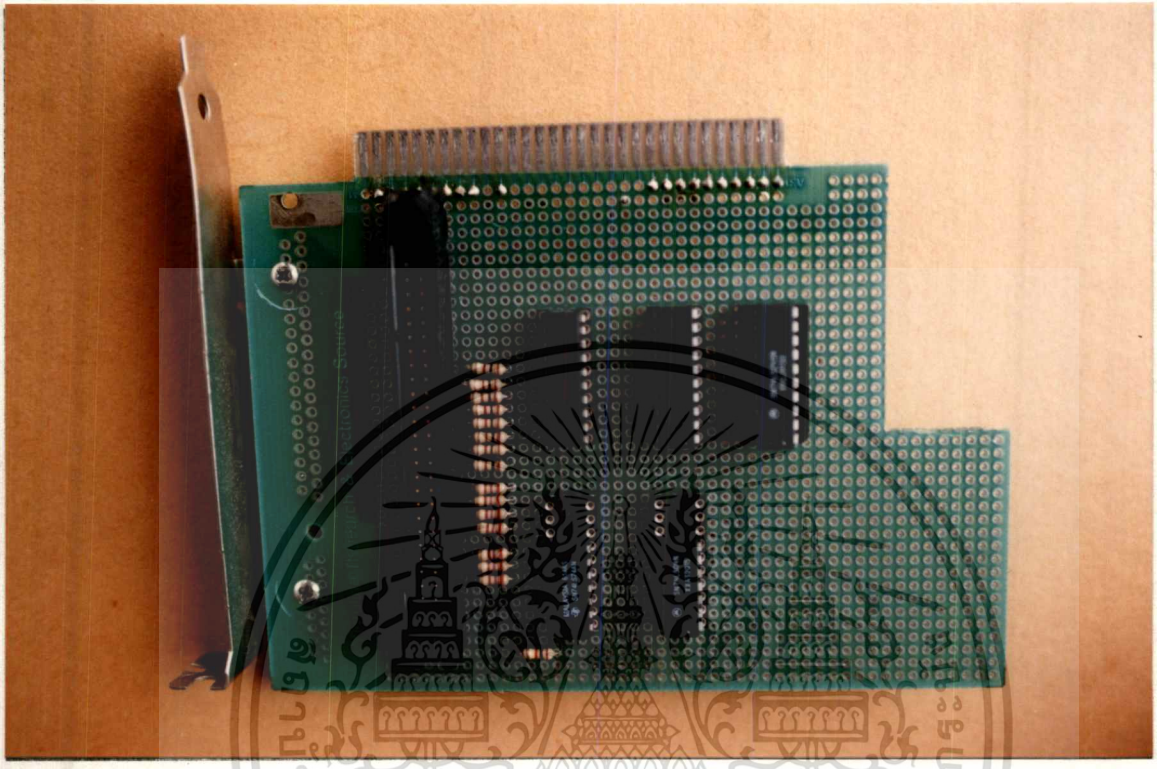


ภาพถ่ายด้านหน้าของชุดจัดเก็บข้อมูลจำลอง



ภาพถ่ายด้านหลังของชุดจัดเก็บข้อมูลจำลอง

ภาพที่ 82



ภาพถ่ายบอร์ดจำลองการทำงานของเครื่อง CT ในส่วนการติดต่อกับอุปกรณ์จัดเก็บข้อมูลแบบฟิกดิส

# บทที่ 6
# สรุปผลการวิจัยและข้อเสนอแนะ

ความก้าวหน้าทางเทคโนโลยี นอกจากจะนำมาซึ่งสิ่งประดิษฐ์ใหม่ๆที่มีความสามารถและรูปแบบการทำงานที่ดีขึ้นแล้ว ยังส่งผลกระทบให้ผลิตภัณฑ์ที่ใช้เทคโนโลยีแบบเก่าถูกยกเลิกการผลิตไปด้วย ทำให้เครื่องมือเครื่องจักรเป็นจำนวนมากไม่สามารถจัดหาอุปกรณ์ที่ใช้เทคโนโลยีแบบเก่ามาทดแทนในส่วนที่หมดสภาพไปตามอายุการใช้งานได้ รวมทั้งเครื่องคอมพิวเตอร์ถ่ายภาพตัดขวาง (computer tomography : CT) [ ยี่ห้อ (รุ่น ) หนึ่งที่มีการติดตั้งให้บริการแก่ผู้ป่วยในประเทศไทยจำนวนหนึ่ง] ซึ่งไม่สามารถจัดหาฟิกดิส ยี่ห้อ SHUGART รุ่น SA4000 มาทดแทนอุปกรณ์ที่หมดสภาพไป ทำให้มีความจำเป็นต้องมีการจำลองสร้างชุดฟิกดิส นี้ขึ้นมา เพื่อช่วยให้เครื่อง CT ทำงานได้อย่างเต็มประสิทธิภาพดังปกติ อีกทั้งช่วยยืดอายุการใช้งานเครื่องมือนี้ต่อไปให้คุ้มค่าที่สุด โดยที่ยังไม่ต้องสั่งซื้อเครื่องรุ่นใหม่เข้ามาทดแทน (ราคาของเครื่องสูงมาก)

เนื่องเพราะเครื่อง CT เป็นอุปกรณ์ทางการแพทย์ ทำให้การออกแบบชุดฟิกดิส นี้ต้องให้ความสนใจในหัวข้อต่างๆดังนี้

๑. ความน่าเชื่อถือของระบบ ได้มีการนำคอมพิวเตอร์ทางอุตสาหกรรม (industrial computer) ซึ่งมีคุณสมบัติเด่นในเรื่องการทนต่ออุณหภูมิ ความชื้น แรงสั่นสะเทือน และใช้พื้นที่ในการติดตั้งน้อย มาเป็นตัวควบคุมของระบบ รวมทั้งการนำอุปกรณ์ FPGA มาใช้ในการออกแบบชุดอินเตอร์เฟส ทำให้ช่วยลดจำนวนของอุปกรณ์ประกอบลงไปได้มาก โอกาสที่อุปกรณ์เสียจึงมีน้อยลง และทำให้ง่ายต่อการบำรุงรักษาหรือซ่อมแซมในภายหลัง

๒. ความน่าเชื่อถือของข้อมูลที่จัดเก็บอยู่บนชุดเลียนแบบ SA4600 รวมทั้งข้อมูลที่ทำการรับ-ส่งระหว่างเครื่อง CT กับชุด SA4600

๓. ความเร็วในการรับ-ส่งข้อมูล ผ่านชุด SA4600 อย่างต่ำต้องไม่ช้ากว่าขณะความเร็วที่ชุดของ SA4600 ตัวจริงสามารถกระทำได้

จากการศึกษาคู่มือระบบฟิกดิส และตามความประสงค์ของเจ้าของเครื่อง ได้มีการตกลงใจทำการจำลองตั้งแต่ส่วนของชุดควบคุมฟิกดิส (รุ่น SA4600) เนื่องจากสามารถแก้ปัญหาการจัดหาอะไหล่สำหรับการ์ดของ SA4600 ได้ด้วย

สัญญาณการติดต่อระหว่างเครื่อง CT และ SA4600 ประกอบไปด้วยสัญญาณ $\overline{\text{RESET}}$ (อินพุต) , $\overline{\text{READ}}$ (อินพุต) , $\overline{\text{WRITE}}$ (อินพุต) , $\overline{\text{CONTROL SELECT}}$ (อินพุต) , $\overline{\text{INTERRUPT}}$ (เอาต์พุต), $\overline{\text{DMA REQUEST}}$ (เอาต์พุต) , $\overline{\text{DMA ACKNOWLEDGE}}$ (อินพุต) , $\overline{\text{ADDRESS}}$ (อินพุต) และบัสข้อมูลขนาด 8 บิต (แบบสองทิศทาง) และจะทำการติดต่อกับเครื่อง CT ผ่านทางอินเตอร์เฟสรีจิสเตอร์ขนาด 8 บิต จำนวน 4 ตัว บน SA4600 ได้แก่

1. รีจิสเตอร์คำสั่ง (command register) (เขียนอย่างเดียว) ใช้เก็บคำสั่งของเครื่อง CT

2. พารามิเตอร์รีจิสเตอร์ (parameter register) (เขียนอย่างเดียว) ใช้เก็บพารามิเตอร์ของคำสั่ง

3. รีจิสเตอร์ผลลัพธ์ (result register) (อ่านอย่างเดียว) ใช้แสดงผลลัพธ์การทำงานตามคำสั่งของ SA4600

4. รีจิสเตอร์สถานะ (status register) (อ่านอย่างเดียว) ใช้เป็นอาณัติสัญญาณ (hand checking) ระหว่าง CT และ SA4600 โดยประกอบด้วย flag ต่างๆ ดังนี้

4.1 command busy ใช้แสดงแทนว่า SA4600 กำลังปฏิบัติงานอยู่

4.2 command register full แสดงสภาวะของ command register ของ SA4600

4.3 parameter register full แสดงสภาวะของ parameter register ของ SA4600

4.4 result register full แสดงสภาวะของ result register ของ SA4600

4.5 interrupt ใช้แสดงสภาวะของสัญญาณขัดจังหวะ ของ SA4600

คำสั่งที่ใช้ในการติดต่อของ SA4600 มีรวมทั้งหมด 21 คำสั่ง แต่จากการทดลองตรวจวัดการทำงานของเครื่อง CT ในสภาวะปกติ ปรากฏว่าเครื่อง CT จะมีการเรียกใช้คำสั่งของ SA4600 เพียง 8 คำสั่งเท่านั้น คือ read ID , seck , recalibrate , terminating sector request , read data , format cylinder , initialize และ write data ดังนั้น ในการจำลองสร้างชุด SA4600 นี้จะทำการสร้างให้สามารถรองรับเฉพาะคำสั่งที่มีการเรียกใช้งานจริงทั้ง 8 คำสั่งดังกล่าวนี้เท่านั้น

สำหรับคำสั่งที่มีการรับ-ส่งข้อมูล จะกระทำผ่านขบวนการที่เรียกว่า DMA ซึ่งจากการตรวจจับสัญญาณ พบว่าเครื่อง CT และ SA4600 จะมีการ DMA ด้วยความเร็วสูงกว่า 1.5 เมกะไบต์ต่อวินาที

สำหรับในการออกแบบชุด SA4600 ขึ้นมาใหม่นี้ ได้จัดแบ่งลักษณะงานออกเป็น 2 ส่วนใหญ่ๆด้วยกัน คือ

1. การสร้างสัญญาณควบคุมต่างๆ ให้เป็นไปตามข้อกำหนดของ SA4600

ซึ่งยังแบ่งย่อยออกเป็นอีกสองส่วนคือ

- สัญญาณการรับ-ส่งคำสั่งและผลลัพธ์ผ่านทาง register ทั้ง 4 ตัวข้างต้น
- สัญญาณการส่งถ่ายข้อมูลผ่านทางขบวนการ DMA

2. การเขียนโปรแกรมควบคุม เพื่อรองรับคำสั่งทั้ง 8 คำสั่งที่ CT มีการเรียกใช้งาน

ในส่วนการออกแบบ interface register ที่เชื่อมต่อกับเครื่องคอมพิวเตอร์ ได้ใช้หลักการของ input/output port ทั่วไป ซึ่งสำหรับในส่วนที่เชื่อมต่อกับเครื่อง CT ได้มีการถอดรหัส (decode) ดังนี้

- รีจิสเตอร์คำสั่ง และ พารามิเตอร์รีจิสเตอร์ ได้นำสัญญาณ $\overline{\text{WRITE}}$ จาก CT มาใช้ในการแลตช์ข้อมูลบนบัสข้อมูลลงในรีจีสเตอร์ตรงๆ เนื่องจากต้องการข้อมูลของบัสข้อมูลในขณะที่สัญญาณ $\overline{\text{WRITE}}$ เปลี่ยนสภาวะจาก low เป็น high และเนื่องจาก CT จะไม่มีการส่งพารามิเตอร์มาให้ในขณะที่ข้อมูลในรีจิสเตอร์คำสั่ง ยังไม่ถูกนำออกไปตีความหรือถอดรหัส ดังนั้นจึงสามารถใช้ รีจีสเตอร์ขนาด 8 บิต เพียงตัวเดียว ในการทำงานเป็นทั้งรีจิสเตอร์คำสั่ง และ พารามิเตอร์รีจิสเตอร์

- รีจิสเตอร์ผลลัพธ์ จะนำสัญญาณ $\overline{\text{READ}}$ ของ CT มาทำการหน่วงเวลาเพิ่ม 80 nSec. ก่อนที่จะใช้เป็นสัญญาณ output enable ของรีจีสเตอร์ เนื่องจากต้องการให้ข้อมูลของรีจิสเตอร์ผลลัพธ์บนบัสข้อมูลคงอยู่อย่างน้อย 42 nSec. ตามข้อกำหนดของ SA4600

- รีจิสเตอร์สถานะ เนื่องจากว่ารีจิสเตอร์สถานะถูกนำมาใช้เป็นอาณัติสัญญาณ ซึ่งต้องสามารถอ่าน-เขียนข้อมูล ได้ทั้งจากเครื่องคอมพิวเตอร์และจากเครื่อง CT จึงได้ใช้ฟลิบฟลอบมาทำหน้าที่ในการเก็บข้อมูลในแต่ละบิต โดยทางคอมพิวเตอร์จะเปลี่ยนแปลงค่าของรีจิสเตอร์สถานะ ผ่านทางขา data ของตัวฟลิบฟลอบ และทางเครื่อง CT จะเปลี่ยนแปลงค่าของบิตต่างๆ ในรีจิสเตอร์สถานะ ผ่านทางขา preset/clear ของตัวฟลิบฟลอบ และในการอ่านข้อมูลจากรีจิสเตอร์สถานะนั้นจะใช้บัฟเฟอร์จำนวนสองตัว ในการแยกนำข้อมูลรีจิสเตอร์สถานะ ส่งต่อไปยังเครื่องคอมพิวเตอร์และ CT เพื่อป้องกันไม่ให้ข้อมูลในบัสข้อมูลชนกัน ในขณะที่ทำการอ่าน-เขียนข้อมูล พร้อมกันทั้งสองด้าน

สำหรับการออกแบบสัญญาณ DMA ซึ่งขบวนการ DMA เป็นตัวกำหนดประสิทธิภาพของระบบโดยรวม และจากอัตราการโอนถ่ายข้อมูลที่มีความเร็วสูงกว่า 1.5 เมกะไบต์ต่อวินาที ทำให้การรับ-ส่งข้อมูล ต้องอาศัยการทำงานด้วยอุปกรณ์ฮาร์ดแวร์ทั้งหมด ซึ่งมีคุณสมบัติดังนี้

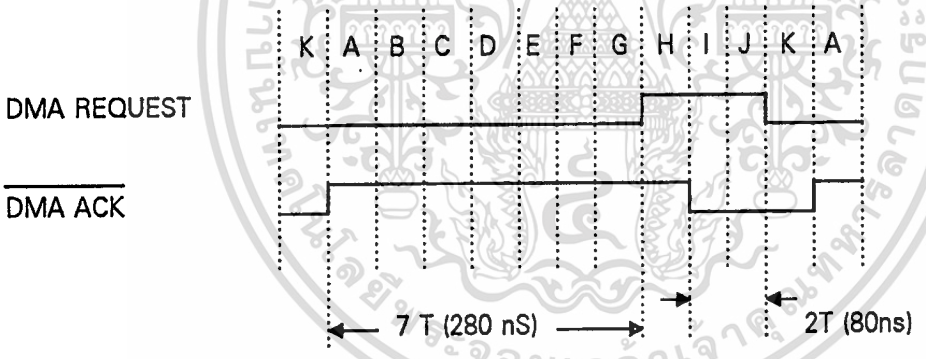
- อ่าน-เขียนข้อมูลได้ทั้งกับเครื่องคอมพิวเตอร์และ CT ได้ทำการออกแบบวงจรแรมแบบสถิตแบบสองทิศทาง (two-way static RAM) เข้ามาทำหน้าที่เป็นตัวกลางในการ DMA และเนื่อง

จากว่าไม่มีสัญญาณแสดงทิศทางของการ DMA จึงต้องทำการสร้างสัญญาณควบคุมเฉพาะขึ้นมา เพื่อกำหนดทิศทางของการ DMA ผ่านทางแรมแบบสถิต

- ไม่มีสัญญาณแสดงตำแหน่งข้อมูลที่ทำการ DMA จึงใช้ presetable 16 bit counter เป็นตัวสร้างสัญญาณตำแหน่งให้กับแรมแบบสถิต และใช้สัญญาณ $\overline{\text{DMA ACK}}$ จากเครื่อง CT มากระตุ้นให้ตัวนับ เพื่อเตรียม DMA ข้อมูลไบต์ต่อไป

- ในบวนการ DMA ชุด SA4600 จะเป็นฝ่ายเริ่มส่งสัญญาณ $\overline{\text{DMA REQUEST}}$ เพื่อทำการร้องขอ DMA ข้อมูลในแต่ละไบต์ ดังนั้นจึงต้องสร้างสัญญาณ $\overline{\text{DMA REQUEST}}$ นี้ให้เป็นแบบ free running และหยุดเองเมื่อทำการ DMA ครบหมดทุกไบต์ ซึ่งในการออกแบบได้มีการสร้างสัญญาณ DMA ENABLE เพื่อเริ่มขบวนการ DMA และใช้สัญญาณ terminate counted จากตัวนับขนาด 16 บิต เป็นตัวจบขบวนการ DMA

ในช่วงที่เป็น free running ได้ออกแบบโดยใช้ time base 40 nSec. แบบซิงโครนัส ตามสถานะดังนี้




ภาพแสดงสถานะของสัญญาณ DMA REQUEST

ในส่วนของการเขียนโปรแกรมควบคุม ส่วนของการรับ-ส่ง คำสั่งและผลลัพธ์ จะใช้หลักการอินพุต/เอาต์พุต polling ผ่านทางอินเตอร์เฟสรีจิสเตอร์ เมื่อได้รับคำสั่งให้ทำงานกับฮาร์ดดีส จะทำการแมป (map) ตำแหน่งของฟิกดิสเดิมให้มาเป็นตำแหน่งในฮาร์ดดีสใหม่ ซึ่งเราควบคุมอยู่ตามหลักการต่อไปนี้

- ทำการแบ่งส่วน (partition) ของฮาร์ดดีส ออกเป็น 2 drive โดยใช้ drive D เป็นตัวเก็บข้อมูลแทนฟิกดิสเดิม และนำโปรแกรมควบคุมไปไว้ที่ drive C

- ใช้คำสั่ง BIOS interrupt 24,25 ซึ่งเป็น absolute read/write ในการอ่าน-เขียนข้อมูล ในโลจิคอล drive D

- คำนวณ absolute sector จากพารามิเตอร์ของ cylinder , head และ sector ดังนี้

absolute sector = [ (cylinder) * 256 ] + (head * 32) + sector

- ในการรับ-ส่งข้อมูลจากเครื่องคอมพิวเตอร์กับแรมแบบสถิต จะใช้วิธีอ้างแรมแบบสถิตเป็นหน่วยความจำของเครื่อง และใช้คำสั่ง REP MOVSB ในการส่งถ่ายข้อมูลเป็นบล็อก

สำหรับในการออกแบบชุดฮาร์ดแวร์ดังกล่าวข้างต้น ได้นำเอาอุปกรณ์ FPGA ของ Xilinx ตระกูล 3000 มาใช้เป็นอุปกรณ์หลัก ซึ่งส่งผลให้มีคุณสมบัติการทำงานได้ตรงตามความต้องการ รวมถึงคุณสมบัติที่ดีหลายประการของอุปกรณ์ FPGA เอง

ในการพัฒนาชุดเลียนแบบ SA4600 (emulator) ได้มีการจำลองสร้างชุด CT ขึ้นมา เพื่อแก้ปัญหาในด้านค่าใช้จ่ายและความไม่สะดวกในการเดินทางไปทดลองกับเครื่อง CT ในสถานที่จริง โดยชุดของ CT ที่จำลองขึ้นมานี้ จะช่วยในการทดสอบเกี่ยวกับหลักการทำงานพื้นฐานของชุด SA4600 เป็นต้นว่า การอ่านคำสั่งและพารามิเตอร์ต่างๆ การเซ็ต รีจิสเตอร์ผลลัพธ์ หรือรีจิสเตอร์สถานะ รวมทั้งการรับ-ส่งข้อมูลผ่านทาง DMA โดยจะทำการทดสอบที่ความเร็วต่ำกว่าเครื่อง CT ของจริง

จากการนำชุดเลียนแบบ SA4600 ที่พัฒนาขึ้นมาโดยใช้อุปกรณ์ FPGA เป็นหลัก ไปติดตั้ง ณ. สถานที่จริง ตั้งแต่เดือนธันวาคม ๒๕๓๘ ปรากฏว่าได้ผลการทำงานถูกต้องเป็นที่น่าพอใจ ทั้งในด้านความน่าเชื่อถือของข้อมูล และความเร็วในการรับ-ส่งข้อมูล และจากการทดลองปรับจังหวะเวลา (timing) ของการ DMA ปรากฏว่า ความเร็วในการทำงานโดยรวมถูกจำกัดอยู่ที่เครื่อง CT ซึ่งใช้ตัวประมวณผล CPU # 8088 เนื่องการการทำงานส่วนใหญ่ของเครื่อง CT จะอยู่ที่การนำข้อมูลไป process (ทดลองโดยการสั่ง full initialize format ซึ่งกินเวลาประมาณ 50 นาที เท่ากันกับ fixed disk ตัวจริงของดั้งเดิม)

# บรรณานุกรม

อัครินทร์ คุณกิตติ. การพัฒนาและการสร้างอุปกรณ์ถ่ายภาพตัดขวาง. ปริญญานิพนธ์ วิศวกรรมศาสตร์มหาบัณฑิต. กรุงเทพฯ : บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2537.

Jesses H. Jenkins. signingDe with FPGAs and CPLDs : Prentice-Hall Inc., 1994.

Ting-Ting Hwang and others. "Logic Synthesis for Field-Programmable Gate Arrays." IEEE Trans. Computer-Aided Design of integrated circuits and systems, vol.13, October 1994, pp.1280-1287.

Neil J. Howard, Andrew M. Tyrrell and Nigel M. Allinson. "The Yield Enhancement of Field-programmable Gate Arrays," IEEE Trans. VLSI system, vol.2, March 1994, pp.115-123.

Kenneth J. Breeding. Digital Design Fundamentals. Prentice-Hall Inc., 1992.

Xilinx, Inc. The Programmable Gate Array Data Book. San Jose:Xilinx, 1994.

# ภาคผนวก

# โปรแกรมควบคุมชุดเลียนแบบ SA4600

```
// SA4600 HEADER FILE
#define HDRD_CMD_PARA    0x1c0   // read command or parameter register
#define HDSET_RSLT       0x1c1   // set result register
#define HDRD_STAT        0x1c2   // read status register
#define HDSET_STAT       0x1c3   // set status register
#define LATCH_CNTR     0x1C7  // Latch Down counter pulse.
#define CMD_FULL_BIT     0x02    // command full mask bit
#define PARA_FULL_BIT    0x04    // parameter full mask bit
#define STAT_BIT         0x1f    // status mask bit
#define RSLT_FULL_BIT    0x08    // result full mask bit
#define INT_BIT          0x10    // interrupt mask bit
#define INIT_STATUS      0x00    // status initial value
#define DMA_IN_OUT       0x20    // DMA in/out mask bit
#define DMA_ENABLE       0x80    // enable DMA request
#define DMA_ACK          0x20    // DMA acknowledge mask bit
 /*
note INIT_STATUS bit is mean that
    bit 0 = 0      :command busy bit off
    bit 1 = 0      :command full bit off
    bit 2 = 0      :parameter full bit off
    bit 3 = 0      :result full bit
    bit 4 = 0      :interrupt not activate
    bit 5 = 0      :DMA out
    bit 6 = 0      :clear status bit off
    bit 7 = 0      :not DMA
 */
```

```
#define CLR_STAT_BIT                    0x40// clear status mask bit
#define READ_ID                 0x00
#define READ_DIAGNOSTIC         0x01
#define VERIFY_DATA             0x02
#define VERIFY_DATA_SPECIAL     0x03
#define SEEK                    0x04
#define RECALIBRATE             0x05
#define TERMINATING_SECTOR_REQUEST  0x07
#define READ_DATA               0x08
#define READ_DATA_SPECIAL       0x09
#define SEARCH_DATA_EQUAL       0x0a
#define SEARCH_DATA_EQUAL_SPECIAL   0x0b
#define SEARCH_HIGH_EQUAL       0x0c
#define SEARCH_HIGH_EQUAL_SPECIAL   0x0d
#define SEARCH_LOW_EQUAL        0x0e
#define SEARCH_LOW_EQUAL_SPECIAL    0x0f
#define WRITE_ID                0x10
#define FORMAT_CYLINDER         0x11
#define INITIALIZE              0x12
#define WRITE_DATA              0x18
#define WRITE_SPECIAL_DATA      0x19
#define WRITE_BUFFER            0x1a
#define ERROR                   0xff
#define TRUE                    0
#define FALSE                   1
#define BUFFSIZE                32*1024
```

```
// ADD-ON SA4600 PORT
#define CMD  0x1C6   // set command for counter
#define L_BYTE 0x1C4 // set low byte in counter
#define H_BYTE 0x1C5 // set high byte in counter
#define TC 0x20 // terminate counter
//---------- GOBAL VARIABLE ----------

// command information
struct
CCOMM {
                unsigned char parameter;
                int function;
} COMM[255];
unsigned char COMMAND;      // current command
unsigned char PARAMETER;    // current parameter lenght
unsigned char PARMIFO[5];   // parameter information

 /*
   note PARMIFO consist of 4 important information are as follows
     PARMIFO[0] : bit 0 - 2 is drive address:drive 0-3 are fix disk,
                          4 - 7 are floppy disks.
                  bit 3     not used.
                  bit 4 - 6 Head address: bit 4 is second side for
                            2 side floppy's.
                  bit 7     Fixed Head:The head address is interpreted
                            as fixed head address.
     PARMIFO[1] : bit 0 - 7 is define as a cylinder address.
     PARMIFO[2] : bit 0 - 6 is define as a starting sector address.
                          bit 7not used.
```

```
    PARMIFO[3] : bit 0 - 7 is define as a multiple sector count
*/

char DISKBUFF[BUFFSIZE];
int SECSIZE =512;
unsigned char HEAD;// current head address
unsigned char CYL;// current cylinder address
unsigned char SEC ;// current sector address
unsigned char nSEC;// current number of cylinder
char INTERLEAVE;    // disk interleave code
int TransSec;      // transfer sector / time (must use 64 for porper opration)
char Logno[20];    // bios s/n
//BIOS DISK TO SA4600 COMPAIR TABLE
unsigned char RESULT[255];
int  done = 1;

//---------- pototype function declaretion ----------
unsigned char GetCmd(void);
unsigned char GetParm(void);
void SetRslt(unsigned char value);
void InitStat(void);
int CheckComm(unsigned char value);
int RcvParm(void);
int (*FUNCTION)(void);
// SA4600 Emulation Command declaretion
// return result of operation
int ReadID(void);
int ReadData(void);
int WriteData(void);
```

```
void DMA_out(unsigned int count,unsigned char *buffer);
void DMA_in(unsigned int count);
void warm_boot(void);
void ChkBiosSN(void);
int initial(void);
int recalib(void);
int seek(void);
int termsec(void);
int SearchEq(void);
int format(void);
int Invalid(void);
extern int sa_read(int head,int track,int sector,int nsects,void *buffer);
extern int sa_write(int head,int track,int sector,int nsects,void *buffer);
extern int get_abs(int head,int cyl,int sec);
extern int sa_seek(int head,int track,int sector);
extern int sa_format(int head,int cyl);
void interrupt reset_ISR(void);
void set_int(void);
void clr_stat(void);
void self_test(void);

// ---------- ADD-ON
extern void ck_tc(void);
extern void RAM2PL(unsigned int count,unsigned char *buffer);
void SetSram(unsigned int count);
void pl2sram(unsigned int count, unsigned char *buffer);
void sram2pl(unsigned int count, unsigned char *buffer);
```

```
// SA4600 FIX DISK CONTROLLER EMULATOR
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>
#include "C:\XRAY\NSA4600.H"
// ---------------------

#define DRIVE_ERR  0x08
int PARMERR = 0;          // error flag
unsigned int OLD_SP;
int save_file(void);
int CommProcess(void);

void main()
{
    set_int();
    enable();
    asm mov ax,sp
    asm mov OLD_SP,ax
    warm_boot();
}

//---------- warm boot ----------
void warm_boot(void)
```

```
{
    unsigned char i;
    asm cli
    asm mov ax,OLD_SP
    asm mov sp,ax
    cprintf("SA4600 FIXED DISK CONTROLLER EMULATOR V.1995\r\n");
    InitStat();
    for(;;) {
        asm sti
        i = GetCmd();
        if (CheckComm(i) != TRUE) {
            printf("Invalid Command ...%x \n",i);
            SetRslt(0x2c);  // invalid command
            while ((inportb(HDRD_STAT) & RSLT_FULL_BIT) != 0);
        }
        else {
            if (RcvParm() != TRUE) {
                printf("Parameter not load...");
                SetRslt(0X0c); // illegal length
                while ((inportb(HDRD_STAT) & RSLT_FULL_BIT) != 0);
            }
            else {
                CommProcess();
                asm {  // wait until result readed by CT
                    mov   dx,HDRD_STAT
                    in    al,dx
                    and   al,RSLT_FULL_BIT
                    jnz   $-3
                }
```

```
            }
        }
    }
}

//---------- read command ----------
unsigned char GetCmd(void)
{
    unsigned char status,cmd,i;
   asm {
        mov   dx,HDRD_STAT
        in    al,dx
        and   al,CMD_FULL_BIT
        jz    $-3
      mov   al,10              // delay untils write signal go up
      dec   al                 // for sure
      jz    $-2
        mov   dx,HDRD_CMD_PARA    // read command
        in    al,dx
        mov   cmd,al
        mov   dx,HDSET_STAT       // clear command full ,keep command busy
        mov   al,1
        out   dx,al
   }
    return (cmd);
}

//---------- set result register ----------
void SetRslt(result)
```

```
unsigned char result;
{
    unsigned char status;
    asm {
        mov  dx,HDSET_RSLT        // load result to result register
        mov  al,result
        out  dx,al
        mov  dx,HDRD_STAT         // point to HDRD_STAT port
      jmp  short $+2
      jmp  short $+2
        in   al,dx                // get old status
      jmp  short $+2
        and  al,STAT_BIT          // mark only status bit
        or   al,18h               // set Result Reg Full & Interrupt flag
        mov  dx,HDSET_STAT        // point to HDSET_STAT port
        out  dx,al
    }
}

//------------ initial system status ------------
 void InitStat(void)
 {
    unsigned char status;
    int i;
    outportb(CMD,0x0C);// LOAD=1 , PL/~CT=1 ,disable DMA
    status = INIT_STATUS | CLR_STAT_BIT;      // reset status on
    outportb(HDSET_STAT,status);
    status &= ~CLR_STAT_BIT;// reset status off
```

```
        and     al,7                // specified bit 0-3
        mov     head,al
        inc     bx                  // point to parameter # 2
        mov     al,[bx]             // get 2nd parameter
        mov     cyl,al              // as starting cylinder addr.
        inc     bx                  // point to parameter # 3
        mov     al,[bx]             // get 3th parameter
        mov     sec,al              // as starting sector addr.
        mov     al,COMMAND          // check if multiple sector option
        and     al,40h
        jz      $+9
        inc     bx                  // point to parameter # 4
        mov     al,[bx]             // get 4th parameter
        or      al,al               // if nsec = 0 then nsec = 1
        jnz     $+4
        mov     al,1                // default with 1
        mov     nSEC,al             // as Multiple sector count
    }

    result =sa_read(head,cyl,sec,nSEC,DISKBUFF);
    if (result != 0x00 ) {
        SetRslt(0x24); //CRC error
        return 1;
    }
    SetRslt(0);// load result to result register
    return 0;
}
```

```
//---------- Write File ----------
int WriteData(void)
{
    unsigned char head,cyl,sec;
    int result;

    asm {                       // get parameter
        mov     bx,offset PARMIFO   //
        mov     al,[bx]             // al = parameter # 1
        mov     cl,4                // get head # by shift right 4 bits
        shr     al,cl
        and     al,7                // specified bit 0-3
        mov     head,al
        inc     bx                  // point to parameter # 2
        mov     al,[bx]             // get 2nd parameter
        mov     cyl,al              // as starting cylinder addr.
        inc     bx                  // point to parameter # 3
        mov     al,[bx]             // get 3th parameter
        mov     sec,al              // as starting sector addr.
        mov     al,COMMAND          // check if multiple sector option
        and     al,40h
        jz      $+9
        inc     bx                  // point to parameter # 4
        mov     al,[bx]             // get 4th parameter
        or      al,al               // if nsec = 0 then nsec = 1
        jnz     $+4
        mov     al,1                // default with 1
        mov     nSEC,al             // as Multiple sector count
    }
```

```
    result =sa_write(head,cyl,sec,nSEC,DISKBUFF);
    if (result != 0x00 ) {
        SetRslt(0x38); // write error
         return 1;
    }
    SetRslt(0);// load result to result register
    return 0;
}

void SetSram(unsigned int count)
{
  // set static ram addr.
  asm {
      mov    ax,count   //cal. start addr. of SRAM for transfer count bytes
      xor    ax,0FFFFh    //start addr. = 10000h - count
      mov    dx,L_BYTE  //point to F/F that store low addr. data
      out    dx,ax          //set counter start addr. to F/F
      mov    dx,CMD      //point to signal control port
      mov    al,0Ch        //set Load high , sele prolog , dis DMA
      out    dx,al
      push   ds          //gen. clock to load start addr. to Counter
      mov    ax,0d000h
      mov    ds,ax
      mov    al,ds:[9]
      pop    ds
      mov    al,04       //clear Load signal , sele prolog , dis DMA
      out    dx,al
  }
}
```

```
void pl2sram(unsigned int count, unsigned char *buffer)
{
  // PL => transfer data to SRAM
  asm {
        push es
        push si
        push di
        mov  dx,CMD     // clear Load signal , sele prolog , dis DMA
        mov  al,04      //
        out  dx,al
        mov  cx,count
        mov  si,buffer // source point by ds:si
        mov  ax,0D000h
        mov  es,ax   // destination point to address D000:0
        mov  di,0
      cld
        rep  movsb      // transfer to SRAM
        pop  di
        pop  si
        pop  es
    }
}

void sram2pl(unsigned int count, unsigned char *buffer)
{
 // PL read SRAM
  asm {
      push si
      push di
```

```
        push es
        push ds
        mov  ax,ds
        mov  es,ax      // es point to old ds
        mov  dx,CMD     // clear Load signal , sele prolog , dis DMA
        mov  al,04      //
        out  dx,al
        mov  cx,count
        mov  ax,0d000h  // source point by d000:0
        mov  ds,ax
        mov  si,0
        mov  di,buffer  // destination point to es:buffer
        cld
        rep  movsb      // SRAM transfer to PL
        pop  ds
        pop  es
        pop  di
        pop  si
    }
}
//---------------- DMA out ----------------------
void DMA_out(unsigned int count,unsigned char *buffer){
    //set counter for PL => SRAM
    SetSram(count);
    // PL => transfer data to SRAM
    pl2sram(count,buffer);
    // set counter for SRAM => CT
    SetSram(count);
    // clear Load signal ,sele CT READ, enable DMA
```

```
    asm   mov  dx,CMD
    asm   mov  al,3
    asm   out  dx,al
}

void ck_tc(void){
  asm  {
        mov  dx,HDRD_STAT     // point to status register
        in   al,dx
        and  al,TC            // check TC bit
        jz   $-3
        mov  dx,CMD           // disable DMA
        mov  al,0Ch           // set Load High , set to Prolog
        out  dx,al
  }
}

//---------------- DMA in ----------------
void DMA_in(unsigned int count){
    //set counter for CT => SRAM
    SetSram(count);
    // clear Load signal ,sele CT WRITE, enable DMA
    asm   mov  dx,CMD
    asm   mov  al,1
    asm   out  dx,al
}

//------ ADD-ON
void RAM2PL(unsigned int count,unsigned char *buffer ){
```

```
    //set counter for SRAM => PL
    SetSram(count);
    // SRAM => transfer data to PROLOG
    sram2pl(count,buffer);
}

//------------ initial fixdisk parameter ------------
int initial(void)
{
    unsigned char head,cyl,sec;
    int result;
    asm {                       // get parameter
        mov     bx,offset PARMIFO   //
        mov     al,[bx]             // al = parameter # 1
        mov     cl,4                // get head # by shift right 4 bits
        shr     al,cl
        and     al,7                // specified bit 0-3
        mov     head,al
        inc     bx                  // point to parameter # 2
        mov     al,[bx]                 // get 2nd parameter
        mov     cyl,al              // as starting cylinder addr.
        inc     bx                  // point to parameter # 3
        mov     al,[bx]                 // get 3th parameter
        mov     sec,al              // as starting sector addr.
        mov     al,COMMAND              // check if multiple sector option
        and     al,40h
        jz      $+9
        inc     bx                  // point to parameter # 4
        mov     al,[bx]                 // get 4th parameter
```

```
        or      al,al               // if nsec = 0 then nsec = 1
        jnz     $+4
        mov     al,1                // default with 1
        mov     nSEC,al             // as Multiple sector count
    }
    SetRslt(0);// good commpatition
    return 0;
}
//---------------- recalibrate ----------------
int recalib(void)
{
    int result;
    result =sa_seek(0,0,0,1);  // seek to home position
    SetRslt(0);// load result to result register
    HEAD = 0;// up date current head,cyl,sec
    CYL  = 0;
    SEC  = 0;
    return (0);
}

//---------------- Seek ----------------
int seek(void)
{
    int result;
    unsigned char head,cyl;

    asm {                       // get parameter
        mov     bx,offset PARMIFO   //
        mov     al,[bx]             // al = parameter # 1
```

```
        mov     cl,4                // get head #  by shift right 4 bits
        shr     al,cl
        and     al,7                // specified bit 0-3
        mov     head,al
        inc     bx                  // point to parameter # 2
        mov     al,[bx]             // get 2nd parameter
        mov     cyl,al              // as cylinder #
    }

    result =sa_seek(head,cyl,1);
    if (result != 0x00) {
        SetRslt(0x04);  // set seek error
        return(1);
    }
    SetRslt(0x40);  // seek complete
}

//---------- terminate sector request ----------
int termsec(void)
{
    unsigned char flag=0; // deflective flag
    unsigned char buff[7];
    buff[0] = CYL;      // sent current sector request
    buff[1] = HEAD;
    buff[2] = SEC;
    buff[3] = flag;
    buff[4] = CYL;
    buff[5] = HEAD;
    buff[6] = SEC;
```

```
    DMA_out(7,buff);
    ck_tc();
    SetRslt(0);          // good completition
    return 0;
}

//------------------ Search ------------------
int SearchEq(void)
{
    SetRslt(0x20);  // set search met equal
    return 0;
}

//------------------ Format Fixed disk ------------------
int format(void)
{
    int result,retry;
    unsigned char head,cyl,j;
    asm {                          // get parameter
        mov     bx,offset PARMIFO   //
        mov     al,[bx]             // al = parameter # 1
        mov     cl,4                // get head #  by shift right 4 bits
        shr     al,cl
        and     al,7                // specified bit 0-3
        mov     head,al
        inc     bx                  // point to parameter # 2
        mov     al,[bx]             // get 2nd parameter
        mov     cyl,al              // as cylinder #
    }
```

```
    result = sa_seek(head,cyl,0);
    if (result != 0x00) {
        SetRslt(0x38);  // set write error
        return(1);
    }
    SetRslt(0);
    return (0);
}

//------------ Invalid Command Service ------------
int Invalid(void)
{
    SetRslt(0x2C);  //set invalid command
    return 0;
}
//-------- sa4600 reset signal interrupt service routine --------
void interrupt reset_ISR(void)
{
    outportb(HDSET_STAT,0); // clear status register
    asm mov word ptr bp[18],offset warm_boot // return main
    asm mov word ptr bp[20],seg warm_boot
    asm mov al,20h// end of interrupt
    asm out 20h,al
}
//-------- set interrupt for service sa4600 reset signal --------
void set_int(void)
{
    asm  cli
    setvect(0x0f,reset_ISR);// set irq7 to service routine
```

```
    asm  in al,21h// enable irq7
    asm  and al,01111101b
    asm  jmp short $+2
    asm  out 21h,al
    asm  sti
}

/*-----------------------
 SA4600 DISK UTILITY PROGRAM
-----------------------*/
#include <bios.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
// SA4600 parameter
const SA_Hd   = 8;      // number of head
const SA_Cy   = 202;    // number of cylinder
const SA_Sec  = 32;     // number of sector
char SEEKBUFF[515];
// gobal variable
extern char DISKBUFF[];
extern unsigned char PARMIFO[];
extern char RESULT[];
extern unsigned char HEAD;
extern unsigned char CYL;
extern unsigned char SEC;
extern int SECSIZE;
// prototype function declaretion
```

```
extern  void SetRslt();
extern  void DMA_out(unsigned int count,unsigned char *buffer);
extern  void DMA_in(unsigned int count);
int     sa_read(int head,int track,int sector,int nsects,void *buffer);
int     sa_write(int head,int track,int sector,int nsects,void *buffer);
unsigned int get_abs(int head,int cyl,int sec);
int     check_para(int head,int cyl,int sec,int nsec,int chknsec);
int     sa_seek(int head,int track,int sector);
//—— ADD-ON
extern void RAM2PL(unsigned int count,unsigned char *buffer );
//—— seek PC fix disk to specific place that define by SA4600 ——
int sa_seek(int head,int track,int sector)
{
    unsigned int abs_sec,result;
    if (check_para(head,track,sector+1,0,0)!=0)
        return (0x07); // command error illegal length
    abs_sec = get_abs(head,track,sector+1);// caculate absulute sector
    result = absread(3,1,abs_sec,SEEKBUFF);//use read instead seek
    if (result != 0)
        return (result);
    HEAD = (unsigned char) head;
    SEC  = (unsigned char) sector;
    CYL  = (unsigned char) track;
    return (0);
}
// ———————— SA4600 READ DISK ————————————
int sa_read(int head,int track,int sector,int nsects,void *buffer)
{
    int result;
```

```
unsigned int abs_sec,k;
div_t x;
int first=1;
if (check_para(head,track,sector,nsects,1) !=0)
    return (0x07); // command error illegal length
while (nsects > 64) {
    abs_sec = get_abs(head,track,sector+1);
     result = absread(3,64,abs_sec,buffer);// read absolute
     head += 2;   //update head,sector
     nsects -=64;
     if (result != 0){
       perror("READ error nsects > 64");
       return (-1);
     }
     // DMA out
     if(!first){
       ck_tc(); //check terminate counter
     }
     DMA_out((64*SECSIZE),buffer);
     first = 0;
}
abs_sec = get_abs(head,track,sector+1);// caculate absulute sector
result = absread(3,nsects,abs_sec,buffer);// write absolute
if (result != 0){
 perror("READ error nsects < 64");
 return (-1);
}
// DMA out
 if(!first)
```

```
        ck_tc();
    DMA_out((nsects*SECSIZE),buffer);
    x = div(nsects-1,SA_Sec);
    SEC  = (unsigned char)( sector + x.rem );
    HEAD = (unsigned char) ( head + x.quot );  // up date current head,cyl,sec
    CYL = (unsigned char) track;
    ck_tc();// check tc before return from last DMA_OUT
    return(0);
}
// ---------- SA4600 WRITE DISK ----------
int sa_write(int head,int track,int sector,int nsects,void *buffer )
{
    int result;
    unsigned int abs_sec,k; div_t x;
    int first=1;
    if (check_para(head,track,sector,nsects,1)!=0)
      return (0x07);    // command error illegal length
    while (nsects > 64) {  // Write data that more than 64 sectors.
      DMA_in((64*SECSIZE));
      if(!first){
          abs_sec = get_abs(head,track,sector+1);// caculate absulute sector
          result = abswrite(3,64,abs_sec,buffer);    // write absolute
          head += 2;        //update head,sector
          if (result != 0){
             perror("WRITE error nsects > 64 ");
             return(-1);
          }
      }
      ck_tc();
```

```
  RAM2PL((64*SECSIZE),buffer); //load data sram to prolog
  first=0;
  nsects -= 64;
 }
 // Write data that less than 64 sectors.
 DMA_in((nsects*SECSIZE));
 if(!first){
  abs_sec = get_abs(head,track,sector+1);// caculate absulute sector
  result = abswrite(3,64,abs_sec,buffer);// write absolute
  head += 2;   //update head,sector
  if (result != 0){
       perror("WRITE error nsects > 64");
       return(-1);
    }
 }
 ck_tc();
 RAM2PL((nsects*SECSIZE),buffer);
 abs_sec = get_abs(head,track,sector+1);// caculate absulute sector
 result = abswrite(3,nsects,abs_sec,buffer);// write absolute
 if (result != 0){
  perror("WRITE error nsects < 64");
  return (-1);
 }
 x = div(nsects-1,SA_Sec);
 SEC  = (unsigned char)( sector + x.rem );
 HEAD = (unsigned char) ( head + x.quot );  // up date current head,cyl,sec
 CYL  = (unsigned char) track;
 return(0);
}
```

```
//------------ get SA4600 absolute sector ------------
// return absolute sector
unsigned int get_abs(int head,int cyl,int sec)
{
/*
  In SA4600 HD => 1 cylinder has (8 heads)*(32 sectors/track)
              =  256 sectors/cylinder and 32 sectors/head
*/
     unsigned int abs_sec;
     abs_sec = (cyl*256) + (head*32) + sec;
     return (abs_sec);
}

//------------ check SA 4600 parameter in lenght ------------
int check_para(int head,int cyl,int sec,int nsec,int chknsec)
{
     int seclen;
     if (head >=0 && head < SA_Hd) {       // 0 -> 7
          if (cyl >=0 && cyl < SA_Cy+1) {      // 0 -> 201
               if(sec >=0 && sec < SA_Sec) {   // 0 -> 31
                    if (chknsec== NULL)   // no need to check nsec
                         return (0);
                    if (nsec>0 && nsec<=  256)  // 1 -> 255
                         return (0);      // no error
                    else return (-1);
               } else return (-1);      // return false
          } else return (-1);
     } else return (-1);
}
```

# ประวัติผู้เขียน

นายสีหชาติ สดับธรรมารักษ์ เกิดเมื่อวันที่ 9 สิงหาคม 2513 ที่จังหวัดนครราชสีมา สำเร็จการศึกษาอุตสาหกรรมศาสตร์บัณฑิต (เทคโนโลยีโทรคมนาคม) จากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2535 ปัจจุบันทำงานที่องค์การโทรศัพท์แห่งประเทศไทย ตำแหน่งวิศวกร ระดับ 4