สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์ การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

Implementation Central Processing Unit Using VHDL And FPGAs Device

นักศึกษา

1. นายจิรวัฒน์ นาคสง รหัสประจำตัว 39031304
2. นายนิคม ยอดมณี รหัสประจำตัว 39031313
3. นายอรรถพล กิจปราชญ์ รหัสประจำตัว 39031340
4. นายเอกรัตน์ นามวงค์ รหัสประจำตัว 39031343

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต สาขาวิชา อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ผู้ควบคุมปริญญานิพนธ์

1. อาจารย์กิติพงศ์ มะโน
2. อาจารย์ปิยะ จิตธรรมมาภิรมย์
3. อาจารย์อำพล ทองระอา

คณะกรรมการสอบปริญญานิพนธ์	ลายมือชื่อ
1. อาจารย์กิติพงศ์ มะโน	.....
2. อาจารย์ปิยะ จิตธรรมมาภิรมย์	.....
3. อาจารย์อำพล ทองระอา	.....
4. อาจารย์ไพบูลย์ พวงวงศ์ตระกูล	.....

วันเดือนปีที่สอบ วันที่ 14 เดือน ธันวาคม พ.ศ.2540 เวล..20.00...ถึง 21.00..น...

สถานที่สอบ ห้อง ค.310 คณะครุศาสตร์อุตสาหกรรม

ภาควิชารับรองแล้ว

..... (ผู้ช่วยศาสตราจารย์ เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ ..... เดือน ..... พ.ศ. 41

ปริญญานิพนธ์

การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL

และอุปกรณ์ FPGAs

IMPLEMENTATION CENTRAL PROCESSING UNIT USING VHDL

AND FPGAs DEVICE

นายจิรวัฒน์ นาคสง

นายนิคม ยอดมณี

นายอรรถพล กิจปราชญ์

นายเอกรัตน์ นามวงค์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์

เรื่อง การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

Implementation Central Processing Unit Using VHDL and FPGAs Device

ผู้จัดทำ

1. นายจิรวัฒน์ นาคสง
2. นายนิคม ยอดมณี
3. นายอรรถพล กิจปราชญ์
4. นายเอกรัตน์ นามวงค์

อาจารย์ที่ปรึกษา

ลงนาม.....

(อาจารย์กิติพงค์ มะโน)

ลงนาม.....

(อาจารย์ปิยะ จิตธรรมมาภิรมย์)

ลงนาม.....

(อาจารย์อำพล ทองระอา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

ลงนาม.....

(ผศ.ดร.ธีรพล เทพหัสดิน ณ อยุธยา)

ปริญญานิพนธ์

เรื่อง การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

Implementation Central Processing Unit Using VHDL and FPGAs Device

วัตถุประสงค์

1. เพื่อศึกษาการเขียนโปรแกรมภาษา VHDL และอุปกรณ์ FPGAs
2. เพื่อศึกษาโปรแกรมสำหรับออกแบบสร้างและเลียนแบบการทำงานของหน่วยประมวลผลกลางที่ใช้ในการโปรแกรมอุปกรณ์ FPGAs
3. เพื่อออกแบบหน่วยประมวลผลโดยใช้ภาษา VHDL
4. เพื่อสร้างหน่วยประมวลผลโดยใช้อุปกรณ์ FPGAs
5. เพื่อนำหน่วยประมวลผลมาทดสอบ
6. เพื่อนำหน่วยประมวลผลมาประยุกต์ใช้งาน

ประโยชน์ที่คาดว่าจะได้รับ

1. ได้ความรู้เกี่ยวกับภาษา VHDL และอุปกรณ์ FPGAs
2. ได้ความรู้เกี่ยวกับโปรแกรมการสร้างวงจรและเลียนแบบการทำงานของหน่วยประมวลผลกลางที่ใช้ในการโปรแกรมอุปกรณ์ FPGAs
3. ได้ความรู้ในการออกแบบวงจรเชิงเลขเฉพาะงาน
4. สร้างวงจรเชิงเลขเฉพาะงานโดยใช้อุปกรณ์ FPGAs ได้
5. สามารถนำวงจรเชิงเลขเฉพาะงานมาทดสอบและใช้งานได้

I

การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDLและอุปกรณ์ FPGAs

นายจิรวัฒน์ นาคสง

นายนิคม ยอดมณี

นายอรรถพล กิจปราชญ์

นายเอกรัตน์ นามวงค์

อาจารย์ที่ปรึกษา

อาจารย์กิติพงค์ มะโน

อาจารย์ปิยะ จิตธรรมมาภิรมย์

อาจารย์อำพล ทองระอา

ปีการศึกษา 2540

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอการสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs ซึ่งได้ออกแบบหน่วยประมวลผลกลางโดยใช้ภาษา VHDL ที่ประกอบไปด้วยบัสข้อมูลขนาด 8 บิท , บัสตำแหน่งขนาด 16 บิท , ALU ขนาด 8บิท และสามารถกระทำคำสั่งได้ 25 คำสั่ง เมื่อได้ไฟล์ของ VHDL ที่จำลองการทำงานที่สมบูรณ์แล้วจึงแปลงไฟล์เพื่อให้ได้ไฟล์ของวงจรโดยใช้โปรแกรมของบริษัทวิวลอจิก และนำไฟล์ของวงจรที่แปลงได้ไปแปลงเป็นไฟล์ XNF และ bit โดยใช้โปรแกรม XECP step ของบริษัท XILINX จากนั้นนำไฟล์ที่มีส่วนขยายเป็น bit ส่งให้อุปกรณ์ FPGAs เบอร์ XC4010 เพื่อให้อุปกรณ์ FPGAs ทำหน้าที่เป็นหน่วยประมวลผลกลางขนาด 8 บิท และได้ลองนำอุปกรณ์ FPGAs ไปทดสอบเป็นหน่วยประมวลผลกลางขนาด 8 บิท ปรากฏว่าได้ผลถูกต้องสอดคล้องตามหลักการที่นำเสนอทุกประการ

IMPLEMENTATION CENTRAL PROCESSING UNIT USING VHDL AND FPGAs DEVICE

MR.JIRAWAT NAKSONG
MR.NIKHOM YODMANEE
MR.ATTHAPHOL KIJPRACH
MR.AKEGARAT NAMVONG

ADVISORS

MR.KITIPONG MANO
MR.PIYA JITTHAMMAPIROM
MR.AMPHON THONGRA-AR

1997

ABSTRACT

This thesis presents the central processing unit using VHDL language and FPGAs device. In this thesis, CPU is designed by VHDL language. It contains of data bus 8 bit, address bus 16 bit, ALU 8 bit and 25 instructions. Then, the VHDL file, which simulated completely, is converted to a schematic file by View Logic software. Next, a schematic file is converted to a XNF file and a bit file by XECP step software from XILINX Co.,Ltd.. After that, a bit file is downloaded into the FPGAs devices XC4010. The FPGAs cevice represents central processing unit. Consequently, the FPGAs device can worked completely.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์นี้สามารถสำเร็จลุล่วงไปได้ด้วยดี จากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน นอกจากนี้ยังได้รับความกรุณาจากอาจารย์ที่ปรึกษาประจำโครงงาน และอาจารย์ประจำภาควิชาครุศาสตร์วิศวกรรมทุกท่านในการให้คำปรึกษาแนะนำ และความช่วยเหลือต่างๆ ตลอดจนให้โอกาสในการทำโครงงานอย่างเต็มที่ ทั้งด้านเวลาสถานที่ เครื่องมือและอุปกรณ์ต่างๆ และขอขอบคุณบุพการีผู้ให้กำเนิดที่ให้โอกาสในการศึกษา เพื่อน ตลอดจนผู้ที่เกี่ยวข้องทุกคนที่ให้คำแนะนำต่างๆ และเป็นกำลังใจในการทำงาน จนทำให้โครงงานนี้สำเร็จลุล่วงไปได้ด้วยดี

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

ในอดีตที่ผ่านมาการออกแบบวงจรเชิงเลขที่ใช้อุปกรณ์ต่อกันเป็นจำนวนมาก ซึ่งจะทำให้วงจรมีขนาดใหญ่และสิ้นเปลืองพื้นที่มาก การออกแบบและการแก้ไขข้อผิดพลาดของวงจรทำได้ยากและเสียเวลามาก เนื่องจากในปัจจุบันนี้มีความก้าวหน้าทางเทคโนโลยีสูงขึ้นและมีการพัฒนาด้านซอต์ฟแวร์ใหม่ๆ ซึ่งช่วยลดความยุ่งยากในการออกแบบวงจรอีกทั้งการแก้ไขข้อผิดพลาดสามารถทำได้ง่าย และลดระยะเวลาในการทำต้นแบบของวงจรเชิงเลขให้น้อยลงด้วย

ปัจจุบันการออกแบบวงจรเชิงเลขจะแบ่งออกเป็น 2 กลุ่ม คือฟูลลี่คัสตอม (Fully Custom) และเซมิคัสตอม (Semi Custom) โดยการออกแบบแบบฟูลลี่คัสตอมนั้น หลังจากที่ทำการออกแบบในขั้นตอนสุดท้ายจะต้องให้โรงงานผลิตออกมาเป็นไอซีสำเร็จรูป แต่แบบเซมิ-คัสตอมเราสามารถที่จะดำเนินการได้เองจนสิ้นสุดกระบวนการออกแบบ ซึ่งทางคณะผู้จัดทำได้เลือกใช้การออกแบบแบบเซมิคัสตอม โดยใช้โปรแกรมภาษา VHDL (Very High Speed Integrated Circuit Hardware Description Language) ทำการออกแบบวงจรเชิงเลข และทำการสังเคราะห์จนได้วงจรระดับเกทแล้วจึงโปรแกรมลงบนอุปกรณ์ประเภทที่สามารถโปรแกรมได้ (Programmable Logic Device) ซึ่งในโครงงานนี้ทางคณะผู้จัดทำได้เลือกใช้อุปกรณ์ FPGAs (Field Programmable Gate Array)

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาการเขียนโปรแกรมภาษา VHDL และอุปกรณ์ FPGAs
2. เพื่อศึกษาโปรแกรมสำหรับออกแบบสร้างและเลียนแบบการทำงานของหน่วยประมวลผลกลางที่ใช้ในการโปรแกรมอุปกรณ์ FPGAs
3. เพื่อออกแบบหน่วยประมวลผลโดยใช้ภาษา VHDL
4. เพื่อสร้างหน่วยประมวลผลโดยใช้อุปกรณ์ FPGAs
5. เพื่อนำหน่วยประมวลผลมาทดสอบ

6. เพื่อนำหน่วยประมวลผลมาประยุกต์ใช้งาน

1.3 เนื้อหาโดยสังเขป

โปรแกรมภาษา VHDL เป็นภาษาที่ใช้สำหรับบรรยายการทำงานทางฮาร์ดแวร์ของวงจรเชิงเลข (Digital Circuit) โดยภาษา VHDL สามารถบรรยายการทำงานของวงจรเชิงเลขได้หลายๆ ลักษณะหรือบรรยายถึงความสัมพันธ์ระหว่างสัญญาณอินพุทและเอาท์พุทที่มีสัญญาณต่างๆ วิ่งอยู่ภายในวงจรเชิงเลขนั้น ซึ่งทำให้การออกแบบวงจรเชิงเลขสามารถทำได้สะดวกและรวดเร็วขึ้นจากเดิมมาก เพียงแต่ทราบถึงการทำงานของวงจรแล้วทำการออกแบบวงจรนั้นเป็นส่วนย่อยๆ เพื่อที่จะเขียนโปรแกรมบรรยายการทำงาน จากนั้นทำการแปลภาษาและจำลองการทำงานเพื่อตรวจสอบผลการทำงาน หากมีข้อผิดพลาดสามารถแก้ไขได้โดยการเข้าไปแก้ไขที่โปรแกรมภาษา VHDL แล้วทำการแปลภาษาและจำลองการทำงานใหม่ หลังจากได้ผลการทำงานตามต้องการแล้วต้องนำไปสังเคราะห์เพื่อให้ได้เป็นวงจรแสดงการเชื่อมต่อซึ่งประกอบด้วยเกท (Gate) ฟลิป-ฟลอป (Flip-Flop) เสร็จแล้วจึงนำไปโปรแกรมลงบนอุปกรณ์ FPGAs ภายในตัวของอุปกรณ์ FPGAs จะประกอบไปด้วยเส้นทางการเชื่อมต่อระหว่างบล็อก ซึ่งเราสามารถโปรแกรมได้ และจะมีเกทให้เราเลือกใช้ได้อยู่ระหว่าง 600 ถึง 25000 เกท ทำให้เลือกใช้งานได้อย่างกว้างขวางอีกทั้งยังสามารถโปรแกรมครั้งใหม่ได้

จากคุณสมบัติและความก้าวหน้าทางเทคโนโลยีของโปรแกรมภาษา VHDL และอุปกรณ์ FPGAs เป็นเรื่องที่น่าศึกษา ทางคณะผู้จัดทำได้เห็นความสำคัญและสนใจที่จะศึกษาค้นคว้าทดลอง จึงได้ทำการออกแบบโครงงานขึ้นมา โดยมีเป้าหมาย 2 ประการ คือใช้โปรแกรมภาษา VHDL เขียนโปรแกรมจำลองการทำงานของหน่วยประมวลผลกลางขนาด 8 บิทและใช้โปรแกรมสังเคราะห์ภาษา VHDL สร้างเป็นหน่วยประมวลผลกลางขนาด 8 บิท โดยใช้อุปกรณ์ FPGAs ซึ่งเนื้อหาในปริญญานิพนธ์ฉบับนี้ประกอบไปด้วยรายละเอียดต่างๆ ที่สำคัญดังนี้

บทที่ 2 ทฤษฎีและหลักการ กล่าวถึงการแนะนำภาษา VHDL อันได้แก่ โครงสร้างภาษา VHDL, การบรรยายเชิงพฤติกรรม, การบรรยายเชิงข้อมูล, การบรรยายเชิงโครงสร้าง, ไวยากรณ์พื้นฐาน ในส่วนต่อมาจะกล่าวถึงขั้นตอนการออกแบบและจำลองการทำงานโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก(View Logic),ขั้นตอนการสังเคราะห์วงจร,โครงสร้างภายใน

อุปกรณ์ FPGAs (เบอร์ XC4010E/L),รายละเอียดการใช้งานและข้อพึงระวังของอุปกรณ์ FPGAs และทฤษฎีไมโครโปรเซสเซอร์

บทที่ 3 การออกแบบและการสร้างหน่วยประมวลผลกลางขนาด 8 บิท กล่าวถึงโครงสร้างและการออกแบบ, วิธีการออกแบบ, ลำดับขั้นการออกแบบหน่วยประมวลผลกลางขนาด 8 บิท และการออกแบบวงจรอินเตอร์เฟส

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึงการทดลอง Simulate ในขั้นตอนที่เป็นบล็อก เป็นอุปกรณ์ FPGAs และเป็นหน่วยประมวลผลกลางขนาด 8 บิท

บทที่ 5 สรุปอภิปรายและข้อเสนอแนะ กล่าวถึงข้อสรุป, ปัญหา, แนวทางการแก้ไข และแนวทางการพัฒนาเพื่อนำไปประยุกต์ใช้งาน

ภาคผนวก ก ผังงานและโปรแกรมภาษา VHDL ของหน่วยประมวลผลกลางขนาด 8 บิท

ภาคผนวก ข ขั้นตอนการ Simulate โดยใช้ซอฟแวร์ของบริษัทวิวลอจิก (Viewlogic) และขั้นตอนการติดตั้งโปรแกรม Workview Office

ภาคผนวก ค เอกสารของอุปกรณ์ FPGAs ตระกูล XC4010E

บทที่ 2
ทฤษฎีและหลักการ

2.1 กล่าวนำ

ปัจจุบันการออกแบบระบบเชิงเลขนั้นมีความยุ่งยากและซับซ้อนมากยิ่งขึ้น ทำให้คอมพิวเตอร์มีบทบาทที่สำคัญและนำมาเพื่อช่วยในการออกแบบวงจรระบบเชิงเลข เราเรียกว่า CAD (CAD ย่อมาจาก Computer Aided Design) วิธีการออกแบบใหม่ ๆ ได้ถูกพัฒนาขึ้นจากซอฟแวร์ใหม่ๆ เพื่ออำนวยความสะดวกให้กับนักออกแบบ ดังนั้นโปรแกรมภาษา VHDL (Very High Speed Integrated Circuit Hardware Description Language) เป็นเครื่องมืออย่างหนึ่งที่มีการพัฒนาขึ้นมา และช่วยในกระบวนการของการออกแบบวงจรเชิงเลขให้สะดวกมากยิ่งขึ้น

สำหรับการออกแบบวงจรระบบเชิงเลข เริ่มจากการกำหนดความคิดในการออกแบบซึ่งจะต้องสร้างรูปแบบในเชิงพฤติกรรม เช่น การกำหนดแผนงาน ผังแสดงแบบ เป็นต้น ต่อไปจะต้องออกแบบเส้นทางของข้อมูล ซึ่งหลังจากออกแบบแล้วเมื่อนำองค์ประกอบมาประกอบกันจะต้องเป็นระบบที่สมบูรณ์ แต่ละองค์ประกอบจะต้องเชื่อมต่อกันได้ เมื่อกำหนดเส้นทางของข้อมูลแล้วจะเป็นการออกแบบเกี่ยวกับการใช้เกทพื้นฐานและฟลิป-ฟลอปซึ่งเป็นส่วนประกอบของอุปกรณ์ย่อยของระบบเชิงเลขที่ออกแบบ ในขั้นตอนต่อมาจะเป็นเครือข่ายของการเชื่อมโยงระหว่างเกทและฟลิป-ฟลอป ซึ่งจัดการวางเกทและฟลิป-ฟลอปหรือแทนด้วยทรานซิสเตอร์ลิส เมื่อเสร็จกระบวนการดังกล่าวแล้ว จะเป็นการส่งระบบที่ออกแบบไปทำการเจือสารที่โรงงานผลิตออกมาเป็นวงจรรวมในที่สุด การออกแบบที่กล่าวตามวิธีข้างต้นเป็นวิธีการออกแบบแบบฟลูลีคัสตอม ยังมีการออกแบบอีกวิธีหนึ่งคือ การออกแบบแบบเซมิคัสตอม คือหลังจากออกแบบในระดับเกทและผ่านการสังเคราะห์แล้ว จึงนำโปรแกรมไปโปรแกรมลงบนอุปกรณ์ FPGAs ซึ่งเราสามารถทำได้เองทุกขั้นตอนจนสิ้นสุดกระบวนการออกแบบ

2.2 แนะนำภาษา VHDL

ในปี 1981 สถาบันเพื่อการวิเคราะห์และป้องกัน (The institute for Defense Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่งเพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยาย

หรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้นเรียกว่า VHDL (VHSIC Hardware Description Language) โดย VHSIC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวข้องกับวงจรรวมที่มีความเร็วสูงมาก (Very High Speed Integrated Cricuit) ต่อมาในปี 1985 IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL ในด้านการกำหนดพฤติกรรมการทำงานของวงจรให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งของวงจรเชิงเลขทั่วไปและในระบบที่แตกต่างออกไป เช่น พฤติกรรมการทำงานของระบบเรดาร์หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในของมนุษย์ ข้อดีหลักที่สำคัญของ VHDL ก็คือภาษานี้สามารถจะใช้ได้ตลอดในทุกระดับขั้นของการออกแบบ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูง (System Level) จนถึงในระดับที่ต่ำกว่า (Lower Hardware Level) สามารถใช้ภาษาเดียวกันได้โดยตลอดทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

2.3 ภาษา VHDL เบื้องต้น

ภาษา VHDL เป็นภาษาสำหรับการออกแบบและการบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและการออกแบบในระดับสูง ความสามารถในการเลียนแบบ (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนี้ภาษา VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับของระบบจนถึงระดับเกท เนื่องจากในการทำงานของระบบเชิงเลขจริงทุกองค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อมกัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งในภาษา VHDL ด้วยเช่นกัน สำหรับภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้วความพร้อมเพรียงจะหมายถึงทุกๆคำสั่ง องค์ประกอบ เกท หรือวงจรตรรกจะนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายก็จะดูเหมือนว่าได้มีการปฏิบัติไปพร้อมๆกัน

ไลบรารี

ภาษา VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่

ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารี หลังจากที่ได้ผ่านการแปลเรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่นๆสามารถนำไปใช้ได้ด้วย

ลำดับคำสั่ง

ภาษา VHDL ได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งเอาไว้ เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if-then-else และ loop การบรรยายแบบลำดับคำสั่ง ทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้โดยสะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของภาษา VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันอยู่

การกำหนดคุณสมบัติ

นอกจากการกำหนดอินพุทเอาท์พุทแล้ว เงื่อนไขอื่นก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์เช่นเดียวกัน ทั้งนี้ได้รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้น ภาษาสำหรับการออกแบบที่ดีควรจะควรที่จะช่วยให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาดทางกายภาพเวลาโหลด และเงื่อนไขทางสภาพแวดล้อมอื่นๆ ความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL

ชนิดของข้อมูล

ภาษา VHDL สามารถกำหนดข้อมูลไม่เพียงแต่ชนิดบิท และบูลีนเท่านั้น แต่ยังสามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับการนับ (Enumerate type) หรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเอง

โปรแกรมย่อย

ความสามารถในการใช้ฟังก์ชั่นและโพรซีเจอร์ (Procedure) เป็นข้อกำหนดอีกอย่างหนึ่งในภาษา VHDL เราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูลการกำหนดหน่วยของลอจิก และการกำหนดตัวกระทำต่างๆทั้งเก่าและใหม่ได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

การควบคุมเวลา

ภาษา VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ การตรวจสอบการออกแบบเกทหรือหน่วงเวลาสามารถกระทำได้โดยการ กำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้

2.3.1 โครงสร้างภาษา VHDL

โครงสร้างของภาษา VHDL มีโครงสร้างที่สำคัญสองส่วนคือการบรรยายการเชื่อมต่อกับโลกภายนอก (Design-Entity) และการบรรยายสถาปัตยกรรม (Architecture body) โดยไวยากรณ์ทั้งสองส่วนดังแสดงในรูปที่ 2.1

```
ENTITY identifier is
    [generic (generic_list)][port_list);]
    [begin
    {passive_concerrent_statement}]
    end[entity][entity_simple_name];
```

```
ARCHITECTURE identifier of entity_name is
architecture_declarative_part
begin
all_concerrent_statements
end[architecture][architecture_simple_name];
```

รูปที่ 2.1 โครงสร้างของภาษา VHDL

ตัวอย่างภาษา VHDL อย่างง่ายสำหรับวงจรเชิงเลขแบบจัดหมวดหมู่ (Combination Circuit) วงจรนอร์เกต

```
ENTITY nor1 is
        port(a,b:in bit;
             s:out bit);
end nor1;
```

```
ARCHITECTURE arcnor of nor1 is
begin
s <= a nor b
end arcnor;
```

รูปที่ 2.2 ตัวตัวอย่างภาษา VHDL อย่างง่ายสำหรับวงจรเชิงเลขแบบจัดหมวดหมู่

ตัวอย่างภาษา VHDL อย่างง่ายสำหรับวงจรเชิงเลขแบบลำดับ (Sequential Circuits) วงจรฟลิบ-ฟลอบ (Flip Flop)

```
ENTITY ff1 is
    port(clk : in bit;
         d : in bit;
         q : out bit);
end ff1;
```

```
ARCHITECTURE arcff of ff1 is
    begin
        p_ff1 :process
        begin
```

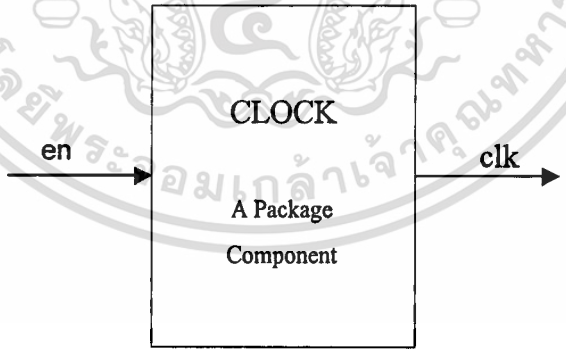
รูปที่ 2.3 ตัวอย่างภาษา VHDL อย่างง่ายสำหรับวงจรเชิงเลขแบบลำดับ

```
        wait utill clk='1';
        q <= d;
    end process p_ff1
end arcff;
```

รูปที่ 2.3 (ต่อ) ตัวอย่างภาษา VHDL อย่างง่ายสำหรับวงจรเชิงเลขแบบลำดับ

การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบในระดับนี้ จะต้องกำหนดพอร์ตสำหรับการติดต่อองค์ประกอบภายนอก ดังตัวอย่างในรูปที่ 2.4 แสดงผังการทำงานและการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับให้สัญญาณนาฬิกา บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็น clock_component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ในวงเล็บ IN และ OUT กำหนดโหมดของสัญญาณเป็นอินพุตหรือเอาท์พุต และ BIT แสดงชนิดของข้อมูล




```
ENTITY  clock_component IS
    PORT (en : IN BIT; clk : OUT BIT)
END clock_component;
```

รูปที่ 2.4 ผังการทำงานและการบรรยายการเชื่อมต่อของ clock_component

โปรแกรมย่อย

การใช้ฟังก์ชั่นและโพรซีเจอร์ในภาษา VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมชั้นสูงทั่วไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรง เช่น ถ้าเราใช้ฟังก์ชั่นแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรตรรกจริงๆ ในขณะที่ถ้าเราใช้โปรแกรมในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าหน่วงเวลาแล้ว ก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

โอเปอร์เรเตอร์

การบรรยายเชิงพฤติกรรมในภาษา VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกับภาษาซอฟแวร์ทั่วไปดังรูปที่ 2.5

PREDEFINED OPERATORS
LOGICAL OPERATORS : NOT AND OR NOR NAND XOR
OPERAND TYPE : BIT BOOLEAN
RESULT TYPE : BIT BOOLEAN
PREDEFINED OPERATORS
RELATIONAL OPERATOR : = /= < <= > >=
OPERAND TYPE : any type
RESULT TYPE : boolean
ARITHMETIC OPERATORS : + - * / MOD REM ABS
OPERAND TYPE : INTEGER REAL physical
RESULT TYPE : INTEGER REAL physical
PREDEFINED OPERATORS
CONCANTENATION OPERATORS : &
OPERAND TYPE : array of any type
RESULT TYPE : array of any type

รูปที่ 2.5 ตัวกระทำในภาษา VHDL

เวลาและความพร้อมเพรียง

ในวงจรอิเล็กทรอนิกส์ อุปกรณ์ทุกๆตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (Always Active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องเสมอในทุกเหตุการณ์ที่เกิดขึ้น ภาษา VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบ และการป้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายใต้ส่วนของสถาปัตยกรรมบรรยาย (Architecture) จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โพรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายโพรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกโพรเซสก็จะทำงานไปพร้อมกันด้วย

สัญญาณและตัวแปร

สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาลักษณ์จะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น w<=a AFTER 12 nS หมายถึง การกำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 nS ในทางตรงกันข้ามตัวแปรมีลักษณะเป็นเสมือนตัวกลางซอฟแวร์ที่ใช้ในการส่งผ่านข้อมูล และไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ := ตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นใน ฟังก์ชั่น โพรซีเจอร์ และโพรเซส

2.3.2 การบรรยายเชิงพฤติกรรม

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรม เป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึม สำหรับการคำนวณผลลัพธ์ที่เกิดขึ้นสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูลที่เข้ามาโดยไม่คำนึงถึงว่าลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายใน ในหัวข้อนี้แสดงให้เห็นถึงประโยชน์ในการใช้โปรแกรมย่อยที่จำลองแบบอินพุทเอาท์พุทในระดับพฤติกรรม แทนการใช้ฮาร์ดแวร์รวมถึงข้อกำหนดต่างๆ ที่ควรรู้

โพรเซส (Process)

โพรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดค่าให้กับสัญลักษณ์ โพรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอ และจะปฏิบัติงานตามคำสั่งพร้อมกันกับโพรเซสอื่นที่อยู่ภายในสถาปัตยกรรมบรรยายเดียวกัน โพรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์

เกิดขึ้น สัญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์การกำหนดค่าให้กับสัญญาณ (<=) การบรรยายโพรเซสจะเริ่มด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS; รูปที่ 2.6 แสดงส่วนประกอบของการบรรยายแบบโพรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้ และส่วนของการปฏิบัติคำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ

```
PROCESS
        declarative part
        ……
BEGIN
        statement part
        ……..
END PROCESS;
```

รูปที่ 2.6 รูปแบบของการบรรยายแบบโพรเซส

การกำหนดตัวประกอบภายในโพรเซส

ตัวกระทำภายในโพรเซสมี 3 ชนิดคือ ตัวแปร (Variable) ไฟล์ (File) และตัวคงที่ (Constant) ซึ่งตัวกระทำทั้งสามชนิดนี้หากมีการประกาศไว้ในโพรเซสใด ก็จะใช้ได้เฉพาะในโพรเซสนั้นเท่านั้น การติดต่อกับภายนอกหรือระหว่างโพรเซส สามารถทำได้โดยใช้สัญญาณหรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE

```
PROCESS
   FILE flush : TEXT IS IN "filename.dat";
   VARIABLE var : BIT ; CONSTANT n : INTEGER := 0;
      BEGIN ……..
END PROCESS;
```

รูปที่ 2.7 ตัวอย่างการประกาศตัวกระทำภายในโพรเซส

รูปที่ 2.7 แสดงการประกาศตัวกระทำภายในโพรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN ค่าเริ่มต้นที่ถูกกำหนดให้กับตัวกระทำภายในโพรเซสจะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียว ค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกโปรแกรมย่อย

การกำหนดการกระทำภายในโพรเซส

การกระทำภายในโพรเซสจะเป็นการปฏิบัติแบบลำดับ (Sequencial) เสมอ ภายในโพรเซสสามารถใช้รูปแบบของการใช้เงื่อนไขหรือการทำซ้ำได้ เช่น IF-THEN-ELSE, CASE-WHENFOR-LOOP และ WHILE-LOOP ดังตัวอย่างในรูปที่ 2.8 และรูปที่ 2.9

```
ARCHITECTURE  demo OF partial_process IS
……..
 BEGIN
    PROCESS
    ……..
    BEGIN
    ……..
    x<= '1';
    IF x = '1' THEN
          perfrom  action_1;
    ELSIF
          perfrom  action_2;
    END IF;
  END PROCESS;
END demo;
```

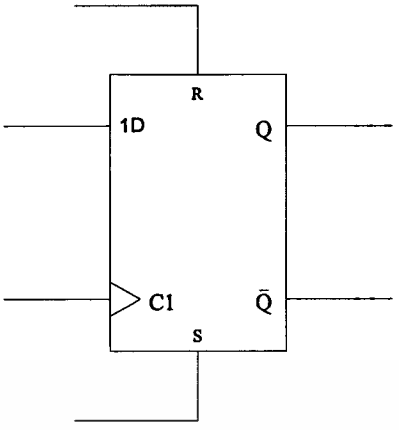
รูปที่ 2.8 เงื่อนไขการกระทำในโพรเซส

```
ARCHITECTURE demo OF partial_process IS
.....
BEGIN
    PROCESS
      BEGIN
        .....
        x <= a AFTER  10 nS;
        y <= b AFTER  6 nS;
        .....
      END PROCESS;
    END demo
```

รูปที่ 2.9 การกระทำในโพรเซส

การกระตุ้นและยับยั้งการกระทำของโพรเซส

การกระทำภายในโพรเซสจะเตรียมพร้อมและมีการปฏิบัติงานตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์ที่เกิดขึ้น อย่างไรก็ดีเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโพรเซสได้ โดยการกำหนดรายการของสัญญาณที่ต้องการให้โพรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่เรากำหนดไว้เท่านั้นเหตุการณ์ใดๆที่เกิดขึ้นกับสัญญาณที่ไม่ได้กำหนดไว้ในรายการจะไม่ส่งผลให้มีการกระทำภายในโพรเซส รายการของสัญญาณนี้เรียกว่า Sensitivity List และถูกกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS รูปที่ 2.10 แสดง (ก) ตัวอย่างโมเดลและ (ข) ตัวอย่างการบรรยายการเชื่อมต่อของ D Flip-Flop รูปที่ 2.11 แสดงการบรรยายเชิงพฤติกรรมของ D Flip-Flop โดยการใช้ตัวกระทำภายนอกโพรเซส และมีรายการของสัญญาณ (rst, set, clk) เป็นตัวกระตุ้นการปฏิบัติงานภายในโพรเซส



รูปที่ 2.10 (ก) ตัวอย่างโมเดล D Flip-Flop

```
ENTITY d_sr_flipflop IS
  GENERIC(sq_delay, rq_delay, cq_delay  :  TIME := 6 nS);
  PORT(d, set, rst, clk : IN BIT; q, qb : OUT BIT);
END d_sr_flipflop;
```

รูปที่ 2.10 (ข) การบรรยายการเชื่อมต่อของ D Flip-Flop

```
ARCHITECTURE  behavioral OF d_sr_flipflop IS
  SIGNAL  state : BIT := '0';
  BEGIN
dff: PROCESS(rst, set, clk)
      BEGIN
        IF set = '1' THEN
            state <= '1' AFTER sq_delay;
      ELSIF rst = '1' THEN
            state <= '0' AFTER rq_delay;
      ELSIF clk = '1' AND clk'event THEN
```

รูปที่ 2.11 การบรรยายเชิงพฤติกรรมของ D Flip-Flop โดยใช้ตัวกระทำภายในโพรเซส

```
            state <= d AFTER cq_delay;
        END IF;
    END PROCESS
     q <= state;
     qb <= NOT state;
END behavioral;
```

รูปที่ 2.11(ต่อ) การบรรยายเชิงพฤติกรรมของ D Flip-Flop โดยใช้ตัวกระทำภายในโพรเซส

การตรวจสอบการกระทำ

ภาษา VHDL ได้จัดเตรียมรูปแบบคำสั่งไว้เพื่อใช้ในการตรวจดูการกระทำต่างๆ ที่เกิดขึ้นภายในอุปกรณ์ดังนี้

```
ASSERT assertion_condition REPORT "reporting_message"SEVERITY severity_level;
```

คำสั่งนี้จะถูกปฏิบัติเมื่อมีเงื่อนไข assertion_condition มีค่าเป็น False และจะแสดงข้อความ "reporting_Message" ส่วนพารามิเตอร์ severity_level จะเป็นตัวกำหนดระดับความรุนแรงที่เกิดขึ้น ซึ่งแบ่งเป็น 4 ระดับ คือ Note, Warning,Error และ False ระดับความรุนแรง Error หรือ Failure จะทำการเลียนแบบการทำงานหยุดทันทีหลังจากที่ได้แสดงข้อความ reporting_message

```
ARCHITECTURE behavioral  OF d_sr_flipflop IS
        ……..
BEGIN
    ASSERT
        (NOT(set= '1' AND  rst = '1'))
```

รูปที่ 2.12 การใช้ ASSERT

```
        REPORT
          "report and rst both 1"
        SEVERITY NOTE;
        ……..
END behavioral;
```

รูปที่ 2.12 (ต่อ) การใช้ ASSERT

ส่วนระดับ Note หรือ Warning จะแสดงข้อความ reporting_message แต่ยังคงเลียนแบบการทำงานต่อไป สมมติว่าเราต้องการตรวจสอบการทำงานของ D Flip-Flop ในขณะที่กำลังทำงานได้เกิดสัญญาณในกรณีที่ไม่พึงปรารถนา เช่น สัญญาณ set ,rst มีค่าเป็น '1' พร้อมกันก็สามารถตรวจสอบได้โดยแทรกคำสั่ง ASSERT เพื่อตรวจสอบสัญญาณในโพรเซสได้ดังรูปที่ 2.12

การหยุดรอ

การหยุดรอเป็นรูปแบบคำสั่งที่ใช้เพื่อหน่วงเวลาของสัญญาณมีอยู่ 4 รูปแบบดังนี้

```
WAIT FOR waiting_time;
WAIT ON waiting_sensitivity_list;
WAIT UNTIL waiting_condition;
WAIT;
```

คำสั่งหยุดรอสามารถใช้ได้ภายในโพรเซสและโพรซีเจอร์ที่ไม่ถูกกำหนด Sensitivity_ List ไว้เท่านั้น WAIT FOR จะอยู่รอเป็นเวลาเท่ากับ waiting_snsitivit_list WAIT UNTIL จะหยุดรอจนกว่าเงื่อนไข waiting_condition เปลี่ยนจาก False เป็น Ture และ WAIT จะหยุดรอตลอดไป

การสร้างองค์ประกอบย่อย

การสร้างองค์ประกอบย่อยเพื่อใช้ไลบรารี เป็นการบรรยายเชิงโครงสร้างซึ่งจะต้องทำการกำหนดจุดเชื่อมต่อระหว่างอุปกรณ์ต่างๆ ด้วยการประกาศไว้ในส่วนของ Architecture

ประกอบด้วย 3 ส่วนตามลำดับ คือส่วนประกาศองค์ประกอบ (Component Specification) ส่วนที่กำหนดการเชื่อมต่อของพอร์ตกับสัญญาณขององค์ประกอบ (Component Specification) ส่วนที่กำหนดการเชื่อมต่อของพอร์ตกับสัญญาณขององค์ประกอบ (Component Instantistion) รูปที่ 2.13 แสดงการบรรยายการเชื่อมต่อวงจรเลือกสัญญาณ 2 ออก 1 (Multiplex)

COMPONENT DECLARATION

format:

```
COMPONENT identifier_name
        PORT (local_port_list);
END COMPONENT
```

COMPONENT SPECIFICATION

format:

```
FOR component_instruction_label:component_name USE binding_indication;
Binding_indication:
ENTITY library.entity_name[(architecture_name)]
```

COMPONENT INSTANTATION

format:

```
label : compinent_declaration_name PORT MAP(association_list);
ENTITY mux IS
    PORT (d0, d1, sel : IN bit ;  q : OUT bit);
END mux;
ARCHITECTURE struct OF mux IS
    COMPONENT and2
PORT (a, b : IN bit ; z : OUT bit);
    END COMPONENT;
```

รูปที่ 2.13 การบรรยายโครงสร้างของวงจรเลือกสัญญาณ 2 ออก1

```
COMPONENT or2
    PORT (a, b : IN bit);
END COMPONENT;
COMPONENT inv
    PORT( i : IN bit ;  z : OUT bit);
END COMPONENT;
SIGNAL   aa, ab, nsel : bit;
FOR  u1  :  inv  USE ENTITY  invert(behav);
FOR  u2, u2  :  and2  USE ENTITY  and_gt(dflw);
FOR  u4  :  or2  USE ENTITY  or_gt(arch 1);
    BEGIN
      u1 : inv  PORT MAP (sel, nsel);
      u2 : and2  PORT MAP (nsel, dl, ab);
      u3 : and2  PORT MAP (d0, nsel, aa);
      u1 : or2  PORT MAP (aa, ab, q);
    END struct;
```

รูปที่ 2.13 (ต่อ) การบรรยายโครงสร้างของวงจรเลือกสัญญาณ 2 ออก 1

แพคเกจ (Packgate)

แพคเกจ คือ กลุ่มของชนิดข้อมูล โปรแกรมย่อยหรืออุปกรณ์ต่างๆ ที่ออกแบบ ได้ออกแบบไว้แล้วได้นำมารวบรวมไว้เป็นกลุ่มๆอยู่ภายใน เพื่อให้ผู้ออกแบบสามารถเรียกใช้ได้สะดวกดังรูปที่ 2.14

```
PACKAGE indentifier IS
        package_declaritive_part
END indentifier ;
```

```
PACKAGE BODYindentifier IS
        package_declaritive_part
END indentifier ;
```

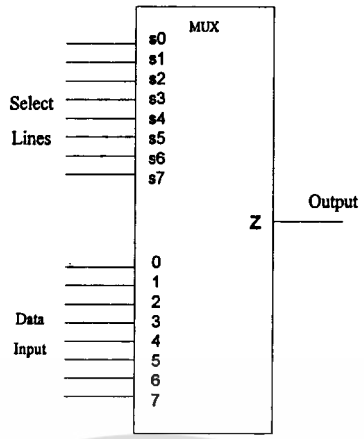
รูปที่ 2.14 โครงสร้างของเพคเกจ

2.3.3 การบรรยายเชิงข้อมูล

การบรรยายเชิงข้อมูลเป็นการบรรยายถึงการเลื่อนไหลของข้อมูลผ่านรีจิสเตอร์และบัสของระบบ เป็นระดับขั้นของการบรรยายที่อยู่ตรงกลางระหว่างการบรรยายเชิงพฤติกรรมและการบรรยายเชิงโครงสร้าง เครื่องมือที่ใช้ในการควบคุมการเลื่อนไหลของข้อมูลได้แก่ Conditional Selected และ Guarded ในหัวข้อนี้จะแสดงให้เห็นถึงคุณลักษณะและรูปแบบของการบรรยายเชิงข้อมูลข้อกำหนดและเงื่องไขต่างๆ พร้อมทั้งตัวอย่างประกอบ

การกำหนดทางเลือกข้อมูล

ในระบบเชิงเลขโครงสร้างของอุปกรณ์ฮาร์ดแวร์ส่วนใหญ่จะถูกใช้สำหรับการคัดเลือกและการนำข้อมูลเข้าสู่บัสและรีจิสเตอร์ รูปที่ 2.15 แสดงตัวอย่างโมเดลและการบรรยายตัวเลขข้อมูลของวงจร 8-to-1 มัลติเพล็ก Sel_line เป็นสัญญาณที่กำหนดขึ้นมาเพื่อรวมสัญญาณเลือกทั้ง 8 อินพุท (S7-S0) เข้าด้วยกัน และนำสัญญาณไปใช้ในการกำหนดเลือกอินพุทตัวใดตัวหนึ่ง ให้กับเอาท์พุท z ถ้าไม้มีอินพุทใดมีค่าเป็น 1 เลย ค่า 0 จะถูกส่งให้กับเอาท์พุท z หลังจากเวลาผ่านไป 3 nS ส่วนอินพุทอื่น (i7,i6,i5,i4,i3,i2,i1,i0) จะถูกส่งให้ z ขึ้นอยู่กับสัญญาณเลือกใด (s7,s6,s5,s4,s3,s2,s1,s0) มีค่าเป็น 1



รูปที่ 2.15(ก) ตัวอย่างโมเดลของวงจร 8-to-1 มัลติเพล็ก

```
ENTITY mux_8_to_1 IS
    PORT (i7, i6, i5, i4, i3, i2, i1, i0 : IN BIT;
        s7, s6, s5, s4, s3, s2, s1, s0 : IN BIT;  z : OUT BIT);
END mux_8_to_1;
ARCHITECTURE  dataflow   OF mux_8_to_1 IS
        SIGNAL  sel_line : BIT_VECTOR(7 DOWNTO 0);
   BEGIN
        sel_line <= s7 & s6 & s5 & s4 & s3 & s2 & s1 & s0;
            WITH sel_line SELECT
                z <= '0' AFTER 3 NS  WHEN  "00000000",
                        i7 AFTER 3 NS  WHEN  "10000000",
                        i6 AFTER 3 NS  WHEN  "01000000",
```

รูปที่ 2.15(ข) การบรรยายเชิงข้อมูลของวงจร 8-to-1 มัลติเพล็ก

```
i5 AFTER 3 NS  WHEN  "00100000",
i4 AFTER 3 NS  WHEN  "00010000",
i3 AFTER 3 NS  WHEN  "00001000",
i2 AFTER 3 NS  WHEN  "00000100",
i1 AFTER 3 NS  WHEN  "00000010",
i0 AFTER 3 NS  WHEN  OTHERS;
```

รูปที่ 2.15 (ต่อ) (ข) การบรรยายเชิงข้อมูลของวงจร 8-to-1 มัลติเพล็ก

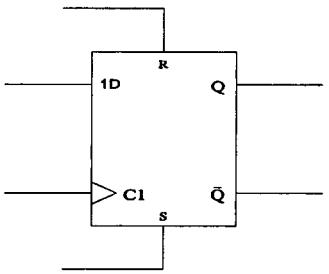
การกำหนดการ์ด

ในการบรรยายเชิงข้อมูล เราสามารถตั้งเงื่อนไขสำหรับการกำหนดค่าสัญญาณใดก็ได้ โดยใช้คำสั่ง GUARDED ซึ่งมีรูปแบบในการเขียนดังนี้

```
target <= GUARDED waveforms or conditional waveforms or selected waveform;
```

รูปที่ 2.16 ตัวอย่างใช้ GUARDED ในการบรรยายการทำงานของ D Flip-Flop

การกำหนด GUARDED ทำได้โดยใช้รูปแบบของคำสั่ง BLOCK ตามด้วยเงื่อนไขในวงเล็บนี้คือ GUARDED นั่นเอง ในรูปที่ 2.17 เป็นการบรรยายการทำงานของ D Flip-Flop ที่มีการทำงานเมื่อมีสัญญาณนาฬิกามีการเปลี่ยนแปลงจาก 0 เป็น 1 หรือสัญญาณขาขึ้น จะเห็นว่า GUARDED ถูกใช้เป็นเงื่อนไขในการกำหนดค่าให้กับ q และ qb นั่นคือ ถ้าเงื่อนไขภายในวงเล็บมีค่าเป็น FALSE (GUARDED เป็น FALSE) ค่า d และ NOT d ก็จะไม่ถูกส่งค่าให้กับ q และ qb



รูปที่ 2.17 (ก) โมเดลของ D Flip-Flop

```
ENTITY d_flipflop IS
    GENERIC(delay 1 : TIME := 4 Ns : delay2 : TIME := 5 nS)
        PORT(d, c : IN BIT ; q, qb : OUT BIT);
END d_flipflop;
ARCHITECTURE  guarding OF d_flipflop IS
    BEGIN
        ff: BLOCK(c= '1' AND NOT c'STABLE)
       BEGIN
          q <= GUARDED d AFTER delay1;
          qb <= GUARDED NOT d AFTER delay2;
       END BLOCK ff;
END guarding;
```

รูปที่ 2.17 (ข) ตัวอย่างการใช้ GUARDED

การกำหนดโครงข่ายของการ์ด

เราสามารถกำหนดการ์ดเป็นโครงข่ายซ้อนกันได้โดยการกำหนดบล็อกซ้อนกัน ซึ่งสามารถจะทำให้เราออกแบบ อุปกรณ์ที่มีการทำงานที่ซับซ้อนมากกว่า D Flip-Flop ในตัวอย่างข้างต้น ดังรูปที่ 2.18 นอกจากสัญญาณอินพุท d จะถูกส่งไปยัง q โดยมีเงื่อนไขว่า ((c='1') AND (NOT c'STABLE)) ต้องเป็น TRUE แล้วยังมีเงื่อนไขเพิ่มเติมอีกว่าสัญญาณ

enable (e) ต้องเป็น TRUE ด้วย GRUADED ใน gate : BOLCK แทนเงื่อนไขใดๆ ที่อยู่ภายนอก ในกรณีนี้หมายถึงเงื่อนไขภายในวงเล็บ ((c='1')AND(NOT c'STABLE)) ในขณะที่ GUARDED จะแทนเงื่อนไขทั้งหมดในกรณีนี้คือ (e='1') AND ((c='1')AND(NOT c'STABLE))

```
ENTITY d_flipflop IS
    GENERIC(delay 1 : TIME := 4 Ns : delay2 : TIME := 5 nS)
        PORT(d, c, e : IN BIT ; q, qb : OUT BIT);
END d_flipflop;
```

```
ARCHITECTURE  guarding OF d_flipflop IS
    BEGIN
        edge: BLOCK((c= '1') AND (NOT c'STABLE))
BEGIN
        gate : BLOCK (e = '1' AND GUARD)
        BEGIN
            q <= GUARDED d AFTER delay1;
            qb <= GUARDED NOT d AFTER delay2;
        END BLOCK gate
    END BLOCK edge;
END guarding;
```

รูปที่ 2.18 ตัวอย่างการใช้ NESTING GUARD

การเลือกสรรข้อมูล

ในทางฮาร์ดแวร์มีหลายกรณีที่มีความจำเป็นต้องเชื่อมต่อหลายเอาท์พุทไปยังอินพุตหนึ่งซึ่งในกรณีเช่นนี้อาจจะทำให้เกิดความสับสนของสัญญาณที่ปะปนกัน ทำให้ไม่สามารถรู้ค่าที่แน่นอนได้ ในทางภาษา VHDL หมายถึง การกำหนดค่าจากหลายสัญญาณให้กับสัญญาณ

เดียวซึ่งจะให้ผลลัพธ์ที่สับสนได้เหมือนกัน ดังตัวอย่างในรูปที่ 2.19 ซึ่งจะไม่สามารถบอกได้เลยว่าค่าของ circuit_node มีค่าเป็นอะไรถ้าสมมติว่าค่าอินพุท a เป็น “1” และค่าอินพุตอื่นเป็น “0”

```
ENTITY y_circuit IS
        PORT(a, b, d, c : IN BIT ; z : OUT BIT);
END y_circuit;
ARCHITECTURE  smoke_generator OF y_circuit IS
  SIGNAL circuit_node : BIT;
   BEGIN
        circuit_node <= a;
        circuit_node <= b;
        circuit_node <= c;
        circuit_node <= d;
        z <= circuit_node;
END smoke_generator;
```

รูปที่ 2.19 การกำหนดค่าสัญญาณที่ทำให้เกิดความสับสน

กรณีเช่นนี้เราสามารถแก้ไขได้โดยการกำหนดรูปแบบฟังก์ชั่นขึ้นมาเพื่อใช้เลือกสรรข้อมูลรูปที่ 2.20 แสดงตัวอย่างฟังก์ชั่นเลือกสรรข้อมูลแบบ AND ซึ่งกำหนดให้สัญญาณที่เข้ามาที่โหนดเดียวกันจะถูกนำมารวมกันในลักษณะของการ AND กันเสียก่อน เมื่อนำฟังก์ชั่นในรูปที่ 2.20 มาใช้ร่วมกับการบรรยายในรูปที่ 2.19 ก็สามารถแก้ปัญหาเรื่องความสับสนของข้อมูลได้ ดังแสดงในรูปที่ 2.21

```
FUNCTION anding(drivers :BIT_VECTOR) RETURN BIT IS
  VARIABLE  accumulate : BIT := '1';
  BEGIN
      FOR I IN drivers'RANGELOOP
          accumulate := accumulate AND drivers(i);
      END LOOP;
  RETURN FUNCION;
END anding;
```

รูปที่ 2.20 ฟังก์ชั่นเลือกสรรข้อมูลแบบ anding

```
ARCHITECTURE wired_and OF y_circuit IS
    FUNCTION anding(drivers :BIT_VECTOR) RETURN BIT IS
    VARIABLE  accumulate : BIT := '1';
    BEGIN
        FOR I IN drivers'RANGELOOP
            accumulate := accumulate AND drivers(i);
        END LOOP;
    RETURN FUNCION;
    END anding;
SIGNAL circuit_node : anding BIT;
 BEGIN
    circuit_node <= a;
    circuit_node <= b;
```

รูปที่ 2.21 การใช้ฟังก์ชั่น anding กับการเลือกสรรข้อมูล

```
        circuit_node <= c;
        circuit_node <= d;
        z <= circuit_node;
END wire_and;
```

รูปที่ 2.21 (ต่อ) การใช้ฟังก์ชั่น anding กับการเลือกสรรข้อมูล

2.3.4 การบรรยายเชิงโครงสร้าง

การบรรยายการทำงานของระบบในเชิงโครงสร้างจะต้องแสดงรายการของอุปกรณ์ทั้งหมดที่ใช้ในระบบและต้องกำหนดการเชื่อมต่อระหว่างอุปกรณ์ต่างๆด้วย เพราะว่าการบรรยายในระดับนี้เป็นการบรรยายที่ใกล้เคียงลักษณะของฮาร์ดแวร์จริงที่สุด ภาษา VHDL ได้จัดเตรียมเครื่องมือและลักษณะโครงสร้างของการบรรยายในระดับนี้ไว้ที่สำคัญ 4 ลักษณะคือ

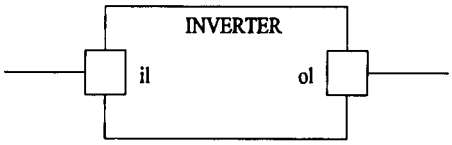
1. ความสามารถในการเลือกหรือกำหนดอุปกรณ์ที่ต้องการได้จากไลบรารี
2. การสร้างไลบรารีเพื่อเก็บอุปกรณ์ที่ผู้ใช้ออกแบบไว้เองได้
3. กลไกในการเชื่อมต่อระหว่างอุปกรณ์
4. โครงสร้างการกำหนดอุปกรณ์ชนิดเดียวกันซ้ำๆกัน

ไลบรารีพาร์ท

ในหัวข้อนี้เป็นตัวอย่างการใช้ภาษา VHDL อธิบายการทำงานของวงจรเกทพื้นฐาน คือ อินเวอร์เตอร์เกท แนนด์เกท2 อินพุต และแนนด์ 3 อินพุต โดยการเขียนอธิบายการทำงานของวงจรเกทขึ้นเองหรือเรียกใช้งานจากไลบรารีในรูปส่วนของเกทก็ได้

อินเวอร์เตอร์เกทโมเดล

รูปที่ 2.22 แสดงโมเดลของอินเวอร์เตอร์เกท โดยในการบรรยายการเชื่อมต่อนั้น il เป็นสัญญาณอินพุต ol เป็นขาสัญญาณเอาท์พุต และการบรรยายการทำงานของอินเวอร์เตอร์เกทนั้น ol จะเป็นกลับของสัญญาณ il หลังจากเวลาผ่านไปได้ 4 nS



รูปที่ 2.22 (ก) โมเดลของอินเวอร์เตอร์เกท

```
Entity inv IS
        PORT(i1 : IN BIT ;   01: OUT BIT);
End inv;
```

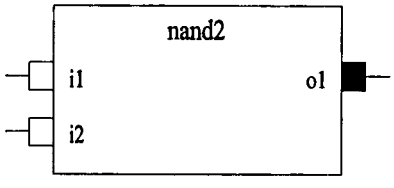
รูปที่ 2.22 (ข) การบรรยายการเชื่อมต่อ

```
ARCHITECTURE single_delay OF inv IS
  BEGIN
    01 <= NOT i1  AFTER  4 nS;
End single_delay;
```

รูปที่ 2.22 (ค) การบรรยายการทำงาน

แนนด์เกทโมเดล

รูปที่ 2.23 แสดงสัญลักษณ์และการบรรยายของแนนด์เกทชนิด 2 อินพุต



รูปที่ 2.23 (ก) โมเดลของแนนด์เกท 2 อินพุต

```
ENTITY nand2 IS
    PORT(i1, i2 : IN BIT ;  01 : OUT BIT);
END nand2;
```

รูปที่ 2.23 (ข) การบรรยายการเชื่อมต่อ

```
ARCHITECTURE single_delay OF nand2 IS
  BEGIN
    01 <= i1 NAND i2 AFTER  5 nS;
End single_delay;
```

รูปที่ 2.23 (ค) การบรรยายการทำงาน

รูปที่ 2.24 แสดงสัญลักษณ์และการบรรยายของแนนด์เกทชนิด 3 อินพุต

รูปที่ 2.24 (ก) โมเดลของแนนด์เกท 3 อินพุต

```
ENTITY nand3 IS
    PORT(i1, i2,i3 : IN BIT ;  01 : OUT BIT);
END nand3;
```

รูปที่ 2.24 (ข) การบรรยายการเชื่อมต่อ

```
ARCHITECTURE single_delay OF nand3 IS
  BEGIN
    01 <= NOT (i1 NAND i2 AND i3) AFTER  6 nS;
End single_delay;
```

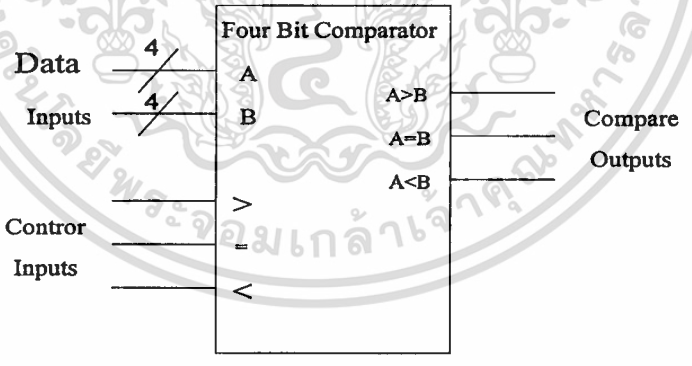
รูปที่ 2.24 (ค) การบรรยายการทำงาน

การเชื่อมต่ออุปกรณ์พื้นฐาน

เมื่อออกแบบเกทพื้นฐานเรียบร้อยแล้ว ขั้นตอนต่อไปเป็นการนำเกทพื้นฐานเหล่านี้มาเชื่อมต่อกันเป็นวงจร ในหัวข้อนี้จะแสดงการออกแบบและการบรรยายเชิงโครงสร้างของวงจรเปรียบเทียบโดยใช้อินเวอร์เตอร์เกทและแนนด์เกท

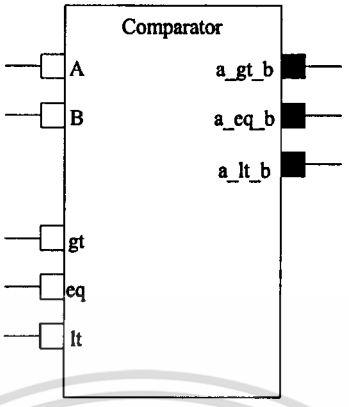
ตัวอย่างการออกแบบวงจรเปรียบเทียบ

วงจรเปรียบเทียบทีละบิต (Bit_comparator) ประกอบด้วยสัญญาณข้อมูล 2 อินพุต สัญญาณควบคุม 3 อินพุต และสัญญาณเปรียบเทียบ 3 เอาท์พุต ดังรูปที่ 2.25



รูปที่ 2.25 วงจรเปรียบเทียบทีละบิต

เอาท์พุต A>B มีค่าเป็น “1” เมื่ออินพุต A มีค่ามากกว่า B (AB = 10) หรือ A เท่ากับ B และอินพุต > มีค่าเป็น “1” เอาท์พุต A=B มีค่าเป็น “1” เมื่ออินพุต A เท่ากับ B และอินพุต = มีค่าเป็น “1” ส่วนเอาท์พุต A<B จะตรงข้ามกับเอาท์พุต A>B นั่นคือมีค่าเป็น “1” เมื่ออินพุต A มีค่าน้อยกว่า B(AB = 01) หรือถ้า A เท่ากับ B และอินพุต < มีค่าเป็น “1” ซึ่งสามารถเขียนเป็นสมการบูลีนในรูปของแนนด์เกทได้ ดังรูปที่ 2.26




```
a_gt_b  = ((a.gt)'.(b'.gt)'.(a.b')')'
a_eq_b  = ((a.b.eq)'.(a'.b'.eq)')'
a_lt_b  = ((a'.lt)'.(b.lt)'.(a'.b)')'
```

รูปที่ 2.26 สัญลักษณ์และสมการบูลีนของวงจรเปรียบเทียบทีละบิต

ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบทีละบิต

รูปที่ 2.27 แสดงการบรรยายเชื่อมต่อของวงจรเปรียบเทียบทีละบิตตามสัญลักษณ์ใน **รูปที่** 2.27 เครื่องหมาย "--" ใช้แทนข้อความอธิบาย (Comment)

```
ENTITY bit_comparator IS
        port (a,b,          --data inputs
              gt,           --previous  greater  than
              eq,           --previous  equal
```

รูปที่ 2.27 การบรรยายการเชื่อมต่อของโปรแกรมเปรียบเทียบทีละบิต

```
        lt : IN  bit        --previous  less then
    a_gt_b,                 --greater
    a_eq_b,                 --equal
    a_lt_b : OUT BIT);      --LESS THEN
END bit_comparator;
```

รูปที่ 2.27 (ต่อ) การบรรยายการเชื่อมต่อของโปรแกรมเปรียบเทียบทีละบิต

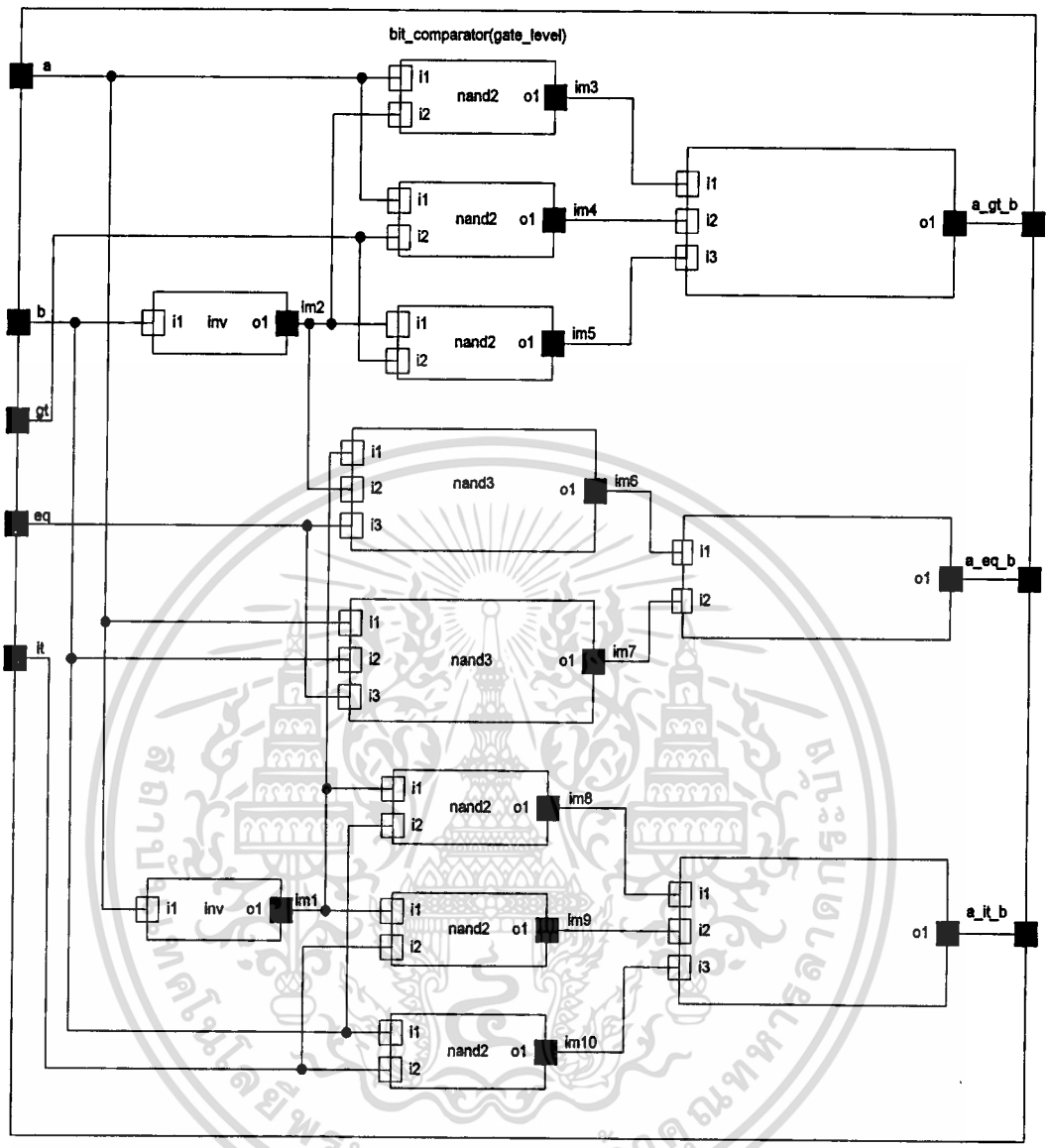
รูปที่ 2.28 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบ โดยกำหนดชื่อการบรรยายเป็น Gate_Level ภายในมีการกำหนดรายการเกทพื้นฐานที่ต้องใช้เอาไว้ 3 ชนิด และกำหนดชื่อเป็น n1,n2,n3 โดยกำหนดให้ n1 อ้างอิงถึง อินเวอร์เตอร์เกท n2 อ้างอิงถึง แนนด์เกท 2 อินพุต และ n3 อ้างอิงถึงแนนด์เกท 3 อินพุต คำสั่ง FOR ALL : n1 เป็นการกำหนดใช้อุปกรณ์ทุกตัวที่ขึ้นต้นชื่อด้วย n1 ให้อ้างอิงกับอินเวอร์เตอร์เกท ในกรณีของ n2 และ n3 ก็เช่นเดียวกัน WORK เป็นการกำหนดที่อยู่หรือไลบรารี่ ที่เก็บอุปกรณ์ที่อ้างอิงถึง ซึ่งอาจจะเป็นชื่อไดเร็กทรอรี่ใดๆ (กรณีที่ใช้คำ WORK จะหมายถึงไดเร็กทรอรี่ปัจจุบัน) รูปที่ 2.29 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิทตามสมการบูลีนโดยใช้เกทพื้นฐานที่ได้ออกแบบไว้แล้ว

```
ARCHITECTURE gate_level OF bit_comparator IS
  COMPONENT n1 PORT(I1 : IN BIT; 01 : OUT BIT); END  COMPONENT
  COMPONENT n1 PORT(I1, I2 : IN BIT; 01 : OUT BIT); END  COMPONENT
  COMPONENT n1 PORT(I1, I2, I3 : IN BIT; 01 : OUT BIT); END  COMPONENT
FOR ALL : n1 USE ENTITY WORK.inv(single_delay);
FOR ALL : n2 USE ENTITY WORK.nand2(single_delay);
FOR ALL : n3 USE ENTITY WORK.nand3(single_delay);
SIGNAL im1, im2, im3, im4, im5, im6, im7, im8, im9, im0 : BIT;
```

รูปที่ 2.28 การบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต

```
BEGIN
-- a_gt_b output
        g0 : n1 PORT MAP(a,im1);
        g1 : n1 PORT MAP(b,im2);
        g2 : n2 PORT MAP(a,im3,im3);
        g3 : n2 PORT MAP(a,gt,im4);
        g4 : n2 PORT MAP(im2,gt,im5);
        g5 : n3 PORT MAP(im3,im4,im4,im5,a_gt_b);

-- a_eq_b output
        g6 : n3 PORT MAP(im1,im2,eq,im6);
        g7 : n3 PORT MAP(a,b,eq,im7);
        g8 : n2 PORT MAP(im6,im7,a_eq_b);

-- a_lt_b output
        g9 : n2 PORT MAP(im1,b,im8);
        g10 : n2 PORT MAP(im1,lt,im9);
        g11 : n2 PORT MAP(b,lt,im10);
        g12 : n3 PORT MAP(im8,im8,im10,a_lt_b);
END gate_level;
```

รูปที่ 2.28 (ต่อ) การบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต



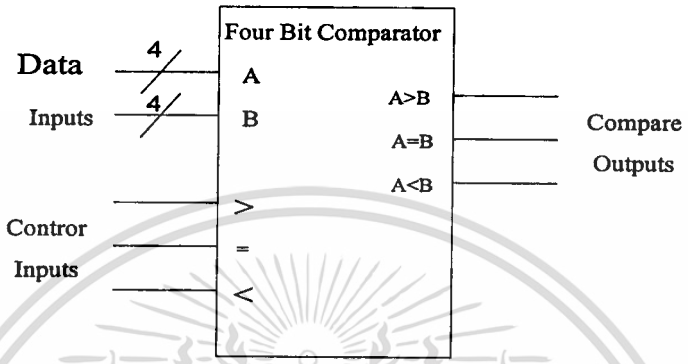
รูปที่ 2.29 องค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิต

การเชื่อมต่ออุปกรณ์ชนิดเดียวกัน

ในการเชื่อมต่ออุปกรณ์ต่างๆ เข้าด้วยกันจะต้องอ้างอิงถึงอุปกรณ์ทีละตัว แต่หากเป็นการเชื่อมต่ออุปกรณ์ชนิดเดียวกัน จำนวนหลายตัวเข้าด้วยกันแล้วภาษา VHDL ได้เตรียมเครื่องมือซึ่งเปรียบเสมือนเส้นทางลัดในการบรรยาย โดยไม่จำเป็นต้องอ้างอิงถึงอุปกรณ์ทุกตัว ในหัวข้อต่อไปจะยกตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิต ที่เรียกว่า nibble_comparator ซึ่งจะประกอบด้วยวงจรเปรียบทีละบิต ที่ได้ออกแบบไว้ในหัวข้อที่กล่าวมาแล้วก่อนหน้านี้จำนวน 4 ตัว ต่อเข้าด้วยกัน

ตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิต

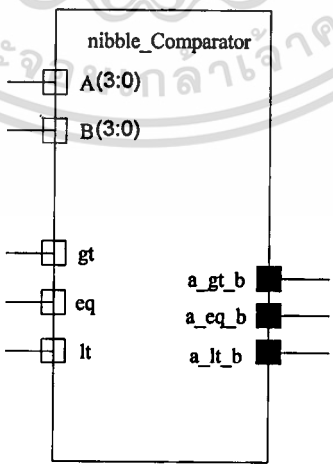
วงจรเปรียบเทียบขนาด 4 บิตมีสัญญาณข้อมูลขนาด 4 บิตจำนวน 2 อินพุต สัญญาณควบคุม 3 อินพุต และสัญญาณผลเปรียบเทียบ 3 เอาท์พุต ดังรูปที่ 2.30



รูปที่ 2.30 วงจรเปรียบเทียบขนาด 4 บิต

ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบขนาด 4 บิต

รูปที่ 2.31 แสดงสัญลักษณ์ของวงจร nibble_comparator และการบรรยายเชื่อมต่อของวงจร อินพุต a และ b เป็นอินพุตสำหรับข้อมูลขนาด 4 บิต เราสามารถกำหนดชนิดของอินพุตเป็น BIT_VECTOR ซึ่งเป็นอาเรย์ของ BIT ได้ด้วย



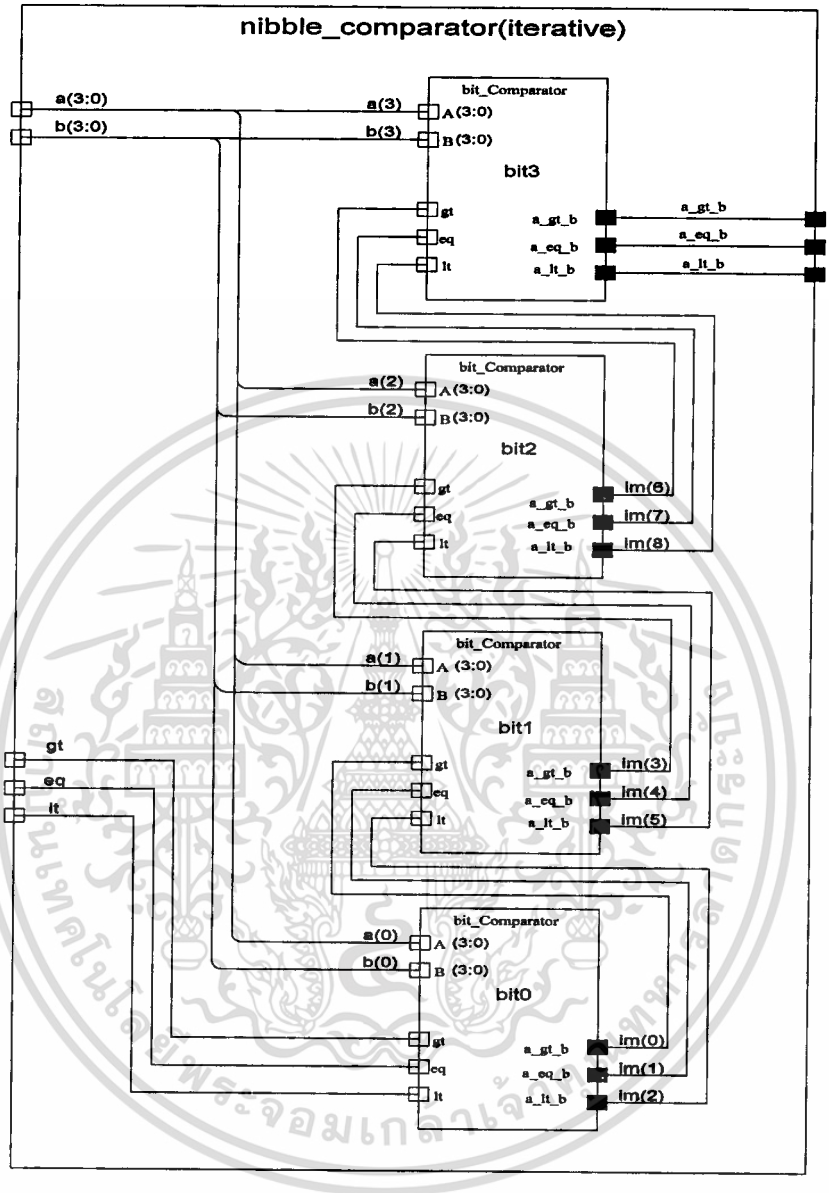
รูปที่ 2.31 (ก) สัญลักษณ์ของวงจร nibble_comparator

```
ENTITY nibble_comparator IS
 PORT (a,b, IN BIT_VECTOR(3 DOWNTO 0);          --data inputs
                    gt,              --previous  greater  than
                    eq,              --previous  equal
                    lt : IN  bit     --previous  less then
              a_gt_b,                --a greater than b
              a_eq_b,                --a equal b
              a_lt_b : OUT BIT);     --a less than b
END nibble_comparator;
```

รูปที่ 2.31 (ข) การบรรยายเชื่อมต่อของวงจร nibble_comparator

```
ARCHITECTURE iterative OF nibble_comparator IS
    COMPONENT comp1
      PORT(a, b, gt, eq, lt : IN BIT ; a_gt_b, a_eq_b, a_lt_b : out bit);
    END COMPONENT;
    FORALL  : comp1  USE ENTITY WORK.bit_comparator(gate_level);
BEGIN
    c0 : comp1 PORT MAP (a(0), b(0), gt, eq, lt, im(0), im(1), im(2));
  clto2 : FOR i  IN 1 TO 2 GENERATE
     c  : comp1 PORT MAP (a(i), b(i), im(I*3-3), im(I*3-2), im(I*3-1),
                                 im(I*3+0), im(I*3+1), im(I*3+2));
END GENERATE;
     c3 : comp1 PORT MAP (a(3), b(3), im(6), im(7), im(8),
                          a_gt_b, a_eq_b, a_lt_b);
end iterative
```

รูปที่ 2.32 การบรรยายการทำงานของวงจร



รูปที่ 2.33 ลักษณะของวงจรแบบสัญลักษณ์

รูปที่ 2.32 แสดงการบรรยายการทำงานของวงจรโดยใช้ comp1 ซึ่งอ้างอิงถึงวงจรเปรียบทีละบิต ค่า n เป็นค่าคงที่ที่ใช้กำหนดจำนวนของวงจร comp1 ซึ่งในกรณีนี้เป็นวงจรเปรียบเทียบขนาด 4 บิต ดังนั้นจึงกำหนดค่า n เป็น 4 การบรรยายได้ใช้ FOR loop และ GENERLATE ในการกำหนดการเชื่อมต่อของอุปกรณ์ชนิดเดียวกัน สังเกตว่าการบรรยายในลักษณะนี้จะช่วยประหยัดเวลาและสะดวกมากขึ้น ทั้งนี้เราสามารถกำหนดจำนวนของอุปกรณ์

ที่มาเชื่อมต่อกันจำนวนเท่าใดก็ได้ โดยไม่ต้องเปลี่ยนรูปแบบใดๆ เลย นอกจากค่าของ n เท่านั้น ในรูปที่ 2.33 แสดงลักษณะของวงจรแบบสัญลักษณ์

2.4 โครงสร้างภายในของอุปกรณ์ FPGAs

FPGAs จัดเป็นวงจรรวมเฉพาะกิจชนิดหนึ่งที่สามารถโปรแกรมเป็นวงจรเชิงเลขใด ๆ ก็ได้ เช่นเดียวกับ EPLD ต่างกันที่ EPLD โปรแกรมลงบน EPROM ภายในและสามารถโปรแกรมใหม่ได้หลังจากนำไปลบด้วยแสง UV แต่ FPGAs โปรแกรมลงบนสแตติกแรมภายในด้วยข้อมูลที่อยู่ภายนอก และสามารถโปรแกรมใหม่ได้โดยการรีเซตด้วยสัญญาณไฟฟ้า นอกจากนั้น FPGAs ยังประหยัดไฟและมีความจุวงจรสูง (จำนวนเกทมาก) ได้อีกด้วย

วงจรรวมชนิดที่ใช้ในโครงงานนี้ผลิตโดยบริษัทไซลิงค์ (Xilinx) ซึ่งเป็นบริษัทที่ทำการค้นคว้าร่วมกับบริษัทเอ็มเอ็มไอ (MMI) สร้างเป็นกลุ่มของเกทจำนวน 600-25,000 เกทดังแสดงในตารางที่ 2.1 การที่ต้องบอกขนาดของวงจรรวมเป็นจำนวนเกทเพราะจะได้รู้ว่าขนาดของวงจรที่ได้ ออกแบบไว้สามารถโปรแกรมลงบนวงจรรวม FPGAs ได้หรือไม่

ตารางที่ 2.1 คุณสมบัติของ FPGAs ตระกูลต่าง ๆ

FPGAs	Appr.Gate Cout	Max I/Os	Flip-Flops	RAM bit	Available CLBs
XC2064	1,000	58	122	0	64
XC2018	1,500	74	174	0	100
XC3020/3120	1,800	64	256	0	64
XC3030/3130	2,700	80	360	0	100
XC3042/3142	3,700	96	480	0	144
XC3064/3164	5,500	120	688	0	244
XC3090/3190	7,500	144	928	0	320
XC3195	9,000	176	1,320	0	484
XC4002A	2,000	64	256	2,048	64
XC4003/4003A	3,000	80	360	3,200	100

ตารางที่ 2.1 (ต่อ) คุณสมบัติของ FPGAs ตระกูลต่าง ๆ

XC4003H	3,000	160	200	3,200	100
XC4004A	4,000	960	480	4,608	144
XC4005/4005A	5,000	122	616	6,072	196
XC4005H	5,000	192	392	6,272	196
XC4006	6,000	128	768	8,192	256
XC4008	8,000	144	936	10,368	324
XC4010	10,000	160	1,120	12,800	400
XC4013	13,000	192	1,536	18,432	576
XC4025	25,000	256	25,560	32,768	1,024

NAIINR2 หมายถึง เกท NAND2 หรือเกท NOR2

ตารางที่ 2.2 ตารางประมาณนับเกทของเกทพื้นฐาน

Gate	Equipvalent gate count	Gate	Equipvalent gate count
INV	1	RS Latch	3
NAIINR2	1	D latch	4
NAIINR3	2	D latch with CLR	5
NAIINR4	2	D latch with PRE	5
NAIINR6	5	D latch with PRE/CLR	6
NAIINR8	6	D F/F	6
NAIINR9	7	D F/F with CLR	7
NAIINR12	8	D F/F with PRE	7
NAIINR16	11	D F/F with PRE/CL	8
BUFF	2	JK F/F with CLR	9

ตารางที่ 2.2 (ต่อ) ตารางประมาณนับเกทของเกทพื้นฐาน

ANIIOR2	2	JK F/F with PRE	12
ANIIOR3	2	JK F/F PRE/CL	13
ANIIOR4	3	T F/F with CLR	8
XOR2	3	T F/F with PRE	8
XNOR2	3	T F/F PRE/CL	9

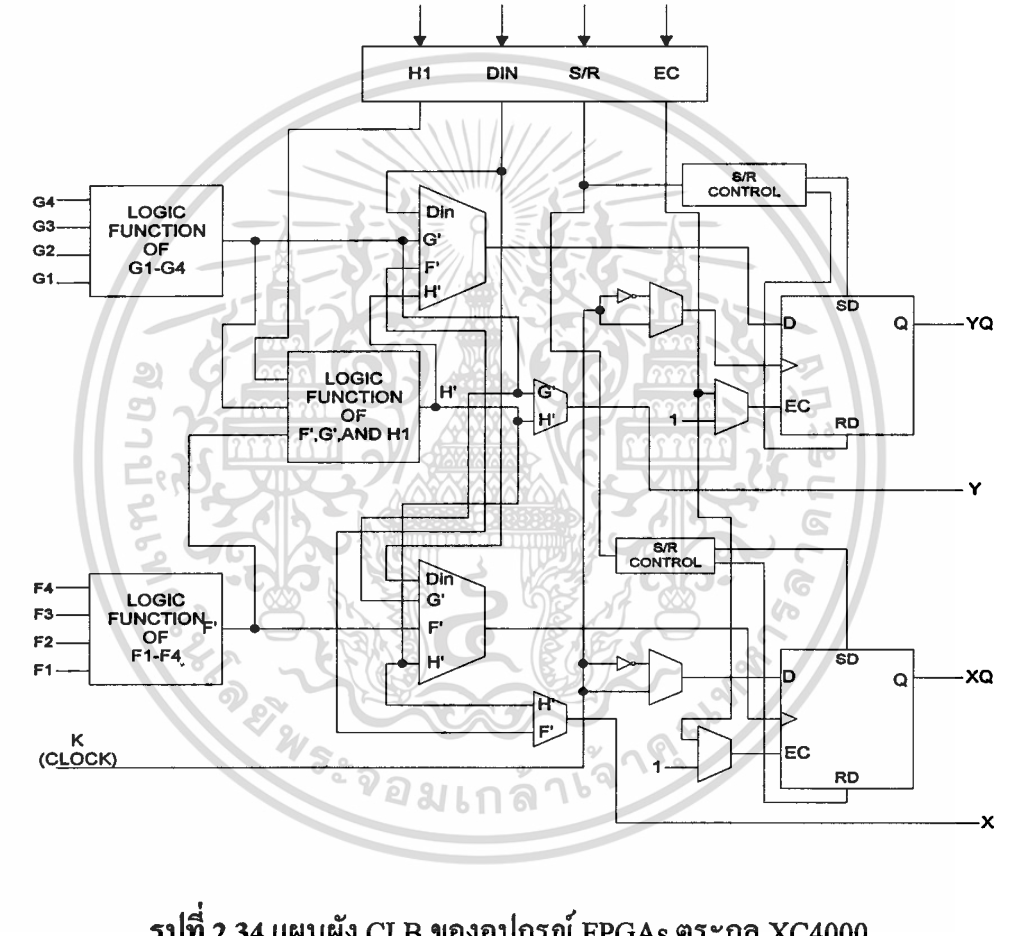
FPGAs มีโครงสร้างภายในใกล้เคียงกับสถาปัตยกรรมของเกทอาเรย์ (GAL, Gate Array Logic)มาก สามารถโปรแกรมและลบคอนฟิกกูเรชั่น (Configuration) ภายในสแตติกแรม (Static RAM) ได้โดยใช้กระแสไฟฟ้า ซึ่งทำการโปรแกรมได้โดยดึงข้อมูลฐานสิบหกจากภายนอกเช่น Parallel EPROM หรือ Serial PROM ต่างกับ EPLD,PAL ที่มี EPROM อยู่ในตัว ภายใน FPGAs จะจัดเรียงเห็นลอจิกเซลล์ล้อมรอบภายนอกด้วยอินพุตเอาท์พุตเซล FPGAs ตัวแรกที่ผลิตโดยบริษัทไซลิงค์คือเบอร์ XC2064 (2000 Family) ประกอบด้วยเซลเรียงกันเป็นเมตริกซ์ (Matrix) เป็นจำนวน 64 เซล หลังจากนั้นผลิต FPGAs ตระกูล 3000 และ 4000 ซึ่งมีโครงสร้างซับซ้อนขึ้นสามารถเพิ่มจำนวนเกทได้มากขึ้นและดีขึ้น แต่ละเซลเรียกว่า CBL (Configurable Logic Block)

2.4.1 ส่วนที่เป็นองค์ประกอบของลอจิก (Configurable Logic Block)

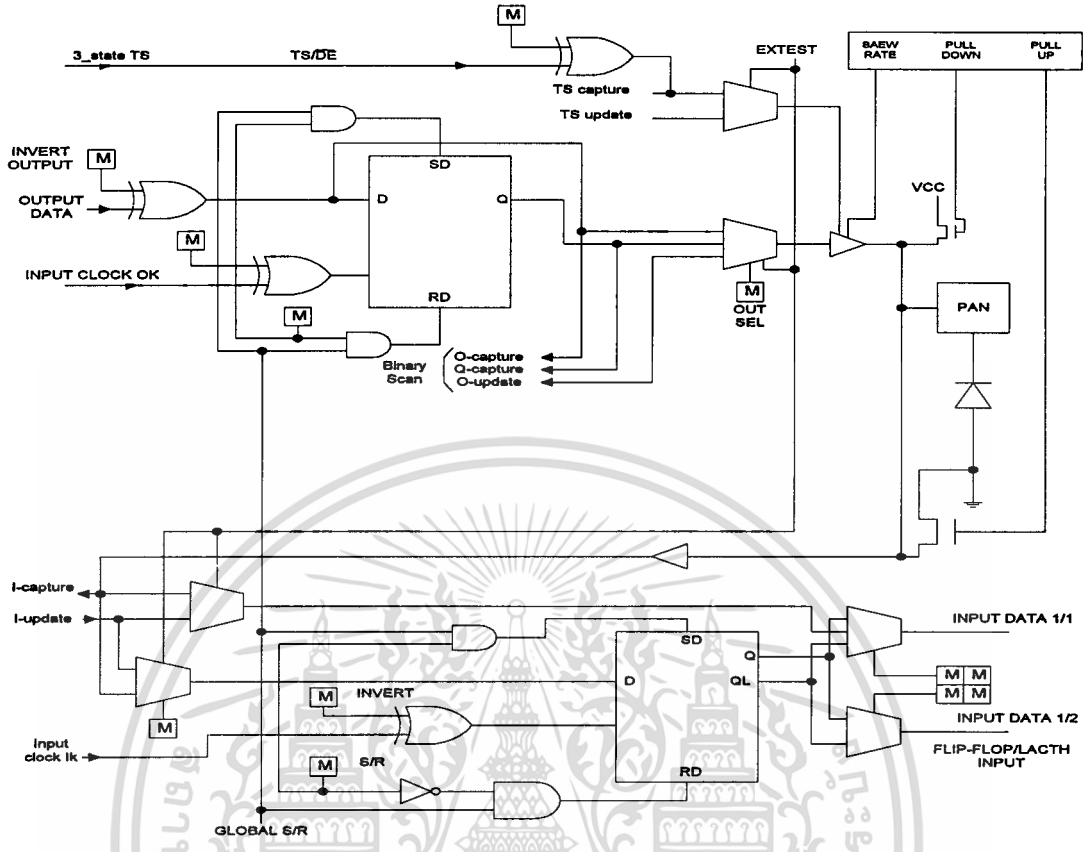
CLB จะจัดเรียงกันแบบเมตริกซ์แบบอาเรย์ขนาด M*N การออกแบบนั้นสามารถทำได้โดยการจัดการวาง CLB และต่อเชื่อมขาของ CLB ให้ต่อถึงกัน เราสามารถจัด CLB ให้เชื่อมต่อถึงกันได้โดยการทำด้วยมือหรือให้โปรแกรมที่สนับสนุน FPGAs ทำให้โดยอัตโนมัติ โดยวิธีของมันเองสำหรับไฟล์ที่ได้จากโปรแกรมเหล่านี้ เราเรียกว่าไฟล์ที่กำหนดการวางอุปกรณ์ (Configuration File) ซึ่งจะบรรจุโครงร่างภายในของ CLB ตามความเหมาะสมอีกด้านหนึ่ง ไฟล์กำหนดการวางอุปกรณ์นั้นจะเป็นไฟล์กระแสข้อมูล (Bit Stream) ซึ่งสามารถใช้โปรแกรมหน่วยความจำภายในของ FPGAs ได้ สำหรับรูปที่ 2.38 แสดง CLB ของ FPGAs ตระกูล 4000

2.4.2 ส่วนอินพุตและเอาท์พุตของอุปกรณ์ FPGAs

รอบนอกของ FPGAs จะประกอบด้วย IOBs ประมาณ 64 ถึง 144 ตัว ซึ่งขึ้นอยู่กับตระกูลของ FPGAs ซึ่ง IOBs จะเป็นตัวเชื่อมต่อระหว่างภายในกับภายนอกของวงจรลอจิกของ FPGAs ลักษณะของ IOBs จะมีลักษณะ 2 ทิศทาง สามารถโปรแกรมให้เป็นอินพุตหรือเอาท์พุตก็ได้ สำหรับรูปที่ 2.39 แสดง IOBs ของ FPGAs ตระกูล 4000



รูปที่ 2.34 แผนผัง CLB ของอุปกรณ์ FPGAs ตระกูล XC4000



รูปที่ 2.35 แผนผัง IOBs ของอุปกรณ์ FPGAs ตระกูล XC4000

2.5 รายละเอียดการใช้งานของอุปกรณ์ FPGAs

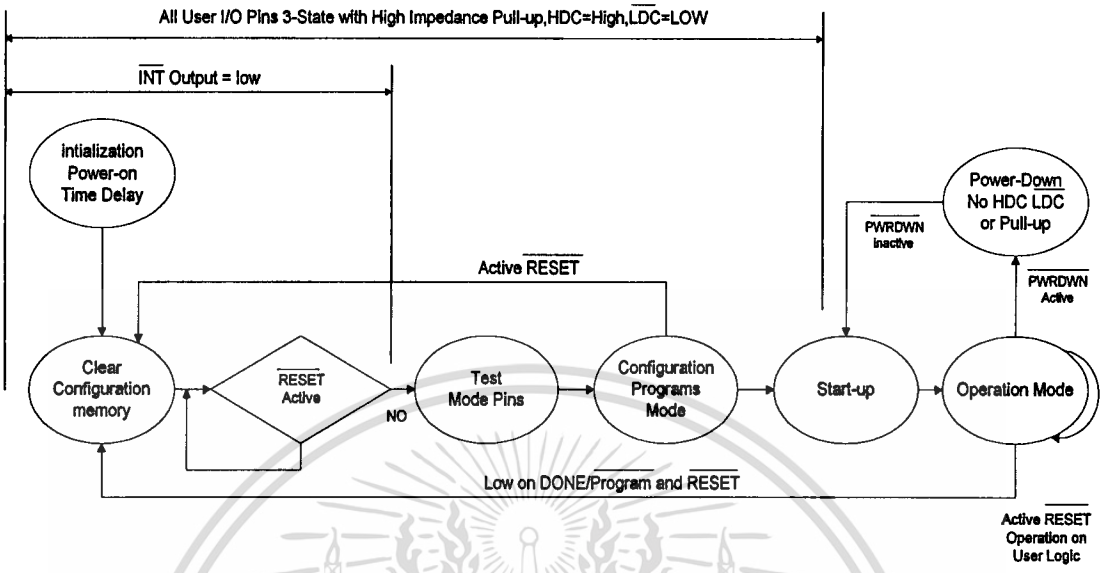
FPGAs สามารถทำงานได้หลายลักษณะ โดยกำหนดได้ที่ขาสัญญาณ M0 M1 M2 ดังแสดงในตารางที่ 2.3 ในลักษณะมาสเตอร์พาราเรล (Master Parallel Mode) รับโปรแกรมคอนฟิกทีละ 1 ไบท์ (Byte) จากหน่วยความจำภายนอกที่เป็นแบบขนาน โดยสามารถรับโปรแกรมคอนฟิก (Config) จากแอดเดรส (Addess) ต่ำหรือสูงก่อนก็ได้ การต่อลักษณะเพอริเฟอรัล (Peripherai) จะรับโปรแกรมคอนฟิกทีละ 1 ไบท์จากไมโครโปรเซสเซอร์ โดยสามารถโต้ตอบกันได้ว่าพร้อมหรือไม่ที่จะรับข้อมูลต่อไป การต่อลักษณะสเลฟซีเรียล (Slave Serial) จะรับโปรแกรมคอนฟิกทีละ 1 บิท (Bit) จากไมโครโปรเซสเซอร์ตามสัญญาณอินพุต CCLK ส่วนการต่อลักษณะมาสเตอร์ซีเรียล (Master Serial) จะรับโปรแกรมคอนฟิกทีละ 1 บิตจากหน่วยความจำภายนอกที่เป็นแบบอนุกรม

ตารางที่ 2.3 โหมดต่าง ๆ ของการคอนฟิกกูเรชั่น

Mode	M2	M1	M0	CCLK	Data
Master Serial	0	0	0	output	Bit-Serial
Slave Serial	1	1	1	input	Bit-Serial
Master Parallel up	1	0	0	output	Byte-Wide,00000 up
Master Parallel down	1	1	0	output	Byte-Wide,3FFFF down
Peripheral Synchr	0	1	1	input	Byte-Wide
Peripheral Asynchr	1	0	1	output	Byte-Wide
Reserved	0	1	0	---	---
Reserved	0	0	1	---	---

จากความต้องการสร้างให้ใช้กระแสไฟฟ้าต่ำจากลักษณะการต่อใช้งานทั้ง 5 แบบจึงมีเพียง 2 แบบเท่านั้นที่เหมาะสมคือแบบมาสเตอร์ซีเรียลและแบบสเลฟซีเรียลส่วนแบบมาสเตอร์พาราแรล ต้องใช้ EPROM 27CXXX ซึ่งกินกระแสมากกว่า PROM XC17XXX เหมาะในการทดสอบต้นแบบก่อนเมื่อวงจรต้นแบบทำงานได้ถูกต้องแล้วจึงทำการอัดโปรแกรมลง PROM อีกทีหนึ่งเพราะว่าในแบบพาราแรลนั้น EPROM สามารถโปรแกรมได้ใหม่ต่างกับ PROM ที่โปรแกรมได้เพียงครั้งเดียว

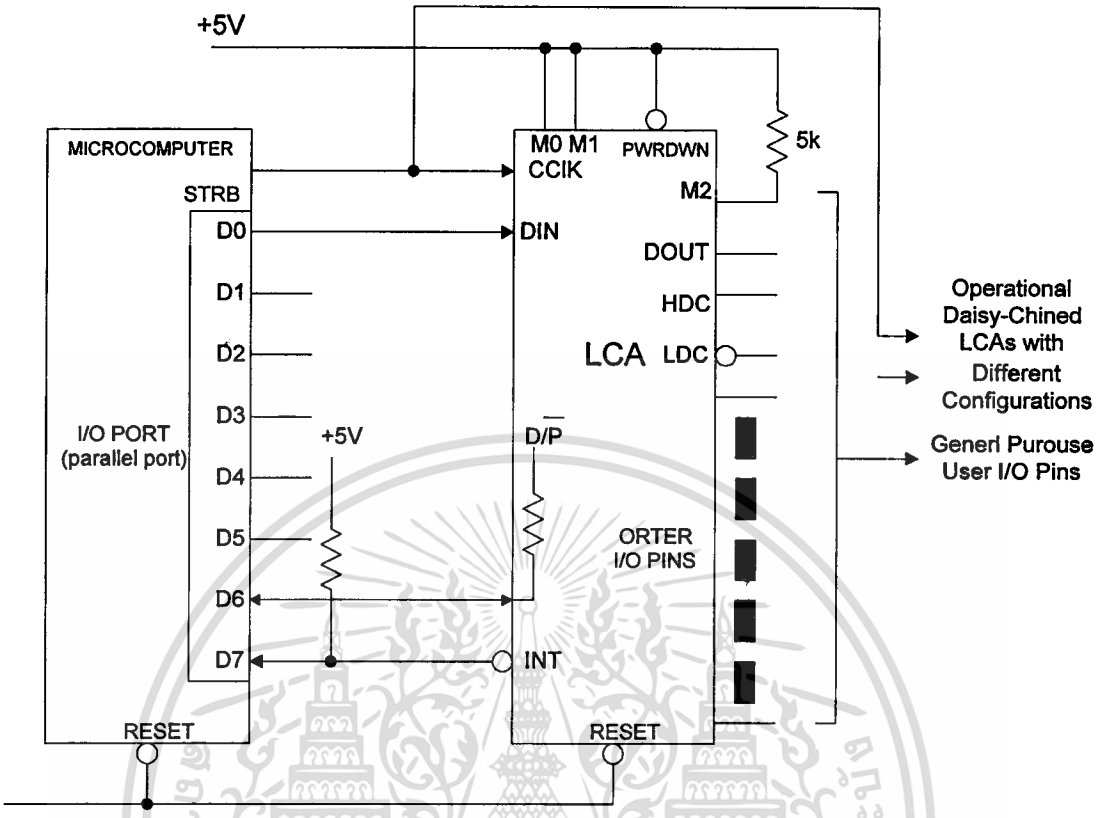
การใช้งาน FPGAs ในการต่อลักษณะสเลฟซีเรียลและมาสเตอร์ซีเรียล เมื่อเริ่มจ่ายไฟเข้าตัว FPGAs จะทำการลบข้อมูลหน่วยความจำที่ใช้ในคอนฟิก (Configuration Memory) ตรวจสอบลักษณะการคอนฟิกว่าเป็นลักษณะใดในตารางที่ 2.3 ว่าเป็นแบบอนุกรมหรือขนาน หลังจากนั้นจะเริ่มทำการโปรแกรมคอนฟิกสัญญาณ DONE/PROGRAM เป็น “0” ซึ่งอยู่ระหว่างโปรแกรมและเมื่อข้อมูลตรงกับที่ส่วนหัวของข้อมูลคอนฟิก สัญญาณ DONE/PROGAM จะเป็น “1” ซึ่งหมายถึงโปรแกรมทำการคอนฟิกเสร็จสิ้น ดูรูปที่ 2.36 ประกอบ



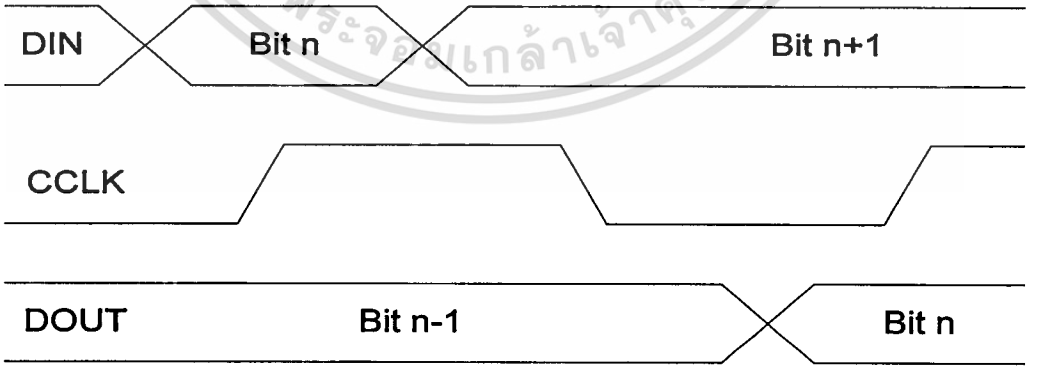
รูปที่ 2.36 ลำดับไดอะแกรมในการคอนฟิกเมื่อเริ่มป้อนแหล่งจ่ายไฟเข้าไอซีและการโปรแกรมใหม่

2.5.1 การใช้งานในลักษณะสเลฟซีเรียล

การต่อใช้งานในลักษณะนี้เหมาะสมกับวงจรที่ออกแบบมาเพื่อทำงานร่วมกับไมโครคอมพิวเตอร์อยู่แล้ว ทั้งนี้เพราะ FPGAs ได้ใช้ความสามารถของไมโครคอมพิวเตอร์ในการเก็บและส่งข้อมูลคอนฟิกให้ เพียงแต่ต้องเขียนโปรแกรมเพื่อส่งโปรแกรมคอนฟิกให้เพิ่มลักษณะการต่อในลักษณะนี้เป็นดังรูปที่ 2.37 ซึ่งไมโครคอมพิวเตอร์จะสร้างสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGAs การป้อนโปรแกรมคอนฟิกให้ FPGAs ทำได้โดยต่อสัญญาณ Strobe เข้ากับขา CCLK และพอร์ต D0 เข้ากับขา DIN สร้างสัญญาณคล็อกป้อนที่ขา CCLK และป้อนโปรแกรมคอนฟิกแบบอนุกรมเข้าที่ขา DIN ดังแผนภูมิในรูปที่ 2.38



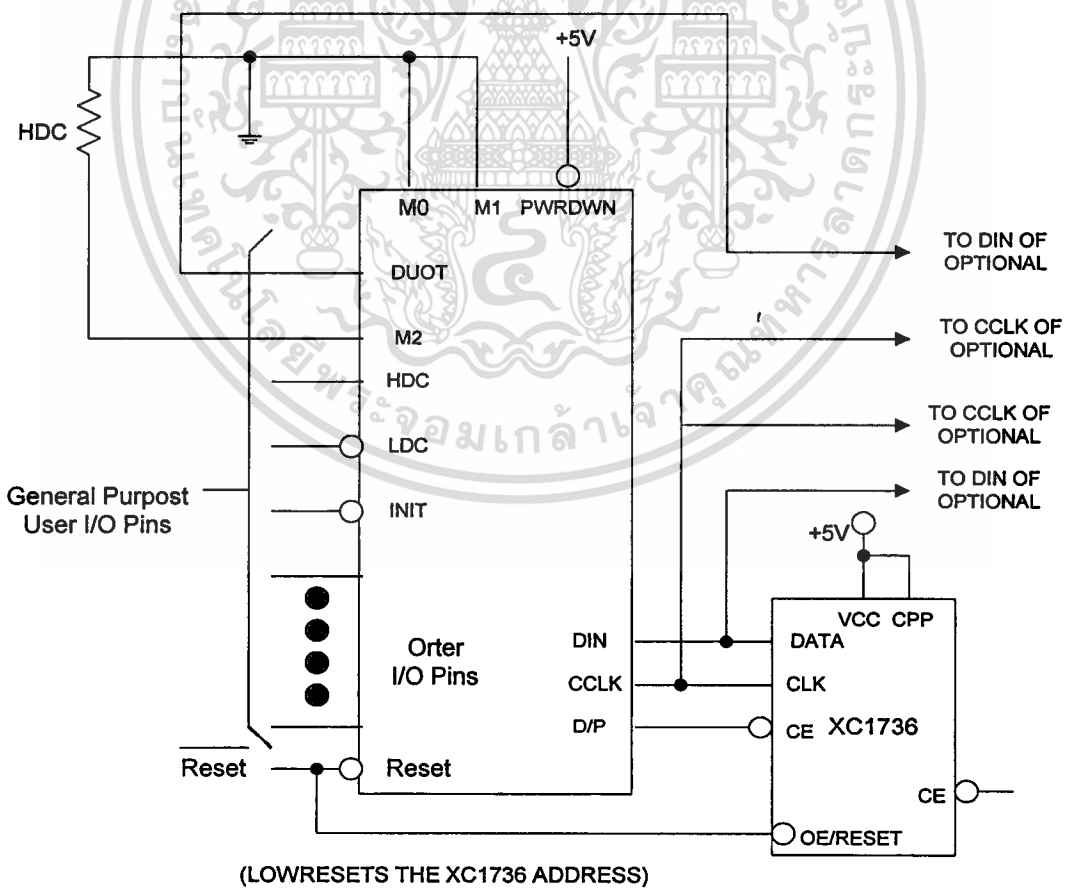
รูปที่ 2.37 การต่อใช้งานในลักษณะสเลฟซีเรียล



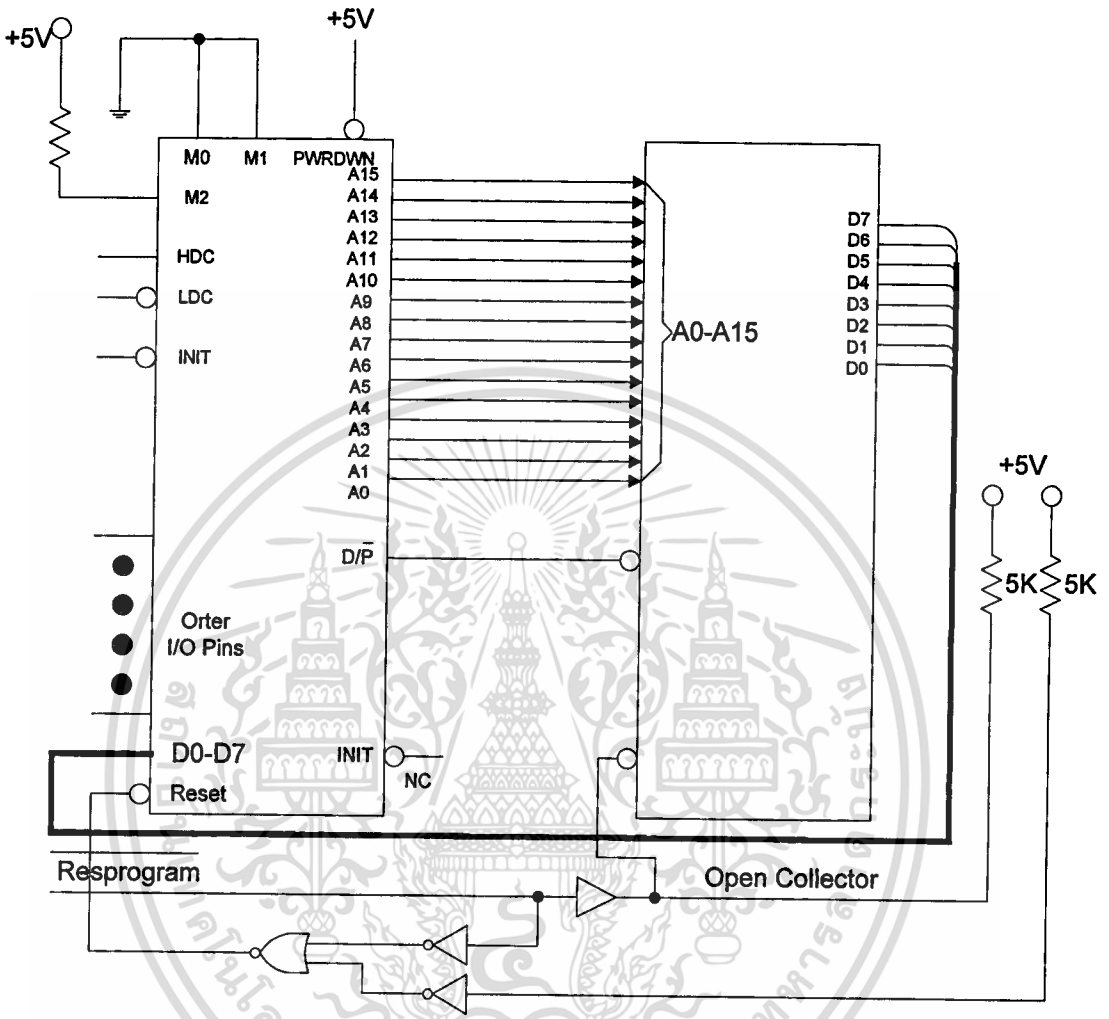
รูปที่ 2.38 แผนภูมิเวลาการป้อนข้อมูลโปรแกรมคอนฟิกในลักษณะสเลฟซีเรียล

2.5.2 การใช้งานลักษณะมาสเตอร์ซีเรียล

การต่อใช้งานในลักษณะนี้ส่วนที่เก็บโปรแกรมคอนฟิกจะต่างจากการต่อลักษณะแรก คือใช้ PROM เบอร์ XC71XXX เป็นตัวเก็บโปรแกรม ทำให้ไม่ต้องเสียเวลาเขียนโปรแกรมเพื่อทำการคอนฟิก ซึ่งวิธีการอัดโปรแกรมคอนฟิกลง PROM ทำตามขั้นตอนดังนี้คือ เมคบิต (MakeBits) สร้างไฟล์ .BIT จากวงจรที่ออกแบบ และใช้โปรแกรม MakePROM สร้าง Hex ไฟล์แล้วทำการอัดโปรแกรมลง PROM ด้วยอุปกรณ์อัด PROM ที่มาพร้อมกับตัวโปรแกรมของไซลิงค์ PROM XC17XXX จะส่งสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGAs ดังแสดงในรูปที่ 2.43 D0-D7 เป็นขารับข้อมูลที่ใช้ในการคอนฟิกแบบขนาน A0-A15 เป็นแอดเดรสที่ FPGAs สร้างให้กับ EPROM เพื่ออ่านข้อมูลจากหน่วยความจำมาเก็บไว้ในสแตติกแรม (StaticRAM) แอดเดรสทั้ง 16 เส้นไม่จำเป็นต้องต่อให้ครบก็ได้ ขึ้นอยู่กับขนาดหน่วยความจำ EPROM ที่ใช้และสามารถกำหนดให้นับขึ้นหรือลงได้



รูปที่ 2.39 การต่อใช้งานในลักษณะมาสเตอร์ซีเรียล



รูปที่ 2.40 การต่อใช้งานในลักษณะมาสเตอร์พาราแรล

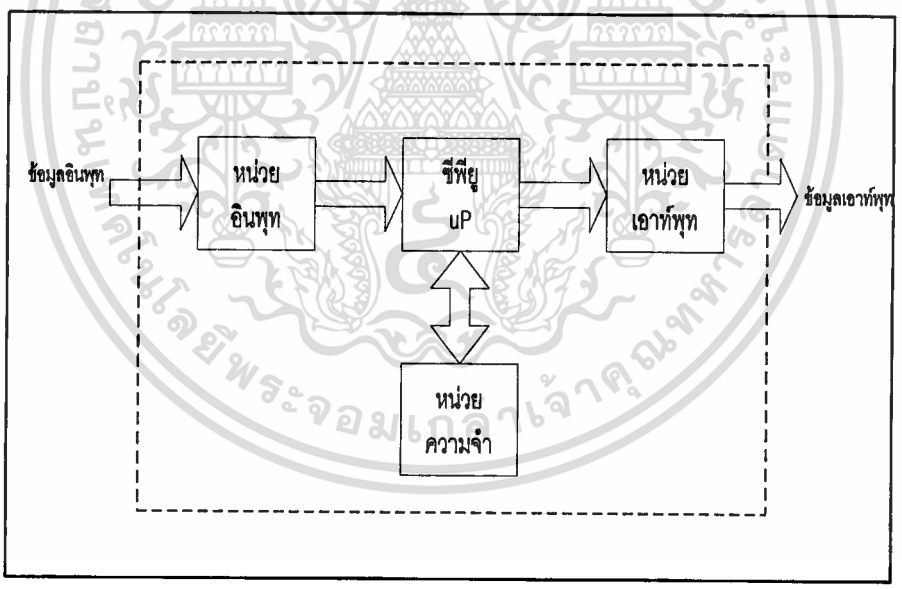
2.6 ข้อควรระวังในการใช้อุปกรณ์ FPGAs

สิ่งที่สำคัญคืออุปกรณ์ FPGAs ไวต่อความร้อนมาก การบัดกรีโดยหัวแร้งกำลังสูงหรือบัดกรีโดยจี้หัวแร้งที่ขาไอซีเป็นเวลานานจะทำให้ไอซีเสียหายได้ ระยะเวลาในการบัดกรีหนึ่งจุดควรไม่เกิน 5-10 วินาที ควรใช้ซ็อกเกต (Socket) ไอซีในการประกอบวงจรลงแผ่นวงจรพิมพ์

การป้องกันไอซีจากแรงดันไฟฟ้าไม่ควรต่อสลับขั้วบวกกับขั้วลบจะทำให้ไอซีเสียได้ นอกจากนั้นแรงดันของแหล่งจ่ายไฟต้องอยู่ในช่วงที่โรงงานกำหนดมาสำหรับ FPGAs ค่าแรงดันที่ใช้งานอยู่ในช่วง VCC = 4.75-5.25 V และแรงดันที่ทนได้อยู่ในช่วง -0.5-7 V ดังนั้นก่อนป้อนแรงดันควรตรวจเช็คให้แน่ใจก่อน

2.7 ทฤษฎีไมโครโปรเซสเซอร์ (Microprocessor)

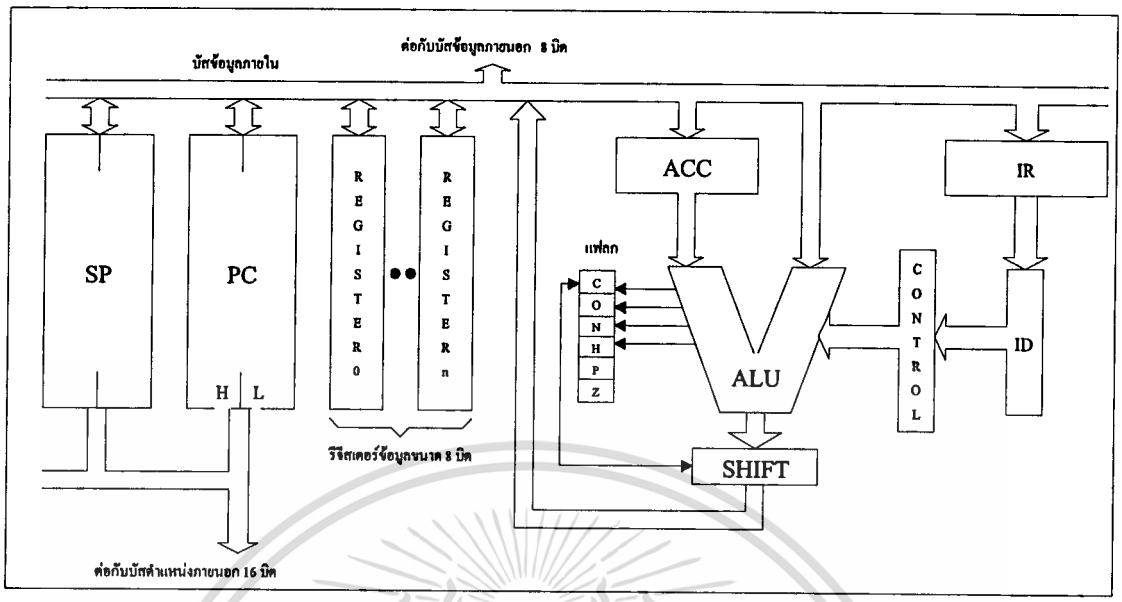
ไมโครโปรเซสเซอร์ คือ อุปกรณ์ที่รวบรวมหน่วยต่างๆของคอมพิวเตอร์อันได้แก่ หน่วยควบคุม (Control Unit) หน่วยความจำบางส่วน หน่วยคำนวณ(ALU) วงจรควบคุม I /O บางส่วน ทั้งหมดนี้รวมไว้ในแผ่นวงจรเดียวกัน ซึ่งสามารถทำหน้าที่ประมวลข้อมูลและควบคุมหน่วยอื่นๆ ให้ทำงานไปด้วยกันได้ เช่น ใช้เป็นตัวประมวลผลกลางในคอมพิวเตอร์ เรียกคอมพิวเตอร์นั้นว่า ไมโครคอมพิวเตอร์ (Microcomputer) ดังมีรูปร่างตามแผนผังนี้



รูปที่ 2.41 แผนผังระบบไมโครคอมพิวเตอร์

2.7.1 โครงสร้างภายในไมโครโปรเซสเซอร์

โครงสร้างของไมโครโปรเซสเซอร์ขนาด 8 บิต โดยทั่วไปในปัจจุบันจะมีสถาปัตยกรรมภายในที่ใช้คล้าย ๆ กัน ดังแสดงในรูปที่ 2.42



รูปที่ 2.42 สถาปัตยกรรมภายในของไมโครโปรเซสเซอร์ 8 บิต

ส่วนประกอบภายในที่สำคัญของไมโครโปรเซสเซอร์ประกอบด้วย วงจรคณิตศาสตร์และลอจิก (Arithemetic Logic Unit หรือ ALU) รีจิสเตอร์คำสั่ง (Instruction Register หรือ IR) วงจรถอดรหัสคำสั่ง (Instruction Decoder หรือ ID) หน่วยควบคุม (Control Unit) และรีจิสเตอร์ตำแหน่ง (Address Register) ส่วนประกอบที่สำคัญภายในเหล่านี้จะติดต่อกันด้วยบัสภายใน (Internal Bus) และบัสภายในนี้ต่อกับบัสภายนอกโดยผ่านบัฟเฟอร์ เพื่อเป็นการติดต่อกับวงจรภายนอกอีกทีหนึ่ง ซึ่งหน้าที่ของหน่วยต่าง ๆ อธิบายได้ดังนี้

หน่วยควบคุม (Control Unit)

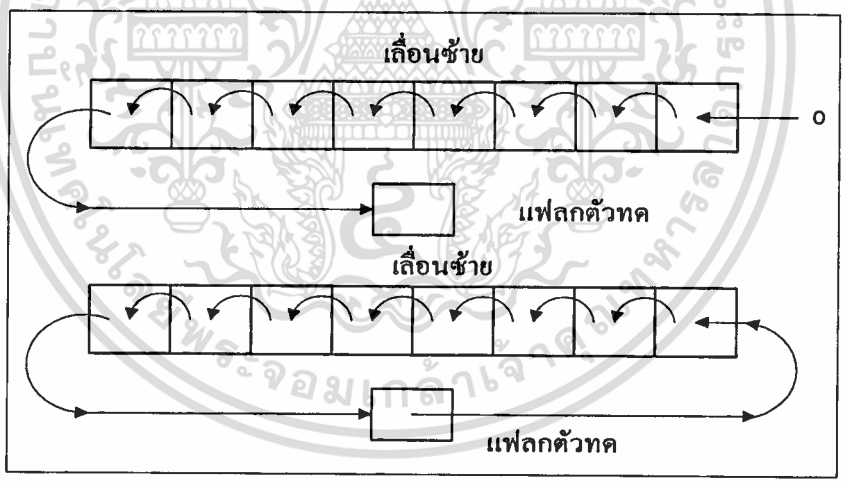
หน่วยควบคุม เป็นหน่วยที่ใช้สร้างสัญญาณเพื่อควบคุมการทำงานภายในของไมโครโปรเซสเซอร์ ให้ทำงานอย่างมีระเบียบและสัมพันธ์กันโดยที่หน่วยควบคุมนี้จะรับสัญญาณมาจากวงจรถอดรหัสคำสั่งซึ่งจะถอดรหัสคำสั่งจากรีจิสเตอร์คำสั่ง ดังนั้นการทำงานภายในไมโครโปรเซสเซอร์จึงขึ้นอยู่กับคำสั่งที่ป้อนเข้ามาให้กับไมโครโปรเซสเซอร์

หน่วยคณิตศาสตร์และลอจิก (Aritric Logic Unit)

หน่วยคณิตศาสตร์และลอจิก หรือเรียกย่อ ๆ ว่า ALU หน่วยนี้มีหน้าที่กระทำทางคณิตศาสตร์และลอจิก หน่วย ALU นี้ จะใช้รีจิสเตอร์พิเศษตัวหนึ่งเป็นอินพุท ซึ่งเรียกว่า แอคคิวมูเลเตอร์ แอคคิวมูเลเตอร์จะเป็นได้ทั้ง อินพุทและเอาต์พุตของ ALU นอกจากนี้ยังสามารถใช้ในการเลื่อน (Shift) และหมุน (Rotate) ข้อมูลได้อีกด้วย

การเลื่อนข้อมูลก็คือการเคลื่อนย้ายข้อมูลใน 1 ไบต์ ไปทางซ้ายหรือทางขวา 1 ตำแหน่งหรือมากกว่า ซึ่งตัวอย่างการเลื่อนและการหมุนข้อมูลแสดงได้ดังรูปที่ 2.43 ซึ่งในรูปเป็นการเลื่อนข้อมูลไปทางซ้าย 1 บิต การเลื่อนข้อมูลนี้อาจเลื่อนผ่านแฟลกตัวทด (Carry Flag) หรือไม่ก็ได้ขึ้นอยู่กับคำสั่งซึ่งรายละเอียด ของการเลื่อนและการหมุนข้อมูลนี้จะได้กล่าวในรายละเอียดในหนังสือ วงจรเลื่อนข้อมูล (Shifter) อาจต่อไว้กับเอาท์พุทของ ALU ดังที่แสดงไว้ดังรูปที่ 2.43

จากรูปที่ 2.42 บล็อกทางด้านซ้ายของ ALU คือ แฟลกหรือรีจิสเตอร์แสดงสถานะ (Status Register หรือ Condition Code Register) แฟลกนี้ใช้เก็บเงื่อนไขพิเศษต่าง ๆ ที่เกิดขึ้นภายในไมโครโปรเซสเซอร์ ข้อมูลในแฟลกสามารถตรวจสอบได้โดยคำสั่งบางคำสั่ง คำสั่งประเภทที่มีการตรวจสอบสถานะที่แฟลก ใช้ในการเขียนโปรแกรมที่มีการทำงานข้าม (Jump หรือ Call) ไปยังส่วนอื่นของโปรแกรม ซึ่งรายละเอียดของคำสั่งประเภทนี้ จะได้กล่าวถึงในเรื่องชุดคำสั่ง



รูปที่ 2.43 การเลื่อนและการหมุนข้อมูล

รีจิสเตอร์ (Register)

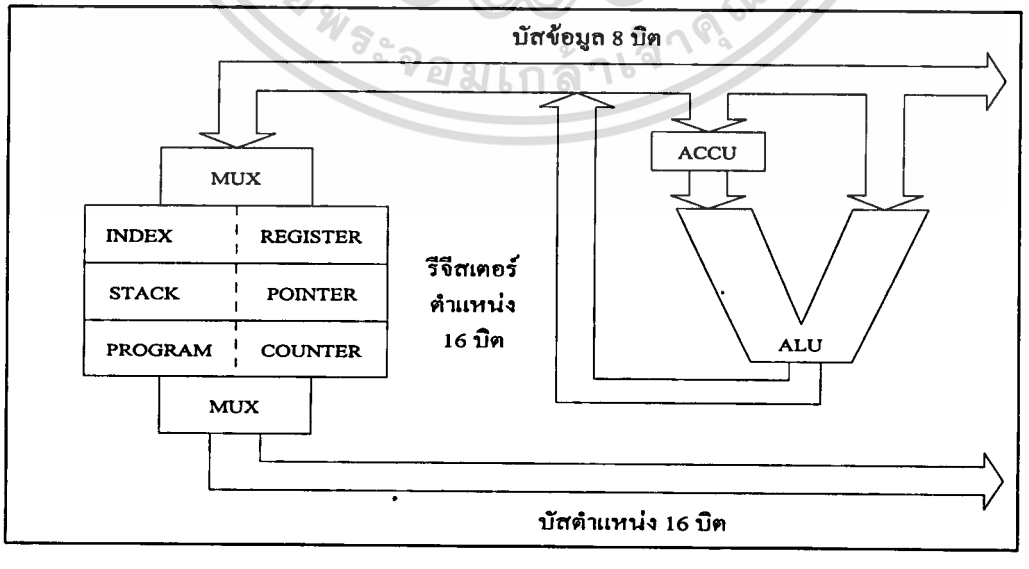
จากสถาปัตยกรรมของไมโครโปรเซสเซอร์ดังรูปที่ 2.42 เราอาจแบ่งริจิสเตอร์ที่มีอยู่ภายในไมโครโปรเซสเซอร์ออกได้เป็น 2 กลุ่ม คือรีจิสเตอร์เพื่อใช้งานทั่วไป (General purpose register) และรีจิสเตอร์สำหรับการอ้างตำแหน่ง (Address register) ซึ่งรีจิสเตอร์ทั้ง 2 อย่าง เราสามารถอธิบายการทำงานได้ดังนี้

รีจิสเตอร์ใช้งานทั่วไป

ในไมโครโปรเซสเซอร์ขนาด 8 บิตรีจิสเตอร์นี้จะมีขนาด 8 บิต หน้าที่ของรีจิสเตอร์เหล่านี้ไม่ได้ถูกกำหนดเฉพาะเจาะจงลงไป แต่โดยทั่วไป จะใช้สำหรับเก็บข้อมูลเพื่อให้ ALU กระทำกับข้อมูลต่างๆ เหล่านั้นด้วยความเร็วสูง เนื่องจากไม่ต้องติดต่อกับหน่วยความจำที่อยู่ภายนอก นอกจากนี้ไมโครโปรเซสเซอร์บางตัว ยังสามารถนำรีจิสเตอร์ 2 ตัวมาต่อร่วมกันได้ เรียกว่า คู่รีจิสเตอร์ (Register pair) โดยที่คู่รีจิสเตอร์นี้จะกลายเป็นรีจิสเตอร์ขนาด 16 บิต หรือใช้เป็นตัวชี้ตำแหน่งข้อมูลในหน่วยความจำก็ได้ ซึ่งจะขึ้นอยู่กับคำสั่งที่ใช้

รีจิสเตอร์สำหรับการอ้างตำแหน่ง

หน้าที่ของรีจิสเตอร์นี้ใช้สำหรับเก็บตำแหน่งของหน่วยความจำที่ต้องการอ้างถึง ซึ่งอาจเป็นการอ้างถึงโดยคำสั่ง หรือการอ้างถึงโดยระบบก็ตาม ขนาดของรีจิสเตอร์นี้อาจเป็น 8 บิต หรือ 16 บิต ก็ได้ ขึ้นอยู่กับชนิดของไมโครโปรเซสเซอร์นั้น ๆ บางครั้งอาจเรียกรีจิสเตอร์เหล่านี้ว่า Data Counter หรือ Pointer ในไมโครโปรเซสเซอร์ทั่วๆ ไป ควรจะมีรีจิสเตอร์สำหรับการอ้างตำแหน่งอย่างน้อย 2 ตัว คือ โปรแกรมเคาท์เตอร์ (PC) และ แสตคพอยเตอร์ (SP) ส่วนรีจิสเตอร์ตัวอื่น ๆ เช่น อินเด็กรีจิสเตอร์ (IX) อาจจะมีหรือไม่ก็ได้ รีจิสเตอร์นี้จะต่อไว้กับบัสตำแหน่งดังแสดงไว้ในรูป เอาท์พุตของรีจิสเตอร์เหล่านี้จะต่อไว้กับบัสตำแหน่ง โดยมีตัวเลือกข้อมูล (Muliplexer) เพื่อทำหน้าที่เลือกว่าจะนำข้อมูลมาจากรีจิสเตอร์ใดเพื่อไปกำหนดตำแหน่งที่ต้องการ



รูปที่ 2.44 รีจิสเตอร์ตำแหน่งขนาด 16 บิตที่ใช้สร้างข้อมูลบนบัสตำแหน่ง

2.7.2 หน้าที่เฉพาะของรีจิสเตอร์สำหรับการอ้างตำแหน่งต่าง ๆ

โปรแกรมเคาท์เตอร์ (Program Counter หรือ PC) โปรแกรมเคาท์เตอร์จะต้องมีอยู่ในทุกโปรเซสเซอร์ข้อมูลในโปรแกรมเคาท์เตอร์คือตำแหน่งของคำสั่งต่อไปที่โปรเซสเซอร์จะต้องอ่านเพื่อปฏิบัติ กลไกการปฏิบัติตามรหัสคำสั่ง และลำดับขั้นของการทำงานตามคำสั่งที่วางไว้จะกำหนดโดยข้อมูลที่อยู่ในโปรแกรมเคาท์เตอร์นี้เอง กล่าวโดยสรุปคือ การปฏิบัติโปรแกรมจะเป็นไปแบบเรียงลำดับ และเพื่อที่จะดึงคำสั่งต่อไป โปรเซสเซอร์จำเป็นต้องดึงคำสั่งมาจากหน่วยความจำ ซึ่งกระบวนการนี้ข้อมูลในโปรแกรมเคาท์เตอร์จะส่งมายังบนบัสตำแหน่ง และส่งต่อไปยังหย่วยความจำ หน่วยความจำจะอ่านข้อมูลจะอ่านจากตำแหน่งที่ถูกอ้างถึงและส่งข้อมูลที่อ่านได้ไปบนบัสข้อมูล โปรเซสเซอร์จะอ่านข้อมูลบนบัสข้อมูลนั้น ซึ่งข้อมูลหรือคำที่อ่านได้คือคำสั่ง Operation Code นั้นเอง

สแตคพอยท์เตอร์ (Stack Pointer หรือ SP) ไมโครโปรเซสเซอร์ที่มีความสามารถสูงโดยทั่วไปจะต้องมีสแตค ซึ่งสแตคอาจเป็นรีจิสเตอร์ที่อยู่ในโปรเซสเซอร์เองหรือใช้หน่วยความจำภายนอกส่วนหนึ่ง เพื่อกำหนดให้เป็นสแตค ในการเก็บรักษาตำแหน่งตำแหน่งสูงสุดของสแตคที่อยู่ในหน่วยความจำ จะใช้รีจิสเตอร์ที่เรียกว่า สแตคพอยท์เตอร์ (SP) สแตคและสแตคพอยท์เตอร์เป็นสิ่งที่จำเป็น และขาดไม่ได้ในการทำโปรแกรมย่อย (Subroutine) และการทำโปรแกรมเกี่ยวกับการอินเตอร์รัพต์

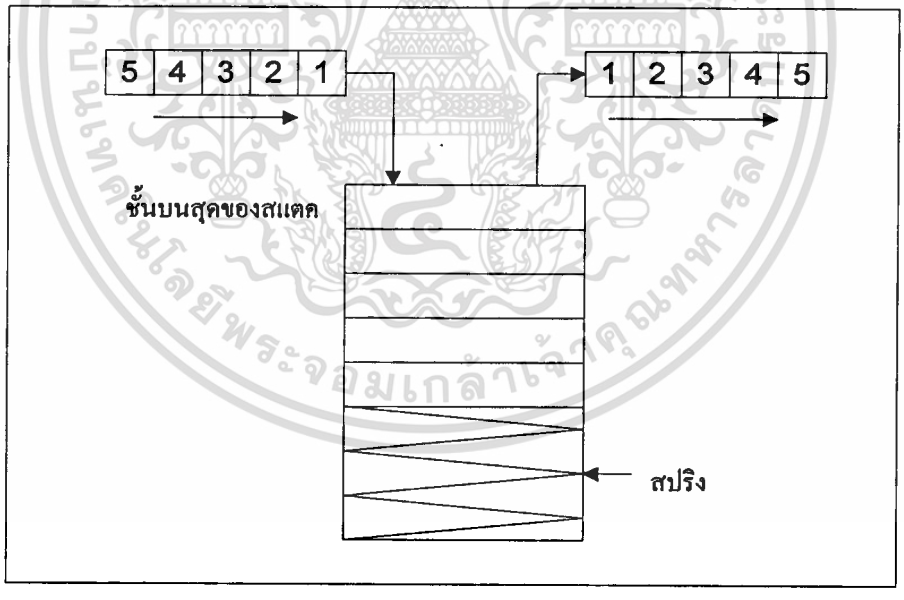
อินเด็กรีจิสเตอร์ (Index Register หรือ IX) อินเด็กรีจิสเตอร์ เป็นรีจิสเตอร์สำหรับการอ้างตำแหน่งอีกตัวหนึ่ง ที่มีในไมโครโปรเซสเซอร์บางชนิดเท่านั้น และใช้กับคำสั่งที่มีการเข้าถึงข้อมูลแบบอินเด็กซ์ (Index Addressing mode) ซึ่งอินเด็กรีจิสเตอร์นี้จะทำให้การเข้าถึงหน่วยความจำเป็นกลุ่มได้โดยสะดวก ข้อมูลในอินเด็กรีจิสเตอร์อาจเป็นค่าระยะห่าง (Displacement) ซึ่งจะนำไปบวกกับค่าตำแหน่งฐาน (Base address) เพื่อใช้ชี้ตำแหน่งของหน่วยความจำที่ต้องการอ้างอิงหรือข้อมูลในอินเด็กรีจิสเตอร์อาจเป็นค่าตำแหน่งฐาน เพื่อนำไปบวกกับค่าระยะห่างที่กำหนดมากับคำสั่งก็ได้ ทั้งนี้แล้วแต่ไมโครโปรเซสเซอร์ตัวนั้น ๆ

สแตค (Strack) สแตค คือกลุ่มของรีจิสเตอร์ หรือส่วนหนึ่งของหน่วยความจำที่ถูกจัดเตรียมไว้ ซึ่งสแตคสามารถจัดแบ่งสแตคออกได้เป็น 2 อย่าง คือ

1. ฮาร์ดแวร์สแตค (Hardware Stack) สแตคแบบนี้เป็นรีจิสเตอร์ที่กำหนดไว้คงที่ภายในตัวของโปรเซสเซอร์เองซึ่งมีข้อดีในเรื่องของความเร็วในการทำงาน แต่มีข้อเสียคือจำนวนของรีจิสเตอร์ที่ใช้เป็นสแตคจำกัด

2. ซอฟต์แวร์สแตค (Software stack) ไมโครโปรเซสเซอร์ที่ใช้งานทั่วๆไป จะใช้ซอฟต์แวร์สแตค เพราะจำนวนของสแตคไม่ได้ถูกจำกัดด้วยจำนวนของรีจิสเตอร์ที่อยู่ภายในแต่จะใช้ส่วนหนึ่งของหน่วยความจำ และใช้คำสั่งในการกำหนดตำแหน่งเริ่มต้นของสแตค ตำแหน่งบนสุดของสแตค จะชี้โดยข้อมูลที่มีอยู่ในสแตคพอยท์เตอร์ ไมโครโปรเซสเซอร์ z-80 ใช้ลักษณะของสแตคแบบนี้

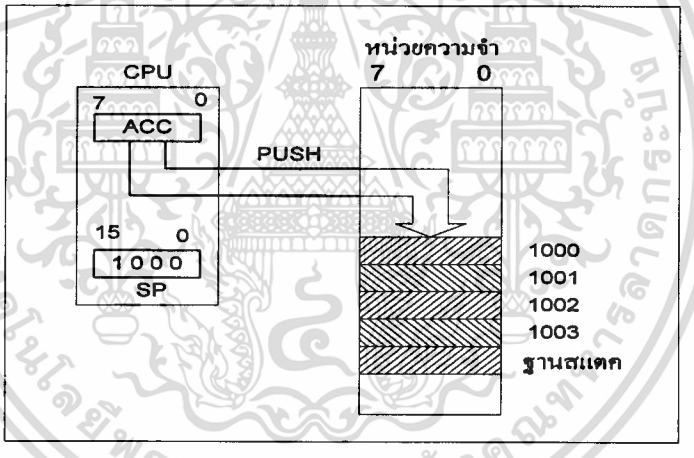
ลักษณะโครงสร้างของสแตคทั้งสองจะเป็นแบบเรียงลำดับ(chronological structure) คือข้อมูลที่ใส่ลงไปในสแตคข้อมูลแรกจะอยู่ที่ชั้นใต้สุดของสแตค และข้อมูลที่ใส่เข้ามาทีหลังจะวางทับด้านบน ต่อไปเรื่อยๆ ส่วนในการนำข้อมูลออก ข้อมูลที่อยู่บนสุด หรือข้อมูลที่ใส่เข้ามาทีหลังที่สุดจะถูกนำออกมาก่อน ดังนั้นลักษณะโครงสร้างลักษณะนี้เรียกได้ว่าเป็นโครงสร้างแบบ LIFO (Last-in First-out) ลองนึกถึงโครงสร้างของแหนบลูกปืน ซึ่งมีสปริงอยู่ด้านล่าง ลูกปืนลูกแรกที่ใส่เข้าไปจะอยู่ล่างที่สุด และลูกปืนที่ใส่เข้าไปทีหลังก็จะซ้อนอยู่ข้างบนเรื่อยๆ เห็นได้ว่าลักษณะโครงสร้างแบบนี้ ลูกปืนที่อยู่บนสุดจะถูกใช้ก่อนเสมอ



รูปที่ 2.45 โครงสร้างของสแตค

สแตคมีประโยชน์ในการทำโปรแกรม 3 ประเภทคือ โปรแกรมย่อย(Subroutine) โปรแกรมบริการการขัดจังหวะ (Interrupt roution)และการใช้เป็นที่เก็บข้อมูลชั่วคราว(Temporary data register) ของการทำโปรแกรมต่างๆ

คำสั่งโดยทั่วไปสำหรับไมโครโปรเซสเซอร์ที่ใช้สำหรับการนำข้อมูลการนำเข้าและออกจากสแตคโดยตรง มีอยู่ 2 คำสั่ง คือPUSH และPOP (หรือPULL) คำสั่ง PUSH มีผลทำให้ใส่ข้อมูลลงไปในสแตค ส่วนคำสั่ง POP ทำการดึงข้อมูลออกจากสแตค และตำแหน่งบนสุดของสแตคจะแสดงที่สแตคพอยท์เตอร์รีจิสเตอร์เสมอ ซึ่งโดยทั่วไป ตำแหน่งบนสุดของสแตค หมายถึง ตำแหน่งหน่วยความจำที่มีค่าน้อยที่สุดของหน่วยความจำที่เป็นสแตค รูปที่ 2.46 แสดงถึงการทำงานของสแตคและสแตคพอท์ยเตอร์ ถ้าในขณะเริ่มแรกข้อมูลใน สแคต-พอท์ยเตอร์ที่ชี้ตำแหน่ง 1002 เมื่อโปรเซสเซอร์ทำคำสั่ง PUSH AF เพื่อทำการนำข้อมูลในแอคคิวมูเลเตอร์ และ แฟลก เข้าไปเก็บไว้ในสแตค หลังจากทำคำสั่ง PUSH AF นี้แล้ว ข้อมูลในสแตคพอย์ทเตอร์จะเปลี่ยนไปเป็น 1000 โดยอัตโนมัติ นั่นคือ สแตคพอย์ทเตอร์จะชี้ที่ตำแหน่งบนสุดของสเตคที่มีข้อมูลอยู่นั่นเอง



รูปที่ 2.46 การใส่ข้อมูลลงในสแตคด้วยคำสั่ง PUSH

บทที่ 3
การออกแบบและการสร้าง

การออกแบบและการสร้างวงจรของหน่วยประมวลผลกลางขนาด 8 บิต มีรายละเอียดดังนี้

3.1 ลักษณะการออกแบบ

3.1.1 การออกแบบจากล่างขึ้นบน (Bottom Up Design)

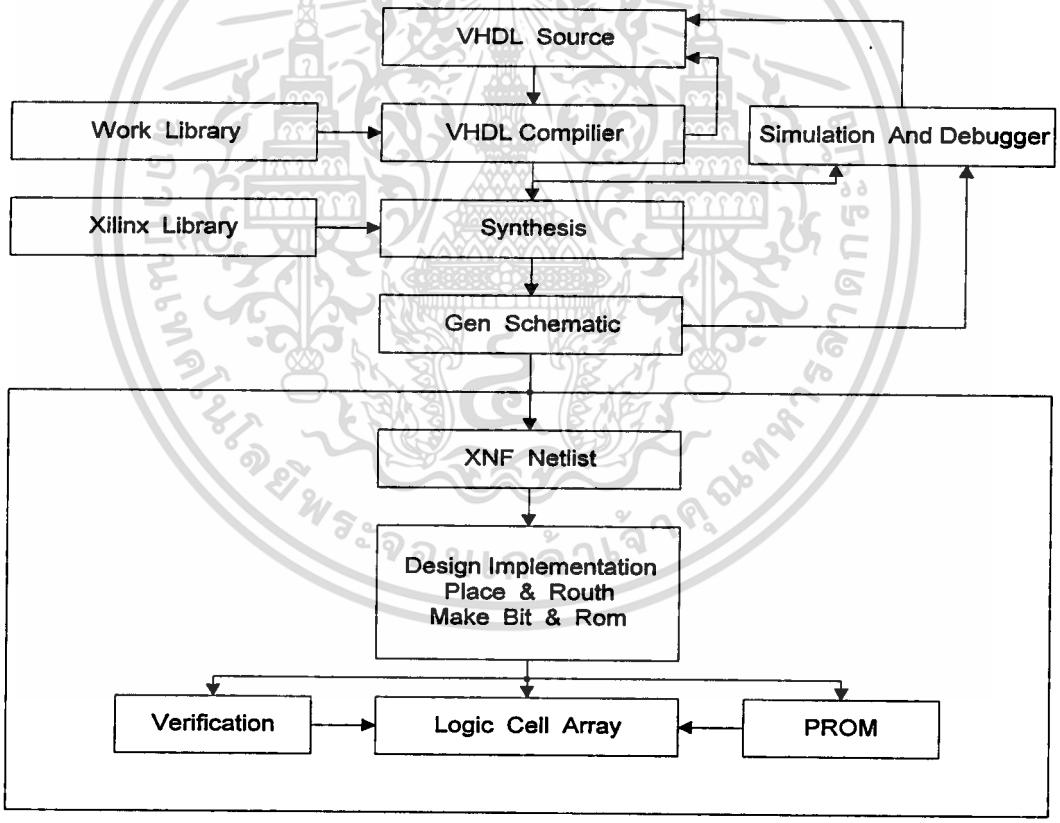
ในอดีตถึงปัจจุบันการออกแบบในระบบดิจิตอลจะเป็นลักษณะที่เรียกว่า "การออกแบบจากล่างขึ้นบน" (Bottom Up Design) คือผู้ออกแบบจะเริ่มต้นกำหนดหัวข้องาน แล้วใช้หลักการทางทฤษฎีแบ่งออกเป็นฟังก์ชั่นการทำงานต่าง ๆ แล้วเริ่มต้นการออกแบบ เมื่อได้วงจรตามที่ต้องการแล้วจะต้องหาอุปกรณ์มาตราฐานต่าง ๆ เช่น IC 74LSXX เป็นต้น เพื่อนำมารองรับฟังก์ชั่นการทำงานต่าง ๆ ที่ได้จากการออกแบบ ถ้าไม่สามารถหาอุปกรณ์มารองรับได้จะต้องออกแบบหรือดัดแปลงวงจรใหม่ ในขั้นตอนสุดท้ายคือการจำลองการทำงานโดยทดลองจากวงจรต้นแบบ

3.1.2 การออกแบบจากบนลงล่าง (Top Down Design)

จากการออกแบบในหัวข้อที่ 3.1.1 นั้นในกรณีที่การทำงานของวงจรไม่ตรงตามข้อกำหนดต้องทำการออกแบบและประกอบวงจรใหม่อีกครั้ง ซึ่งจะเห็นว่าขั้นตอนและกระบวนการดังกล่าวยุ่งยากและใช้เวลามาก ดังนั้นในการออกแบบสมัยใหม่สามารถออกแบบระบบดิจิตอลได้จากแนวคิดโดยสังเขป โดยวิธีการเขียนรูปแบบแล้วทดลองการทำงานของรูปแบบนั้นจนเป็นที่น่าพอใจแล้วค่อย ๆ เพิ่มรายละเอียดของระบบไปทีละขั้น ซึ่งในแต่ละขั้นตอนสามารถที่จะจำลองการทำงานภายใต้สภาวะแวดล้อมเดิมได้ ทำให้ไม่มีโอกาสที่รูปแบบการทำงานจะผิดไปจากวัตถุประสงค์เดิม เมื่อเกิดข้อผิดพลาดก็สามารถแก้ไขได้ทันที หลังจากนั้นจึงนำไปลงอุปกรณ์และทดสอบการทำงานของวงจรซ้ำอีกครั้ง การออกแบบในลักษณะนี้เรียกว่า "การออกแบบจากบนลงล่าง" (Top Down Design)

3.2 วิธีการออกแบบ

จากหัวข้อที่ 3.1 จะเห็นว่าการออกแบบจากบนลงล่างนั้น มีความสะดวกและรวดเร็วกว่าการออกแบบจากล่างขึ้นบนมาก เพราะสามารถจำลองการทำงานของวงจรที่ออกแบบจนแน่ใจว่าทำงานได้ตามวัตถุประสงค์ที่ต้องการแล้ว จึงนำไปสร้างวงจรจริงในภายหลัง โครงงานนี้จึงได้เลือกวิธีการออกแบบจากบนลงล่างในการสร้างวงจรหน่วยประมวลผลกลาง โดยทำการออกแบบวงจรหน่วยประมวลผลกลางขนาด 8 บิตด้วยภาษา VHDL ซึ่งเป็นซอร์ฟแวร์ของบริษัทวิวลอจิก และสร้างวงจรหน่วยประมวลผลกลางที่ได้ออกแบบไว้ด้วยอุปกรณ์ FPGAs ของบริษัทไซลิงค์ ขั้นตอนการออกแบบโดยใช้ซอฟแวร์และอุปกรณ์ดังกล่าวได้ดังรูปที่ 3.1



รูปที่ 3.1 ผังการทำงานของการออกแบบวงจรโดยใช้ซอฟแวร์ของบริษัทวิวลอจิกและอุปกรณ์ FPGAs ของบริษัทไซลิงค์

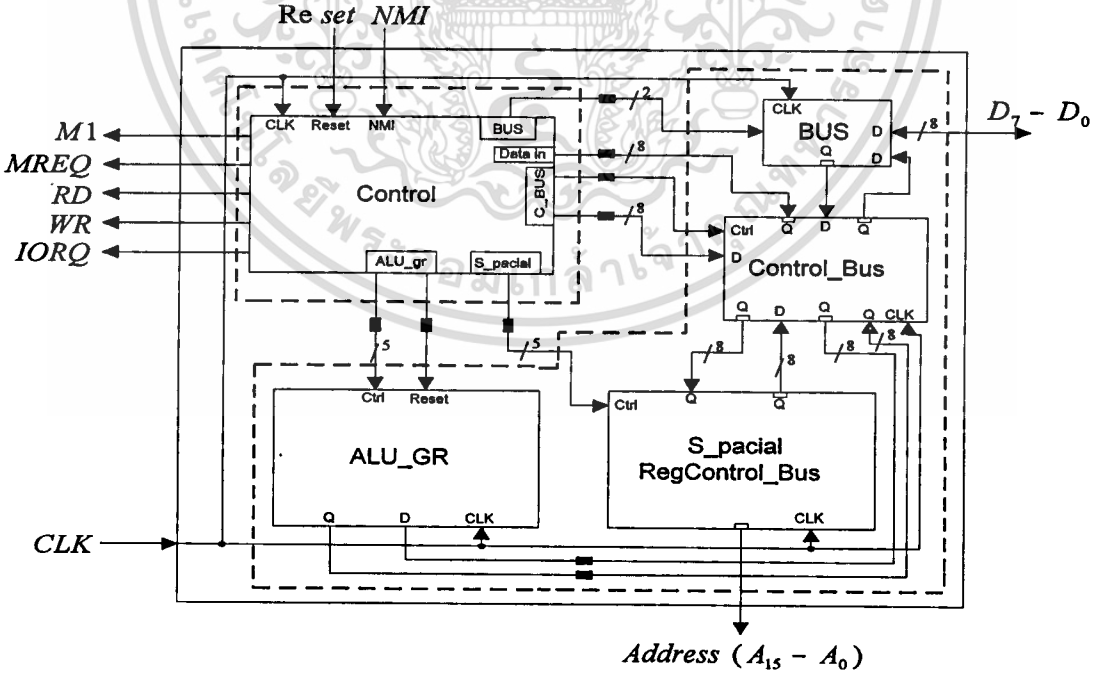
3.3 คุณสมบัติของวงจรหน่วยประมวลผลกลางขนาด 8 บิต

วงจรหน่วยประมวลผลกลางขนาด 8 บิตที่ทำการออกแบบได้กำหนดคุณสมบัติไว้ดังนี้คือ

- กระทำคำสั่งได้ไม่เกิน 25 คำสั่ง
- ใช้สัญญาณนาฬิกา 2 MHz เป็นฐานเวลาในการทำงาน
- สามารถต่อหน่วยความจำภายนอกได้ 65,536 ตำแหน่ง(64 K)
- สามารถรับข้อมูลได้ครั้งละ 8 บิต
- มีคำสั่งในการทำงานทางลอจิก
- สามารถใช้หน่วยความจำภายนอกเพื่อใช้เป็นสแตค

3.4 ผังการทำงานของวงจรหน่วยประมวลผลกลางขนาด 8 บิต

ผังการทำงานของวงจรหน่วยประมวลผลกลางขนาด 8 บิต แสดงได้ดังรูปที่ 3.2 ซึ่งอธิบายการทำงานของวงจรหน่วยประมวลผลกลางขนาด 8 บิต ในแต่ละส่วนได้ดังนี้คือ



รูปที่ 3.2 ผังการทำงานของวงจรหน่วยประมวลผลกลางขนาด 8 บิต

ภาค ALU_GR ทำหน้าที่ประมวลผลทางคณิตศาสตร์และลอจิก โดยคำสั่งทางคณิตศาสตร์จะมีคำสั่ง ADD,SUB,INC,DEC ส่วนคำสั่งทางลอจิกจะมีคำสั่ง OR,AND,XOR ซึ่งจะกระทำกับข้อมูลขนาด 8 บิต และยังทำหน้าที่เก็บข้อมูลเพื่อให้ ALU กระทำคำสั่งกับข้อมูลต่างๆ ซึ่งมีขนาด 8 บิต ภายในมีรีจิสเตอร์ขนาด 8 บิตด้วยกัน 4 ตัว คือ แอคคิวมูเลเตอร์, แฟกซ์, รีจิสเตอร์ H และรีจิสเตอร์ L

ภาค S_pacial_Reg ทำหน้าที่เป็นรีจิสเตอร์ในการเก็บตำแหน่ง ของหน่วยความจำที่ต้องการอ้างถึงซึ่งอาจจะเป็นการอ้างถึงโดยคำสั่ง หรืออ้างถึงโดยระบบก็ตาม ภายในประกอบด้วยรีจิสเตอร์ขนาด 16 บิต ด้วยกัน 3 ตัว คือ โปรแกรมเคาน์เตอร์(PC) สแตคพอยน์เตอร์(SP) และอินเด็กรีจิสเตอร์(IX)นอกจากนี้ยังประกอบด้วยวงจรบวก วงจรลบ วงจรเพิ่มค่าและลดค่าขนาด 16 บิตอีกด้วย

ภาค CTRL_BUS ทำหน้าที่ควบคุมการป้อนข้อมูลให้กับหน่วยต่าง ๆ ภายใน CPU เพื่อการจัดสรรข้อมูลที่ถูกต้องให้กับภาคต่าง ๆ

ภาค BUS ทำหน้าที่รับ-ส่ง ข้อมูลให้กับอุปกรณ์ภายนอกที่เชื่อมต่อกับ CPU

ภาค Control ทำหน้าที่ควบคุมการทำงานทุกภาคภายใน CPU เพื่อให้ทำงานอย่างเป็นระบบ

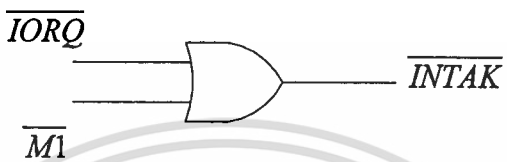
3.5 หน้าที่การทำงานของขาสัญญาณแต่ละขา

ขา CLK เป็นขาอินพุทแอคทีฟขอบขาขึ้น ทำหน้าที่รับสัญญาณฐานเวลาให้แก่วงจรของหน่วยประมวลผลกลางขนาด 8 บิต ซึ่งมีความถี่เท่ากับ 2 MHz

ขา A_{15}-A_0 เป็นสายสัญญาณกำหนดตำแหน่ง(Address Bus) โดยที่ A_0 เป็นบิตทางด้านต่ำ(LSB)ขาเหล่านี้เป็นเอาต์พุตแบบ 3 สถานะ (Tri-State) และจะให้แอคตีฟที่ลอจิก 1 บัสนี้มีด้วยกันทั้งหมด 16 สาย ดังนั้นจึงสามารถติดต่อกับหน่วยความจำได้ถึง 2^{16} = 65536 ตำแหน่ง (64 Kbytes) นอกจากนั้นยังสามารถกำหนดตำแหน่งของพอร์ต อินพุท/เอาต์พุต เมื่อใช้คำสั่งกลุ่มอินพุท/เอาต์พุต

ขา D_7-D_0 เป็นสายสัญญาณข้อมูล (Data Bus) D_0 เป็นบิตทางด้านต่ำลักษณะเป็นบัสแบบสองทิศทาง แบบสามสถานะ ขนาด 8 บิต และแอคตีฟที่ลอจิก 1 ใช้เพื่อเป็นเส้นทางผ่านของข้อมูลระหว่างไมโครโปรเซสเซอร์กับหน่วยความจำ หรืออุปกรณ์อินพุท/เอาต์พุตต่าง ๆ

ขา M1 (Machine Cycle One) เป็นขาเอาต์พุตและแอคตีฟที่ลอจิก 0 เมื่อขานี้แอคตีฟ ชี้ให้เห็นว่าขณะนี้กำลังอยู่ในสภาวะของการเฟทช์คำสั่ง และสัญญาณ M1 นี้ จะใช้ร่วมกับ IORQ เพื่อสร้างสัญญาณตอบรับการอินเตอร์รัพต์ (Interrupt Acknowledge) โดยใช้วงจรลอจิกง่าย ๆ ดังรูป



รูปที่ 3.3 วงจรสร้างสัญญาณตอบรับการอินเตอร์รัพต์

ขา MREQ (Memory Reguest) เป็นเอาต์พุตแบบสามสถานะและแอคตีฟที่ลอจิก 0 เมื่อสายสัญญาณนี้แอคตีฟ บอกให้ทราบว่า ขณะนี้ไมโครโปรเซสเซอร์ต้องการติดต่อกับหน่วยความจำเพื่ออ่านหรือเขียนข้อมูล โดยที่ตำแหน่งของหน่วยความจำจะปรากฏอยู่บนบัสตำแหน่งแล้ว

ขา IORQ (Input/Output Request) เป็นเอาต์พุตแบบสามสถานะและแอคตีฟที่ลอจิก 0 สายสัญญาณนี้แอคตีฟบอกให้ทราบว่า ขณะนี้ทางด้านไบต์ต่ำ (A7-A0) ของบัสตำแหน่งบรรจุตำแหน่งของพอร์ต ที่จะส่งถ่ายข้อมูลระหว่างไมโครโปรเซสเซอร์กับอุปกรณ์อินพุต/เอาต์พุต นอกจากนี้จะใช้ร่วมกับสัญญาณ M1 เพื่อตอบรับการอินเตอร์รัพต์ดังรูปที่ 3.3 และขณะนี้เวกเตอร์ของการอินเตอร์รัพต์จะส่งผ่านเข้ามาในบัสข้อมูลเพื่อกำหนดตำแหน่งของโปรแกรมบริการการอินเตอร์รัพต์

ขา RD (Memory Read) เป็นขาเอาต์พุตแบบสามสถานะและแอคตีฟที่ลอจิก 0 สัญญาณนี้ขณะนี้ไมโครโปรเซสเซอร์ต้องการอ่านข้อมูลขจากหน่วยความจำ หรือจากอุปกรณ์อินพุตเอาต์พุต

ขา WR (Memory Write) เป็นขาเอาต์พุตแบบสามสถานะ และแอคตีฟที่ลอจิก 0 เมื่อสัญญาณนี้แอคตีฟชี้ว่าขณะนี้ไมโครโปรเซสเซอร์ต้องการเขียนข้อมูลเข้าหน่วยความจำหรือเข้าอุปกรณ์อินพุต/เอาต์พุต

ขา NMI (Non Maskable Interrupt) เป็นขาอินพุตและแอคทีฟที่ขอบพัลซ์ขาลง (Negative edge Trigger) ไมโครโปรเซสเซอร์จะทำการตรวจสอบขานี้ที่สเตทสุดท้ายของคำสั่ง เมื่อไมโครโปรเซสเซอร์ได้รับสัญญาณที่ขา NMI จะทำให้เริ่มต้นการทำงานใหม่ที่ตำแหน่ง 0066H ส่วนค่าในโปรแกรมเคาน์เตอร์ที่ชี้ตำแหน่งของคำสั่งต่อไปก่อนซีพียูจะถูกอินเตอร์รัพต์จะเก็บไว้ในสแตค (ที่ RAM) เพื่อที่ซีพียูสามารถกลับมาทำงานต่อได้หลังจากที่ทำโปรแกรมบริการการอินเตอร์รัพต์เสร็จสิ้นแล้ว ในขณะที่ซีพียูอยู่ในจังหวะ Wait มันจะไม่รับสัญญาณ NMI นี้

ขา RESET เป็นอินพุต แอคตีฟที่ลอจิก 0 เมื่อไมโครโปรเซสเซอร์ได้รับสัญญาณ RESET จะทำให้ค่าในโปรแกรมเคาน์เตอร์เริ่มต้นที่ศูนย์ และตั้งต้นการทำงานของไมโครโปรเซสเซอร์ใหม่

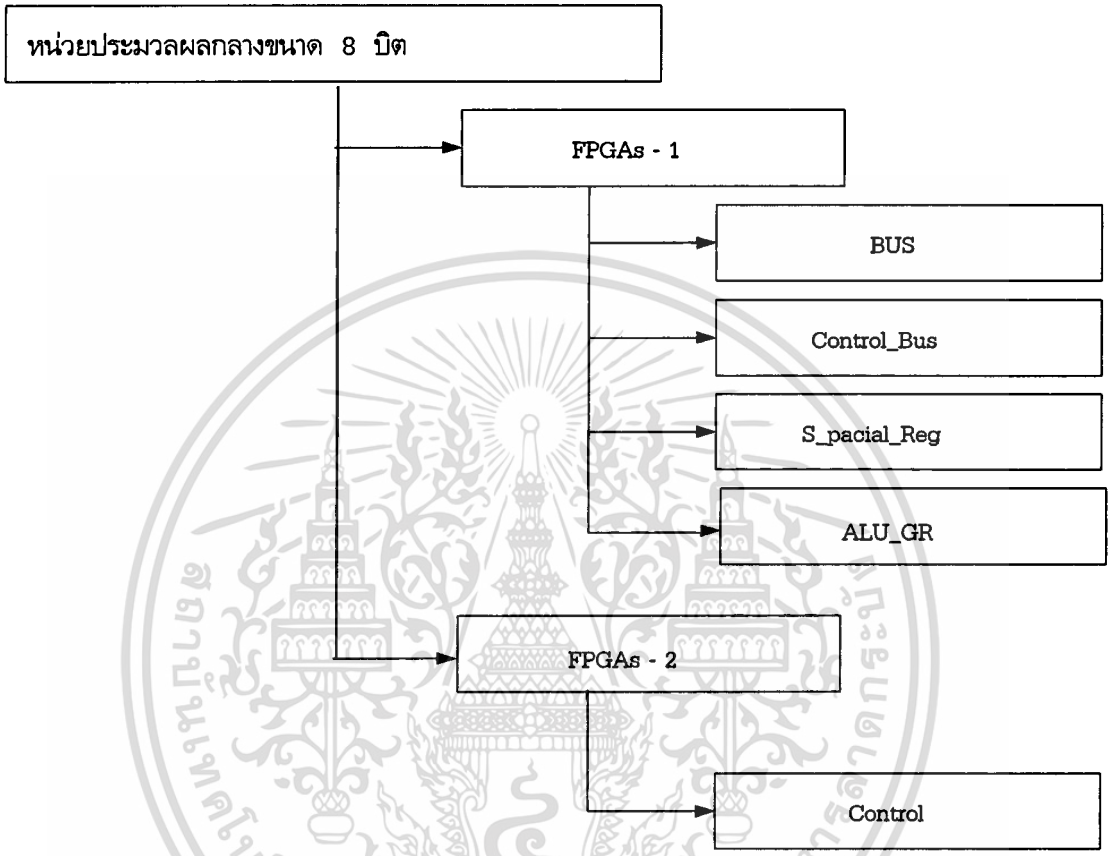
3.6 ขั้นตอนการออกแบบวงจรประมวลผลกลางขนาด 8 บิต

3.6.1 การออกแบบวงจรหน่วยประมวลผลกลาง

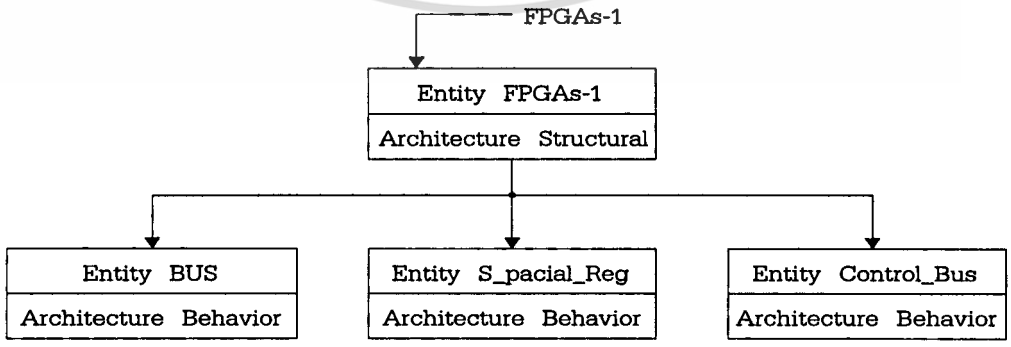
หลังจากที่ได้กำหนดคุณสมบัติของวงจรหน่วยประมวลผลกลางขนาด 8 บิตเสร็จแล้ว โดยทราบว่าวงจรหน่วยประมวลผลกลางขนาด 8 บิต มีขาสัญญาณอินพุตและเอาต์พุตกี่ขา มีการทำงานเป็นอย่างไร การออกแบบที่มีขนาดใหญ่ควรออกแบบเป็นผังการทำงาน ดังแสดงในรูปที่ 3.2 เป็นต้น ซึ่งยึดหลักการออกแบบให้สามารถดาวน์โหลดลงอุปกรณ์ FPGAs ได้ เนื่องจากวงจรที่ได้ทำการออกแบบนั้นมีขนาดใหญ่มากจึงได้แบ่งส่วนต่าง ๆ ออกเพื่อให้สามารถดาวน์โหลดลงอุปกรณ์ FPGAs ได้ซึ่งประกอบด้วย หน่วยกระทำคำสั่งทางคณิตศาสตร์และลอจิกและหน่วยพักข้อมูล (ALU_GR) หน่วยพักข้อมูลพิเศษ (S_pacial_Reg) หน่วยควบคุม(Control)และหน่วยบัส (BUS) หลังจากนั้นทำการแยกส่วนในแต่ละวงจรออกเป็นส่วนย่อยลงมาเรื่อย ๆ เป็นลำดับขึ้นดังรูปที่ 3.4

การออกแบบในลักษณะการแยกส่วนออกเป็นส่วนย่อยนี้ เมื่อเราใช้ภาษา VHDL ในการออกแบบนั้น หน้าที่การทำงานของแต่ละส่วนสามารถอธิบายได้ในลักษณะส่วนของโปรแกรม ซึ่งแสดงการทำงานของวงจรในส่วนย่อยนั้นการแตกวงจรใหญ่ ๆ เป็นส่วนย่อยนี้ทำให้ง่ายต่อการออกแบบและง่ายต่อการทำความเข้าใจ เช่น หน่วยกระทำทางข้อมูล ประกอบไปด้วยหน่วยประมวลผลทางคณิตศาสตร์และลอจิก กลุ่มรีจิสเตอร์ กลุ่มรีจิสเตอร์พิเศษ หน่วยควบคุมบัส จากรูปที่ 3.4 เมื่อแยกส่วนของวงจรหน่วยประมวลผลกลางขนาด 8 บิต ออกเป็น

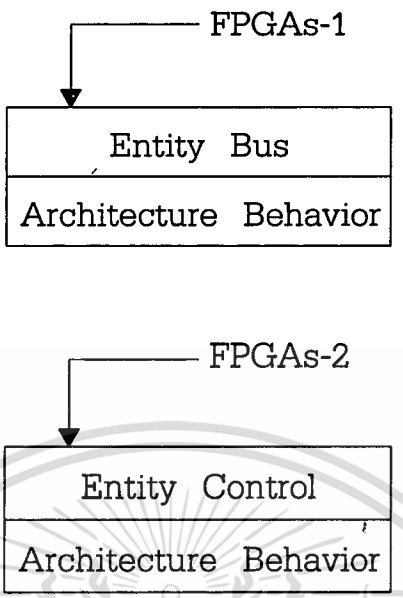
ส่วนย่อยต่าง ๆ แล้วใช้ภาษา VHDL อธิบายการทำงานของวงจรในส่วนย่อย เมื่อเสร็จในแต่ละส่วนก็ประกอบส่วนย่อยแต่ละส่วนเข้าด้วยกันเป็นวงจรขนาดใหญ่ที่สมบูรณ์



รูปที่ 3.4 การแยกส่วนวงจรของหน่วยประมวลผลกลางขนาด 8 บิตออกเป็นส่วนย่อย



รูปที่ 3.5 การอธิบายการทำงานแต่ละส่วนย่อยด้วยภาษา VHDL



รูปที่ 3.5 (ต่อ) การอธิบายการทำงานแต่ละส่วนย่อยด้วยภาษา VHDL

ขอยกตัวอย่างให้เห็นถึงโปรแกรมเพียงบางส่วน เช่น วงจรในอุปกรณ์ FPGAs-1(FPGAs-1) โดยโปรแกรมที่สมบูรณ์ดูได้จากภาคผนวก ก ภายในวงจรในอุปกรณ์ FPGAs-1 ประกอบด้วยหน่วยควบคุมการทำงานของบัส (Control_Bus) หน่วยพักข้อมูลพิเศษ (S_pacial_Reg) และหน่วยกระทำเกี่ยวกับบัส (BUS) ซึ่งสามารถใช้ภาษา VHDL อธิบายการทำงานของวงจรในอุปกรณ์ FPGAs-1 ได้ ดังรูปที่ 3.6

```
----------------------------------
-- Latch VHDL Test By Kitipong --
----------------------------------
library ieee;
use ieee.std_logic_1164.all;
library synth;
use synth.vhdlsynth.all;
```

รูปที่ 3.6 การใช้ภาษา VHDL อธิบายการทำงานของหน่วยกระทำเกี่ยวกับบัส(BUS)

```
entity b_s is
 port( signal clk : in std_logic;
     signal CH        :        in std_logic_vector ( 1 downto 0);
     signal D         :        in std_logic_vector ( 7 downto 0);
     signal Q         :        out std_logic_vector( 7 downto 0);
     in_out1          :        inout std_logic_vector(7 downto 0)bus);
end b_s;
architecture a_b_s of b_s is
begin
G:process(clk)
begin
          if( clk ='1' and clk'event ) then
                    case CH is
                              when "01" => in_out1 <= std_logic_vector(D);
                              when "10" => Q <= std_logic_vector(in_out1);
                              when others => in_out1 <= "ZZZZZZZZ";
                    end case;
     end if;
end process G;
end a_b_s;
```

รูปที่ 3.6 (ต่อ) การใช้ภาษา VHDL อธิบายการทำงานของหน่วยกระทำเกี่ยวกับบัส (BUS)

```
---------------------------------
-- Latch VHDL Test By Kitipong --
---------------------------------
-- Spacial Register
-- 27 NOV 97
library ieee;
use ieee.std_logic_1164.all;
library synth;
use synth.vhdlsynth.all;

entity s_gr is
 port( clk :in std_logic;
     CH      :       in std_logic_vector ( 4 downto 0);
     D       :       in std_logic_vector ( 7 downto 0);
     Q       :       out std_logic_vector( 7 downto 0);
     addrr   :       out std_logic_vector( 15 downto 0));
end s_gr;

architecture a_s_gr of s_gr is
begin

S:process(clk)
variable clr,S_PCL,S_PCH,S_SPL : std_logic_vector(7 downto 0):= "00000000";
variable S_SPH,S_IXL,S_IXH     : std_logic_vector(7 downto 0):= "00000000";
variable PC1,clr1: std_logic_vector( 15 downto 0) :="0000000000000000";
variable PCS,S_C : std_logic_vector( 15 downto 0) :="0000000000000000";
```

รูปที่ 3.7 การใช้ภาษา VHDL อธิบายการทำงานของหน่วยพักข้อมูลพิเศษ (S_pacial_Reg)

```
begin
    if( clk'event and clk = '1' )then
                Q <="ZZZZZZZZ";
            case CH is
                when "00001" => S_PCH := clr;        S_PCL := clr;
                                S_SPH := clr;        S_SPL := clr;
                                addrr <= clr1;
                when "00010" => S_PCL := D;
                when "00011" => S_PCH := D;
                when "00100" => S_SPL := D;
                when "00101" => S_SPH := D;
                when "00110" => S_IXL := D;
                when "00111" => S_IXH := D;
                when "01000" => S_PCL := S_C(7 downto 0);
                                S_PCH := S_C(15 downto 8);
                when "01001" => S_SPL := S_C(7 downto 0);
                                S_SPH := S_C(15 downto 8);
                when "01010" => S_IXL := S_C(7 downto 0);
                                S_IXH := S_C(15 downto 8);
                when "01011" => S_C(7 downto 0) := S_PCL;
                                S_C(15 downto 8) := S_PCH;
                when "01100" => S_C(7 downto 0) := S_SPL;
                                S_C(15 downto 8) := S_SPH;
                when "01101" => S_C(7 downto 0) := S_IXL;
                                S_C(15 downto 8) := S_IXH;
                when "01110" => S_C(15 downto 8) := S_PCH;
```

รูปที่ 3.7 (ต่อ) การใช้ภาษา VHDL อธิบายการทำงานของหน่วยพักข้อมูล (S_pacial_Reg)

```
                S_C(7 downto 0) := S_PCL;
        if( S_C ="1111111111111111" ) then
                                S_C := clr1;
                        else
                                S_C := ( S_C + 1);
                        end if;
                        S_PCH := S_C(15 downto 8);
                        S_PCL := S_C(7 downto 0);
                when "01111" => S_C(15 downto 8) := S_SPH;
                        S_C(7 downto 0) := S_SPL;
        if( S_C ="1111111111111111" ) then
                                S_C := clr1;
                        else
                                S_C := ( S_C + 1);
                        end if;
                        S_SPH := S_C(15 downto 8);
                        S_SPL := S_C(7 downto 0);
                when "10000" => S_C(15 downto 8) := S_IXH;
                        S_C(7 downto 0) := S_IXL;
        if ( S_C ="1111111111111111" ) then
                                S_C := clr1;
                        else
                                S_C := ( S_C + 1);
                        end if;
                        S_IXH := S_C(15 downto 8);
                        S_IXL := S_C(7 downto 0);
```

รูปที่ 3.7 (ต่อ) การใช้ภาษา VHDL อธิบายการทำงานของหน่วยพักข้อมูล (S_pacial_Reg)

```
when "10010" => PCS( 7 downto 0) := D;
                S_C := ( S_C - PCS );
when "10011" => S_SPH := S_PCH;
                S_SPL := S_PCL;
when "10100" => S_PCH := S_SPH;
                S_PCL := S_SPL;
when "10101" => S_SPH := S_IXH;
                S_SPL := S_IXL;
when "10110" => S_IXH := S_SPH;
                S_IXL := S_SPL;
when "10111" => PC1(15 downto 8) := S_PCH;
                PC1(7 downto 0) := S_PCL;
when "11000" => S_PCH := PC1(15 downto 8);
                S_PCL := PC1(7 downto 0);
when "11001" => S_PCH := S_IXH;
                S_PCL := S_IXL;
when "11010" => Q <= S_C(15 downto 8);
when "11011" => Q <= S_C(7 downto 0);
when "11100" => addrr(15 downto 8) <= S_PCH;
                addrr(7 downto 0) <= S_PCL;
when "11101" => addrr(15 downto 8) <= S_SPH;
                addrr(7 downto 0) <= S_SPL;
when "11110" => S_PCH := clr;
                S_PCL := "00001111";
                addrr(15 downto 8) <= S_PCH;
                addrr(7 downto 0) <= S_PCL;
```

รูปที่ 3.7 (ต่อ) การใช้ภาษา VHDL อธิบายการทำงานของหน่วยพักข้อมูล (S_pacial_Reg)

```
                    when "11111" => S_C := (S_C - 1 );
                    when others => NULL;
                end case;
    end if;
end process S;
end a_s_gr;
```

รูปที่ 3.7 (ต่อ) การใช้ภาษา VHDL อธิบายการทำงานของหน่วยพักข้อมูล (S_pacial_Reg)

```
---------------------------------
-- Latch VHDL Test By Kitipong --
---------------------------------
-- Control Bus Unit
-- 27 NOV 97
library ieee;
use ieee.std_logic_1164.all;
library synth;
use synth.vhdlsynth.all;

entity c_bus is
 port( signal clk :in std_logic;
    signal CH : in std_logic_vector ( 2 downto 0);
    signal alu_gr_i, sgr_i, ctrl_i, data_i  : in std_logic_vector ( 7 downto 0);
    signal alu_gr_o, sgr_o,ctrl_o, data_o : out std_logic_vector( 7 downto 0));
end c_bus;
```

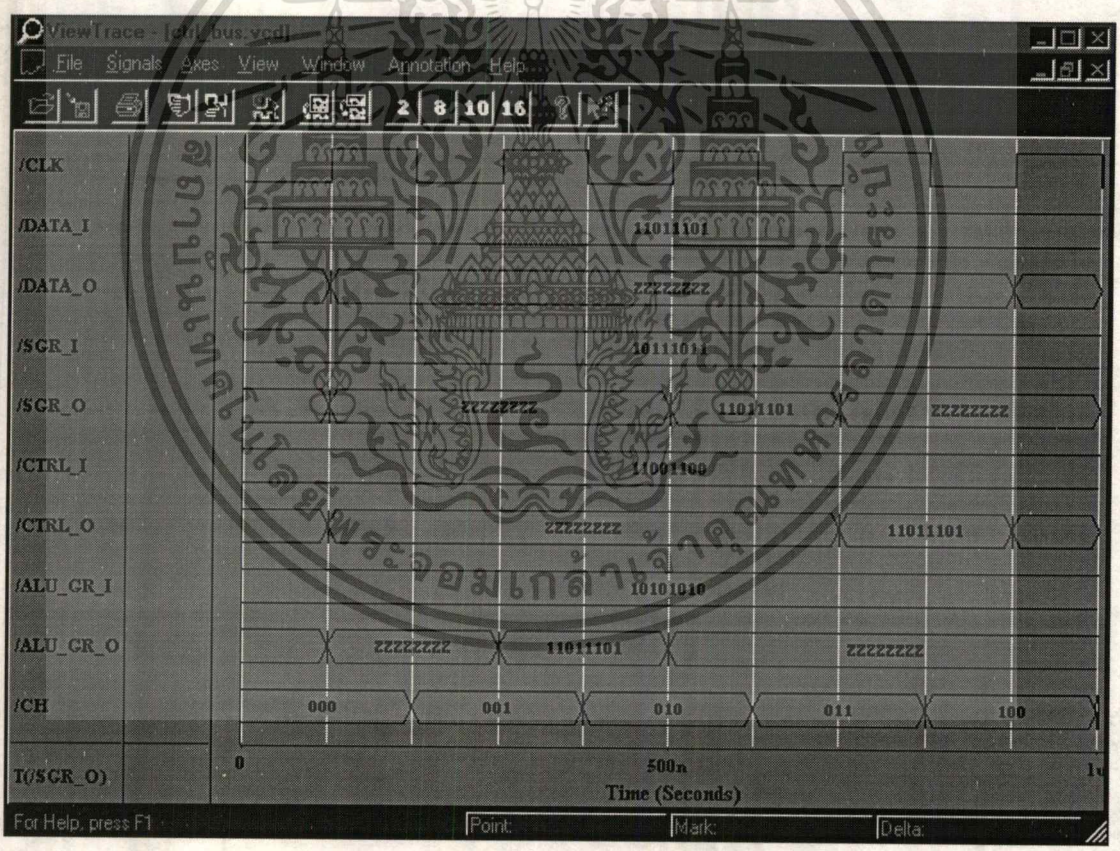
รูปที่ 3.8 การใช้ภาษา VHDL อธิบายการทำงานของหน่วยควบคุมบัส (Control_Bus)

```
architecture ac_bus of c_bus is
begin

B:process(clk)
begin
    if( clk'event and clk = '1' )then
            alu_gr_o <="ZZZZZZZZ";
            sgr_o <="ZZZZZZZZ";
            ctrl_o <="ZZZZZZZZ";
            data_o <="ZZZZZZZZ";

                case CH is
                        when "001" => alu_gr_o <= data_i;
                        when "010" => sgr_o  <= data_i;
                        when "011" => ctrl_o  <= data_i;
                        when "100" => data_o  <= alu_gr_i;
                        when "101" => data_o  <= sgr_i;
                        when "110" => sgr_o  <= ctrl_i;
                        when "111" => ctrl_o <= alu_gr_i;
                        when others => NULL;
                end case;
            end if;
 end process B;
end ac_bus;
```

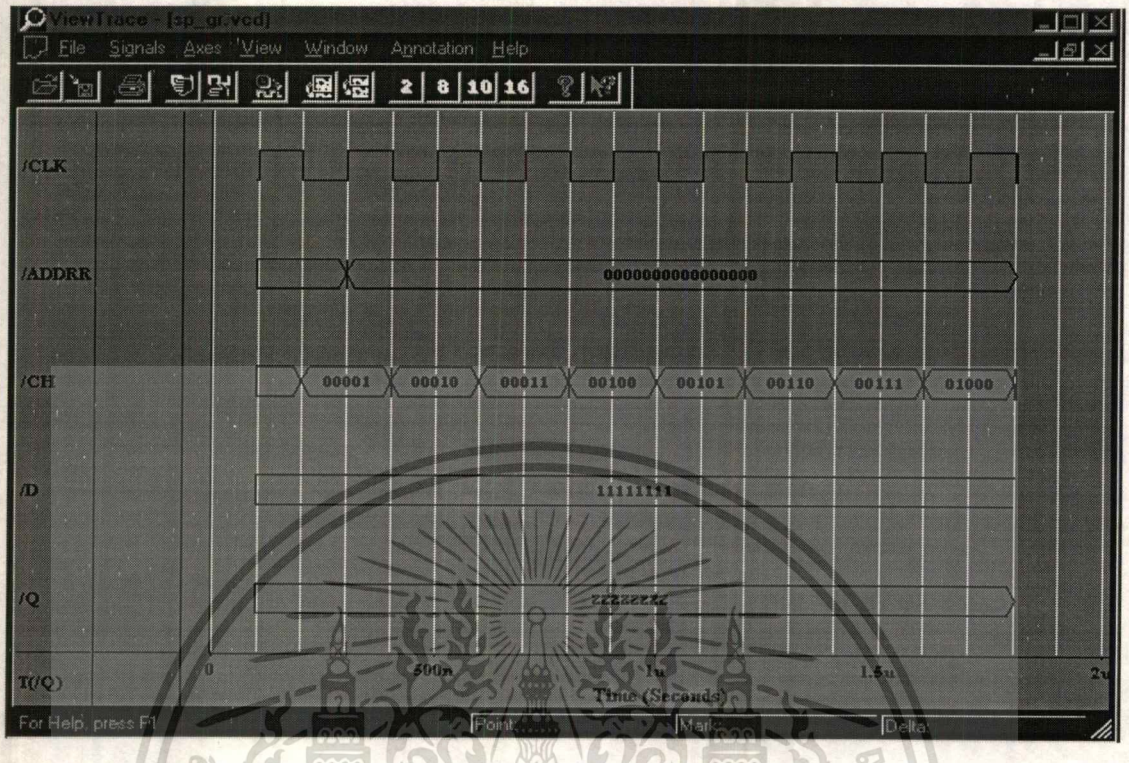
รูปที่ 3.8 (ต่อ) การใช้ภาษา VHDL อธิบายการทำงานของหน่วยควบคุมบัส (Control_Bus)

3.6.2 การจำลองการทำงาน

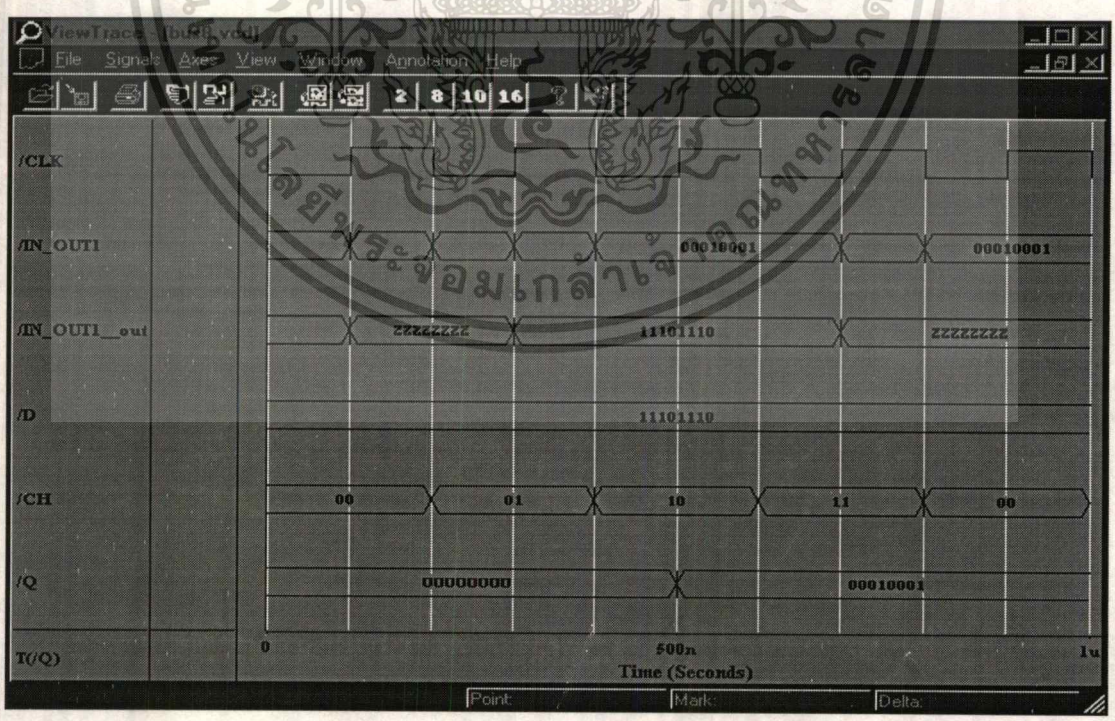
หลังจากที่เขียนโปรแกรมหน่วยกระทำทางข้อมูลเสร็จสมบูรณ์และทำการแปลงภาษาจนไม่พบข้อผิดพลาดขั้นต้น ในขั้นต่อมาก็จะเป็นการทดสอบการทำงานของโปรแกรมที่เขียนขึ้นมาเพื่อดูว่าการทำงานเป็นไปตามวัตถุประสงค์ที่ได้ออกแบบไว้ ในการจำลองการทำงานของโปรแกรมจะต้องสร้างสัญญาณทดสอบวงจรในทุกสภาวะ เพื่อให้แน่ใจว่าผลลัพธ์ที่ได้นั้นตรงตามที่ได้ออกแบบไว้ โดยการใช้โปรแกรมวิวเทรซเป็นตัวแสดงผลของสัญญาณต่างๆ ดังรูปที่ 3.9 - 3.11 และรูปที่ 3.12-3.14 เป็นผลของการใช้โปรแกรมวิวเทรซแสดงสัญญาณต่าง ๆ ของวงจรของหน่วยประมวลผลกลางขนาด 8 บิตซึ่งการทำงานและการใช้โปรแกรมวิวเทรซได้กล่าวไว้แล้ว



รูปที่ 3.9 การใช้โปรแกรมวิวเทรซแสดงสัญญาณต่าง ๆ ของหน่วย Control_Bus



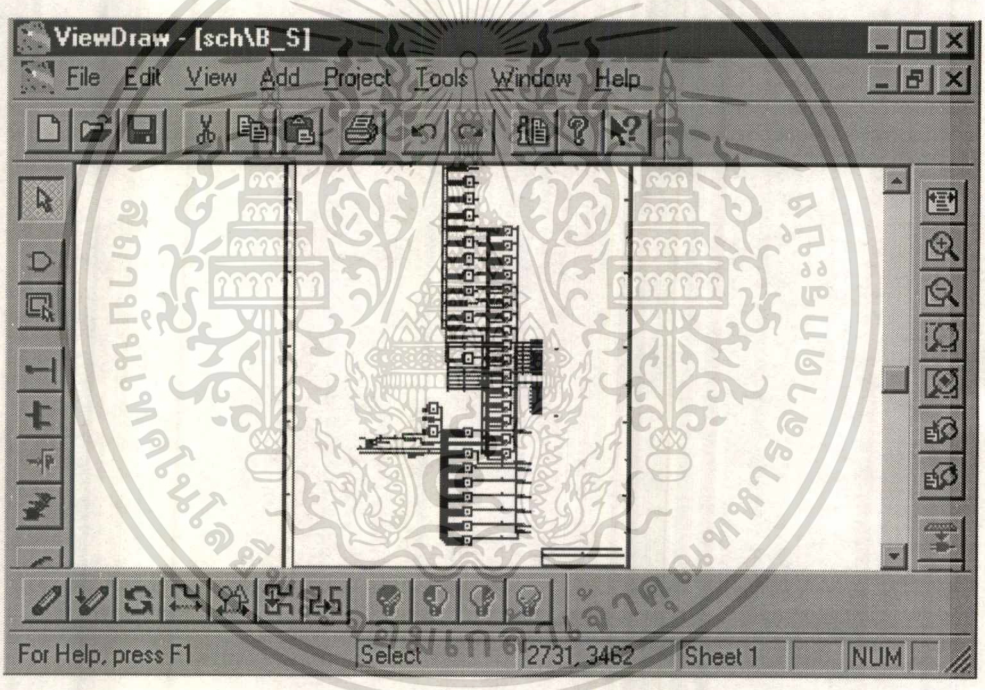
รูปที่ 3.10 การใช้โปรแกรมวิวเทรซแสดงสัญญาณต่าง ๆ ของหน่วย S_pacial_Reg



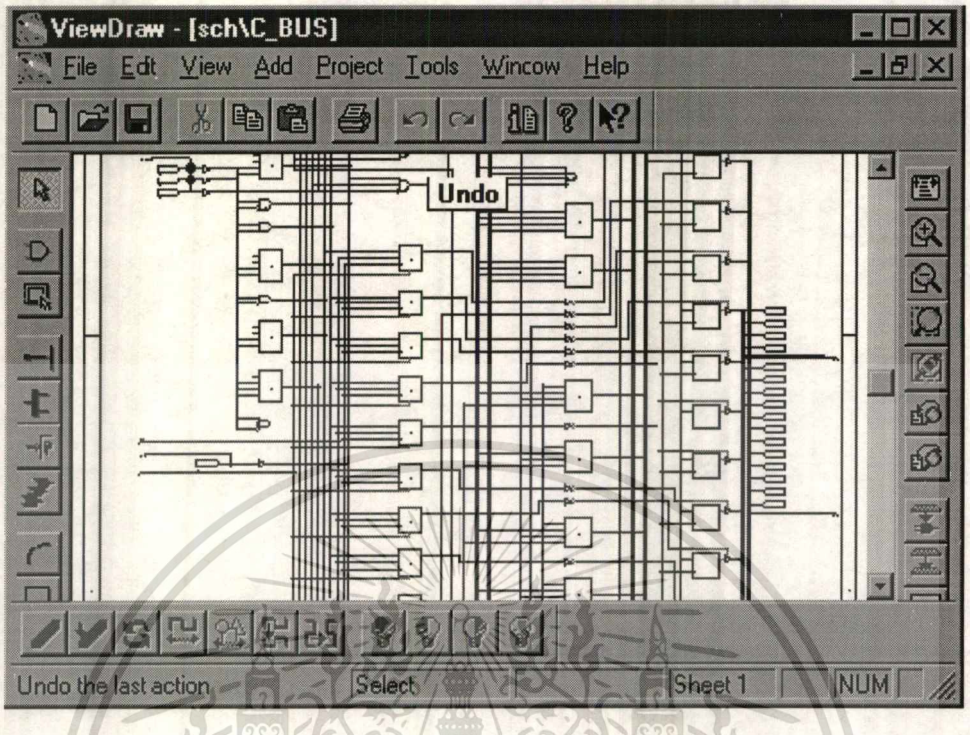
รูปที่ 3.11 การใช้โปรแกรมวิวเทรซแสดงสัญญาณต่าง ๆ ของวงจรบัส (BUS)

3.6.3 การสังเคราะห์เป็นวงจรระดับเกท

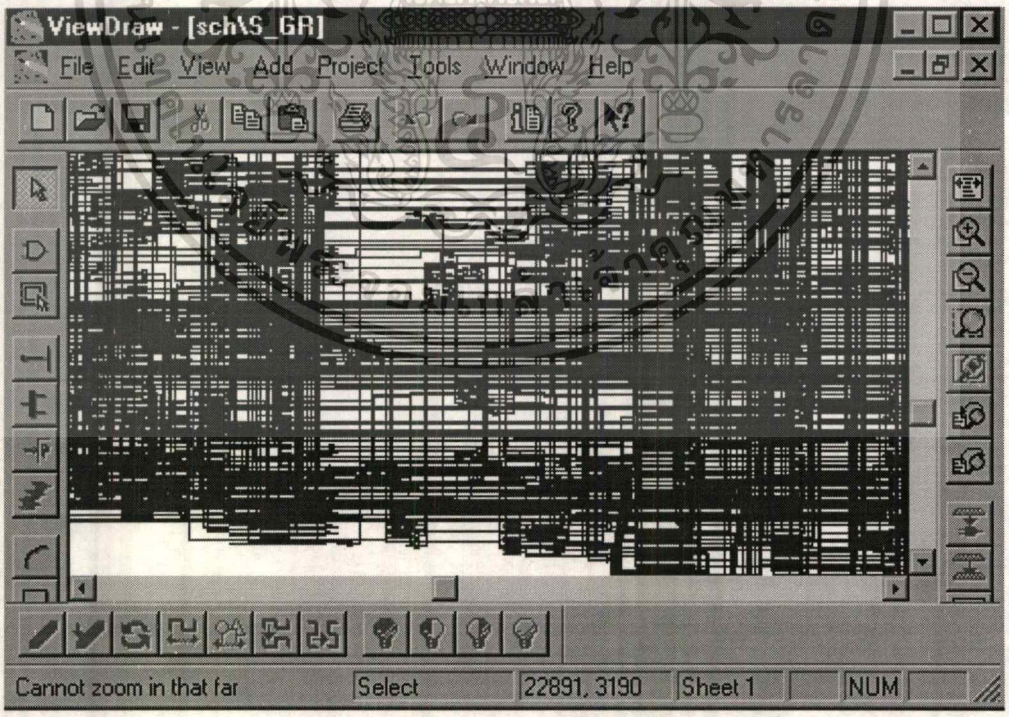
หลังจากการจำลองผลการทำงานจนได้ผลการทำงานตรงตามวัตถุประสงค์ที่ออกแบบไว้แล้วขั้นตอนต่อมาคือ แปลงโปรแกรมของวงจรของหน่วยประมวลผลกลางขนาด 8 บิต ให้เป็นวงจรซึ่งประกอบไปด้วยอุปกรณ์ดิจิตอลพื้นฐานจำพวกเกทและฟลิปฟลอปต่าง ๆ ในการสังเคราะห์เป็นวงจรระดับเกทนี้ต้องกำหนดด้วยว่าจะใช้อุปกรณ์พื้นฐานของบริษัทใด ซึ่งโครงงานนี้เลือกทำการพัฒนาวงจรที่ได้ลงบนอุปกรณ์ FPGAs เบอร์ XC4010 ของบริษัทไซลิงค์ ดังรูปที่ 3.12-3.14 ซึ่งเป็นวงจรระดับเกทของวงจรบัส (BUS) ,วงจรควบคุมบัส (Control_BUS) และวงจรพักข้อมูลพิเศษ (S_pachial_Reg) ตามลำดับ



รูปที่ 3.12 วงจรระดับเกทของวงจรบัส (BUS)



รูปที่ 3.13 วงจรระดับเกทของวงจรควบคุมบัส (Control_BUS)



รูปที่ 3.14 วงจรระดับเกทของวงจรพักข้อมูลพิเศษ (S_pachial_Reg)

3.6.4 การโปรแกรมวงจรลงบนไอซี FPGAs

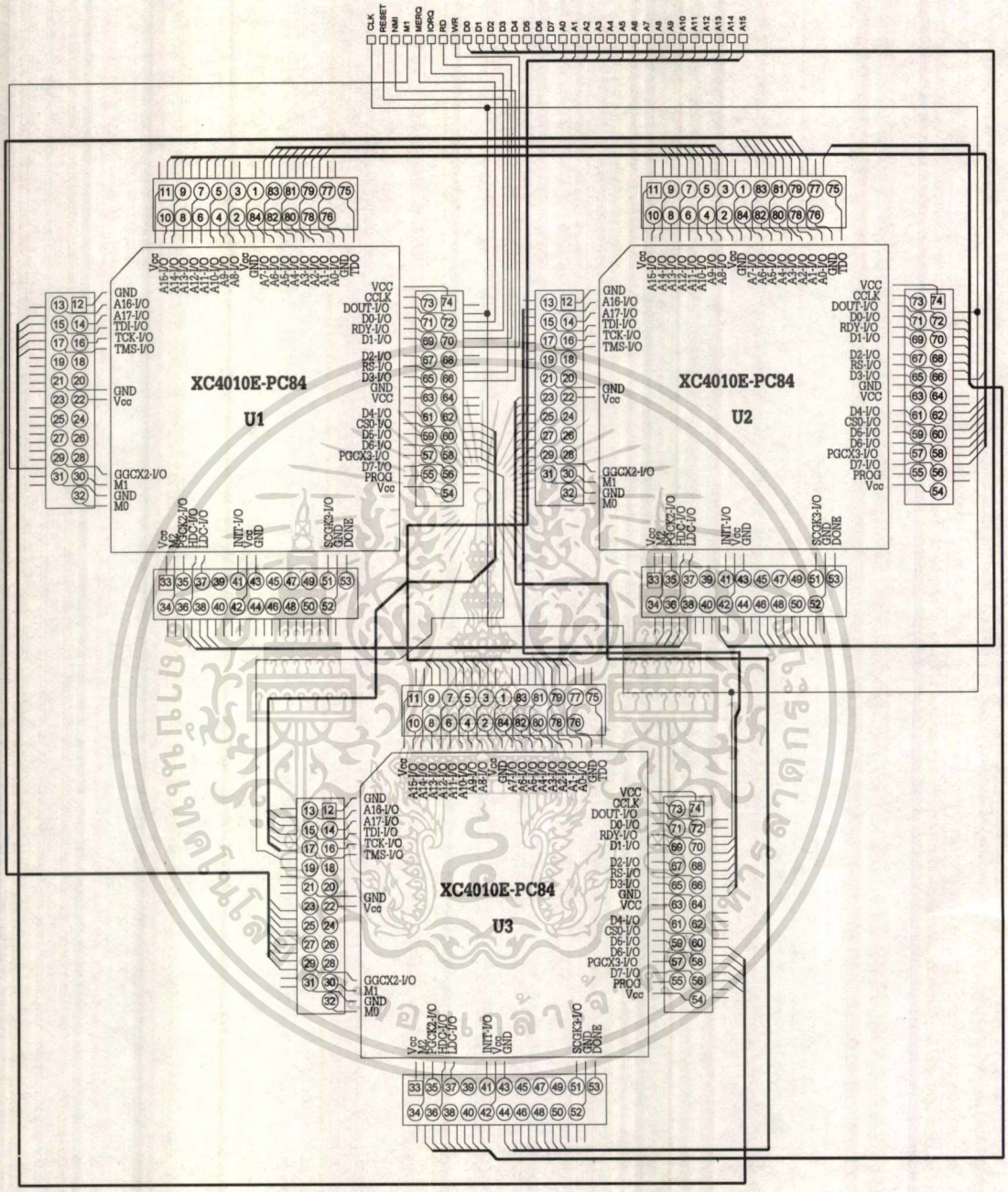
หลังจากสังเคราะห์วงจรระดับเกทของวงจรของหน่วยประมวลผลกลางขนาด 8 บิต แล้ว ขั้นตอนต่อมาเป็นการนำเอาวงจรที่ได้มาทำการดาวน์โหลดลงไอซี FPGAs โดยใช้ซอฟ-แวร์ XACT Development Tools ของบริษัทไซลิงค์ในการดาวน์โหลดวงจรหน่วยประมวลผลกลางขนาด 8 บิตที่ได้ลงบนเซลล์ต่าง ๆ ภายในไอซี FPGAs ผลสุดท้ายจะได้ออกมาเป็นวงจรหน่วยประมวลผลกลางขนาด 8 บิตบนไอซี FPGAs ซึ่งสามารถนำไปสร้างวงจรต้นแบบได้ทันที

3.7 การสร้างวงจรทดสอบ

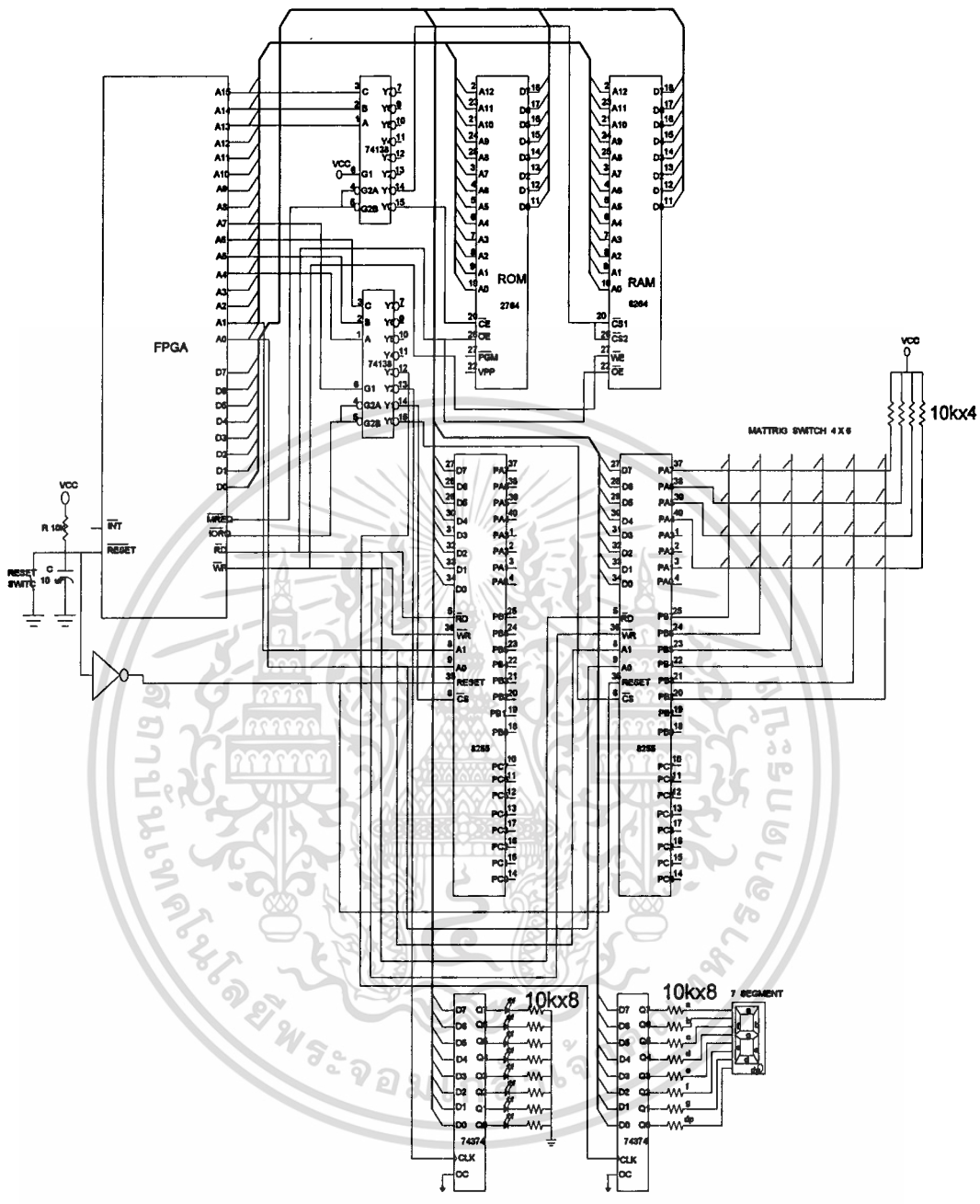
วงจรที่ใช้ในการทดสอบหน่วยประมวลผลกลางขนาด 8 บิตนั้นเป็นแบบบอร์ดตัวอย่างของ FPGAs ซึ่งจะทำการโปรแกรมผ่านทางดาวน์โหลดเคเบิล (Download Cable) โดยวงจรที่ใช้ทดสอบหน่วยประมวลผลกลางขนาด 8 บิตจะเป็นวงจรส่วนต่าง ๆ ดังนี้

3.7.1 วงจรไอซี FPGAs

วงจรไอซี FPGAs เบอร์ XC4005APC84C-6 ทำหน้าที่ควบคุมการทำงานของหน่วยประมวลผลกลางขนาด 8 บิต โดยรับโปรแกรมผ่านทางดาวน์โหลดเคเบิล



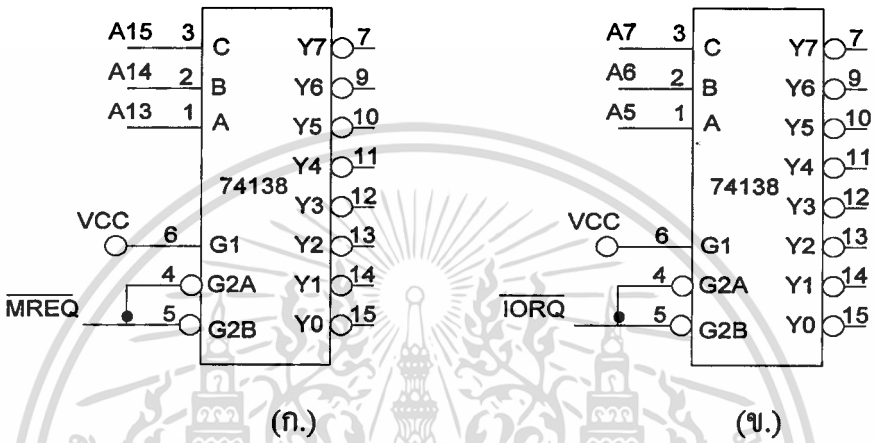
รูปที่ 3.15 วงจรไอซี FPGAs



รูปที่ 3.16 วงจรอินเตอร์เฟสไอซี FPGAs

3.7.2 วงจรดีโค๊ดเดอร์

จากรูปที่ 3.17 เป็นวงจรดีโค๊ดเดอร์ของตำแหน่งหน่วยความจำและวงจรดีโค๊ดเดอร์ของตำแหน่งพอร์ท ทำหน้าที่เป็นวงจรที่ติดต่อกับหน่วยความจำภายนอกและพอร์ทอุปกรณ์อินพุทเอาท์พุท

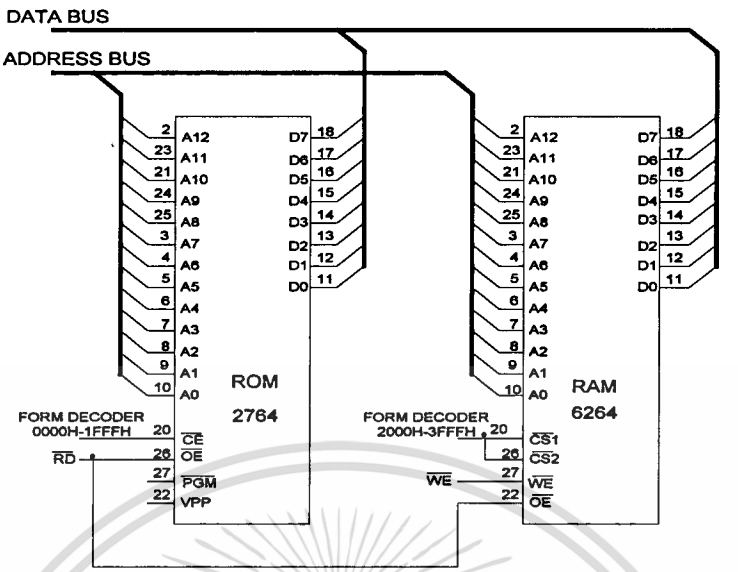


(ก.) (ข.)

รูปที่ 3.17 (ก) วงจรดีโค๊ดเดอร์ตำแหน่งหน่วยความจำ
(ข) วงจรดีโค๊ดเดอร์ตำแหน่งพอร์ต

3.7.3 วงจรหน่วยความจำ ROM และ RAM

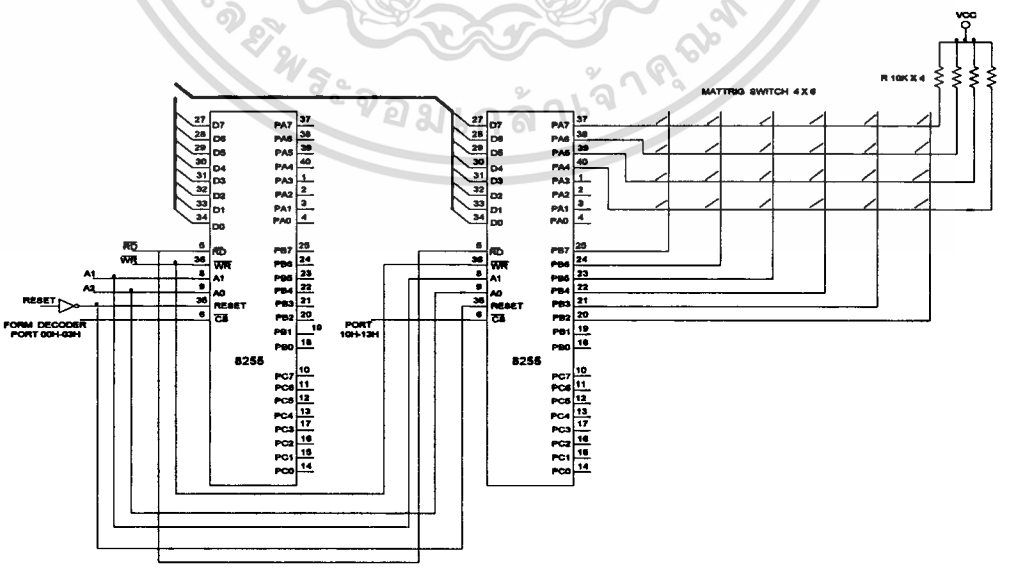
จากรูปที่ 3.18 ไอซีทางด้านซ้ายมือเป็นวงจรหน่วยความจำชนิดอ่านอย่างเดียว หรือเรียกว่า ROM (Ready Only Memory) ไอซีทางด้านขวาเป็นวงจรชนิดอ่านและเขียนได้หรือเรียกว่า RAM (Random Access Memory) ทำหน้าที่เก็บข้อมูลหรือคำสั่งที่ใช้ในโปรแกรมเพื่อสั่งให้หน่วยประมวลผลกลางขนาด 8 บิตทำงาน



รูปที่ 3.18 วงจรหน่วยความจำ ROM และ RAM

3.7.4 วงจรคีย์บอร์ด

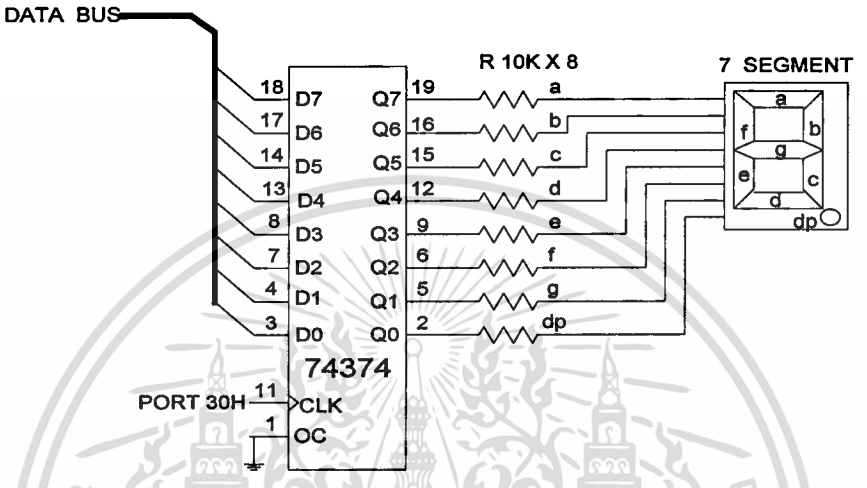
จากรูปที่ 3.19 เป็นวงจรคีย์บอร์ดซึ่งทางด้านซ้ายมือเป็นวงจรพอร์ทขยายสำหรับต่อวงจรอินเตอร์เฟสทั่วไปมีหมายเลขพอร์ท 10H-13H และวงจรทางด้านขวาเป็นวงจรพอร์ทที่ใช้สำหรับการสแกนคีย์บอร์ดโดยใช้ไอซีเบอร์ 8255 มีหมายเลขพอร์ทคือ 00H- 03H



รูปที่ 3.19 วงจรคีย์บอร์ด

3.7.5 วงจรภาคแสดงผล

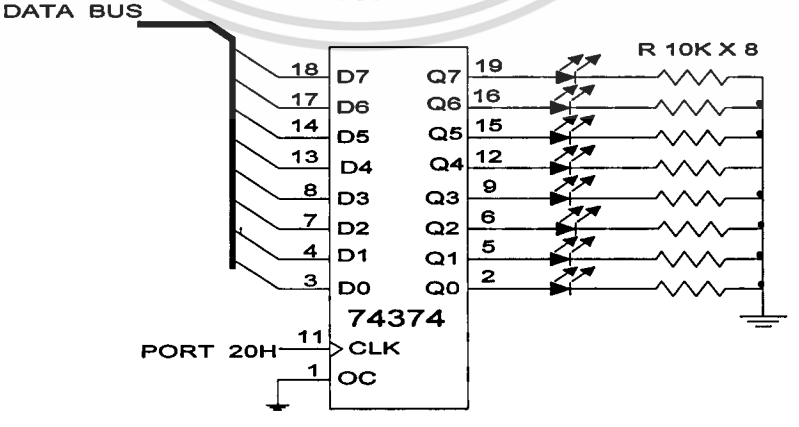
จากรูปที่ 3.20 เป็นวงจรพอร์ต 7 - SEGMENT โดยใช้ไอซี 74374 เป็นพอร์ตเอาท์พุท สำหรับการแสดงผลตัวเลข 0-1 และ A-F และหลายเลขประจำพอร์ตนี้คือ 30H



รูปที่ 3.20 วงจรภาคแสดงผล 7-SEGMENT

3.7.6 วงจรภาคแสดงผลขับ LED ขนาด 8 ตัว

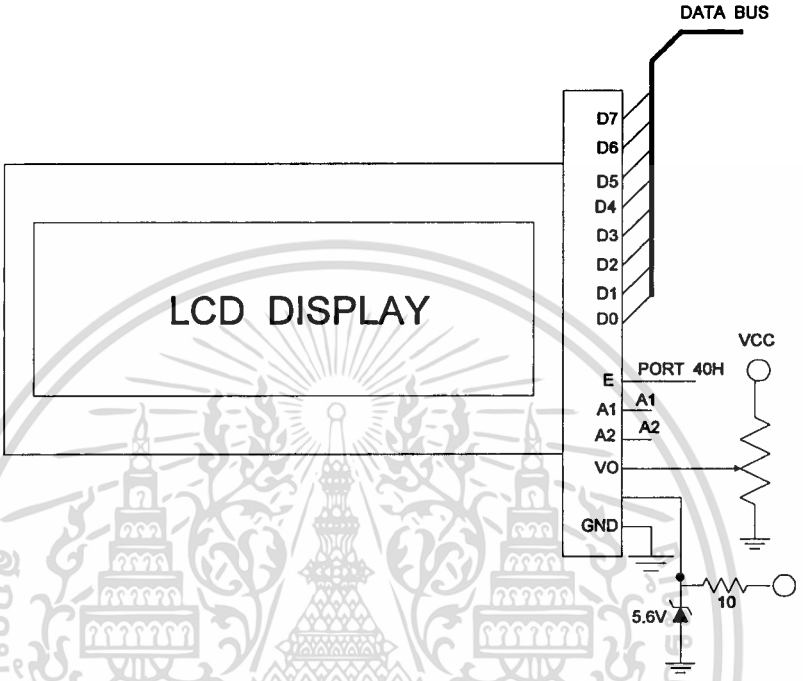
จากรูปที่ 3.21 คือวงจรขับ LED จำนวน 8 ตัว โดยใช้ไอซีเบอร์ 74374 เป็นพอร์ตโดยมีหมายเลขตำแหน่งพอร์ตคือ 20H



รูปที่ 3.21 วงจรแสดงผลขับ LED ขนาด 8 ตัว

3.7.7 วงจรขับ LCD

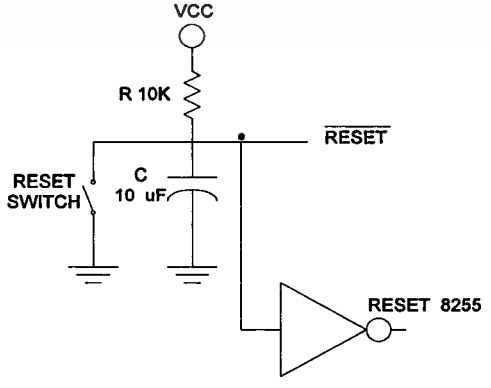
จากรูปที่ 3.26 คือวงจรขับ LCDโดยมีหมายเลขตำแหน่งพอร์ตคือ 40H



รูปที่ 3.22 วงจรขับ จอ LCD

3.7.8 วงจรรีเซ็ท

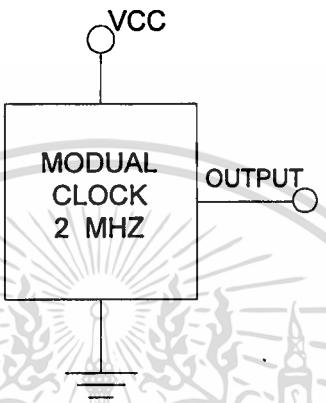
จากรูปที่ 3.23 คือวงจรรีเซ็ทระบบและยังรีเซ็ทให้กับไอซี 8255 ด้วย โดยต้องต่อผ่านเกทอินเวิร์ทติ้งก่อน



รูปที่ 3.23 วงจรรีเซ็ท

3.7.9 วงจรโมดูลคล็อก

จากรูปที่ 2.24 เป็นวงจรโมดูลคล็อกที่ป้อนสัญญาณนาฬิกาให้กับระบบ โดยจะทำหน้าที่เป็นตัวผลิตความถี่ 2 MHZ



รูปที่ 2.24 วงจรโมดูลคล็อกที่ป้อนสัญญาณนาฬิกาให้กับระบบ

บทที่ 4

การทดลองและผลการทดลอง

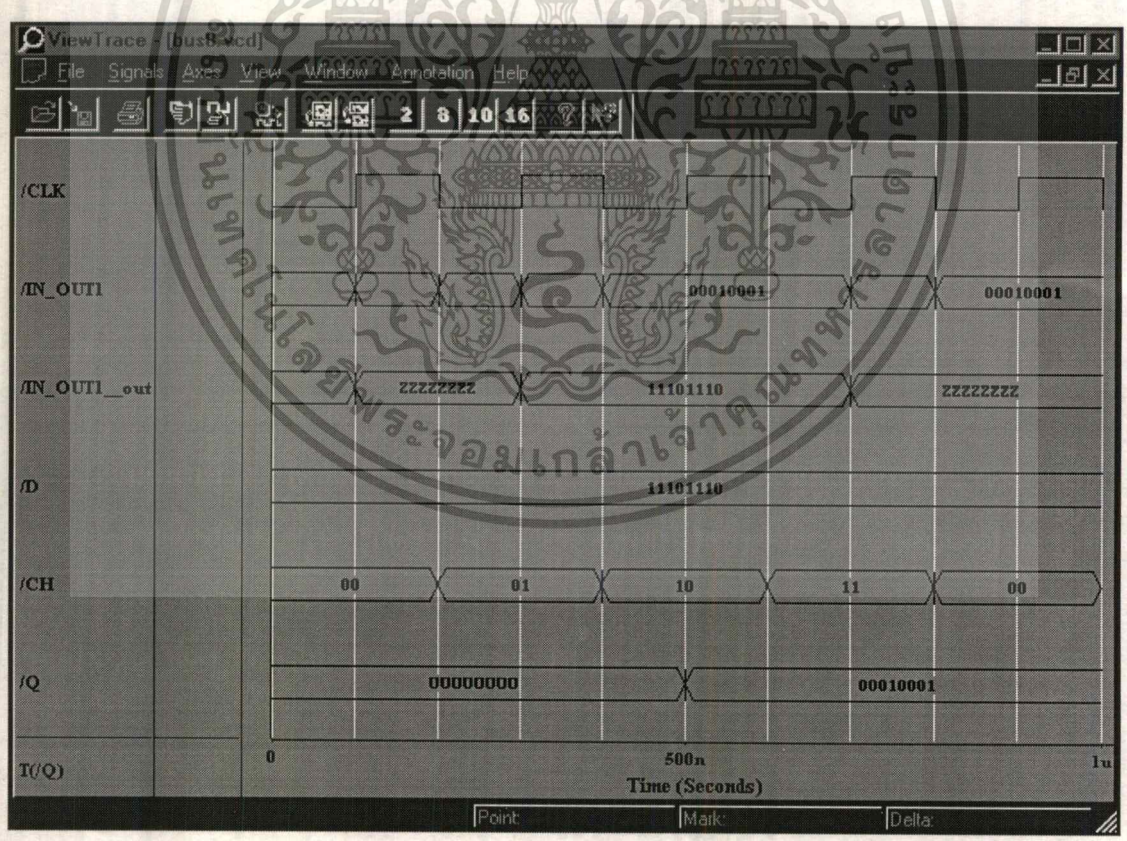
จากการออกแบบหน่วยประมวลผลกลางขนาด 8 บิทตามขั้นตอนที่กล่าวมาคือ เขียนโปรแกรมภาษา VHDL แล้วนำไปจำลองการทำงานจนได้ผลการทำงานตามวัตถุประสงค์ที่ออกแบบ จึงนำโปรแกรมไปสังเคราะห์เป็นวงจรระดับเกทและแปลงไฟล์ที่ได้จากการสังเคราะห์เป็นไฟล์บิทสตรีมเพื่อทำการดาวน์โหลดบนอุปกรณ์ FPGAs ปรากฏว่าหน่วยประมวลผลกลางขนาด 8 บิทที่ออกแบบมีขนาดใหญ่ไม่สามารถโปรแกรมลงบนไอซี FPGAs เบอร์ XC4010E ได้ และคณะผู้จัดทำไม่มีไอซี FPGAs เบอร์ที่มีจำนวนเกทมากกว่านี้จึงจำเป็นต้องลดคำสั่งของหน่วยประมวลผลกลางลงและใช้ FPGAs เบอร์ XC4010E จำนวน 3 ตัวมาต่อกันเพื่อให้สามารถโปรแกรมลงไอซี FPGAs เบอร์ XC4010E ได้ เนื้อหาในบทนี้จึงเป็นการทดลองการทำงานของหน่วยประมวลผลกลางขนาด 8 บิท โดยมีขั้นตอนและผลการทดลองดังนี้

รูปที่ 4.1 ภาพถ่ายบอร์ดตัวอย่างของ FPGAs

4.1 ผลการ Simulate Block ส่วนย่อย

จากการที่ได้ออกแบบวงจรส่วนต่าง ๆ ของหน่วยประมวลผลกลางขนาด 8 บิทด้วยภาษา VHDL เสร็จเรียบร้อยแล้วขั้นต่อมาก็ได้ทำการจำลองการทำงานในส่วนต่าง ๆ เพื่อตรวจสอบข้อผิดพลาดในการออกแบบในวงจรส่วนย่อย ๆ ก่อนในที่นี้คือ วงจรควบคุมบัส (Control_Bus) วงจรบัส (BUS) วงจรพักข้อมูลพิเศษ (S_pacial_Reg) ซึ่งจะแสดงดังรูปที่ 4.2 - 4.4 ก่อนที่จะนำเชื่อมต่อให้เป็น Block ใหญ่เพื่อนำไปดาวน์โหลดลงบนอุปกรณ์ FPGAs ต่อไป สำหรับขั้นตอนการ Simulate สามารถดูได้ที่ภาคผนวก ข

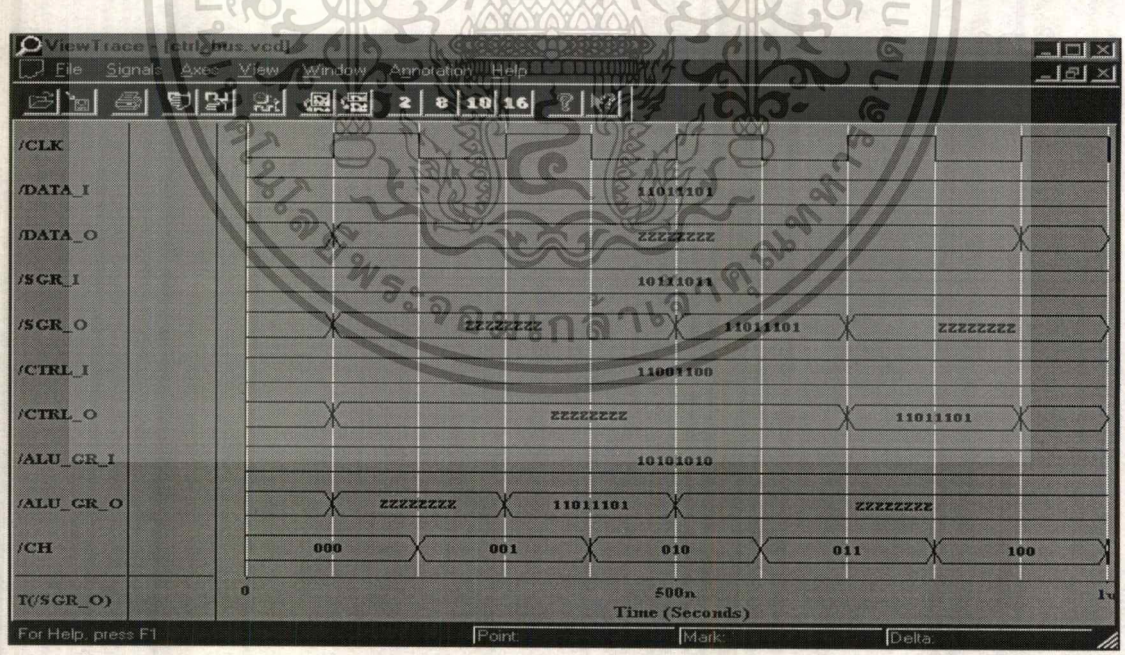
ขั้นตอนที่ 1 ได้ทำการทดสอบวงจรบัส โดยการป้อนสัญญาณ D = “11101110”, in_out1 = “00010001” CH = “00-11” และ CLK ซึ่งผลที่ได้เป็นไปตามโปรแกรมที่ได้เขียนขึ้น คือ Q จะมีค่าเท่ากับ in out1 เมื่อ CH = “10”ส่วนที่ CH = “01” in_out1 จะมีค่าเท่ากับ D ส่วน CH อื่น ๆ in_out1 มีค่าเป็น “ZZZZZZZZ” ดังรูปที่ 4.2



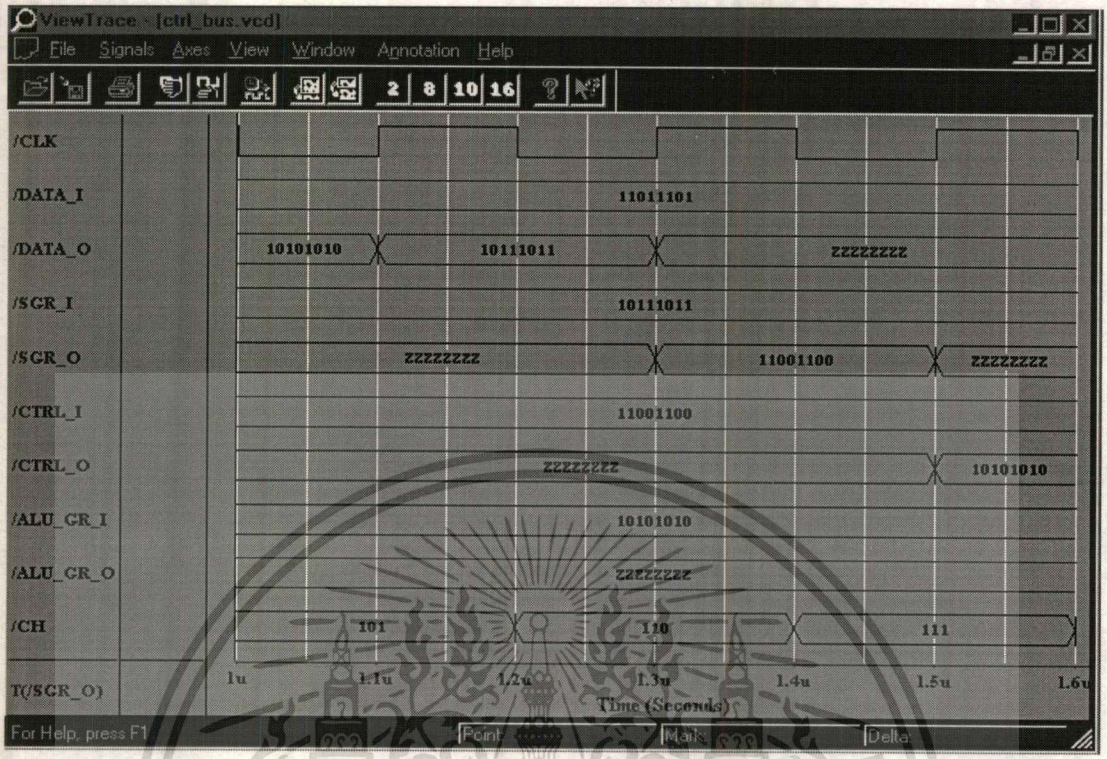
รูปที่ 4.2 ผลการ Simulate ของ Block BUS

ขั้นตอนที่ 2 ได้ทำการทดสอบวงจรควบคุมบัส โดยการป้อนสัญญาณ alu_gr_1 = "10101010", sgr_i = "10111011", ctrl_I = "11001100" , data_I = "11011101" ,CH = "000-111" และ CLK ซึ่งผลที่ได้เป็นไปตามโปรแกรมที่ได้เขียนขึ้นดังรูปที่ 4.2 คือ

1. ข้อมูลใน data_I จะถูกส่งให้ alu_gr เมื่อ CH = "001"
2. ข้อมูลใน data_I จะถูกส่งให้ sgr_o เมื่อ CH = "010"
3. ข้อมูลใน data_I จะถูกส่งให้ ctrol_o เมื่อ CH = "011"
4. ข้อมูลใน alu_gr_i จะถูกส่งให้ data_o เมื่อ CH = "100"
5. ข้อมูลใน sgr_i จะถูกส่งให้ data_o เมื่อ CH = "101"
6. ข้อมูลใน ctrl_i จะถูกส่งให้ sgr_o เมื่อ CH = "110"
7. ข้อมูลใน alu_gr จะถูกส่งให้ ctrol_o เมื่อ CH = "111"
8. ส่วน CH อื่น ๆ ข้อมูลทั้งหมดจะเป็น "ZZZZZZZZ"

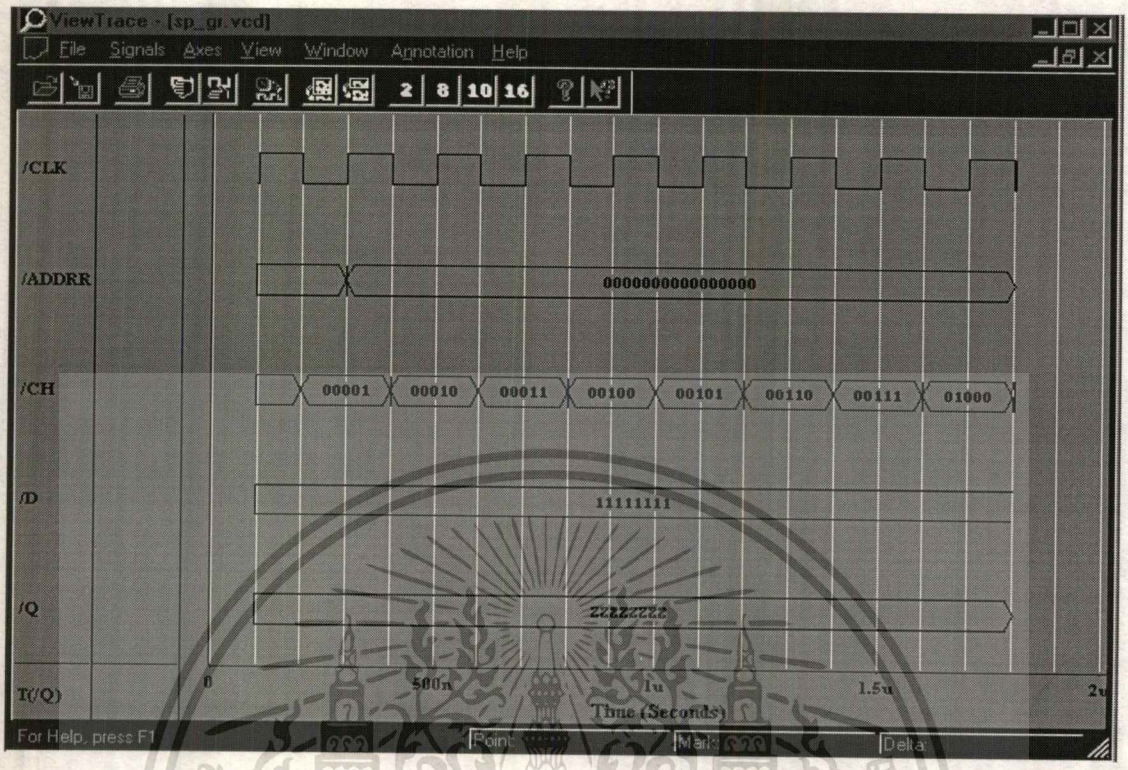


รูปที่ 4.3 ผลการ Simulate ของ Block Control_Bus

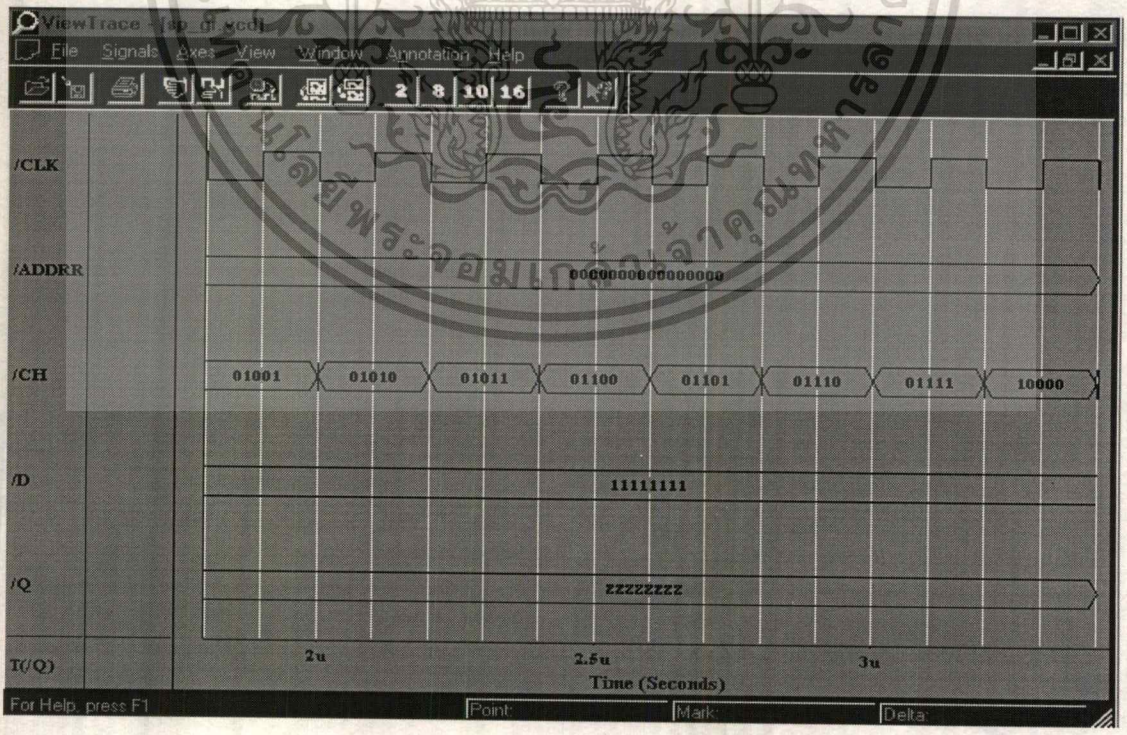


รูปที่ 4.3 (ต่อ) ผลการ Simulate ของ Block Control_Bus

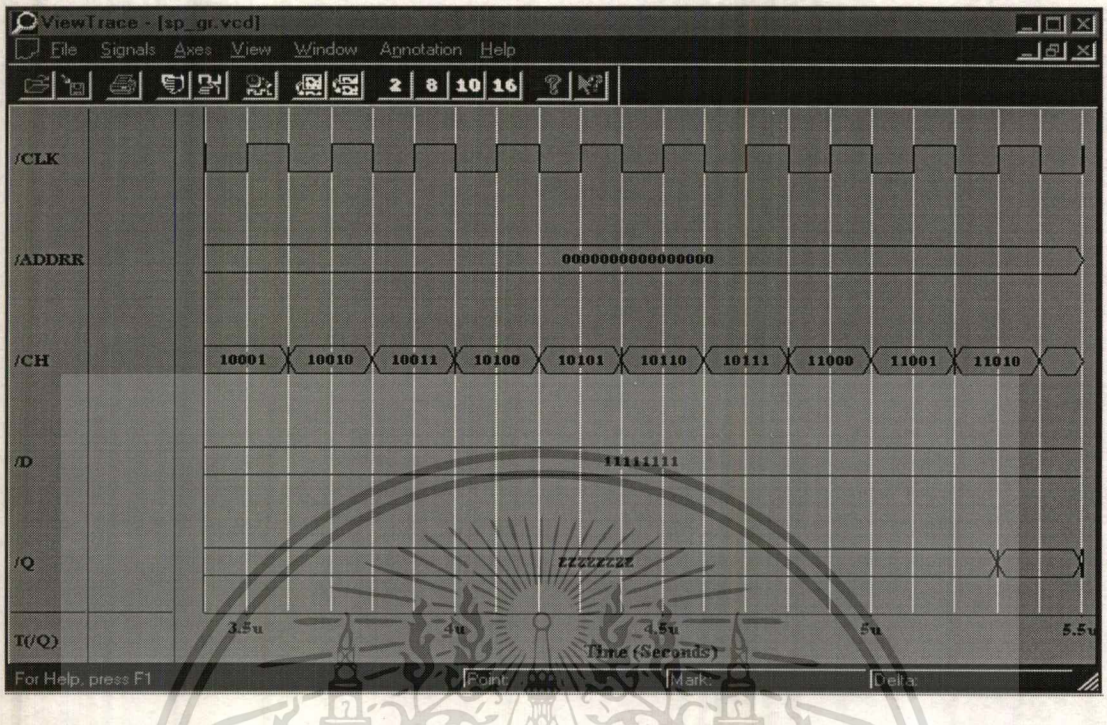
ขั้นตอนที่ 3 ได้ทำการทดสอบหน่วยพักข้อมูลพิเศษ (Control_Bus) โดยการป้อนสัญญาณ D= "11111111", CH = "00000-11111" และ CLK ซึ่งผลที่ได้เป็นไปตามโปรแกรมที่ได้เขียนขึ้นดังรูปที่ 4.4 คือ สัญญาณ CH = "00001-11001" จะเป็นการกระทำคำสั่งภายใน เช่น การบวกลบ การเก็บข้อมูลต่าง ๆ ส่วนที่สัญญาณ CH = "11010-11011" จะเป็นการส่งผลลัพธ์ให้แก่ Q ทางด้านบิตด้านสูงและต่ำตามลำดับ และที่สัญญาณ CH = "11100-11101" จะเป็นการส่งสัญญาณของโปรแกรมเคาท์เตอร์ (PC), สแต็คพอยเตอร์ (SP) ตามลำดับให้กับแอดเดรส (Addrr)



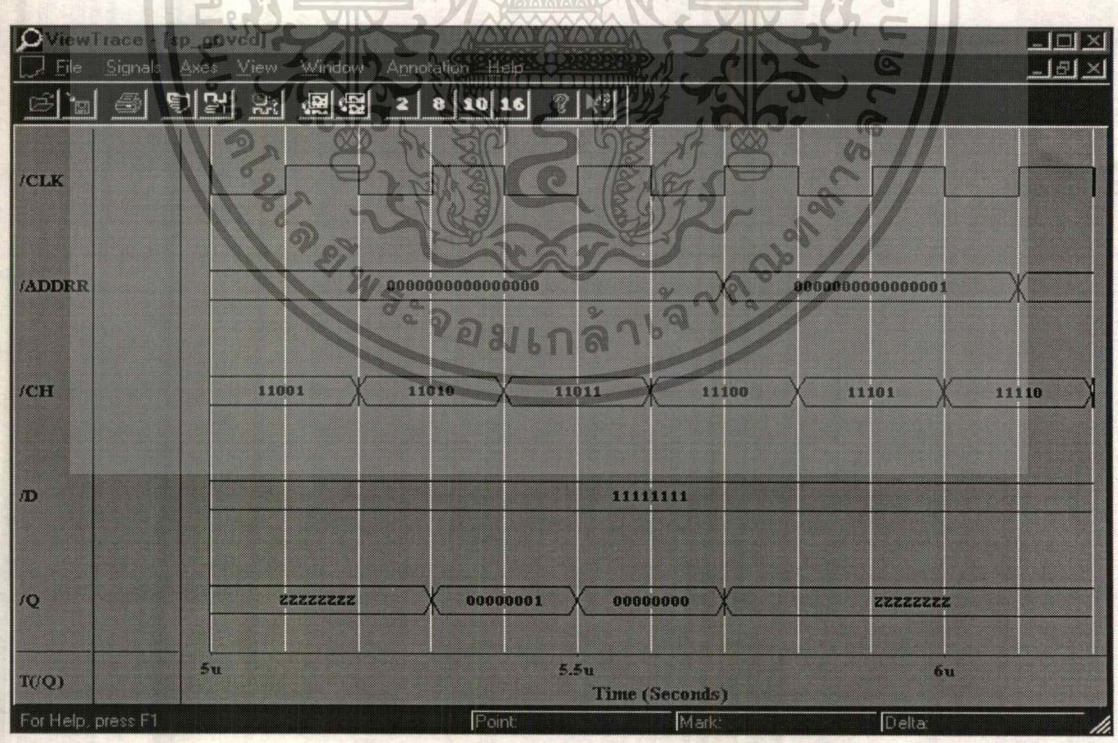
รูปที่ 4.4 ผลการ Simulate ของ Block S_pacail_Reg



รูปที่ 4.4 (ต่อ) ผลการ Simulate ของ Block S_pacail_Reg



รูปที่ 4.4 (ต่อ) ผลการ Simulate ของ Block S_pacail_Reg

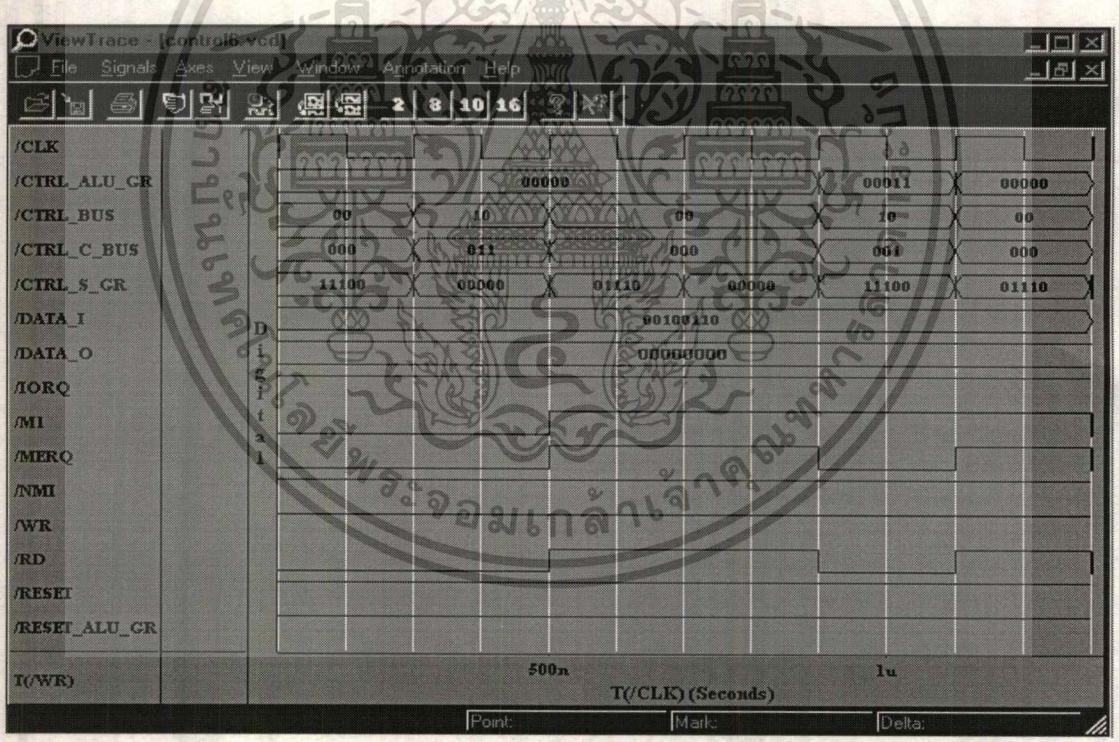


รูปที่ 4.4 (ต่อ) ผลการ Simulate ของ Block S_pacail_Reg

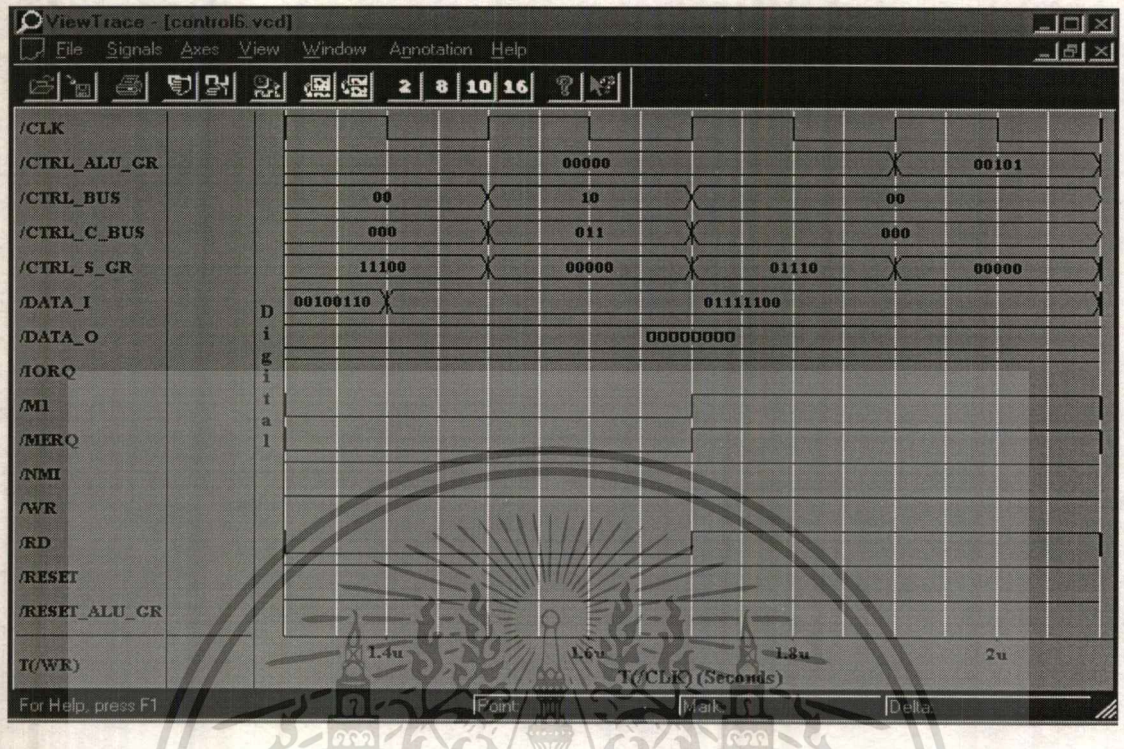
4.2 ผลการ Simulate ของ Block รวม

จากการที่ได้จำลองการทำงานของส่วนย่อย ๆ แล้วก็ทำการรวมวงจรส่วนย่อย ๆ เข้าด้วยกันแล้วทำการจำลองการทำงานอีกครั้งหนึ่ง เพื่อตรวจสอบข้อผิดพลาดในการเชื่อมต่อคือวงจรในส่วนอุปกรณ์ FPGAs - 1 , FPGAs - 2 ซึ่งจะแสดงดังรูป ก่อนที่จะนำไปดาวน์โหลดลงบนอุปกรณ์ FPGAs ต่อไป

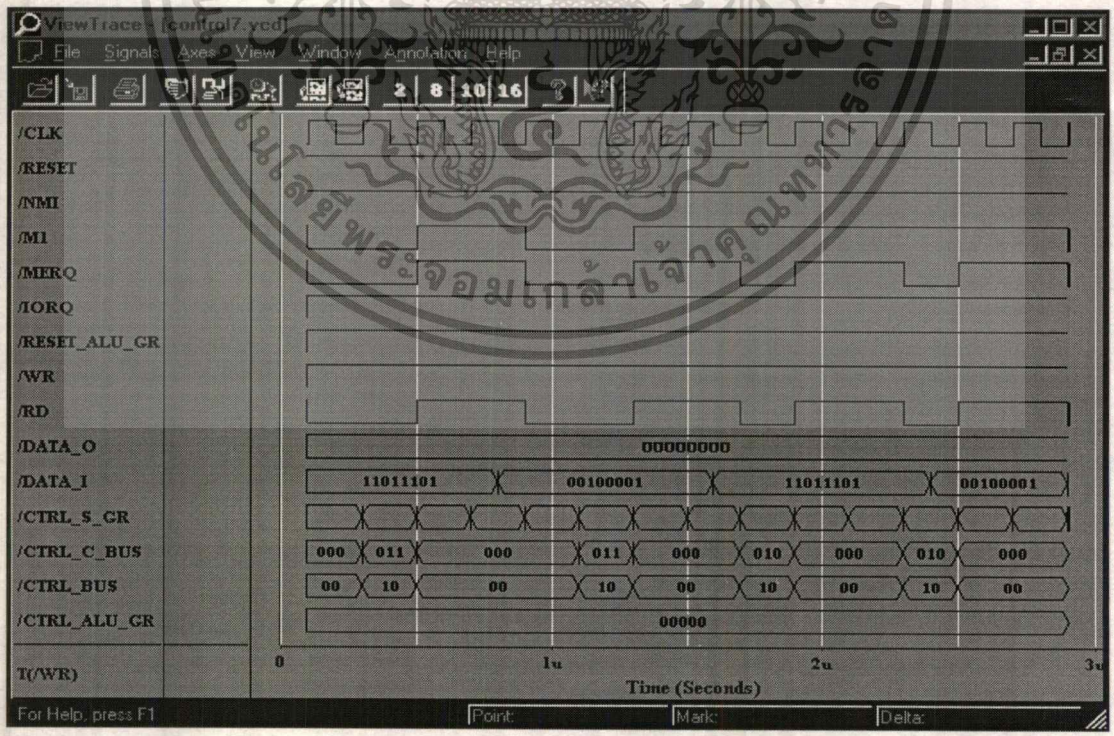
ขั้นตอนที่ 1 ได้ทำการทดสอบหน่วยควบคุม (Control) โดยการป้อนสัญญาณ Data = "00100011"และ CLK ซึ่งในส่วนของหน่วยควบคุม (Control) จะทำการผลิตสัญญาณเพื่อส่งไปควบคุมหน่วยต่าง ๆให้ทำงานตามต้องการ สัญญาณที่สร้างขึ้นก็มีสัญญาณ M1, MREQ, IORQ, RD , WR, ctrl_alu_gr, reset_alu_gr, ctrl_s_gr, ctrl_c_bus, ctrl_bus, data_o ผลที่ได้จากการทดสอบเป็นไปตามโปรแกรมที่ได้เขียนขึ้นดังรูปที่ 4.5



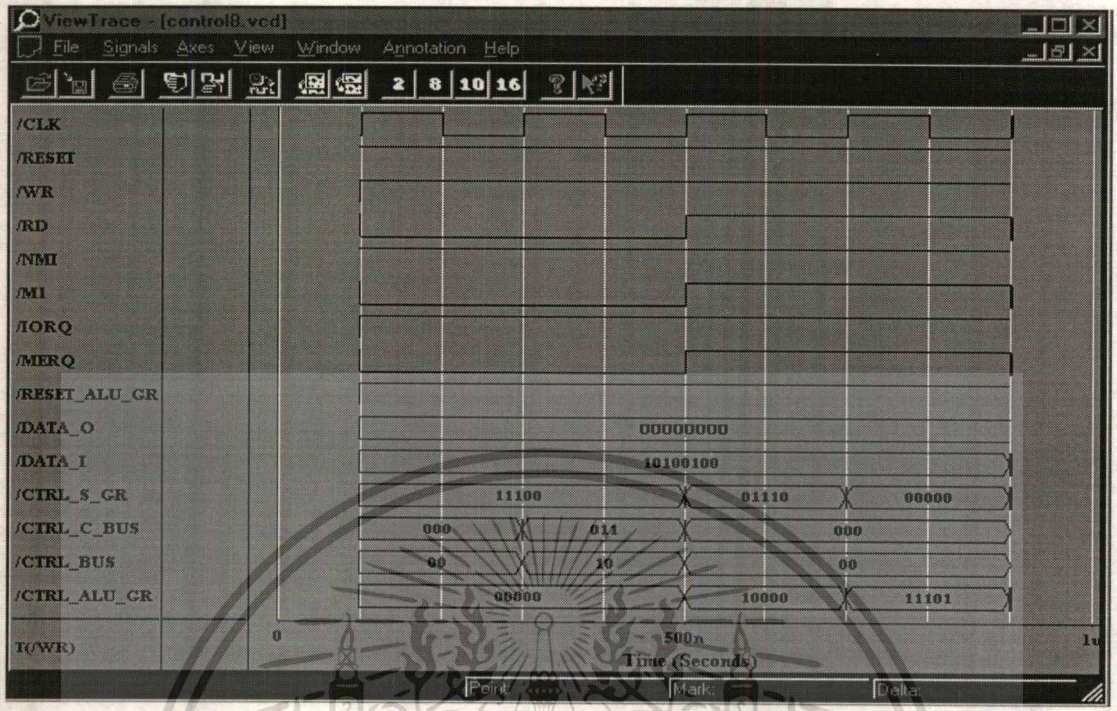
รูปที่ 4.5 สัญญาณควบคุมของคำสั่ง LD H,26H



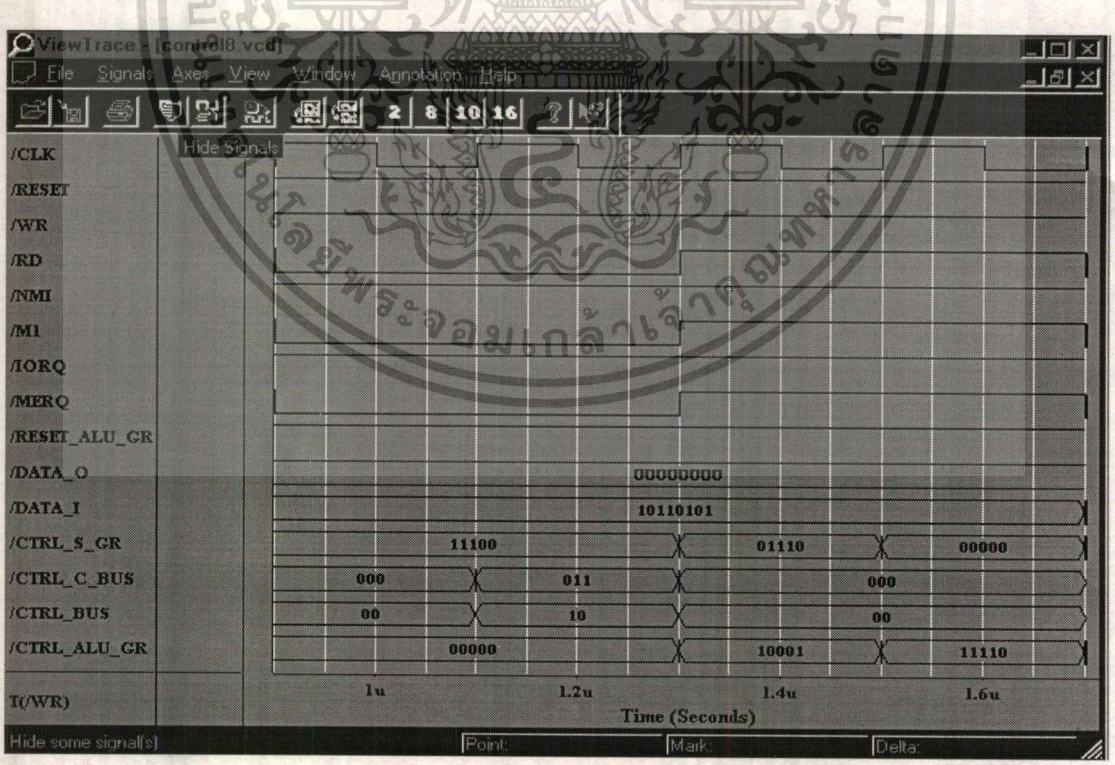
รูปที่ 4.6 สัญญาณควบคุมของคำสั่ง LD A,H



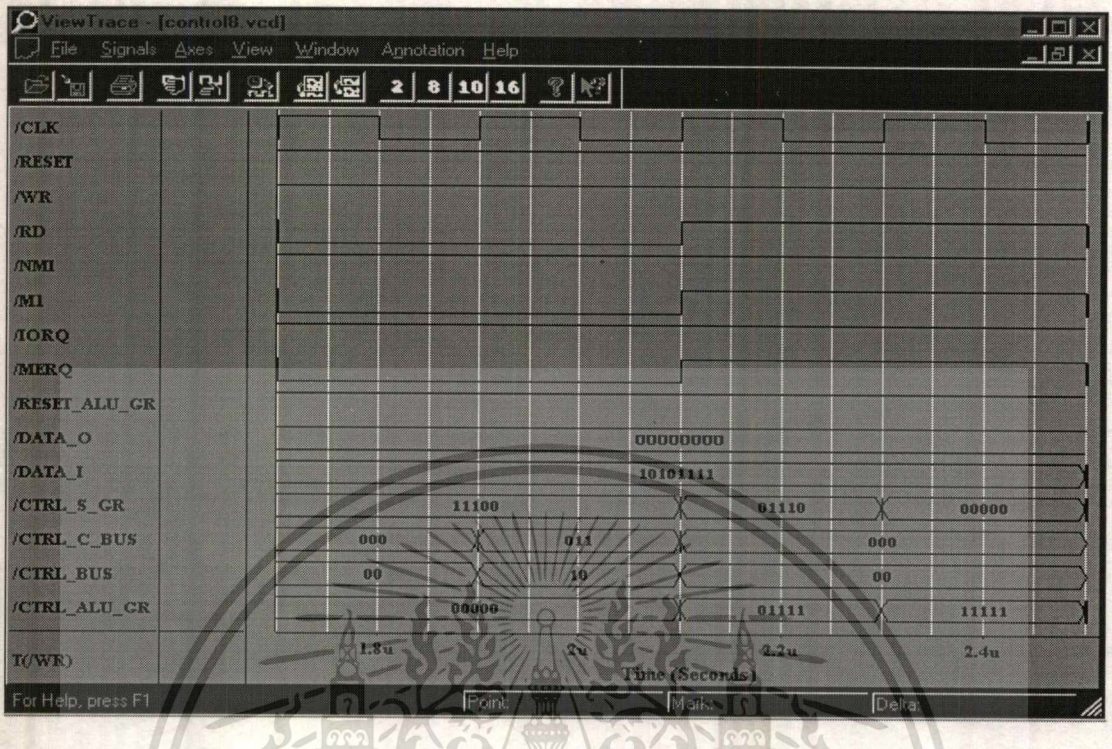
รูปที่ 4.7 สัญญาณควบคุมของคำสั่ง LD IX,DD21H



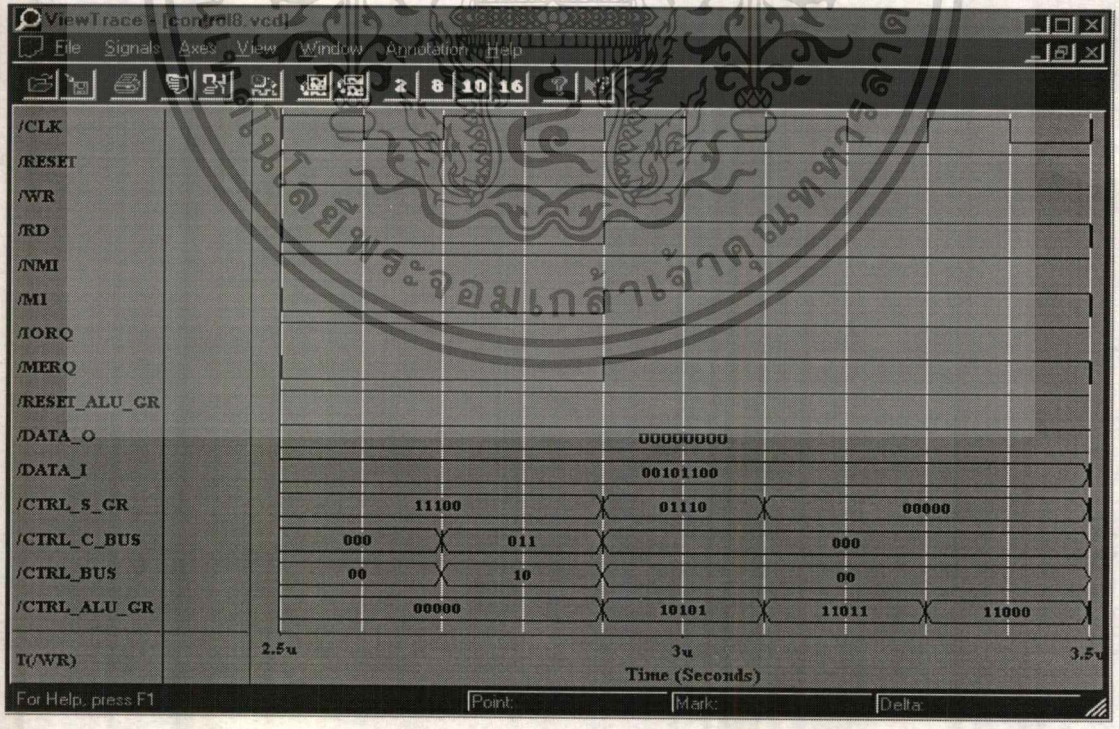
รูปที่ 4.8 สัญญาณควบคุมของคำสั่ง AND A,H



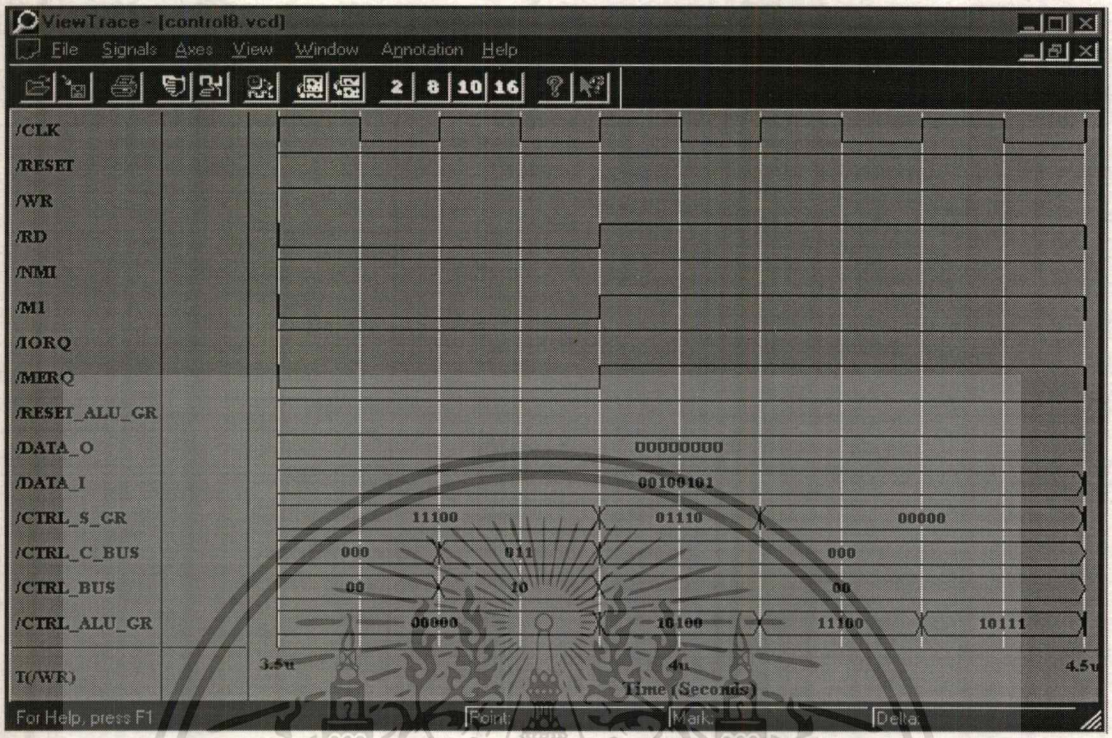
รูปที่ 4.9 สัญญาณควบคุมของคำสั่ง OR A,L



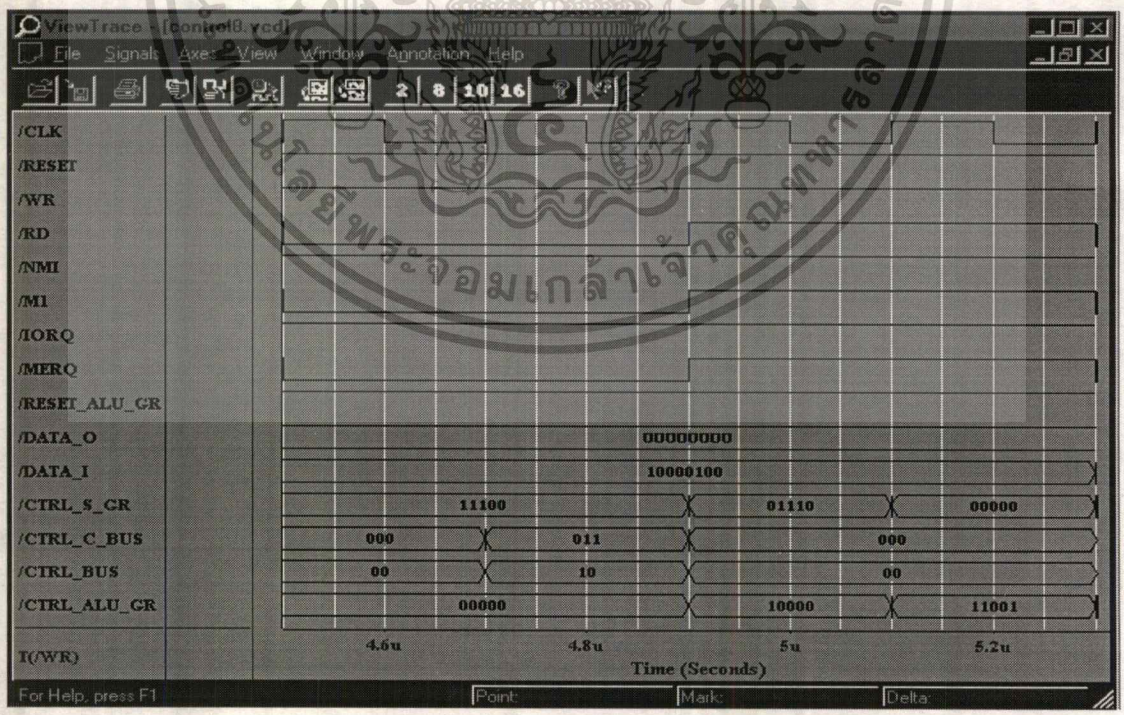
รูปที่ 4.10 สัญญาณควบคุมของคำสั่ง XOR A,A



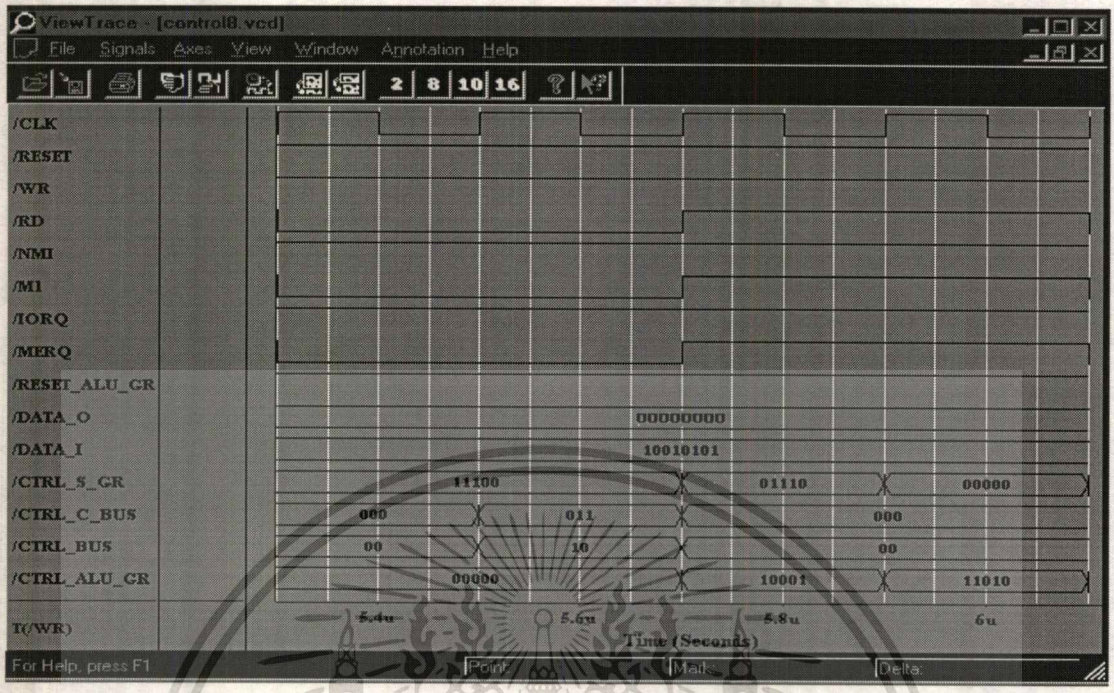
รูปที่ 4.11 สัญญาณควบคุมของคำสั่ง INC L



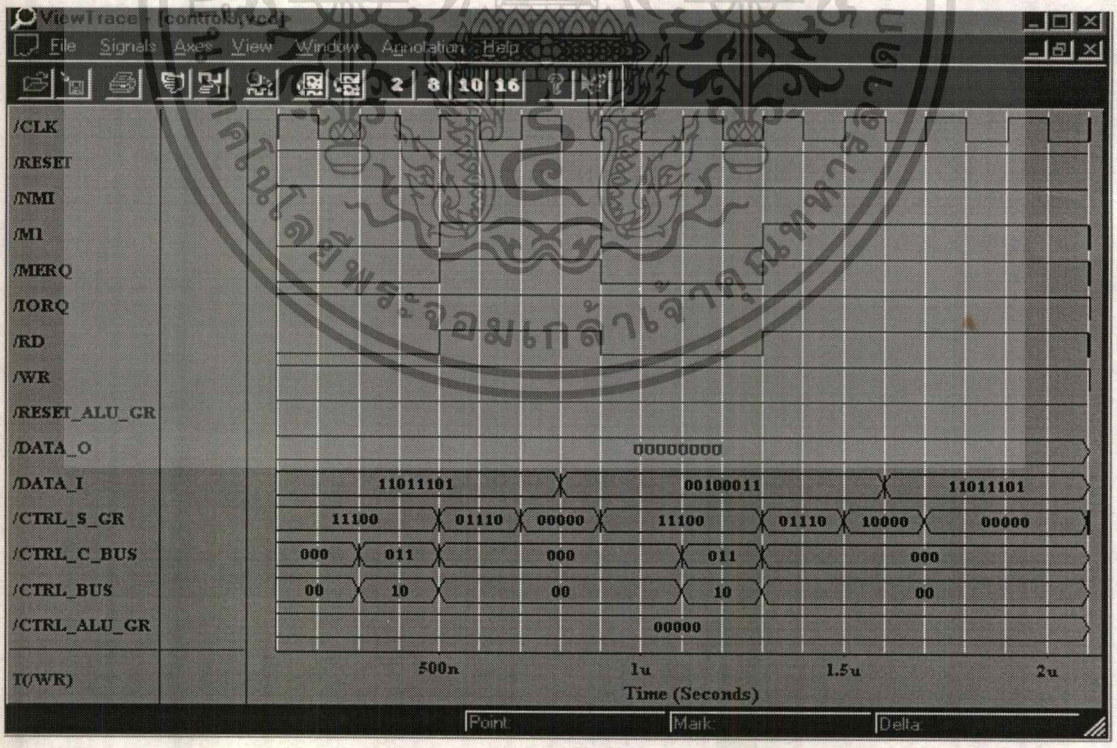
รูปที่ 4.12 สัญญาณควบคุมของคำสั่ง DEC H



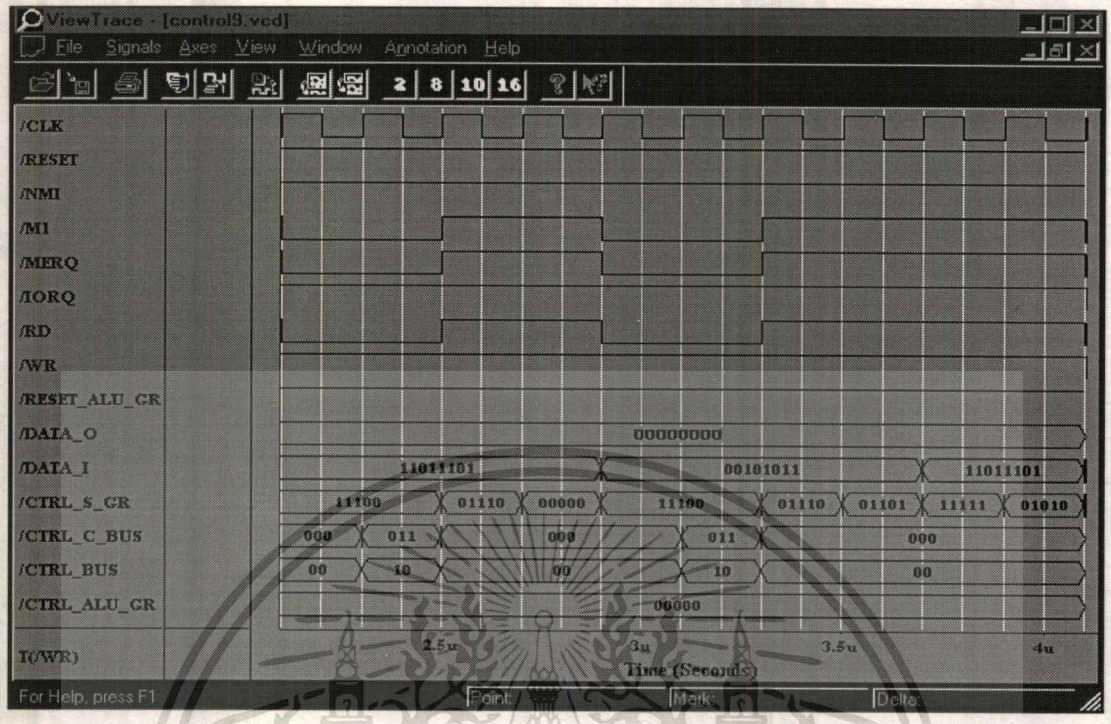
รูปที่ 4.13 สัญญาณควบคุมของคำสั่ง ADD A,H



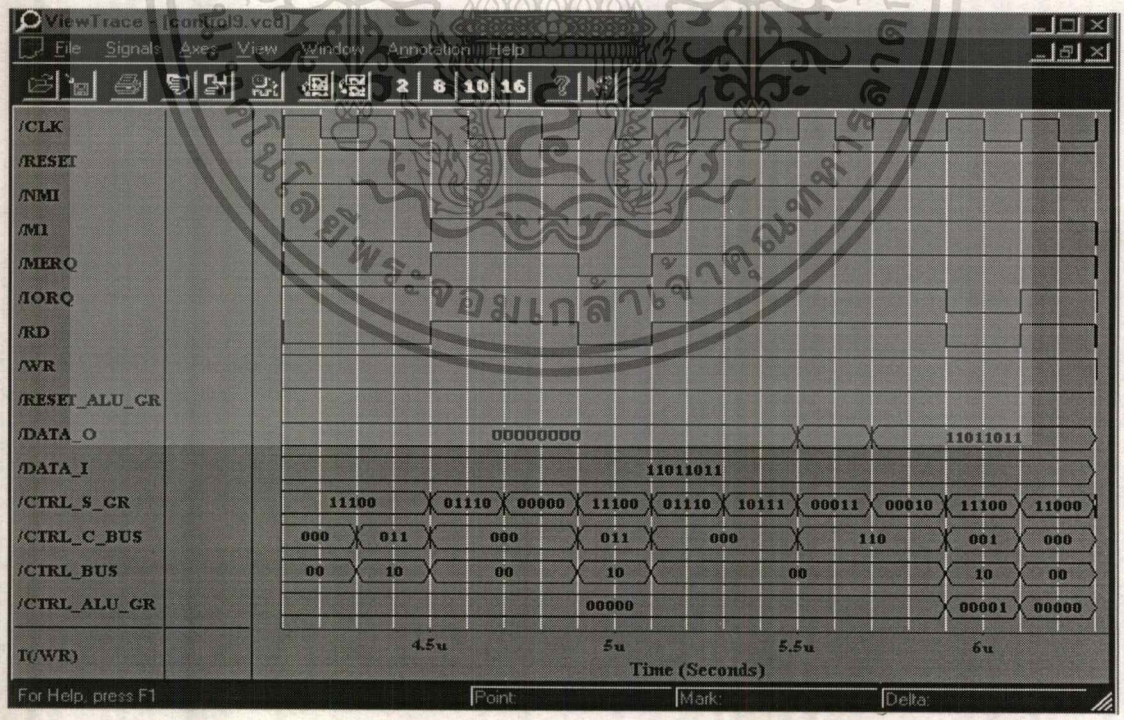
รูปที่ 4.14 สัญญาณควบคุมของคำสั่ง SUB L



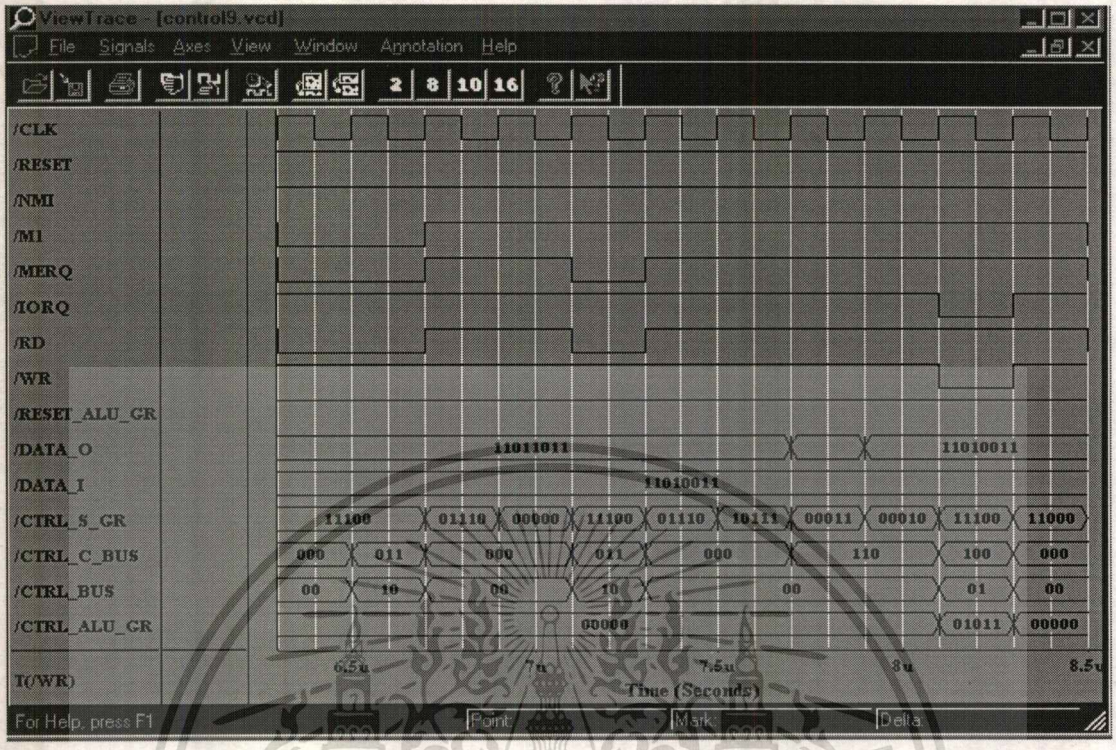
รูปที่ 4.15 สัญญาณควบคุมของคำสั่ง INC IX



รูปที่ 4.16 สัญญาณควบคุมของคำสั่ง DEC IX

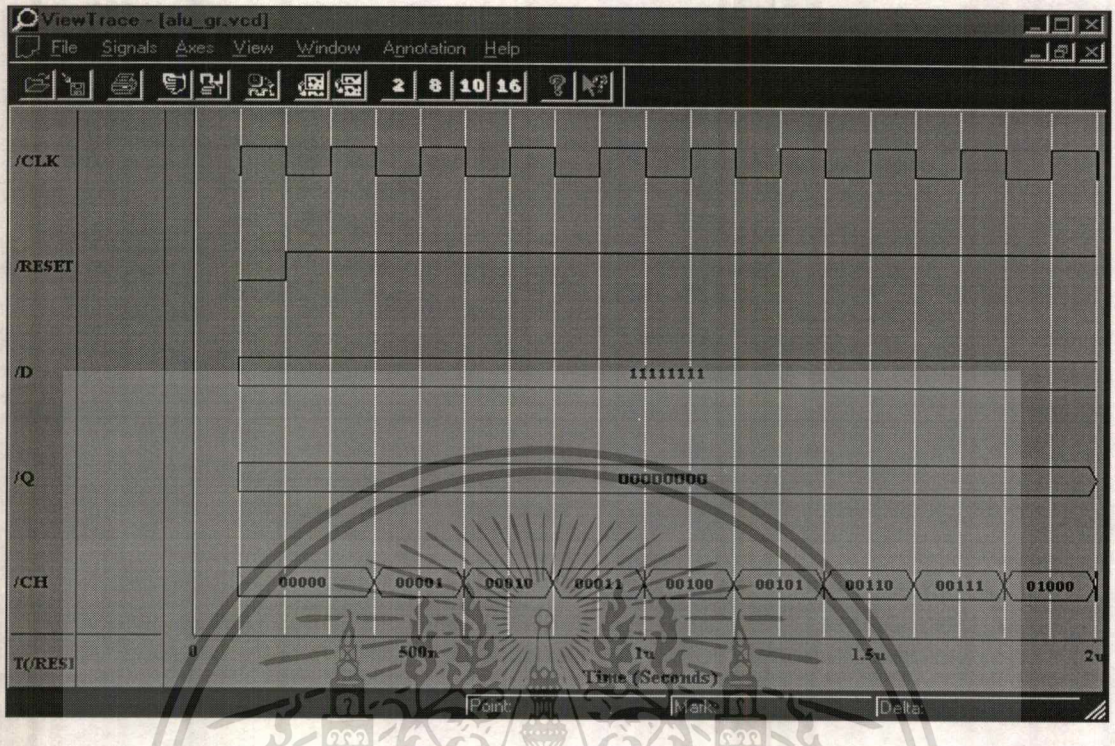


รูปที่ 4.17 สัญญาณควบคุมของคำสั่ง IN A,(m)

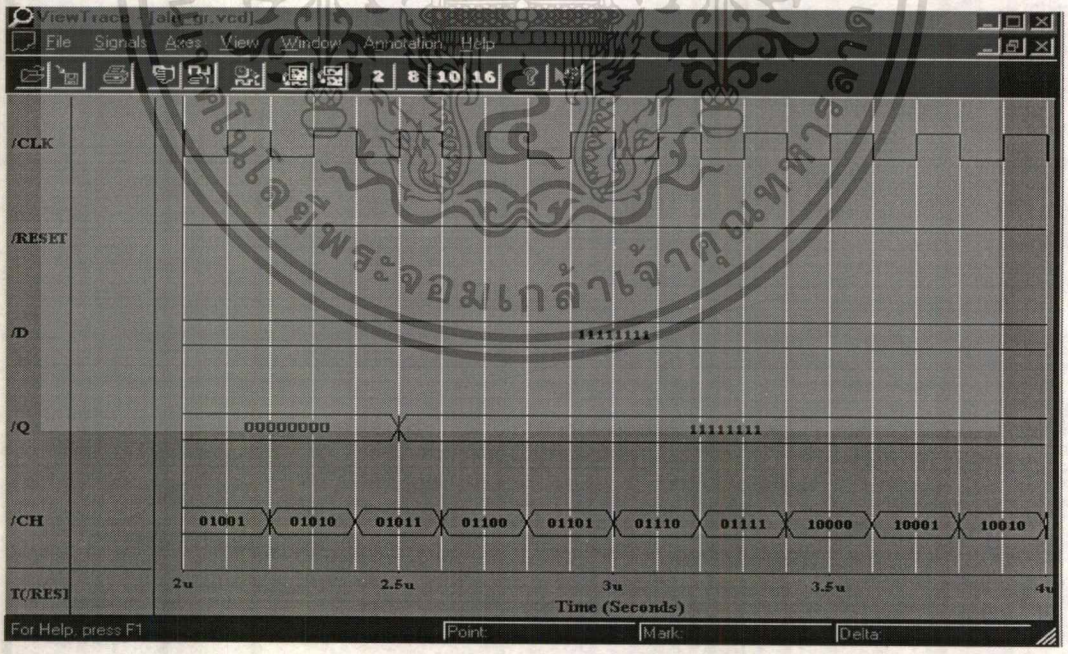


รูปที่ 4.18 สัญญาณควบคุมของคำสั่ง OUT (m),A

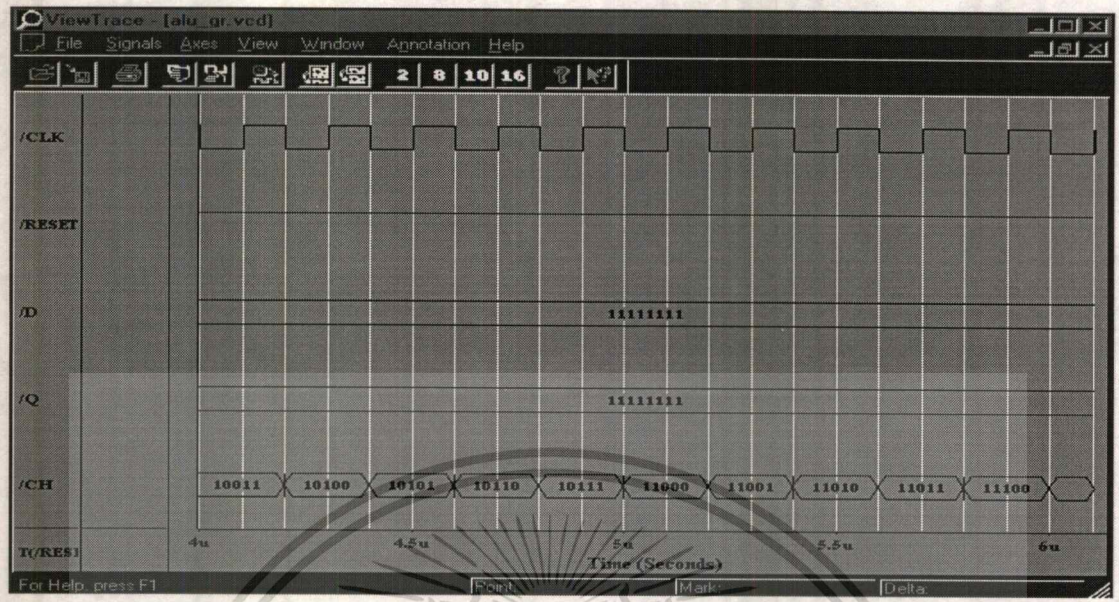
ขั้นตอนที่ 2 ได้ทำการทดสอบหน่วย ALU_GR โดยการป้อนสัญญาณ Data = "11111111", CH = "00000-11111"และ CLK ซึ่งในส่วนของ ALU_GR จะทำการประมวลผลทางคณิตศาสตร์และลอจิก ซึ่งมีคำสั่ง ADD, SUB, INC, DEC, OR, AND, XOR นอกจากนี้ยังมีวงจรรีจิสเตอร์ A,F,H,L อยู่ภายในอีกด้วย ผลที่ได้จากการทดสอบเป็นไปตามโปรแกรมที่ได้เขียนขึ้นดังรูปที่ 4.19



รูปที่ 4.19 การทดสอบ ALU_GR



รูปที่ 4.19 (ต่อ) การทดสอบ ALU_GR



รูปที่ 4.19 (ต่อ) การทดสอบ ALU_GR

ขั้นตอนที่ 3 ได้ทำการทดสอบ FPGAs-1 ซึ่งภายในประกอบด้วย วงจรบัส (BUS) หน่วยควบคุมบัส (Control_Bus) และหน่วยพักข้อมูลพิเศษ (S_pacail_Bus) ซึ่งในการตรวจสอบข้อผิดพลาดจะทำได้โดยการทดสอบการทำงานทีละส่วนไม่สามารถทำการพอร์ตแมพได้ ซึ่งได้กล่าวในบทที่ 5 ในส่วนของปัญหาแล้ว

4.3 ผลการ Simulate หน่วยประมวลผลกลางขนาด 8 บิต

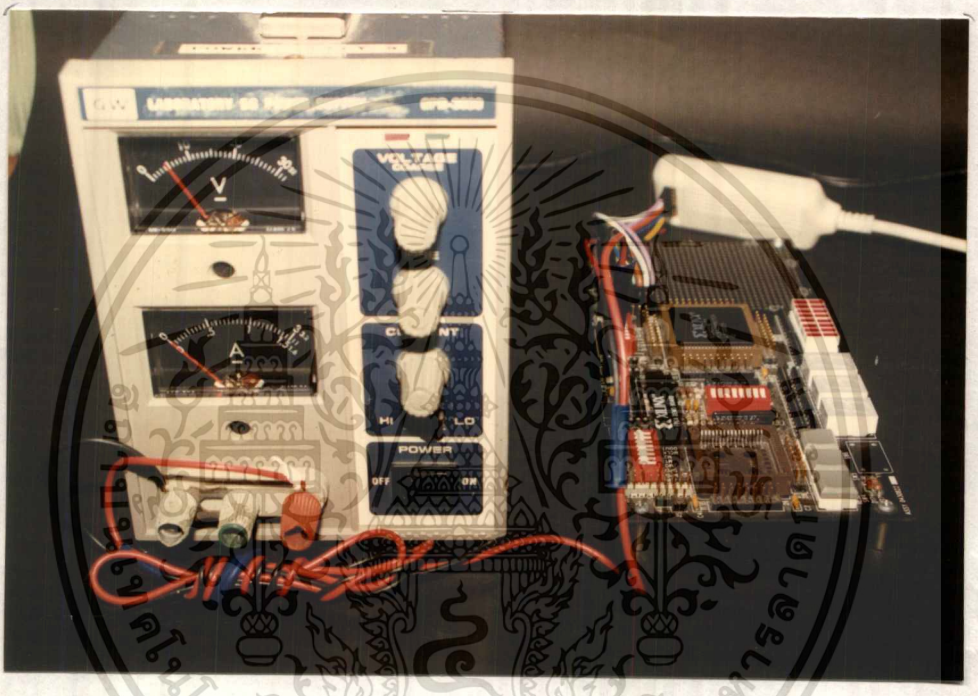
จากการที่ได้ทดสอบในส่วนต่าง ๆ ที่ได้ออกแบบมาด้วยการจำลองการทำงาน (Simulate) ผลการทดสอบเป็นไปตามที่ได้ออกแบบไว้ ซึ่งขั้นตอนทดสอบในส่วนนี้กระทำได้โดยนำผลการจำลองการทำงานทั้งหมดมาเปรียบเทียบกันแทนที่จะใช้วิธีการพอร์ตแมพเพราะเนื่องจากประสบปัญหาซึ่งได้กล่าวไว้ในบทที่ 5 ในส่วนของปัญหา

4.4 ผลการทดลองของวงจรอินเตอร์เฟส

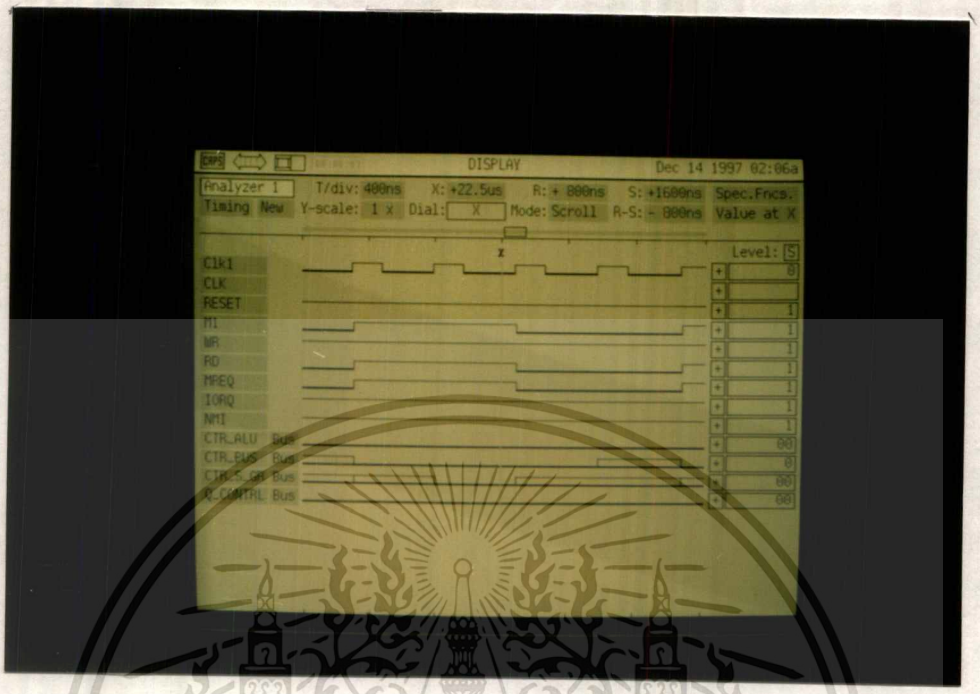
เนื่องจากหน่วยประมวลผลกลางขนาด 8 บิทที่ได้ออกแบบนั้นมีส่วนคล้ายกับหน่วยประมวลผลกลางเบอร์ Z-80 ทางคณะผู้จัดทำจึงได้ นำวงจรอินเตอร์เฟสที่ได้ออกแบบมานั้นไปทดลอง ET-Board ของหน่วยประมวลผลกลางเบอร์ Z-80 ซึ่งจากผลการทดลองเป็นไปตาม

ที่ได้ออกแบบไว้ทุกประการ ไม่ว่าจะเป็นส่วนสัญญาณนาฬิกาของระบบ ส่วนติดต่อกับหน่วยความจำ ส่วนติดต่อการใช้งานของพอร์ตอินพุทและเอาท์พุท สามารถทำงานได้ตามที่ได้ทำการออกแบบไว้

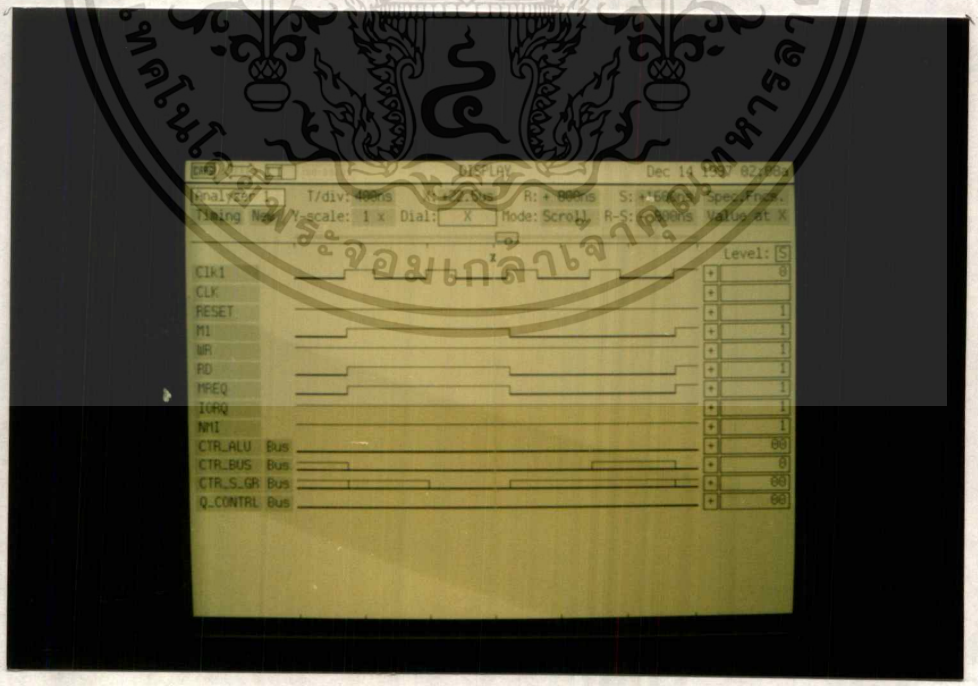
สำหรับภาพถ่ายแสดงการทำงานของหน่วยประมวลผลกลางขนาด 8 บิท ที่ต่อร่วมกับส่วนของวงจรอินเตอร์เฟส แสดงไว้ดังรูปที่ 4.20



รูปที่ 4.20 ภาพถ่ายการเชื่อมต่อสายดาวน์โหลดเคเบิลกับบอร์ดตัวอย่าง FPGAs

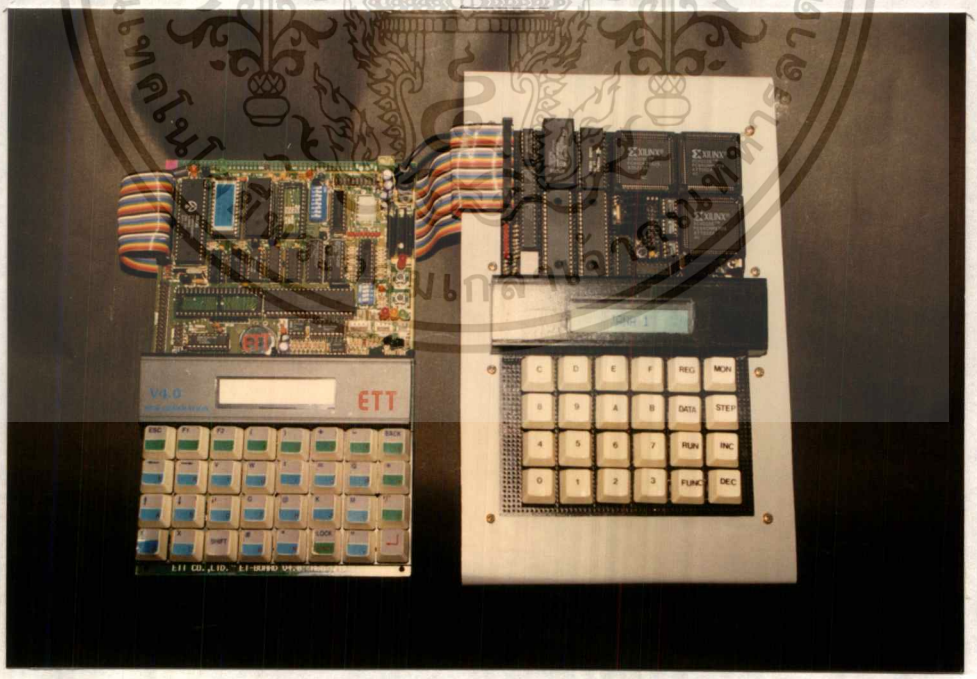


รูปที่ 4.21 ภาพถ่ายการวัดสัญญาณควบคุมที่ขาต่างๆ ของ FPGAs



รูปที่ 4.21 (ต่อ) ภาพถ่ายการวัดสัญญาณควบคุมที่ขาต่างๆ ของ FPGAs

รูปที่ 4.22 ภาพถ่ายแสดงการโปรแกรม SPROM



รูปที่ 4.23 ภาพถ่ายแสดงการทดสอบ DEMO บอร์ด ต่อร่วมกับ ET-Board

บทที่ 5
สรุปผลอภิปรายข้อเสนอแนะ

5.1 สรุป

หน่วยประมวลผลกลางขนาด 8 บิตที่สร้างขึ้นซึ่งทำการออกแบบด้วยภาษา VHDL ซึ่งเป็นภาษาที่ใช้ในการบรรยายการทำงานทางฮาร์ดแวร์ของหน่วยประมวลผลกลางขนาด 8 บิท โดยภาษา VHDL นั้นมีลักษณะคล้ายกับภาษาคอมพิวเตอร์ทั่วไป ทำให้ง่ายต่อความเข้าใจและการแก้ไข เมื่อทำการเขียนโปรแกรมภาษา VHDL เพื่ออธิบายการทำงานทุกส่วนของหน่วยประมวลผลกลางขนาด 8 บิทเสร็จเรียบร้อยแล้ว ขั้นต่อมาคือการนำโปรแกรมไปจำลองการทำงาน เพื่อดูการทำงานของหน่วยประมวลผลกลางขนาด 8 บิทที่ได้ทำการออกแบบว่ามีการทำงานเป็นไปตามวัตถุประสงค์ที่ต้องการหรือไม่ ถ้าผลการจำลองการทำงานถูกต้องเป็นไปตามวัตถุประสงค์ที่ต้องการแล้ว จึงนำโปรแกรมไปสังเคราะห์เป็นวงจรระดับเกทและแปลงไฟล์ที่สังเคราะห์ได้เป็นไฟล์บิทสตีมโดยใช้ซอฟแวร์ XACT Development Tool เพื่อทำการดาวน์โหลดลงบนบอร์ดตัวอย่างของ FPGAs ผ่านทางดาวน์โหลดเคเบิล

การทดสอบการทำงานของหน่วยประมวลผลกลางขนาด 8 บิท หลังจากที่ได้ทำการจำลองการทำงานแล้ว ปรากฏว่าหน่วยประมวลผลกลางขนาด 8 บิทสามารถทำงานได้ถูกต้องตามต้องการที่ออกแบบไว้ เพียงแต่ในส่วนของคำสั่งได้มีการลดคำสั่งที่ไม่จำเป็นออกบ้าง โดยหน่วยประมวลผลกลางที่ได้ทำการสร้างขึ้นนี้มีขอบเขตการทำงานตามที่ได้ออกแบบไว้ดังนี้

1. ใช้สัญญาณนาฬิกา 2 MHz เป็นฐานเวลาในการทำงาน
2. สามารถต่อหน่วยความจำภายนอกได้ 65,536 ตำแหน่ง (64 K)
3. สามารถรับข้อมูลได้ครั้งละ 8 บิท
4. มีคำสั่งในการทำงานทางคณิตศาสตร์และลอจิกเช่น ADD,SUB,DEC,INC,XOR, NOR, OR เป็นต้น
5. สามารถใช้หน่วยความจำภายนอกเพื่อใช้เป็นสแตค

5.2 ปัญหาและแนวทางแก้ไข

ในการจัดทำโครงงานนี้ สามารถสรุปปัญหาที่เกิดขึ้นระหว่างทำโครงงานได้ดังนี้

1. การลงโปรแกรม XACT กระทำได้ยากและไม่สามารถใช้โปรแกรมในการดาวน์โหลดลงบนอุปกรณ์ FPGAs ได้

แนวทางการแก้ไข ประการแรกได้ทำการลงโปรแกรมหลาย ๆ ครั้งและได้ทำการเปิดคู่มือในการลงโปรแกรม ประการที่สองได้ทำการขอใช้เครื่องคอมพิวเตอร์ที่ได้ทำการลงโปรแกรมเรียบร้อยแล้วและสอบถามขั้นตอนการลงโปรแกรม

2. ซอฟแวร์ของบริษัท Viewlogic เวอร์ชัน 7.20 ที่ได้ทำการออกแบบนั้นเปราะบางเสียได้ง่าย

แนวทางแก้ไข ประการแรกได้ทำการสันนิษฐานว่าเกิดจากการจัดฮาร์ดดิสไม่เป็นระเบียบก็ได้ทำการแก้ไขโดยจัดฮาร์ดดิสสำหรับซอฟแวร์ของบริษัท Viewlogic เวอร์ชัน 7.20 เพียงโปรแกรมเดียว ประการที่สองสันนิษฐานว่าเกิดจากซอฟแวร์ของโปรแกรม WINDOWS 95 และส่วนฮาร์ดแวร์ของเครื่องมีปัญหาจึงได้เปลี่ยนเครื่องคอมพิวเตอร์ที่ใช้ ซอฟแวร์ตัวนี้

3. มีความยากลำบากในการเขียน เพราะจากคู่มือเดิมที่มีอยู่จะบรรยายแบบฮาร์ดแวร์ แต่ถ้าหากเขียนโปรแกรมขนาดใหญ่จะทำให้เกิดความซับซ้อนในการเขียนโปรแกรมมาก

แนวทางการแก้ไข พยายามศึกษาการใช้งานของซอฟแวร์จากเอกสารการใช้โปรแกรมที่เมนู Help และคู่มือการใช้โปรแกรมของวิวลอจิกเวอร์ชั่นเก่าที่มีลักษณะการเขียนโปรแกรมภาษา VHDL ในลักษณะการบรรยายเชิงการทำงานให้มากขึ้น

4. โปรแกรมภาษา VHDL ของหน่วยคำนวนคณิตศาสตร์และลอจิก จะต้องประกาศใช้ไลบรารี่ synth; และเรียกใช้ไลบารี synth.Vhdlsynth.all ไม่สามารถจำลองการทำงานได้

แนวทางแก้ไข มีอยู่ 2 ประการคือ ประการแรกทำการแก้ไขโดยใช้แนวทางของปริญญานิพนธ์การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL และการใช้วงจรเชิงเลขโดยใช้อุปกรณ์ FPGAs และแนวทางที่สองคือการเปลี่ยนวิธีการเขียนโปรแกรมใหม่ คือ หลีกเลี่ยงการเรียกใช้ไลบรารี synth; และไลบรารี synth.vhdlsyth.all แต่ให้พยายามเขียนโปรแกรมภาษา VHDL ออกมาในลักษณะบรรยายทางฮาร์ดแวร์ให้มากขึ้น

5. ไม่สามารถดาวน์โหลดโปรแกรมที่เขียนขึ้นมาลงบนอุปกรณ์ FPGAs ได้ เพราะไม่สามารถเขียนโปรแกรมภาษา VHDL ซึ่งใช้การบรรยายแบบโครงสร้างเพื่อให้สามารถเรียกใช้งาน D ฟลิปฟลอปได้

แนวทางการแก้ไข ได้ทำการค้นคว้าวิธีการเขียนโปรแกรมพบว่าในส่วนของขั้นตอนการเขียนโปรแกรมจะต้องมีคำสั่ง if (clk='1' and clk'event) ครอบคลุมการเขียนโปรแกรมทั้งหมดและต้องปิดท้ายด้วย end if; เสมอจึงจะสามารถเรียกใช้งาน D ฟลิปฟลอปได้

6. เขียนโปรแกรมภาษา VHDL สำเร็จแล้วสามารถจำลองการทำงานได้แต่ไม่สามารถสังเคราะห์โปรแกรม VHDL ให้เป็นวงจรในระดับเกทได้

แนวทางการแก้ไข เมื่อจำลองการทำงานของโปรแกรมเสร็จทุกครั้ง ควรมีการสังเคราะห์โปรแกรมภาษา VHDL ให้เป็นอุปกรณ์ระดับเกททุกครั้ง เมื่อไม่สามารถสังเคราะห์ได้ควรดูเอกสารการใช้โปรแกรมภาษา VHDL ที่เป็นเวอร์ชั่นใกล้เคียง เพราะการใช้ไวยากรณ์อาจจะมีข้อผิดพลาด

7. ไม่สามารถออกแบบสร้างหน่วยประมวลผลกลางได้ทั้งนี้เนื่องจากอุปกรณ์ FPGAs เบอร์ XC4010E มีโครงสร้างจำนวนเกทภายในไม่เพียงพอที่จะดาวน์โหลดหน่วยประมวลผลกลางที่มีบัสภายในขนาด 8 บิทลงไปได้

แนวทางการแก้ไข ทำการแบ่งโปรแกรมภาษา VHDL ออกเป็นส่วน ๆ ให้มีขนาดพอดีกับอุปกรณ์ FPGAs เบอร์ XC4010E ในแต่ละชิป จากนั้นจึงนำอุปกรณ์ FPGAs เบอร์ XC4010E นำมาเชื่อมโยงกันภายนอกอุปกรณ์ FPGAs

8. การนำโปรแกรมภาษา VHDL แต่ละโปรแกรมมาพอร์ตแมพรวมกัน ไม่สามารถจำลองการทำงานของหน่วยประมวลผลกลางได้

แนวทางแก้ไข ทำการจำลองการทำงานในแต่ละหน่วย ทีละหน่วย โดยคำนึงถึงการป้อนสัญญาณอินพุทในการจำลองการทำงาน เสร็จแล้วจึงนำอินพุทและเอาท์พุทแต่ละหน่วยมาเปรียบเทียบว่าถูกต้องตามคาบเวลาหรือไม่ ส่วนอีกวิธีหนึ่งคือ นำโปรแกรมภาษา VHDL มาแปลงเป็นวงจรระดับเกทลงอุปกรณ์ FPGAs เบอร์ 4010E ทดลองป้อนสัญญาณอินพุทและตรวจสอบสัญญาณเอาท์พุท

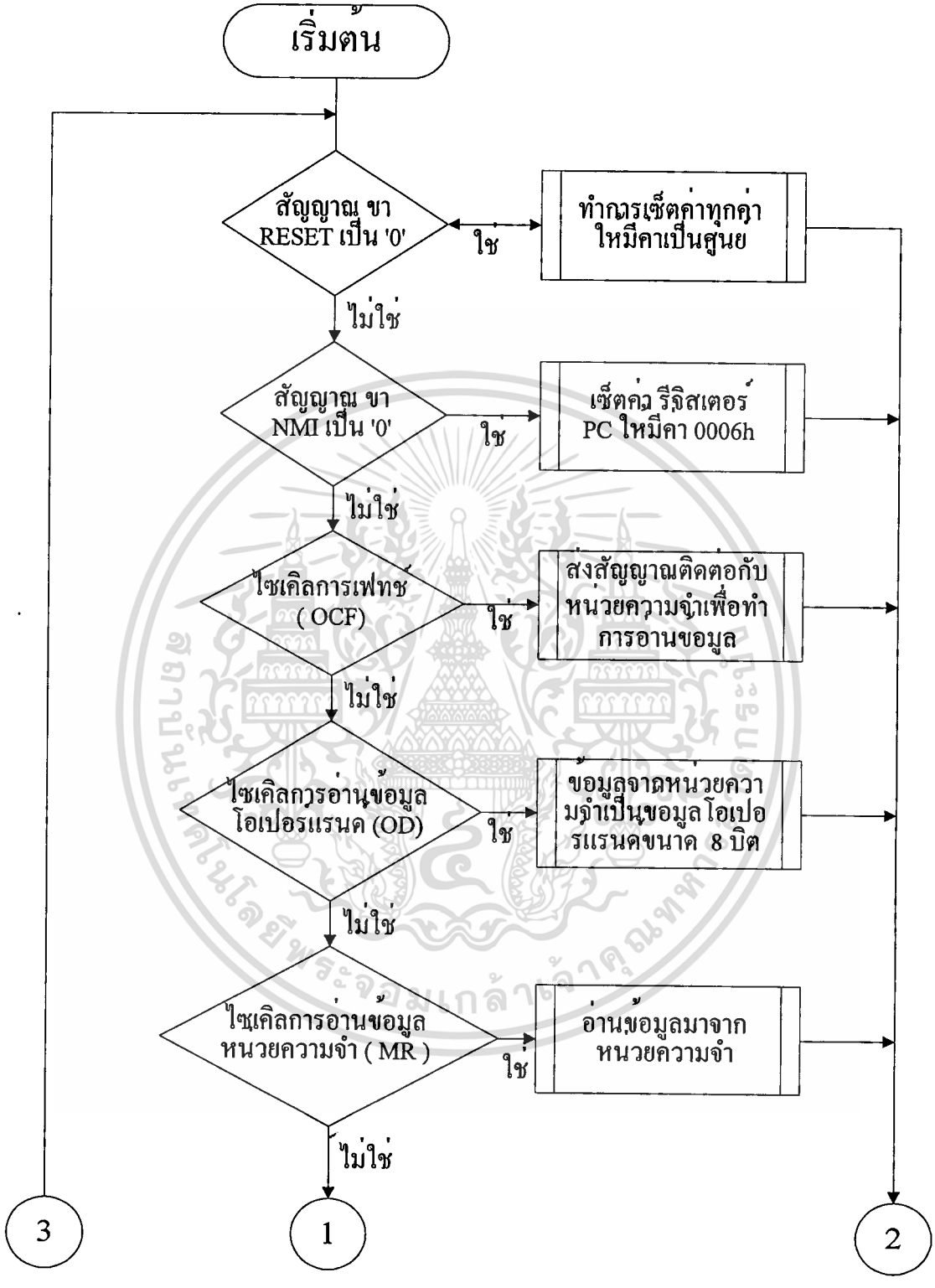
5.3 แนวทางการพัฒนาต่อ

1.การออกแบบวงจรโดยใช้ภาษา VHDL โดยการเรียกใช้ฟังก์ชั่นของภาษา VHDL จะช่วยลดจำนวนเกทในการสังเคราะห์วงจรได้

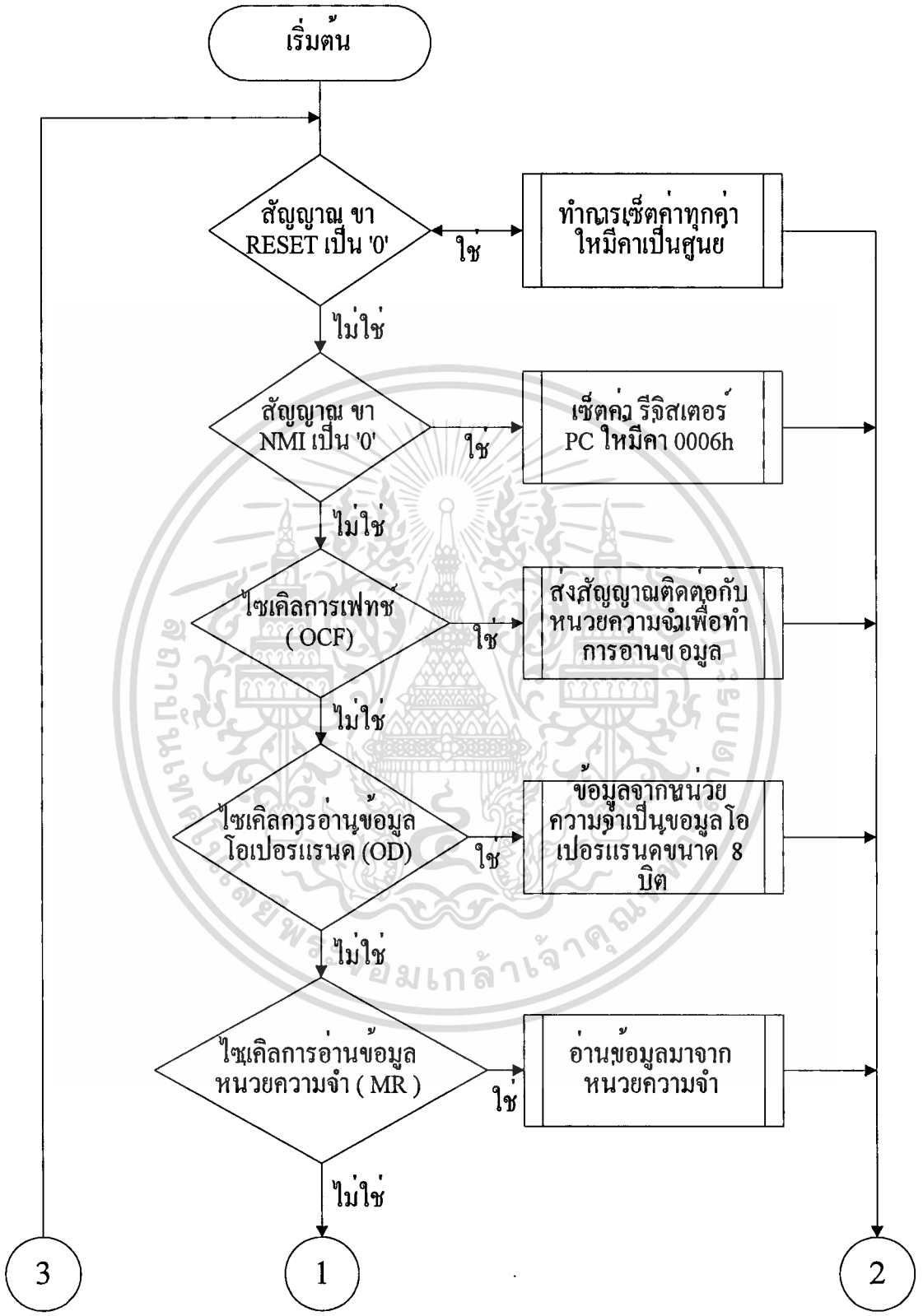
2. สามารถใช้ภาษา VHDL และอุปกรณ์ FPGAs ออกแบบวงจรไปประยุกต์ใช้งานตามความสามารถและความต้องการของผู้ออกแบบแต่ละคนได้อย่างกว้างขวาง

3. ภาษา VHDL เป็นภาษาที่มีลักษณะคล้ายกับภาษาคอมพิวเตอร์ทั่วไป เช่น ภาษาซีหรือภาษาปาสคาล ตัวภาษา VHDL ทำความเข้าใจและแก้ไขได้ง่าย สามารถเขียนอธิบายการทำงานของหน่วยประมวลผลกลางขนาด 8 บิท ได้อย่างกว้างขวาง และมีฟังก์ชั่นการคำนวณตรรกและการทำงานในส่วนของลอจิก จึงสามารถใช้ภาษา VHDL บรรยายการทำงานทางฮาร์ดแวร์ของหน่วยประมวลผลกลางที่มีเทคโนโลยีใหม่ ๆ ได้
4. สามารถใช้ภาษา VHDL ออกแบบวงจรที่ใช้เฉพาะงานลงบนอุปกรณ์ FPGAs โดยเฉพาะวงจรที่สร้างขึ้นเองไม่ต้องการลอกเลียน
5. หน่วยประมวลผลกลางที่ได้ทำการออกแบบสามารถเพิ่มจำนวนคำสั่งให้มากขึ้นได้หรืออาจจะลดขนาดเป็น 4 บิทแต่ใช้เทคโนโลยีการออกแบบใหม่ๆ ลงไปก็ได้ โดยใช้อุปกรณ์ FPGAs ที่มีจำนวนเกทภายในเพิ่มมากขึ้น และเพิ่มในส่วนของฮาร์ดแวร์

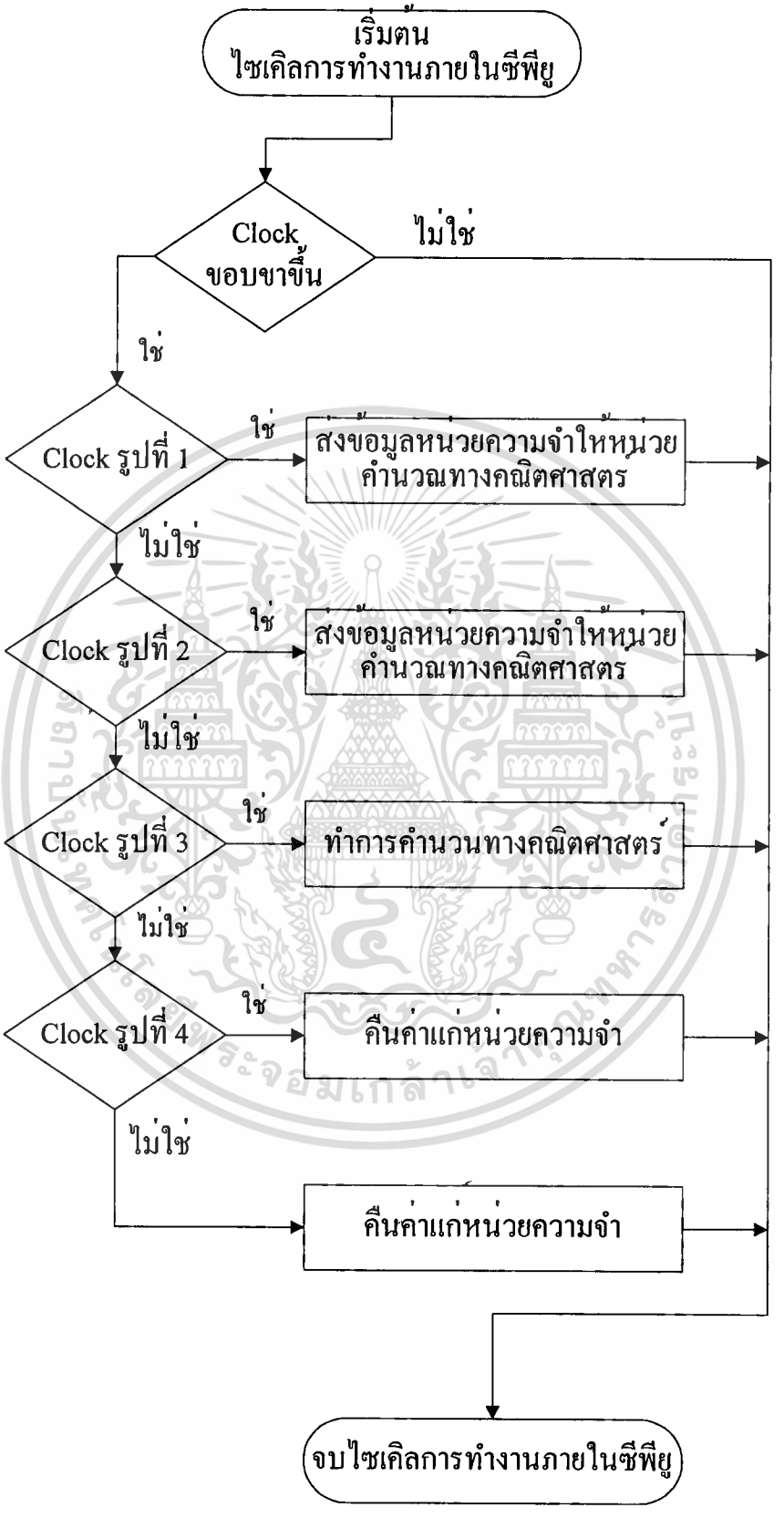
ภาคผนวก ก
ผังงานและโปรแกรมภาษา VHDL ของหน่วยประมวลผลกลางขนาด 8 บิท



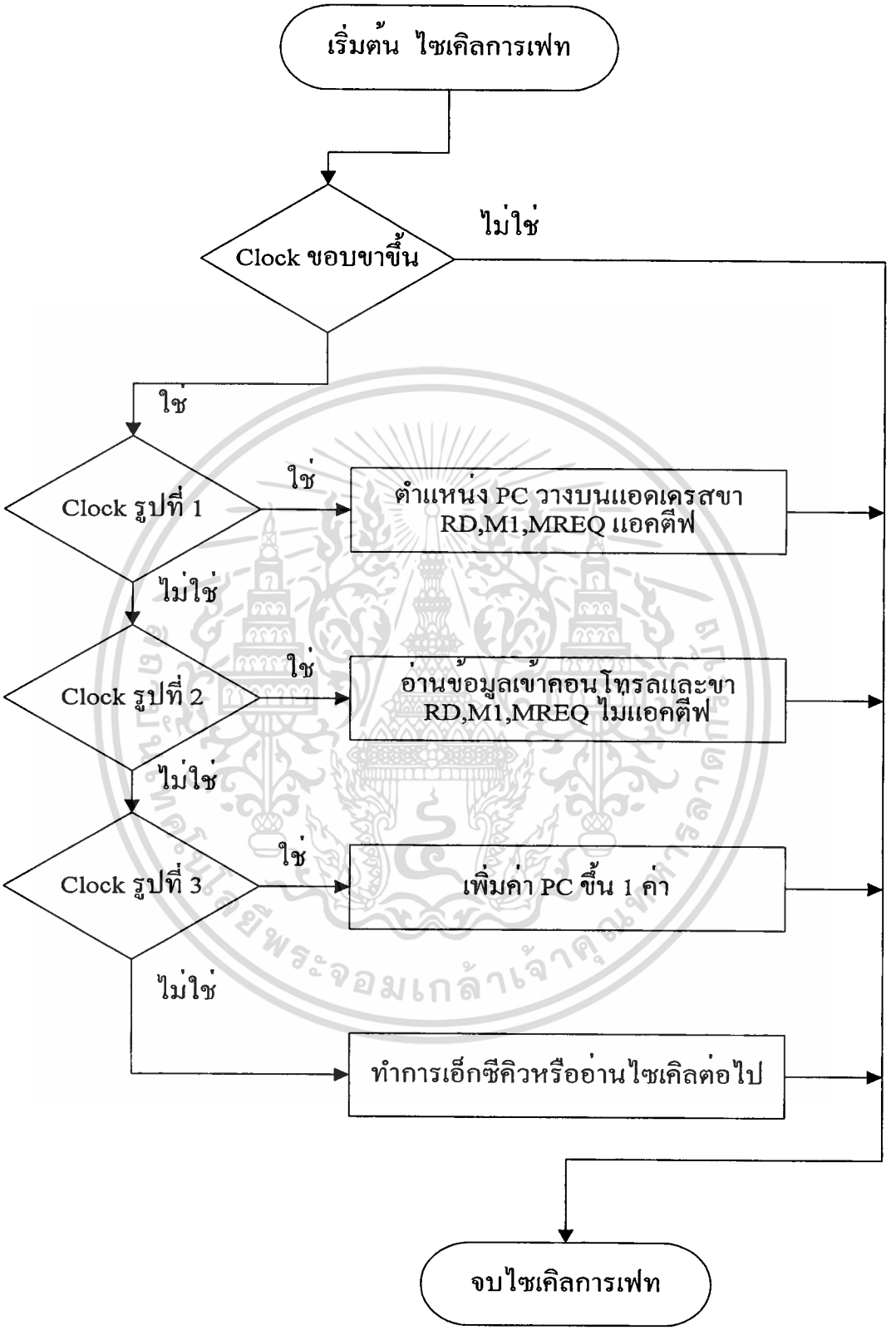
รูปที่ ก.1 ผังงานการทำงานของหน่วยประมวลผลกลางขนาด 8 บิท



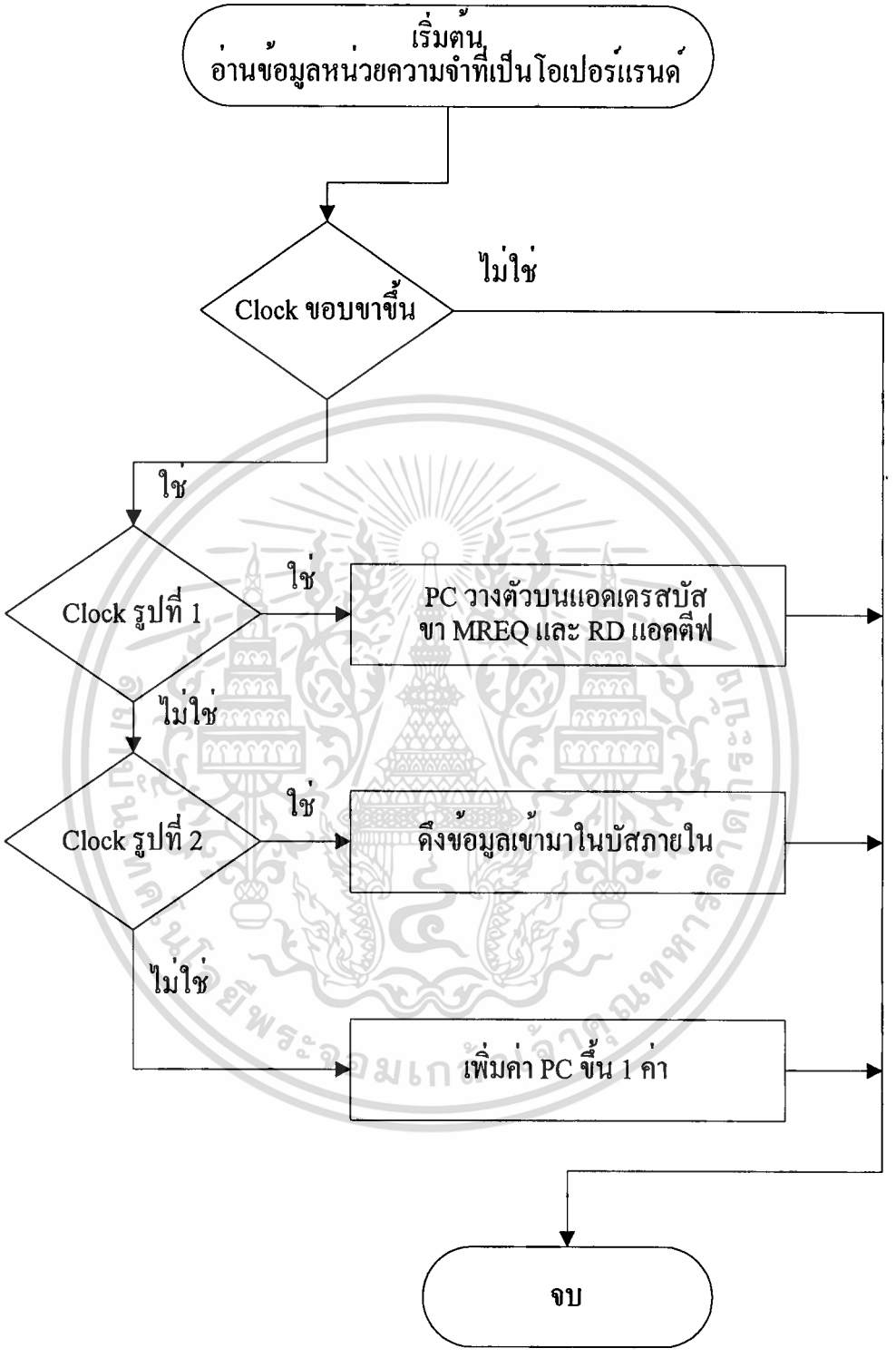
รูปที่ ก.2 (ต่อ) ผังงานการทำงานของหน่วยประมวลผลกลางขนาด 8 บิต



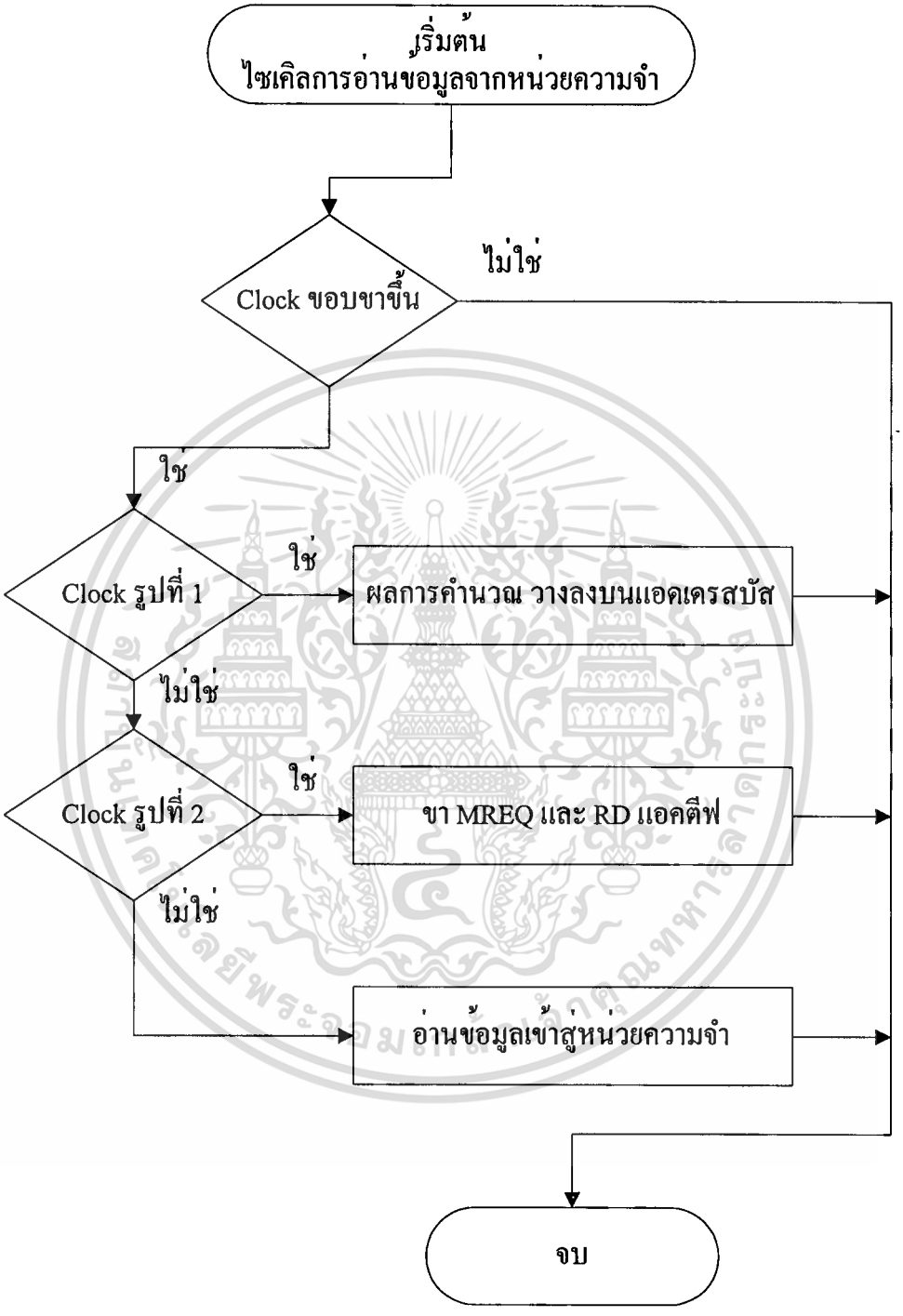
รูปที่ ก.3 ผังงานไซเคิลการทำงานภายในซีพียู



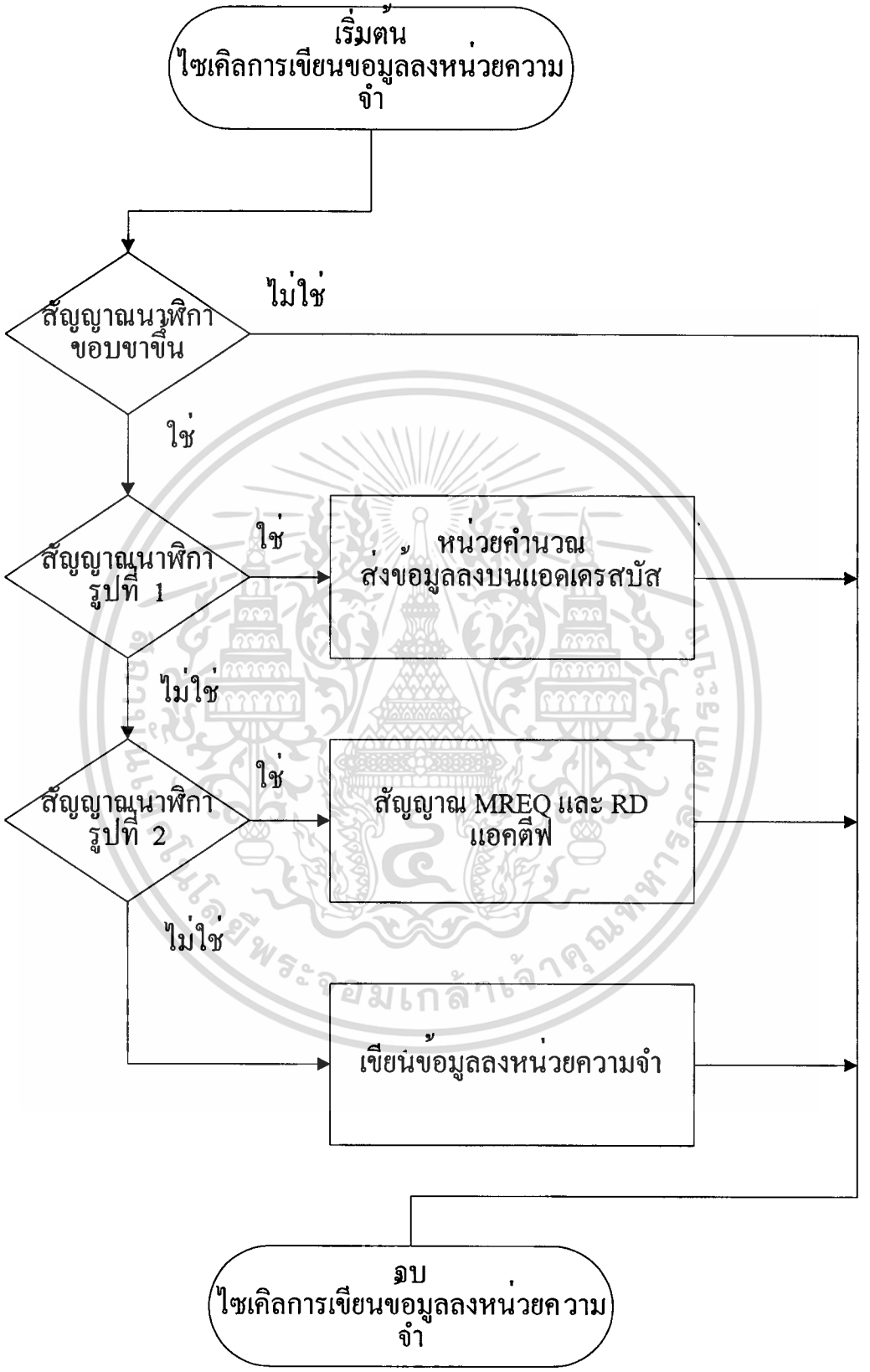
รูปที่ ก.4 ผังงานไซเคิลการเฟท



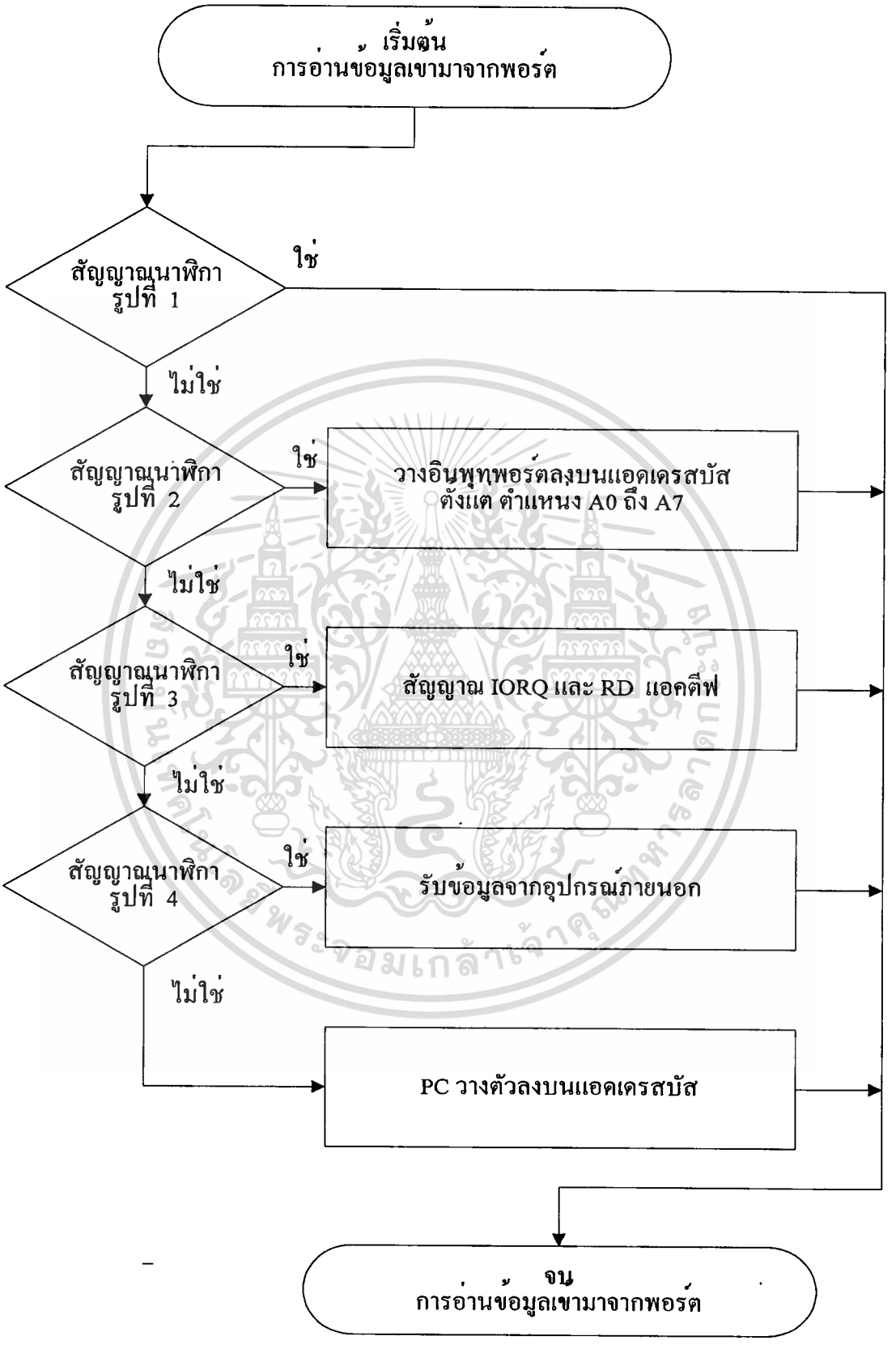
รูปที่ ก.5 ผังงานอ่านข้อมูลหน่วยความจำที่เป็นโอเปอร์แรนด์



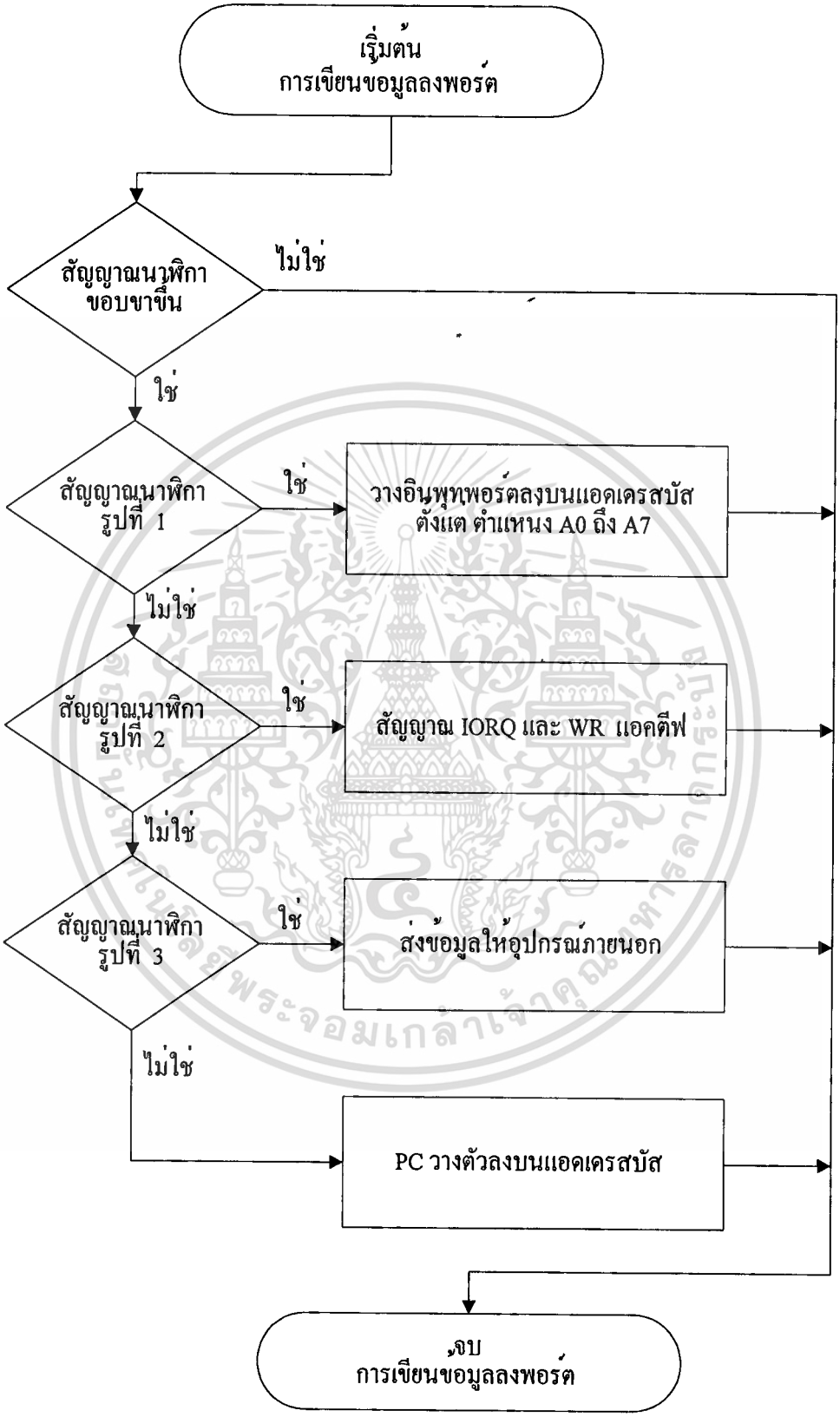
รูปที่ ก.6 ผังงานไซเคิลการอ่านข้อมูลจากหน่วยความจำ



รูปที่ ก.7 ผังงานไซเคิลการเขียนข้อมูลลงหน่วยความจำ



รูปที่ ก.8 ผังงานอ่านข้อมูลเข้ามาจากพอร์ต



รูปที่ ก.9 ผังงานการเขียนข้อมูลลงพอร์ต

```
---------------------------------
-- Latch VHDL Test By Kitipong --
---------------------------------

-- This program have complete block
-- control  27 NOV 97

library ieee;
use ieee.std_logic_1164.all;

--library synth;
--use synth.vhdlsynth.all;


entity control is
port (          clk, reset, nmi   :      in std_logic;
                data_I            :      in std_logic_vector(7 downto 0);
                M1, MERQ, IORQ, RD, WR    :      out std_logic;
                ctrl_alu_gr       :      out std_logic_vector(4 downto 0);
                reset_alu_gr      :      out std_logic;
                ctrl_s_gr         :      out std_logic_vector(4 downto 0);
                ctrl_c_bus        :      out std_logic_vector(2 downto 0);
                ctrl_bus          :      out std_logic_vector(1 downto 0);
                data_o            :      out std_logic_vector( 7 downto 0));
end control;
```

```
architecture a_control of control is
begin
        process(clk)
        variable  data_s, data_s1, sum1   :  std_logic_vector(7 downto 0);
                        variable  count   :  std_logic_vector(4 downto 0) := "00001";
                        variable  c_nmi   :  std_logic_vector(1 downto 0)  := "01";
                        variable  c_load  : std_logic_vector(2 downto 0)  :=  "000";
                        variable c_ix_in_out  :  std_logic_vector( 1 downto 0) :="00";
                        variable DS2      :  std_logic_vector(7 downto 0) :="00000001";
                        variable  c       :  std_logic_vector(8 downto 0);
                        variable  c_jump          : std_logic :='0';

        begin
                if( clk = '1' and  clk'event )then

                        if(reset ='0') then
-- input signal RESET
                                M1 <='1';         MERQ <='1';
                                IORQ <= '1';     RD <= '1';
                                WR <='1';         ctrl_alu_gr   <="00001";
                                reset_alu_gr <='0';        ctrl_s_gr <="00001";

                        elsif( nmi ='0' and  reset ='1')then          -- input signal NMI

                                if( c_nmi = "01" )then
                                        M1 <='1';         MERQ <='1';
                                        IORQ  <= '1';     RD <= '1';
                                        WR <='1';         ctrl_s_gr <="01011";
                                        c_nmi := "10";
```

```
elsif( c_nmi ="10" )then
        ctrl_s_gr <="11010";
        c_nmi := "11";

else
        ctrl_s_gr <="11011";
        c_nmi := "01";
end if;

elsif( reset='1' and  nmi='1') then

    if(count = "00001" )then
        ctrl_alu_gr  <= "00000";        ctrl_bus <="00";
        ctrl_c_bus <= "000";    ctrl_s_gr <="11100";
        reset_alu_gr <='1';       M1 <='0';
        MERQ <='0';    IORQ  <= '1';
        RD <= '0';       WR <='1';
        count := "00010";

elsif(count = "00010" )then
        ctrl_bus  <="10";         ctrl_c_bus <= "011";
        data_s := data_i;                 count := "00011";

elsif(count = "00011") then
        ctrl_bus  <="00";          ctrl_c_bus <= "000";
        M1 <='1';          MERQ <='1';    RD <= '1';
        ctrl_s_gr <="01110";              -- increasement PC
```

```
                    if( data_s(7 downto 6) ="10")then

                        case  data_s(2 downto 0)  is
-- instruction ALU A,A
                        when "111" => ctrl_alu_gr <="01111";
-- instruction ALU A,H
                        when "100" => ctrl_alu_gr <="10000";
-- instruction ALU A,L
                        when "101" => ctrl_alu_gr <="10001";

                        when others => NULL;

                        end case;

                    elsif( data_s(7 downto 6) ="00"
                        and data_s(2 downto 1) ="10" )then

                        case  data_s(5 downto 3)  is
-- instruction INC & DEC A
                        when "111" => ctrl_alu_gr <="10011";
-- instruction INC & DEC H
                        when "100" => ctrl_alu_gr <="10100";
-- instruction INC & DEC L
                        when "101" => ctrl_alu_gr <="10101";

                        when others => NULL;
                        end case;
                end if;
                        count := "00100";
```

```
elsif(count = "00100") then
        ctrl_s_gr <="00000";
        count :="00001";

        if( data_s ="01111100" )then     -- load A with H
         ctrl_alu_gr <="00101";

        elsif( data_s ="01111101" )then -- load A with L
         ctrl_alu_gr <="00110";

        elsif( data_s ="01100111" )then -- load H with A
         ctrl_alu_gr <="00111";

        elsif( data_s ="01100101" )then -- load H with L
         ctrl_alu_gr <="01000";

        elsif( data_s ="01101111" )then -- load L with A
         ctrl_alu_gr <="01001";

        elsif( data_s ="01101100" )then -- load L with H
         ctrl_alu_gr <="01010";

        elsif( data_s ="00111110" )then
-- load A with DATA
         c_load := "001"; count :="00101";

        elsif( data_s ="00100110" )then
```

```
-- load H with DATA
                                    c_load := "010"; count :="00101";

                                elsif( data_s ="00101110" )then
-- load L with DATA
                                    c_load := "011"; count :="00101";

                                elsif( data_s ="11011101" )then
-- instruction LD IX,mn 1
                                    c_load :="100";      c_ix_in_out := "01";
-- instruction LD IX,mn 2
                                elsif( data_s ="00100001" )then
                                    if( c_load ="100" )then
                                        c_load :="101"; count :="01000";
                                        c_ix_in_out := "00";
                                    end if;
-- instruction LD A,(mn)
                                elsif( data_s ="00111010" )then
                                    c_load :="111";      count :="01000";
                                    ctrl_s_gr <="10111";
-- instruction INC & DEC
-- instruction INC IX
                                elsif( data_s ="00100011" ) then
                                    if( c_ix_in_out ="01" )then
                                        c_ix_in_out :="10";
                                        ctrl_s_gr <="10000";
                                        c_load :="000";
                                        count :="01111";
                                    end if;
```

```
-- instruction DEC IX
                                elsif( data_s ="00101011" ) then
                                        if( c_ix_in_out ="01" )then
                                                c_ix_in_out :="11";
                                                ctrl_s_gr <="01101";
                                                c_load :="000";
                                                count :="01111";
                                        end if;
-- instruction IN A,(n)
                                elsif( data_s ="11011011" ) then
                                c_ix_in_out :="10";    count := "00101";
 -- instruction OUT (n),A
                                elsif( data_s ="11010011" ) then
                                        c_ix_in_out :="11";        count :=
"00101";
-- instruction  JR C,e
                                elsif( data_s ="00111000" )then
                                        c_load :="100";          count := "00101";
-- instruction  JR NC,e
                                elsif( data_s ="00110000" )then
                                        c_load :="101";          count := "00101";
-- instruction  JR Z,e
                                elsif( data_s ="00101000" )then
                                        c_load :="110";                  count  :=
"00101";
-- instruction  JR NZ,e
                                elsif( data_s ="00100000" )then
                                        c_load :="111";          count := "00101";
```

```
                    end if;
                    if( data_s(7 downto 6) ="10")then
-- instruction ADD A,R
                        if( data_s(5 downto 3) ="000") then
                            ctrl_alu_gr <="11001";
-- instruction SUB A,R
                        elsif( data_s(5 downto 3) ="010") then
                            ctrl_alu_gr <="11010";
-- instruction  AND A,R
                        elsif( data_s(5 downto 3) ="100") then
                            ctrl_alu_gr <="11101";
-- instruction  OR A,R
                        elsif( data_s(5 downto 3) ="110") then
                            ctrl_alu_gr <="11110";
-- instruction  XOR A,R
                        elsif( data_s(5 downto 3) ="101") then
                            ctrl_alu_gr <="11111";

                        end if;

                    elsif( data_s(7 downto 6) ="00"  )then

                        if( data_s(2 downto 0) ="100") then
                            ctrl_alu_gr <="11011";
                            count :="01110";

                        elsif( data_s(2 downto 0) ="101") then
                            ctrl_alu_gr <="11100";
                            count :="01110";
```

```
                    end if;
                end if;

            elsif( count ="00101" )then
                ctrl_s_gr  <="11100";   -- PC put down on ADDR
                RD <='0';        MERQ <='0';
                ctrl_bus   <="10";                  -- In data

                if( c_load >"000" and c_load <"100")then
                    ctrl_c_bus <="001";     -- Data In to
Register
                    case c_load is
                    when "001" => ctrl_alu_gr <="00001";
-- load A with DATA
                    when "010" => ctrl_alu_gr <="00011";
-- load H with DATA
                    when "011" => ctrl_alu_gr <="00100";
-- load L with DATA
                    when others => NULL;
                    end case;

                else
                    ctrl_c_bus <="011";     -- Data In to Control
                    data_s := data_i;           -- Accept Data
To Control
                end if;

                count := "00110";
```

```
elsif( count ="00110" )then
        ctrl_bus  <="00";  ctrl_c_bus <="000";
        RD <='1';          MERQ <='1';
        count :="00001";          -- Goto Timing Fecth
        ctrl_s_gr <="01110";              -- increase PC

        case c_load is
            when "001" => ctrl_alu_gr <="00000";
                          c_load :="000";
-- load A with DATA
            when "010" => ctrl_alu_gr <="00000";
                          c_load :="000";
-- load H with DATA
            when "011" => ctrl_alu_gr <="00000";
                          c_load :="000";
-- load L with DATA
            when "100" => count :="00111";
-- Back Hold
            when "101" => count :="00111";
-- Back Hold
            when "110" => count :="00111";
-- Back Hold
            when "111" => count :="00111";
-- Back Hold
            when others => NULL;
        end case;
```

```
                    if( c_ix_in_out >"01" )then   -- Back Hold
                        count := "00111";
                    end if;

                elsif( count ="00111") then
                    ctrl_s_gr <="10111";    -- Save Pc To Pc saved

                    if( c_ix_in_out ="10")then    -- instruction IN A,(m)
                        count := "10001";

                elsif( c_ix_in_out ="11" )then
-- instruction OUT (m), A
                    count := "10101";
                end if;
                    data_s1 := data_s;

                    if( c_load > "011") then
                        count :="11001";
                    end if;

                elsif( count ="01000")then
                    ctrl_s_gr <="11100";    RD <='0';
                    MERQ <='0';             ctrl_bus  <="10";
                    ctrl_c_bus <="010";     count :="01001";

                elsif( count ="01001") then
                    ctrl_c_bus <="000";     ctrl_bus  <="00";
                    RD <='1';         MERQ <='1';
```

```
                    if( c_load ="101")then    -- instruction LD IX,mn 1
                            ctrl_s_gr <="00110";

                    elsif( c_load ="100") then   -- instruction LD IX,mn
2
                            ctrl_s_gr <="00111";

                    elsif( c_load ="111" )then   -- instruction LD A,(mn)
1
                            ctrl_s_gr <="00010";

                    elsif( c_load ="110" )then  -- instruction LD A,(mn)
2
                            ctrl_s_gr <="00011";

                    end if;
                    count :="01010";

            elsif( count ="01010") then
                    ctrl_c_bus <="000";
                    ctrl_s_gr  <="01110";   -- instruction LD IX,mn 1

                    if( c_load ="101")then
                            count := "01000";  c_load :="100";
-- instruction LD IX,mn 2
                    elsif( c_load ="100") then
                            count := "00001";  c_load :="000";
-- instruction LD A,(mn) 1
                    elsif( c_load ="111") then
```

```
                count := "01000"; c_load :="110";
-- instruction LD A,(mn) 2
            elsif( c_load ="110") then
                count := "01011"; c_load :="001";
            end if;
-- instruction data transfer A
        elsif( count ="01011") then
            ctrl_s_gr <="11100";    RD <='0';
            MERQ <='0';   ctrl_bus <="10";
            ctrl_c_bus <="001";     count :="01100";
        elsif( count ="01100") then
            ctrl_s_gr  <="00000";   RD <='1';
            MERQ <='1';             ctrl_bus  <="00";
            ctrl_c_bus <="000";
            count :="01101";
        elsif( count ="01101") then
            ctrl_c_bus <="000";

            if( c_load ="001") then
                ctrl_alu_gr <="00001";
                c_load  :="000";
            end if;
            ctrl_s_gr  <="11000";
            count :="00001";

        elsif( count ="01110") then
```

```
                    if( data_s(7 downto 6) ="00"
                    and data_s(2 downto 1) ="10" )then

                        case  data_s(5 downto 3)  is
-- instruction INC & DEC A
                        when "111" => ctrl_alu_gr <="10110";
-- instruction INC & DEC H
                        when "100" => ctrl_alu_gr <="10111";
-- instruction INC & DEC L
                        when "101" => ctrl_alu_gr <="11000";

                        when others => NULL;

                        end case;

                    end if;
                        count :="00001";

                elsif( count ="01111") then

                    if( c_ix_in_out ="10" )then
                        ctrl_s_gr <="00000";
                    elsif( c_ix_in_out ="11" )then
                        ctrl_s_gr <="11111";
                    end if;
                    count :="10000";


            elsif( count ="10000") then
                    if( c_ix_in_out ="11") then
```

```
                    ctrl_s_gr <="01010";
                end if;

                c_ix_in_out :="00"; count :="00001";
-- Start  instruction IN A,(m)
            elsif( count ="10001") then

                if( c_ix_in_out ="10" ) then
                    data_o <="00000000";
                    ctrl_c_bus <="110";  ctrl_s_gr <="00011";
                end if;
                count :="10010";

            elsif( count ="10010") then
                if( c_ix_in_out ="10" ) then
                    data_o <= data_s;
                    ctrl_c_bus <="110";  ctrl_s_gr <="00010";
                end if;
                count :="10011";

            elsif( count ="10011") then

                if( c_ix_in_out ="10" ) then
                    ctrl_s_gr <="11100";
                end if;
                IORQ <='0';  RD <='0';
                ctrl_alu_gr <="00001";
                ctrl_c_bus <="001";
                ctrl_bus <="10";
```

```
                count :="10100";

        elsif( count ="10100") then
                RD <='1';               IORQ <= '1';
                ctrl_c_bus <="000";             ctrl_bus <="00";
                ctrl_alu_gr <="00000";
                 ctrl_s_gr <="11000";
                count := "00001";
-- Stop instruction IN A,(m)
-- Start instruction OUT (m),A

        elsif( count ="10101") then
                if( c_ix_in_out ="11" ) then
                        data_o <="00000000";
                        ctrl_c_bus <="110";  ctrl_s_gr <="00011";
                end if;
                count :="10110";

        elsif( count ="10110") then
                if( c_ix_in_out ="11" ) then
                        data_o <= data_s;
                        ctrl_c_bus <="110";  ctrl_s_gr <="00010";
                end if;
                count :="10111";
        elsif( count ="10111") then
                if( c_ix_in_out ="11" ) then
                        ctrl_s_gr <="11100";
                end if;
                IORQ <='0';  WR <='0';
```

```
                ctrl_alu_gr <="01011";
                ctrl_c_bus <="100";
                ctrl_bus <="01";
                count :="11000";

        elsif( count ="11000") then
                WR <='1';               IORQ <= '1';
                ctrl_c_bus <="000";     ctrl_bus <="00";
                ctrl_alu_gr <="00000";
                ctrl_s_gr <="11000";
                c_ix_in_out :="00";
                count := "00001";
-- Stop instruction OUT (m),A
-- Start State Jump
        elsif( count ="11001") then
                ctrl_s_gr <="00000";
                ctrl_alu_gr <="01100"; ctrl_c_bus <="111";
                data_s := data_i;               count :="11010";

        elsif( count ="11010") then
                ctrl_alu_gr <="00000";  ctrl_c_bus <="000";
                ctrl_s_gr  <="01011";
                c_jump := data_s1(7);

                if( data_s1(7) ='1') then
                        data_s1 := not( data_s1);
                        c :="000000000";

                        for i in 0 to 7 loop
```

```
                    sum1(i) := (data_s1(i) xor DS2(i)) xor
c(i);

                    c(i+1)  := (data_s1(i) and DS2(i)) or (c(i)
                        and data_s1(i)) or (DS2(i) and
                c(i));
                end loop;

                data_s1 := sum1;
            end if;
            count :="11011";

        elsif( count ="11011") then
            data_o(6 downto 0) <=data_s1(6 downto 0);
            data_o(7) <='0';
            ctrl_c_bus <="110";

            if( c_load ="100" )then
                if( data_s(0) ='1')then     -- JR C,e
                    if( c_jump ='0')then
                        ctrl_s_gr <="10001";
                    else
                        ctrl_s_gr <="10010";
                    end if;
                end if;


        elsif( c_load ="101") then
            if( data_s(0) ='0')then    -- JR NC,e
                if( c_jump ='0') then
                    ctrl_s_gr <="10001";
```

```
                else
                        ctrl_s_gr <="10010";
                end if;
        end if;

elsif( c_load ="110") then
        if( data_s(6) ='1') then        -- JR Z,e
                if( c_jump ='0') then
                        ctrl_s_gr <="10001";
                else
                        ctrl_s_gr <="10010";
                end if;
        end if;

elsif( c_load ="111") then
        if( data_s(6) ='0') then        -- JR NZ,e
                if( c_jump ='0') then
                        ctrl_s_gr <="10001";
                else
                        ctrl_s_gr <="10010";
                end if;
        end if;
else
        ctrl_s_gr <="00000";
end if;
count :="11100";
```

- End State Jump

```
                elsif( count ="11100") then
                        ctrl_c_bus <="000";
                        ctrl_s_gr <="01000";
                        c_load :="000";
                        count :="00001";

                elsif( count ="11101") then
                elsif( count ="11110") then
                elsif( count ="11111") then
                else
                        ctrl_alu_gr <="00000";

                end if;
            end if;
        end if;

    end process ;
end a_control;
```

```
--------------------------------
-- Latch VHDL Test By Kitipong --
--------------------------------

-- Addition ALU unit & GROUP REGISTER  unit
-- 27 NOV 97

library ieee;
use ieee.std_logic_1164.all;

--library synth;
--use synth.vhdlsynth.all;


entity alu_gr is
 port( signal clk :in std_logic;
                signal  reset   : in std_logic;
                signal  CH      : in std_logic_vector ( 4 downto 0);
                signal  D       : in std_logic_vector ( 7 downto 0);
                signal  Q       : out std_logic_vector( 7 downto 0));
end alu_gr;

architecture A_alu_gr of alu_gr is
--****************************************
 function  C_CH (  I : std_logic_vector )return   std_logic is
        variable  da : std_logic := '0';
                variable  r1 : integer := 0;
                begin
                for v in 0 to 7 loop
```

```
                if( I(v) ='1' ) then
                        r1 := r1+1;
                        if( r1 = 2 ) then
                                da :='1'; r1 := 0;
                        else
                                da :='0';
                        end if;
                end if;
        end loop;
                return  da;
end  C_CH;

-- ****************************************

begin
P8:process(clk)
        variable  clr,S_A1,S_F1,S_H1,S_L1: std_logic_vector(7 downto 0):= "00000000";
        variable  c : std_logic_vector(8 downto 0):= "000000000";
        variable  D1, D2, DC,DS1 : std_logic_vector(7 downto 0):= "ZZZZZZZZ";
        variable  DS2,DS22,DF1,sum1: std_logic_vector(7 downto 0):= "ZZZZZZZZ";

begin
        if( clk'event and clk = '1' )then
                if( reset ='0')then
                        S_A1 := clr;     S_F1 := clr;
                        S_H1 := clr;     S_L1 := clr;
                else
                        case CH is
                                when "00001" => S_A1 := D;
```

```
            when "00010" => S_F1 := D;
            when "00011" => S_H1 := D;
            when "00100" => S_L1 := D;
            when "00101" => S_A1 := S_H1;
            when "00110" => S_A1 := S_L1;
            when "00111" => S_H1 := S_A1;
            when "01000" => S_H1 := S_L1;
            when "01001" => S_L1 := S_A1;
            when "01010" => S_L1 := S_H1;
            when "01011" => Q <= S_A1;
            when "01100" => Q <= S_F1;
            when "01101" => Q <= S_H1;
            when "01110" => Q <= S_L1;
            when "01111" => D2 := S_A1;
            when "10000" => D2 := S_H1;
            when "10001" => D2 := S_L1;
            when "10010" => D2 := D;
            when "10011" => D1 := S_A1;
            when "10100" => D1 := S_H1;
            when "10101" => D1 := S_L1;
            when "10110" => S_A1 := D1;
            when "10111" => S_H1 := D1;
            when "11000" => S_L1 := D1;
            when others  => NULL;
        end case;

    if( CH > "11000" )then
            DF1 := S_F1;
                    if( CH/="11011" and CH/="11100" )then
```

```
                D1 := S_A1;
            end if;

DS1 := D1;    DS22 := D2;

case CH is
        when "11010" =>  DS2 := NOT(D2);
        when "11011" =>  DS2 := clr;  DS2(0) :='1';
        when "11100" =>  DS2 := "11111111";
        when others =>   DS2 := D2;
end case;

if( CH ="11001" or CH ="11010" or CH ="11011"
        or CH ="11100" ) then
        if( CH ="11010" )then
                c(0) := '1';
        else
                c(0) := '0';
        end if;
        for i in 0 to 7 loop
                sum1(i) := (DS1(i) xor DS2(i)) xor c(i);
                c(i+1) := (DS1(i) and DS2(i)) or (c(i) and
                        DS1(i)) or (DS2(i) and c(i));
        end loop;

        if( CH="11001" or CH ="11011" ) then

                if( CH="11001")then
```

```
                DF1(0) := c(8);          -- flag C

                if( DS1(7) ='1' and DS2(7)='1' ) then
                        DF1(2) := '1';   -- flag V

                else
                        DF1(2) := '0';
                end if;
        else

                if( DS1 ="10000000") then
                        DF1(2) := '1';   -- flag V
                else
                        DF1(2) :='0';
                end if;
        end if;

        c(0):= '0';         DC := clr;

        for i in 0 to 3 loop
                DC(i) := (DS1(i) xor DS2(i)) xor
c(i);

                c(i+1) := (DS1(i) and DS2(i)) or
(c(i)
                        and DS1(i)) or (DS2(i)   and
                        c(i));
        end loop;

        DF1(4) := c(4);          -- flag H
```

```
                DF1(1) := '0';          -- flag N
            else
                if( CH="11010")then
                    if( DS22 > DS1)then
                        DF1(0) :='1'; -- flag C
                    else
                        DF1(0) :='0';
                    end if;

                if( DS1 > DS22 AND  DS1(7)='1' AND
                DS22(7)='0' AND sum1 > "10000000" )
    then
                    DF1(2) := '1';    -- flag V
                else
                    DF1(2) := '0';
                end if;

                if( DS1(3 downto 0) < DS22 (3 downto
    0))then
                    DF1(4) := '1';    -- flag H
                else
                    DF1(4) := '0';
                end if;
            else

                if( DS1 ="01111111") then
                    DF1(2) := '1';    -- flag V
                else
                    DF1(2) := '0';
```

```
                end if;

                if( DS1( 3 downto 0) ="0000" ) then
                        DF1(4) :='1';     -- flag H
                else
                        DF1(4) :='0';
                end if;
            end if;

                DF1(1) := '1';          -- flag N
        end if;
        if( sum1 = clr ) then
                DF1(6) := '1';    -- flag Z
        else
                DF1(6) := '0';
        end if;
        DF1(7) := sum1(7);        -- flag S

elsif( CH ="11101" or CH ="11110" or CH ="11111" )then
        if( CH="11101") then
                S_A1 := (DS1 and DS22);
        DF1(4) := '1';                -- flag H

elsif( CH="11110") then
        S_A1 := (DS1 or DS22);
        DF1(4) := '0';                -- flag H

else
        S_A1 := (DS1 xor DS22);
```

```
                    DF1(4) := '0';              -- flag H
                end if;

                DF1(0) := '0';              -- flag C
                DF1(1) := '0';              -- flag N
                DF1(7) := D1(7);                -- flag S

                if( S_A1 = clr ) then
                    DF1(6) :='1';    -- flag Z
                else
                DF1(6) :='0';
                end if;

                DF1(2) := C_CH(S_A1);          -- flag P
                end if;

            if( CH ="11011" or CH ="11100" )then
                D1 := sum1;

            elsif( CH ="11001" or CH ="11010" )then
                S_A1 := sum1;
            end if;
            S_F1 := DF1;
        end if;
    end if;
end if;
 end process P8;
end A_alu_gr;
```

```
---------------------------------
-- Latch VHDL Test By Kitipong --
---------------------------------

-- Spacial Register
-- 27 NOV 97

library ieee;
use ieee.std_logic_1164.all;

library synth;
use synth.vhdlsynth.all;



entity s_gr is
 port(  clk :in std_logic;
                CH      : in std_logic_vector ( 4 downto 0);
                D       : in std_logic_vector ( 7 downto 0);
                Q       : out std_logic_vector( 7 downto 0);
                addrr   : out std_logic_vector( 15 downto 0));
end  s_gr;

architecture a_s_gr of s_gr is
begin

S:process(clk)
        variable  clr,S_PCL,S_PCH,S_SPL        : std_logic_vector(7 downto 0):=
"00000000";
```

```
        variable  S_SPH,S_IXL,S_IXH          : std_logic_vector(7 downto 0):=
"00000000";
        variable  PC1,clr1       : std_logic_vector( 15 downto 0) ="0000000000000000";
        variable  PCS,S_C        : std_logic_vector( 15 downto 0) :="0000000000000000";
begin
        if( clk'event and clk = '1' )then
                        Q <="ZZZZZZZZ";

                case CH is
                        when "00001" => S_PCH := clr;          S_PCL := clr;
                                        S_SPH := clr;                  S_SPL := clr;
                                        addrr <= clr1;
                        when "00010" => S_PCL := D;
                        when "00011" => S_PCH := D;
                        when "00100" => S_SPL := D;
                        when "00101" => S_SPH := D;
                        when "00110" => S_IXL := D;
                        when "00111" => S_IXH := D;
                        when "01000" => S_PCL := S_C(7 downto 0);
                                        S_PCH := S_C(15 downto 8);
                        when "01001" => S_SPL := S_C(7 downto 0);
                                        S_SPH := S_C(15 downto 8);
                        when "01010" => S_IXL := S_C(7 downto 0);
                                        S_IXH := S_C(15 downto 8);
                        when "01011" => S_C(7 downto 0)  := S_PCL;
                                        S_C(15 downto 8) := S_PCH;
                        when "01100" => S_C(7 downto 0)  := S_SPL;
                                        S_C(15 downto 8) := S_SPH;
                        when "01101" => S_C(7 downto 0) := S_IXL;
```

```
                S_C(15 downto 8) := S_IXH;

when "01110" => S_C(15 downto 8) := S_PCH;
                S_C(7 downto 0) := S_PCL;
               if( S_C ="1111111111111111" ) then
                    S_C := clr1;
               else
                    S_C := ( S_C + 1);
               end if;
               S_PCH := S_C(15 downto 8);
               S_PCL := S_C(7 downto 0);

when "01111" => S_C(15 downto 8) := S_SPH;
                S_C(7 downto 0) := S_SPL;
               if( S_C ="1111111111111111" ) then
                    S_C := clr1;
               else
                    S_C := ( S_C + 1);
               end if;
                    S_SPH := S_C(15 downto 8);
                    S_SPL := S_C(7 downto 0);

when "10000" => S_C(15 downto 8) := S_IXH;
                S_C(7 downto 0) := S_IXL;
               if( S_C ="1111111111111111" ) then
                    S_C := clr1;
               else
                    S_C := ( S_C + 1);
               end if;
```

```
                    S_IXH := S_C(15 downto 8);
                    S_IXL := S_C(7 downto 0);

when "10001" => PCS( 7 downto 0) := D;
                    S_C := ( S_C + PCS );
when "10010" => PCS( 7 downto 0) := D;
                    S_C := ( S_C - PCS );

when "10011" => S_SPH := S_PCH;
                    S_SPL := S_PCL;

when "10100" => S_PCH := S_SPH;
                    S_PCL := S_SPL;

when "10101" => S_SPH := S_IXH;
                    S_SPL := S_IXL;

when "10110" => S_IXH := S_SPH;
                    S_IXL := S_SPL;

when "10111" => PC1(15 downto 8) := S_PCH;
                    PC1(7 downto 0)  := S_PCL;

when "11000" => S_PCH := PC1(15 downto 8);
                    S_PCL := PC1(7 downto 0);

when "11001" => S_PCH := S_IXH;
                    S_PCL := S_IXL;
```

```
                when "11010" => Q <= S_C(15 downto 8);
                when "11011" => Q <= S_C(7 downto 0);

                when "11100" => addrr(15 downto 8) <= S_PCH;
                                addrr(7 downto 0)  <= S_PCL;

                when "11101" => addrr(15 downto 8) <= S_SPH;
                                addrr(7 downto 0)  <= S_SPL;

                when "11110" => S_PCH := clr;
                                S_PCL := "00001111";
                                addrr(15 downto 8) <= S_PCH;
                                addrr(7 downto 0)  <= S_PCL;

                when "11111" => S_C := (S_C - 1 );
                when others => NULL;
            end case;
        end if;
    end process S;
end a_s_gr;
```

```
---------------------------------
-- Latch VHDL Test By Kitipong --
---------------------------------

-- Control Bus Unit
-- 27 NOV 97

library ieee;
use ieee.std_logic_1164.all;

--library synth;
--use synth.vhdlsynth.all;



entity c_bus is
 port( signal  clk :in std_logic;
        signal  CH  : in std_logic_vector ( 2 downto 0);
        signal  alu_gr_i, sgr_i,  ctrl_i, data_i   : in std_logic_vector ( 7 downto 0);
        signal  alu_gr_o, sgr_o,ctrl_o, data_o  : out std_logic_vector( 7 downto 0));
end c_bus;



architecture ac_bus of c_bus is
begin

B:process(clk)
begin
        if( clk'event and clk = '1' )then
                alu_gr_o <="ZZZZZZZZ";
```

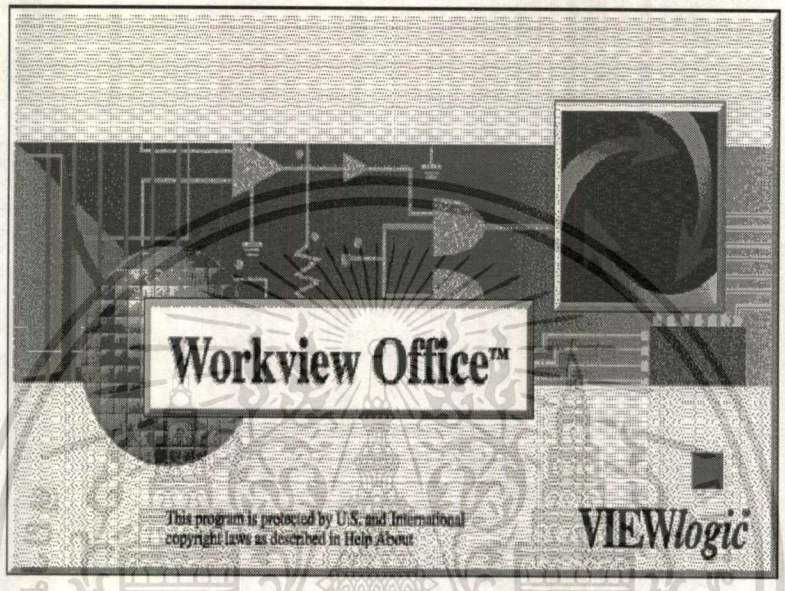
```
            sgr_o <="ZZZZZZZZ";
            ctrl_ò <="ZZZZZZZZ";
            data_o <="ZZZZZZZZ";

                case CH is
                    when "001" => alu_gr_o <= data_i;
                    when "010" => sgr_o    <= data_i;
                    when "011" => ctrl_o   <= data_i;
                    when "100" => data_o   <= alu_gr_i;
                    when "101" => data_o   <= sgr_i;
                    when "110" => sgr_o    <= ctrl_i;
                    when "111" => ctrl_o   <= alu_gr_i;
                    when others => NULL;
                end case;
            end if;
    end process B;
    end ac_bus;
```

```
---------------------------------
-- Latch VHDL Test By Kitipong --
---------------------------------
library ieee;
use ieee.std_logic_1164.all;

--library synth;
--use synth.vhdlsynth.all;

entity b_s is
 port( signal clk  : in std_logic;
                signal  CH      : in std_logic_vector ( 1 downto 0);
                signal  D       : in std_logic_vector ( 7 downto 0);
                signal  Q       : out std_logic_vector( 7 downto 0);
                in_out1                 : inout std_logic_vector(7 downto 0)bus);
end b_s;
architecture a_b_s of b_s is
begin
G:process(clk)
        begin
                if( clk ='1' and clk'event ) then
                        case CH is
                                when "01" => in_out1 <= std_logic_vector(D);
                                when "10" => Q <= std_logic_vector(in_out1);
                                when others => in_out1 <= "ZZZZZZZZ";
                        end case;
            end if;
end process G;
end a_b_s;
```

ภาคผนวก ข
ขั้นตอนการ Simulate โดยใช้ซอฟแวร์ของบริษัทวิวลอจิก (Viewlogic) และขั้นตอนการติดตั้งโปรแกรม Workview Office

ขั้นตอนการจำลองการทำงานโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก (Viewlogic)

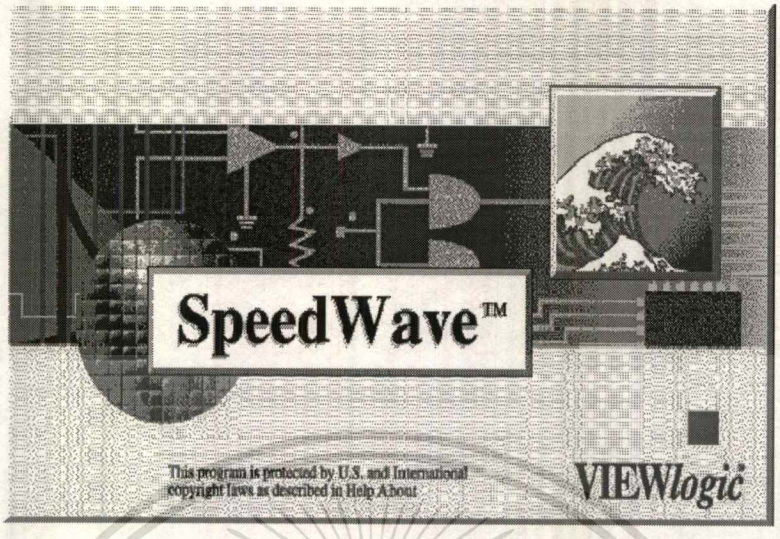
ซอฟแวร์ของบริษัทวิวลอจิกเวอร์ชั่น 7.20 เป็นเวอร์ชั่นที่ทำงานบน Windows 95 เมื่อเริ่มต้นใช้โปรแกรม ที่หน้าจอจะปรากฏรูปภาพดังรูปที่ 1



รูปที่ 1 หน้าจอเริ่มต้นของโปรแกรมวิวลอจิก

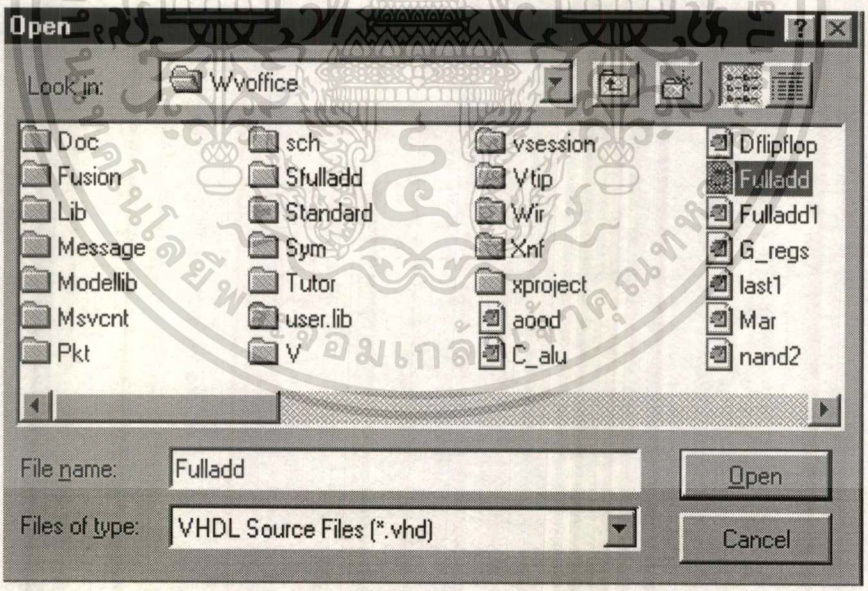
การใช้โปรแกรมสปีดเวฟ (Speed wave)

โปรแกรมสปีดเวฟ ใช้สำหรับเขียนโปรแกรมซอร์สโค้ดด้วยภาษา VHDL โดยจะยกตัวอย่างการใช้โปรแกรมภาษา VHDL เพื่อจำลองการทำงานของวงจร Fulladder ใช้ชื่อไฟล์ Fulladd.vhd คลิกเมาส์ที่ไอคอนสปีดเวฟบริเวณมุมขวาของจอภาพ หน้าจอจะปรากฏดังรูปที่ 2



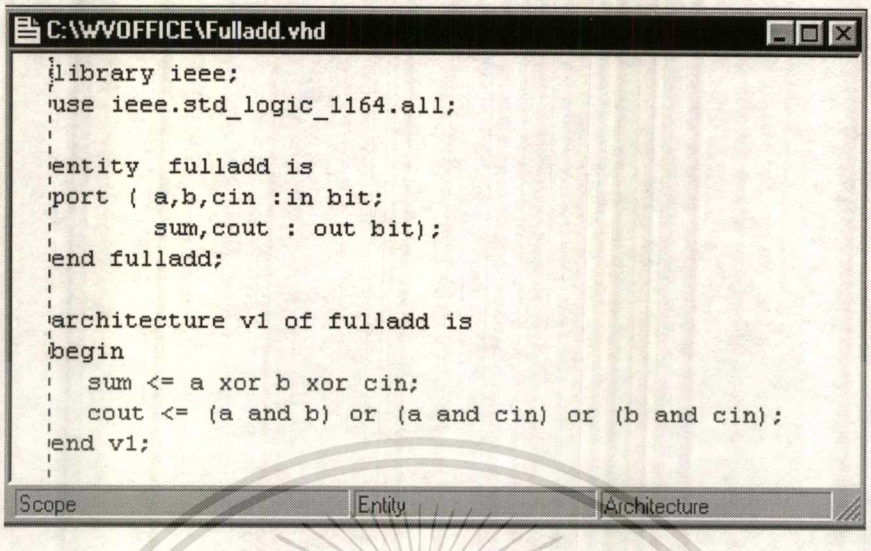
รูปที่ 2 ภาพเริ่มต้นของการใช้โปรแกรม Speed Wave

คลิกเมาส์ที่เมนู File และ Open เพื่อเรียกใช้โปรแกรม Fulladd.vhd ที่ได้เขียนไว้แล้ว หรือ New เพื่อเขียนโปรแกรมใหม่ ดังรูปที่ 3



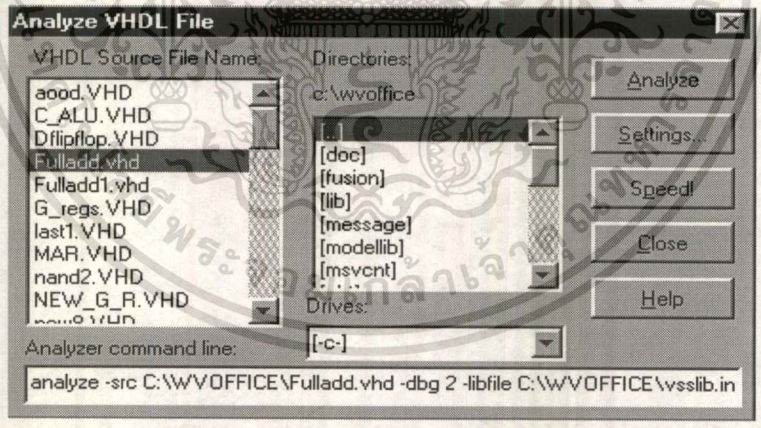
รูปที่ 3 หน้าจอ Open

เปิดไฟล์ Fulladd.vhd ซึ่งจะแสดงหน้าต่างของไฟล์ Fulladd.vhd ดังรูปที่ 4



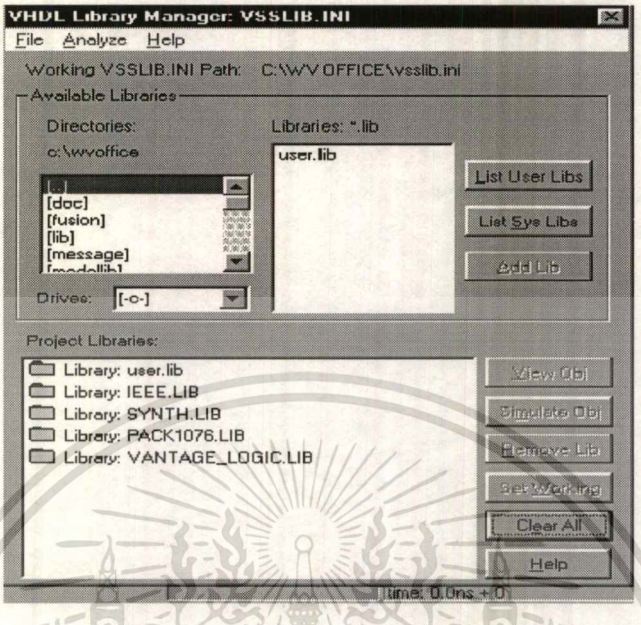
รูปที่ 4 หน้าต่างของไฟล์ Fulladd.vhd

เมื่อทำการเขียนโปรแกรมเรียบร้อยแล้วให้ทำการบันทึกและคลิกเมาส์ที่เมนู Tools พร้อมทั้งเลือกที่ Analyze VHDL เพื่อทำการคอมไพล์โปรแกรม Fulladd.vhd ดังรูปที่ 5



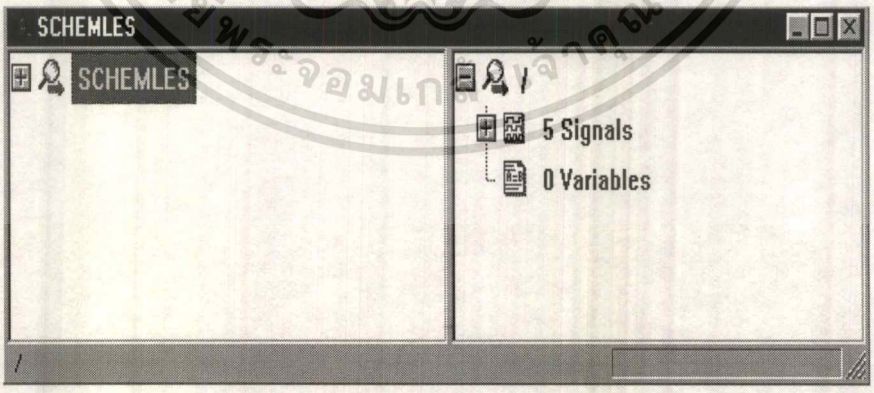
รูปที่ 5 หน้าต่างของ Anlyze VHDL File

คลิกเมาส์ที่ Analyze เพื่อทำการคอมไพล์โปรแกรม Fulladd.vhd หากเกิด Error ให้กลับไปแก้ไขที่โปรแกรม Fulladd.vhd และทำการคอมไพล์โปรแกรมใหม่จนกว่าจะผ่านเมื่อทำการคอมไพล์ไฟล์ Fulladd.vhd ผ่านแล้วให้คลิกเมาส์ที่เมนู Tools และเลือกคำสั่ง VHDL Manager ซึ่งแสดงหน้าต่างของ VHDL Lidrary Manager ดังรูปที่ 6



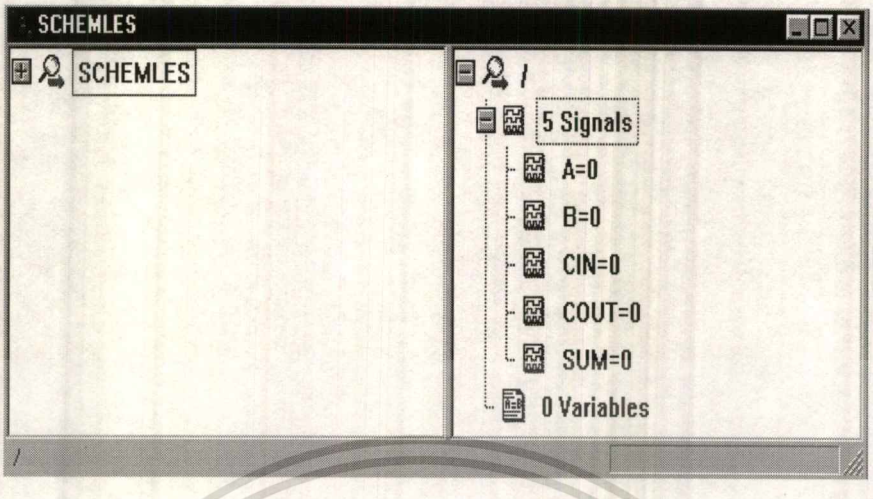
รูปที่ 6 หน้าต่างของ VHDL Library Manager

ดับเบิ้ลคลิกที่ไดเร็คทอรี Library.use.lib ซึ่งจะปรากฏไดเร็คทอรีย่อยของโปรแกรม *.vhd โปรแกรมต่างๆเลือกไดเร็คทอรี Fulladd:Entity และคลิกเมาส์ที่ Simulate Obj ซึ่งจะแสดงหน้าต่างของสัญญาณต่างๆทั้งอินพุทและเอาท์พุทของโปรแกรม Fulladd โดยจะซ่อนอยู่หลังหน้าต่างโปรแกรม Fulladd.vhd ที่เรียกขึ้นมาก่อน ดังรูปที่ 7



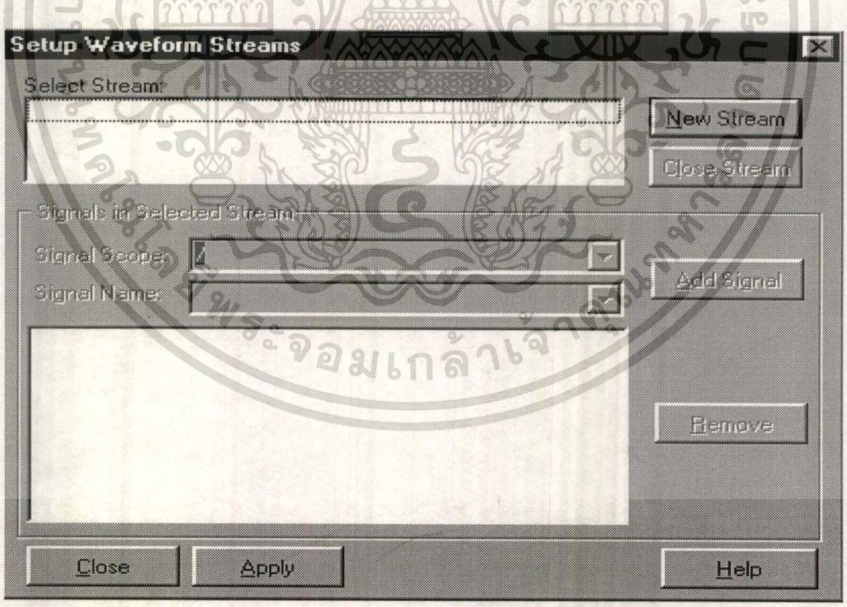
รูปที่ 7 หน้าต่างของ SCHEMLES

คลิกเมาส์ที่ไดเร็คทอรีทางหน้าต่างด้านขวา ซึ่งจะแสดงสัญญาณอินพุทและเอาท์พุททั้ง 5 สัญญาณของโปรแกรม Fulladd.vhd ให้เห็น ดังรูปที่ 8



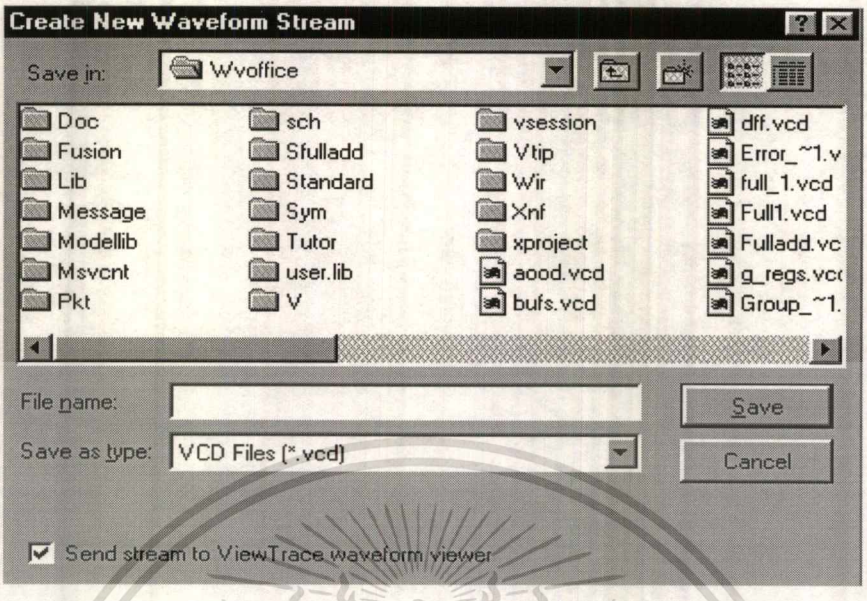
รูปที่ 8 สัญญาณอินพุตและเอาท์พุตของโปรแกรม Fulladd.vhd

คลิกเมาส์ที่เมนู Setup และเรียกใช้คำสั่ง Wavefrom Streams จะขึ้นหน้าต่าง Setup Waveform Streams ให้เห็น ดังรูปที่ 9



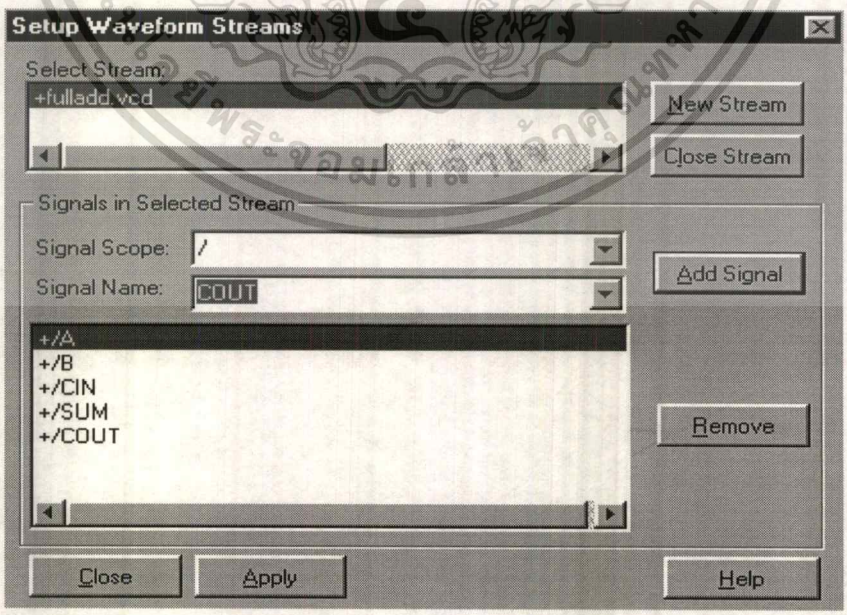
รูปที่ 9 หน้าต่างของเมนู Setup wavefrom Streams

คลิกเมาส์ที่ New Streams ซึ่งจะปรากฏหน้าต่าง Create New Wavefrom Stream ดังรูปที่ 10

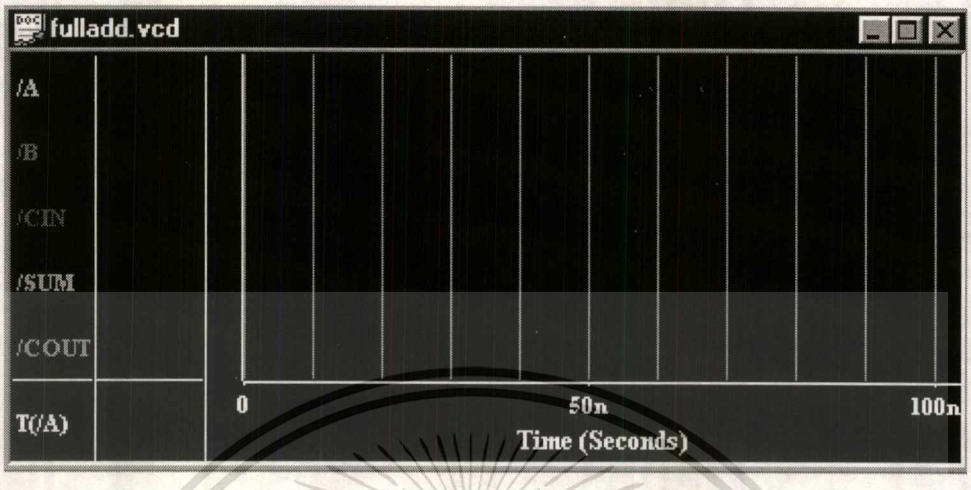


รูปที่ 10 หน้าต่างของ Create New Stream

ใส่ชื่อไฟล์ที่จะบันทึกลงในช่อง File Name และทำการเซฟ หลังจากนั้นใส่ชื่อสัญญาณทั้ง 5 สัญญาณในช่อง Signal Name ดังรูปที่ 11 แล้วคลิกเมาส์ที่ Add Signal จนครบทั้ง 5สัญญาณแล้ว ให้คลิกเมาส์ที่ Apply เพื่อเรียกใช้โปรแกรมวิวเทรซ (View Trace) ในการดูผลจำลองการทำงานของโปรแกรม Fulladd.vhd ดังรูปที่ 12

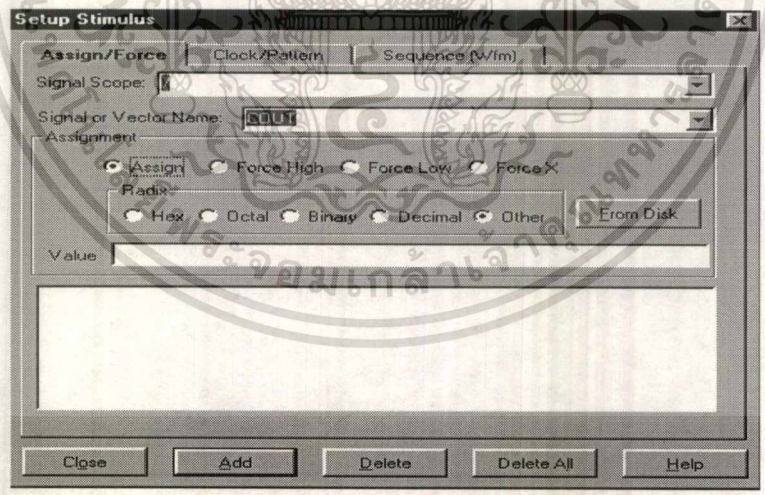


รูปที่ 11 การ Add Signal



รูปที่ 12 หน้าต่างจำลองการทำงานของ Fulladd.vhd

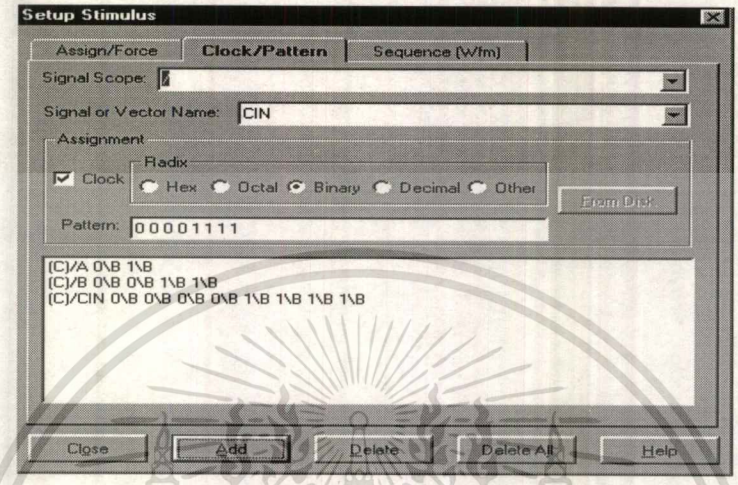
กลับไปที่หน้าต่างโปรแกรม SCHEMLES อีกครั้ง คลิกเมาส์ที่เมนู Setup และเลือกใช้คำสั่ง Stimulus ซึ่งจะปรากฏหน้าต่าง Setup stimulus ขึ้นมาเพื่อใช้กำหนดค่าสัญญาณในการจำลองการทำงาน ดังรูปที่ 13



รูปที่ 13 หน้าต่าง Setup Stimulus

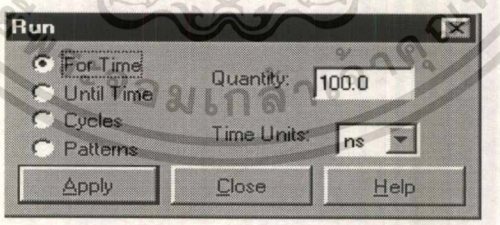
คลิกเมาส์ที่ Clock/Pattern เพื่อกำหนดค่าให้สัญญาณอินพุท ที่มีลักษณะเป็นสัญญาณนาฬิกา (Clock) โดยใส่สัญญาณอินพุทที่ Signal or Vector Name เช่น อินพุท A (/A) และคลิกที่

Clock ในกรณีที่เป็นสัญญาณนาฬิกา และที่ Binary ให้ใส่ค่า 01 ที่Pattern และสัญญาณอินพุท B มีค่า 0011 และ Cin มีค่าเป็น 00001111 ดังรูปที่ 14



รูปที่ 14 การกำหนดค่าสัญญาณอินพุท A B และ Cin

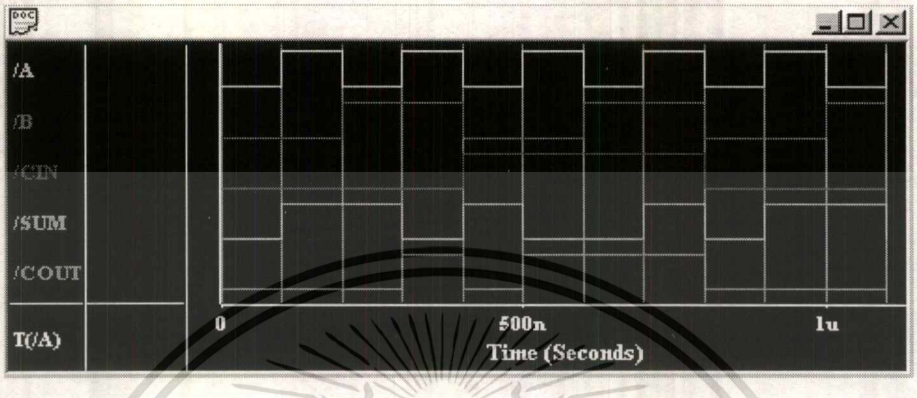
ในกรณีที่ต้องการกำหนดสัญญาณให้เป็น High หรือ Low ก็ให้คลิกเมาส์ที่ Assign/Force และเลือกที่ Force High หรือ Force Low จากรูปที่ 14 เมื่อทำการกำหนดค่าให้สัญญาณอินพุทเสร็จแล้ว ให้คลิกที่เมนู Simulate และเลือกคำสั่ง Run จะปรากฏหน้าต่าง Run ดังรูปที่ 15



รูปที่ 15 หน้าต่าง Run

การกำหนดช่วงเวลาการทำงานโดยคลิกเมาส์ที่ Cycle เพื่อให้จำลองการทำงานเป็นวงรอบหรือถ้าหากต้องการจำลองการทำงานเป็นช่วงเวลาให้คลิกที่ For Time และใส่ค่าเวลาที่ Quantity เมื่อใส่ค่าเสร็จแล้วให้คลิกที่ Apply และไปดูผลการจำลองการทำงาน ดังรูปที่ 12 โดย

ที่โปรแกรมวิวเทรซให้คลิกเมาส์ที่เมนู View และเลือกคำสั่ง Time Range และคลิกที่ OK จะแสดงผลการจำลองการทำงานของโปรแกรม Full Adder ทั้งหมดให้เห็นดังรูปที่ 16



รูปที่ 16 ผลการทำงานของโปรแกรม Fulladd.vhd

สำหรับขั้นตอนที่กล่าวมาข้างต้นเป็นเพียงตัวอย่างขั้นตอนการ Simulate ซึ่งในขั้นตอนการจำลองการทำงานจริงก็จะใช้วิธีการตามที่ได้กล่าวมาแล้ว

ขั้นตอนการติดตั้งโปรแกรม Workview Office

ขีดจำกัดของ Software Workview Office

1. CPU	80486/66 MHz	-	Pentium/90 MHz
2. Monitor Resolution	800 x 600 dpi	-	1024 x 768 dpi (หรือ ดีกว่า)
3. Hard Drive	500 MB	-	1 Gb (Full install ~240 MB)
4. RAM	16 MB	-	32,48, หรือ 64 MB
5. CD ROM	2X	-	4X,6X หรือ 8X
6. ระบบปฏิบัติการ	Windows 95	-	Windows NT3.51

การติดตั้งโปรแกรม

1. ใส่แผ่น CD ของ Workview Office ลงในไดร์ฟของ CD ROM
2. คลิ๊กเมาท์ที่ปุ่ม Install Workview Office เครื่องจะทำการติดตั้งโปรแกรมโดยอัตโนมัติ
3. ทำตามขั้นตอนของการ Install ถ้าสงสัยในคำถามให้ดูได้จาก Help
4. เมื่อทำการติดตั้งโปรแกรมเสร็จเรียบร้อยแล้วคลิ๊กเมาท์ที่ปุ่ม OK
5. ออกจาก Windows 95 และทำการ Reboot เครื่องใหม่

ภาคผนวก ค

เอกสารของอุปกรณ์ FPGAs ตระกูล XC4010E

XILINX®

XC4000 Series
Field Programmable Gate Arrays

July 30, 1996 (Version 1.03)

Product Specification

XC4000-Series Features

Note: XC4000-Series devices described in this data sheet include the XC4000E, XC4000EX, XC4000L, and XC4000XL. This information does not apply to the older Xilinx families: XC4000, XC4000A, XC4000D or XC4000H. For information on these devices, see the Xilinx WEBLINX at http://www.xilinx.com.

- Third Generation Field-Programmable Gate Arrays
  - Select-RAM™ memory: on-chip ultra-fast RAM with
    - synchronous write option
    - dual-port RAM option
  - Fully PCI compliant (speed grades -3 and faster)
  - Abundant flip-flops
  - Flexible function generators
  - Dedicated high-speed carry logic
  - Wide edge decoders on each edge
  - Hierarchy of interconnect lines
  - Internal 3-state bus capability
  - 8 global low-skew clock or signal distribution networks
- System Performance to 66 MHz
- Flexible Array Architecture
- Systems-Oriented Features
  - IEEE 1149.1-compatible boundary scan logic support
  - Individually programmable output slew rate
  - Programmable input pull-up or pull-down resistors
  - 12-mA sink current per XC4000E output (4 mA per XC4000L output)
- Configured by Loading Binary File
  - Unlimited reprogrammability
- Readback Capability
- Backward Compatible with XC4000 Devices
- XACT*step* Development System runs on '386/'486/ Pentium-type PC, Sun-4, and Hewlett-Packard 700 series
  - Interfaces to popular design environments
  - Fully automatic mapping, placement and routing
  - Interactive design editor for design optimization
  - RAM/ROM compiler

Low-Voltage Versions Available

- Low-Voltage Devices Function at 3.0 - 3.6 Volts
- XC4000L: Low-Voltage Versions of XC4000E devices
- XC4000XL: Low-Voltage Versions of XC4000EX devices

Additional XC4000EX/XL Features

- Highest Capacity — Over 130,000 Usable Gates
- Additional Routing Over XC4000E
  - almost twice the routing capacity for high-density designs
- Buffered Interconnect for Maximum Speed
- New Latch Capability in Configurable Logic Blocks
- Improved VersaRing™ I/O Interconnect for Better Fixed Pinout Flexibility
- Flexible New High-Speed Clock Network
  - 8 additional Early Buffers for shorter clock delays
  - 4 additional FastCLK™ buffers for fastest clock input
  - Virtually unlimited number of clock signals
- Optional Multiplexer or 2-input Function Generator on Device Outputs
- High-Speed Parallel Express™ Configuration Mode
- Improved I/O Setup and Clock-to-Output with FastCLK and Global Early Buffers
- 4 Additional Address Bits in Master Parallel Configuration Mode

Introduction

XC4000-Series high-performance, high-capacity Field Programmable Gate Arrays (FPGAs) provide the benefits of custom CMOS VLSI, while avoiding the initial cost, long development cycle, and inherent risk of a conventional masked gate array.

The result of eleven years of FPGA design experience and feedback from thousands of customers, these FPGAs combine architectural versatility, on-chip Select-RAM memory with edge-triggered and dual-port modes, increased speed, abundant routing resources, and new, sophisticated software to achieve fully automated implementation of complex, high-density, high-performance designs.

The XC4000 Series currently has 19 members, as shown in Table 1.

Table 1: XC4000-Series Field Programmable Gate Arrays

Device	Max Logic Gates (No RAM)	Max. RAM Bits (No Logic)	Typical Gate Range (Logic and RAM)*	CLB Matrix	Total Logic Blocks	Number of Flip-Flops	Max. Decode Inputs per side	Max. User I/O
XC4003E	3,000	3,200	2,000 - 5,000	10 x 10	100	360	30	80
XC4005E/L	5,000	6,272	3,000 - 9,000	14 x 14	196	616	42	112
XC4006E	6,000	8,192	4,000 - 12,000	16 x 16	256	768	48	128
XC4008E	8,000	10,368	6,000 - 15,000	18 x 18	324	936	54	144
XC4010E/L	10,000	12,800	7,000 - 20,000	20 x 20	400	1,120	60	160
XC4013E/L	13,000	18,432	10,000 - 30,000	24 x 24	576	1,536	72	192
XC4020E	20,000	25,088	13,000 - 40,000	28 x 28	784	2,016	84	224
XC4025E	25,000	32,768	15,000 - 45,000	32 x 32	1,024	2,560	96	256
XC4028EX/XL	28,000	32,768	18,000 - 50,000	32 x 32	1,024	2,560	96	256
XC4036EX/XL	36,000	41,472	22,000 - 65,000	36 x 36	1,296	3,168	108	288
XC4044EX/XL	44,000	51,200	27,000 - 80,000	40 x 40	1,600	3,840	120	320
XC4052XL	52,000	61,952	33,000 - 100,000	44 x 44	1,936	4,576	132	352
XC4062XL	62,000	73,728	40,000 - 130,000	48 x 48	2,304	5,376	144	384
Larger Devices Available in the First Half of 1997								

* Max values of Typical Gate Range include 20-30% of CLBs used as RAM.

Note: *Throughout the functional descriptions in this document, references to the XC4000E device family include the XC4000L, and references to the XC4000EX device family include the XC4000XL, unless explicitly stated otherwise. References to the XC4000 Series include the XC4000E, XC4000EX, XC4000L, and XC4000XL families. All functionality in low-voltage families is the same as in the corresponding 5-Volt family, except where numerical references are made to timing, power, or current-sinking capability.*

Description

XC4000-Series devices are implemented with a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), interconnected by a powerful hierarchy of versatile routing resources, and surrounded by a perimeter of programmable Input/Output Blocks (IOBs). They have generous routing resources to accommodate the most complex interconnect patterns.

The devices are customized by loading configuration data into internal memory cells. The FPGA can either actively read its configuration data from an external serial or byte-parallel PROM (master modes), or the configuration data can be written into the FPGA from an external device (slave, peripheral and Express modes).

XC4000-Series FPGAs are supported by powerful and sophisticated software, covering every aspect of design from schematic or behavioral entry, floorplanning, simulation, automatic block placement and routing of interconnects, to the creation, downloading, and readback of the configuration bit stream.

Because Xilinx FPGAs can be reprogrammed an unlimited number of times, they can be used in innovative designs where hardware is changed dynamically, or where hardware must be adapted to different user applications. FPGAs are ideal for shortening design and development cycles, and also offer a cost-effective solution for production rates well beyond 5,000 systems per month. For lowest high-volume unit cost, a design can first be implemented in the XC4000E or XC4000EX, then migrated to one of Xilinx' compatible HardWire mask-programmed devices.

Table 2 shows density and performance for a few common circuit functions that can be implemented in XC4000-Series devices.

Table 2: Density and Performance for Several Common Circuit Functions in XC4000E[1]

Design Class	Function	CLBs Used	XC4000E-3	XC4000E-2	Units
Memory	256 x 8 Single Port (read/modify/write)	72	63	80	MHz
	32 x 16 bit FIFO				
	simultaneous read/write	48	63	80	MHz
	MUXed read/write	32	63	80	MHz
Logic	9 bit Shift Register (with enable)	5	170	200	MHz
	16 bit Pre-Scaled Counter	8	142	170	MHz
	16 bit Loadable Counter	8	65	76	MHz
	16 bit Accumulator	9	65	76	MHz
	8 bit, 16 tap FIR Filter sample rate				
	parallel	400	55	65	MHz
	serial	68	8.1	10	MHz
	8 x 8 Parallel Multiplier				
	single stage, register to register	73	37	30	ns
	16 bit Address Decoder (internal decode)	3	4.7	3.9	ns
	9 bit Parity Checker	1	4.3	2.7	ns

Note: 1. Most functions are faster in XC4000EX due to faster carry logic, direct connects, and other additional interconnect.

Taking Advantage of Reconfiguration

FPGA devices can be reconfigured to change logic function while resident in the system. This capability gives the system designer a new degree of freedom not available with any other type of logic.

Hardware can be changed as easily as software. Design updates or modifications are easy, and can be made to products already in the field. An FPGA can even be reconfigured dynamically to perform different functions at different times.

Reconfigurable logic can be used to implement system self-diagnostics; create systems capable of being reconfigured for different environments or operations, or implement multi-purpose hardware for a given application. As an added benefit, using reconfigurable FPGA devices simplifies hardware design and debugging and shortens product time-to-market.

XC4000E and XC4000EX Families Compared to the XC4000

For readers already familiar with the XC4000 family of Xilinx Field Programmable Gate Arrays, the major new features in the XC4000-Series devices are listed in this section. The biggest advantages of XC4000E and XC4000EX devices are significantly increased system speed, greater capacity, and new architectural features, particularly Select-RAM memory. The XC4000EX devices also offer many new routing features, including special high-speed clock buffers that can be used to capture input data with minimal delay.

Any XC4000E device is pinout- and bitstream-compatible with the corresponding XC4000 device. An existing XC4000 bitstream can be used to program an XC4000E device. However, since the XC4000E includes many new features, an XC4000E bitstream cannot be loaded into an XC4000 device.

Most XC4000EX devices have no corresponding XC4000 devices, because of the larger CLB arrays. The XC4028EX has the same array size as the XC4025 and XC4025E, but is not bitstream-compatible. However, the XC4025, XC4025E, and XC4028EX are all pinout-compatible.

Improvements in XC4000E and XC4000EX

Increased System Speed

Delays in FPGA-based designs are layout dependent. There is a rule of thumb designers can consider—the system clock rate should not exceed one third to one half of the specified toggle rate. Critical portions of a design, such as shift registers and simple counters, can run faster—approximately two thirds of the specified toggle rate.

XC4000E and XC4000EX devices can run at synchronous system clock rates of up to 66 MHz, and internal performance can exceed 150 MHz. This increase in performance over the previous families stems from improvements in both device processing and system architecture. XC4000-Series devices use a sub-micron triple-layer metal process. In addition, many architectural improvements have been made, as described below.

PCI Compliance

XC4000-Series -3 and faster speed grades are fully PCI compliant. XC4000E and XC4000EX devices can be used to implement a one-chip PCI solution.

Carry Logic

The speed of the carry logic chain has increased dramatically. Some parameters, such as the delay on the carry chain through a single CLB (TBYP), have improved by as much as 50% from XC4000 values. See "Fast Carry Logic" on page 21 for more information.

Select-RAM Memory: Edge-Triggered, Synchronous RAM Modes

The RAM in any CLB can be configured for synchronous, edge-triggered, write operation. The read operation is not affected by this change to an edge-triggered write.

Dual-Port RAM

A separate option converts the 16x2 RAM in any CLB into a 16x1 dual-port RAM with simultaneous Read/Write.

The function generators in each CLB can be configured as either level-sensitive (asynchronous) single-port RAM, edge-triggered (synchronous) single-port RAM, edge-triggered (synchronous) dual-port RAM, or as combinatorial logic.

Configurable RAM Content

The RAM content can now be loaded at configuration time, so that the RAM starts up with user-defined data.

H Function Generator

In XC4000-Series devices, the H function generator is more versatile than in the XC4000. Its inputs can come not only from the F and G function generators but also from up to three of the four control input lines. The H function generator can thus be totally or partially independent of the other two function generators, increasing the maximum capacity of the device.

IOB Clock Enable

The two flip-flops in each IOB have a common clock enable input, which through configuration can be activated individually for the input or output flip-flop or both. This clock enable operates exactly like the EC pin on the XC4000 CLB. This new feature makes the IOBs more versatile, and avoids the need for clock gating.

Output Drivers

The output pull-up structure defaults to a TTL-like totem-pole. This driver is an n-channel pull-up transistor, pulling to a voltage one transistor threshold below Vcc, just like the XC4000 outputs. Alternatively, XC4000-Series devices can be globally configured with CMOS outputs, with p-channel pull-up transistors pulling to Vcc. Also, the configurable pull-up resistor in the XC4000 Series is a p-channel transistor that pulls to Vcc, whereas in the XC4000 it is an n-channel transistor that pulls to a voltage one transistor threshold below Vcc.

Input Thresholds

The input thresholds can be globally configured for either TTL (1.2 V threshold) or CMOS (2.5 V threshold), just like XC2000 and XC3000 inputs. The two global adjustments of input threshold and output level are independent of each other.

Global Signal Access to Logic

There is additional access from global clocks to the F and G function generator inputs.

Configuration Pin Pull-Up Resistors

During configuration, the three mode pins, M0, M1, and M2, have weak pull-up resistors. For the most popular configuration mode, Slave Serial, the mode pins can thus be left unconnected.

The three mode inputs can be individually configured with or without weak pull-up or pull-down resistors after configuration.

The $\overline{\text{PROGRAM}}$ input pin has a permanent weak pull-up.

Soft Start-up

Like the XC3000A, XC4000-Series devices have "Soft Start-up." When the configuration process is finished and the device starts up, the first activation of the outputs is automatically slew-rate limited. This feature avoids potential ground bounce when all outputs are turned on simultaneously. Immediately after start-up, the slew rate of the individual outputs is, as in the XC4000 family, determined by the individual configuration option.

XC4000 and XC4000A Compatibility

Existing XC4000 bitstreams can be used to configure an XC4000E device. XC4000A bitstreams must be recompiled for use with the XC4000E due to improved routing-resources, although the devices are pin-for-pin compatible.

Additional Improvements in XC4000EX Only

Increased Routing

New interconnect in the XC4000EX includes twenty-two additional vertical lines in each column of CLBs and twelve new horizontal lines in each row of CLBs. The twelve "Quad Lines" in each CLB row and column include optional repowering buffers for maximum speed. Additional high-performance routing near the IOBs enhances pin flexibility.

Faster Input and Output

A fast, dedicated early clock sourced by global clock buffers is available for the IOBs. To ensure synchronization with the regular global clocks, a Fast Capture latch driven by the early clock is available. The input data can be initially loaded into the Fast Capture latch with the early clock, then transferred to the input flip-flop or latch with the low-skew global clock. A programmable delay on the input can be used to avoid hold-time requirements. See "IOB Input Signals" on page 24 for more information.

Latch Capability in CLBs

Storage elements in the XC4000EX CLB can be configured as either flip-flops or latches. This capability makes the FPGA highly synthesis-compatible.

IOB Output MUX From Output Clock

A multiplexer in the IOB allows the output clock to select either the output data or the IOB clock enable as the output to the pad. Thus, two different data signals can share a single output pad, effectively doubling the number of device outputs without requiring a larger, more expensive package. This multiplexer can also be configured as an AND-gate to implement a very fast pin-to-pin path. See "IOB Output Signals" on page 27 for more information.

Express Configuration Mode

A new slave configuration mode accepts parallel data input. Data is processed in parallel, rather than serialized internally. Therefore, the data rate is eight times that of the six conventional configuration modes.

Additional Address Bits

Larger devices require more bits of configuration data. A daisy chain of several large XC4000EX devices may require a PROM that cannot be addressed by the eighteen address bits supported in the XC4000E. The XC4000EX family therefore extends the addressing in Master Parallel configuration mode to 22 bits.

Detailed Functional Description

XC4000-Series devices achieve high speed through advanced semiconductor technology and improved architecture. The XC4000E and XC4000EX support system clock rates of up to 66 MHz and internal performance in excess of 150 MHz. Compared to older Xilinx FPGA families, XC4000-Series devices are more powerful. They offer on-chip edge-triggered and dual-port RAM, clock enables on I/O flip-flops, and wide-input decoders. They are more versatile in many applications, especially those involving RAM. Design cycles are faster due to a combination of increased routing resources and more sophisticated software.

Basic Building Blocks

Xilinx user-programmable gate arrays include two major configurable elements: configurable logic blocks (CLBs) and input/output blocks (IOBs).

- CLBs provide the functional elements for constructing the user's logic.
- IOBs provide the interface between the package pins and internal signal lines.

Three other types of circuits are also available:

- 3-State buffers (TBUFs) driving horizontal longlines are associated with each CLB.
- Wide edge decoders are available around the periphery of each device.
- An on-chip oscillator is provided.

Programmable interconnect resources provide routing paths to connect the inputs and outputs of these configurable elements to the appropriate networks.

The functionality of each circuit block is customized during configuration by programming internal static memory cells. The values stored in these memory cells determine the logic functions and interconnections implemented in the FPGA.

Each of these available circuits is described in this section.

Configurable Logic Blocks (CLBs)

Configurable Logic Blocks implement most of the logic in an FPGA. The principal CLB elements are shown in Figure 1. The number of CLBs needed to implement selected soft macros is shown in Table 3.

Two 4-input function generators (F and G) offer unrestricted versatility. Most combinatorial logic functions need four or fewer inputs. However, a third function generator (H) is provided. The H function generator has three inputs. Either zero, one, or both of these inputs can be the outputs of F and G; the other input(s) are from outside the CLB. The CLB can, therefore, implement certain functions of up to nine variables, like parity check or expandable-identity comparison of two sets of four inputs.

Each CLB contains two storage elements that can be used to store the function generator outputs. However, the storage elements and function generators can also be used independently. These storage elements can be configured as flip-flops in both XC4000E and XC4000EX devices; in the XC4000EX they can optionally be configured as latches. DIN can be used as a direct input to either of the two storage elements. H1 can drive the other through the H function generator. Function generator outputs can also drive two outputs independent of the storage element outputs. This versatility increases logic capacity and simplifies routing.

Thirteen CLB inputs and four CLB outputs provide access to the function generators and storage elements. These inputs and outputs connect to the programmable interconnect resources outside the block.

Function Generators

Four independent inputs are provided to each of two function generators (F1 - F4 and G1 - G4). These function generators, with outputs labeled F' and G', are each capable of implementing any arbitrarily defined Boolean function of four inputs. The function generators are implemented as memory look-up tables. The propagation delay is therefore independent of the function implemented.

A third function generator, labeled H', can implement any Boolean function of its three inputs. Two of these inputs can optionally be the F' and G' functional generator outputs. Alternatively, one or both of these inputs can come from outside the CLB (H2, H0). The third input must come from outside the block (H1).

Signals from the function generators can exit the CLB on two outputs. F' or H' can be connected to the X output. G' or H' can be connected to the Y output.

A CLB can be used to implement any of the following functions:

- any function of up to four variables, plus any second function of up to four unrelated variables, plus any third function of up to three unrelated variables[1]
- any single function of five variables
- any function of four variables together with some functions of six variables
- some functions of up to nine variables.

1. When three separate functions are generated, one of the function outputs must be captured in a flip-flop internal to the CLB. Only two unregistered function generator outputs are available from the CLB.

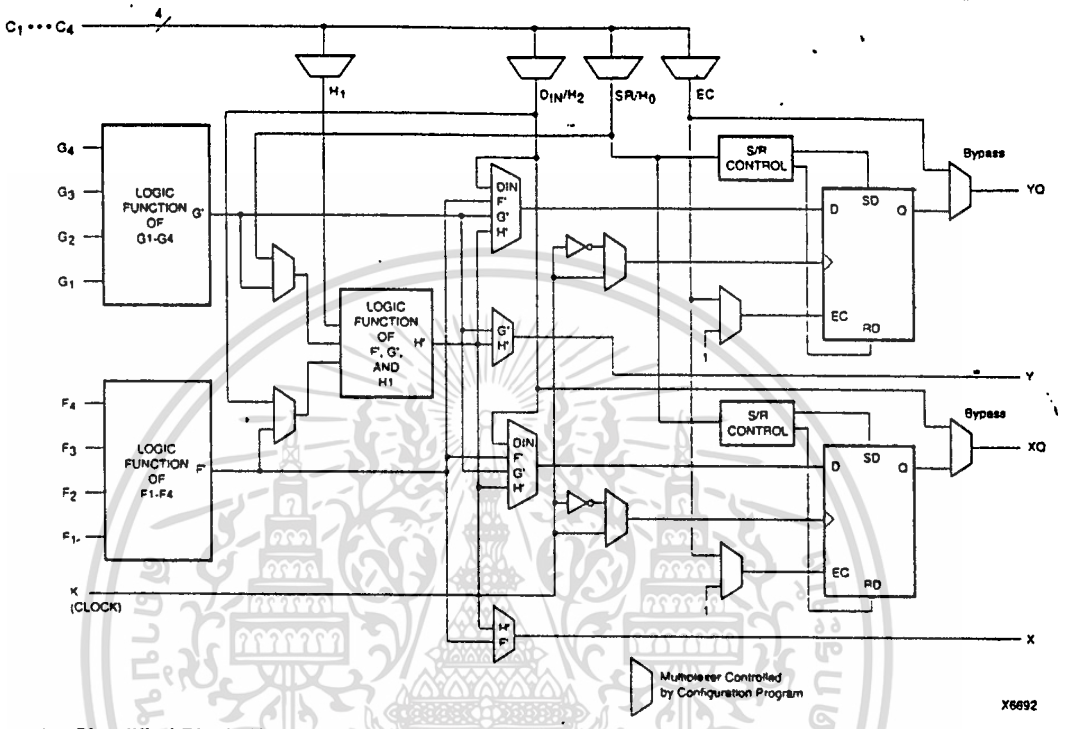


Figure 1: Simplified Block Diagram of XC4000-Series CLB (RAM and Carry Logic functions not shown)

Implementing wide functions in a single block reduces both the number of blocks required and the delay in the signal path, achieving both increased capacity and speed.

The versatility of the CLB function generators significantly improves system speed. In addition, the design-software tools can deal with each function generator independently. This flexibility improves cell usage.

Flip-Flops

The CLB can pass the combinatorial output(s) to the interconnect network, but can also store the combinatorial results or other incoming data in one or two flip-flops, and connect their outputs to the interconnect network as well.

The two edge-triggered D-type flip-flops have common clock (K) and clock enable (EC) inputs. Either or both clock inputs can also be permanently enabled. Storage element functionality is described in Table 4.

Latches (XC4000EX only)

The CLB storage elements can also be configured as latches. The two latches have common clock (K) and clock enable (EC) inputs. Storage element functionality is described in Table 4.

Table 4: CLB Storage Element Functionality (active rising edge is shown)

Mode	K	EC	SR	D	Q
Power-Up or GSR	X	X	X	X	SR
Flip-Flop	X	X	1	X	SR
	⎍	1*	0*	D	D
	0	X	0*	X	Q
Latch	1	1*	0*	X	Q
	0	1*	0*	D	D
Both	X	0	0*	X	Q

Legend:

X	Don't care
⎍	Rising edge
SR	Set or Reset value. Reset is default.
0*	Input is Low or unconnected (default value)
1*	Input is High or unconnected (default value)

Clock Input

Each flip-flop can be triggered on either the rising or falling clock edge. The clock pin is shared by both storage elements. However, the clock is individually invertible for each storage element. Any inverter placed on the clock input is automatically absorbed into the CLB.

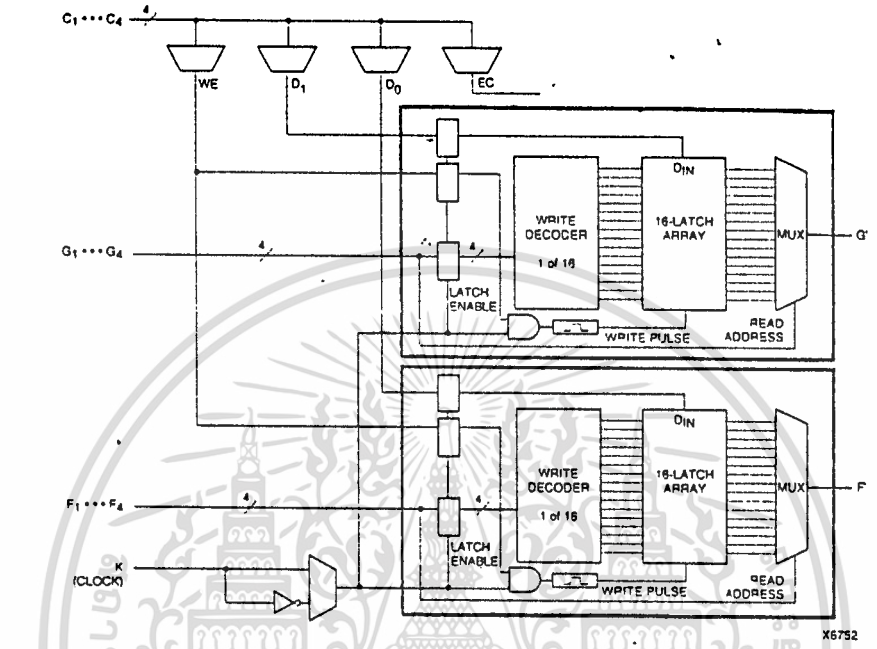


Figure 3: 16x2 (or 16x1) Edge-Triggered Single-Port RAM

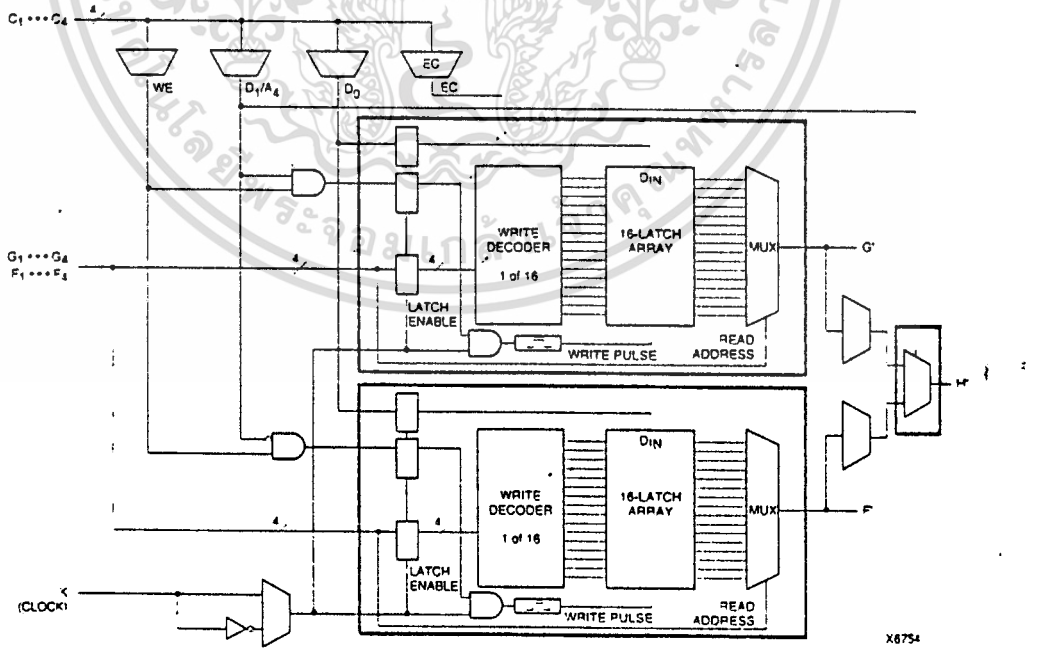


Figure 4: 32x1 Edge-Triggered Single-Port RAM (F and G addresses are identical)

RAM Inputs and Outputs

The F1-F4 and G1-G4 inputs to the function generators act as address lines, selecting a particular memory cell in each look-up table.

The functionality of the CLB control signals changes when the function generators are configured as RAM. The DIN/H2, H1, and SR/H0 lines become the two data inputs (D0, D1) and the Write Enable (WE) input for the 16x2 memory. When the 32x1 configuration is selected, D1 acts as the fifth address bit and D0 is the data input.

The contents of the memory cell(s) being addressed are available at the F' and G' function-generator outputs. They can exit the CLB through its X and Y outputs, or can be captured in the CLB flip-flop(s).

Configuring the CLB function generators as Read/Write memory does not affect the functionality of the other portions of the CLB, with the exception of the redefinition of the control signals. In 16x2 and 16x1 modes, the H' function generator can be used to implement Boolean functions of F', G', and D1, and the D flip-flops can latch the F', G', H', or D0 signals.

Single-Port Edge-Triggered Mode

Edge-triggered (synchronous) RAM simplifies timing requirements. XC4000-Series edge-triggered RAM timing operates like writing to a data register. Data and address are presented. The register is enabled for writing by a logic High on the write enable input, WE. Then a rising or falling clock edge loads the data into the register, as shown in Figure 5.

Complex timing relationships between address, data, and write enable signals are not required, and the external write enable pulse becomes a simple clock enable. The active edge of WCLK latches the address, input data, and WE signals. An internal write pulse is generated that performs the write. See Figure 3 and Figure 4 for block diagrams of a CLB configured as 16x2 and 32x1 edge-triggered, single-port RAM.

The relationships between CLB pins and RAM inputs and outputs for single-port, edge-triggered mode are shown in Table 7.

The Write Clock input (WCLK) can be configured as active on either the rising edge (default) or the falling edge. It uses the same CLB pin (K) used to clock the CLB flip-flops, but it can be independently inverted. Consequently, the RAM output can optionally be registered within the same

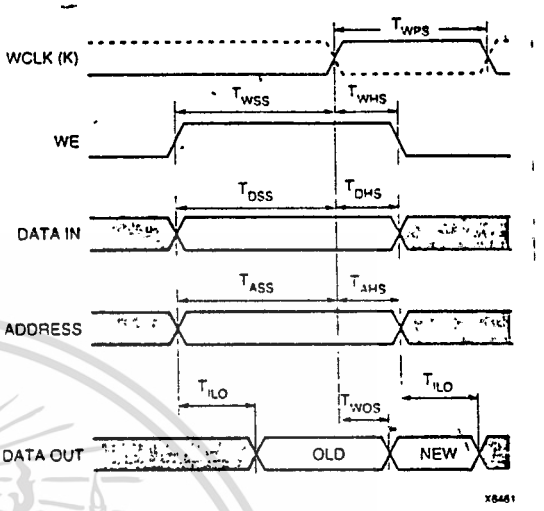


Figure 5: Edge-Triggered RAM Write Timing

CLB either by the same clock edge as the RAM, or by the opposite edge of this clock. The sense of WCLK applies to both function generators in the CLB when both are configured as RAM.

The WE pin is active-High and is not invertible within the CLB.

Note: The pulse following the active edge of WCLK (T_{WPS} in Figure 5) must be less than one millisecond wide. For most applications, this requirement is not overly restrictive; however, it must not be forgotten. Stopping WCLK at this point in the write cycle could result in excessive current and even damage to the larger devices if many CLBs are configured as edge-triggered RAM.

Table 7: Single-Port Edge-Triggered RAM Signals

RAM Signal	CLB Pin	Function
D	D0 or D1 (16x2, 16x1) D0 (32x1)	Data In
A[3:0]	F1-F4 or G1-G4	Address
A[4]	D1 (32x1)	Address
WE	WE	Write Enable
WCLK	K	Clock
SPO (Data Out)	F' or G'	Single Port Out (Data Out)

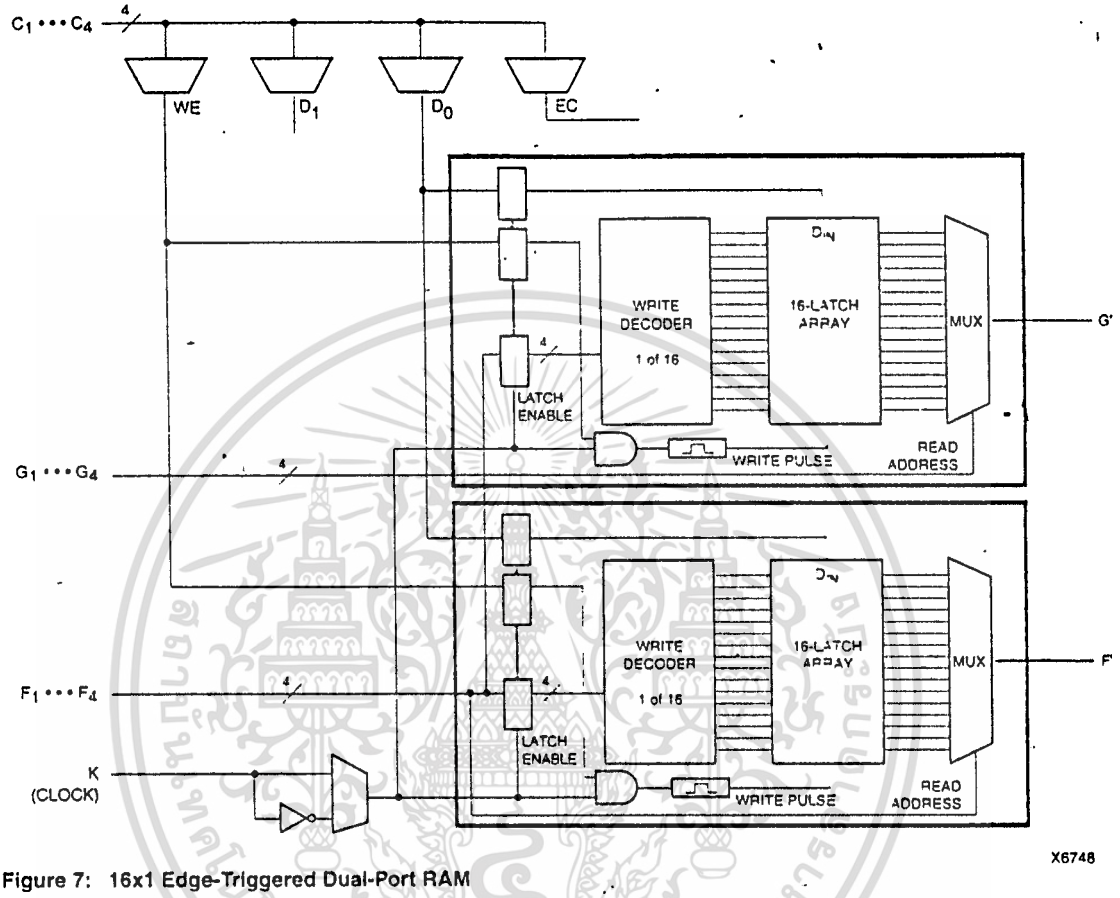


Figure 7: 16x1 Edge-Triggered Dual-Port RAM

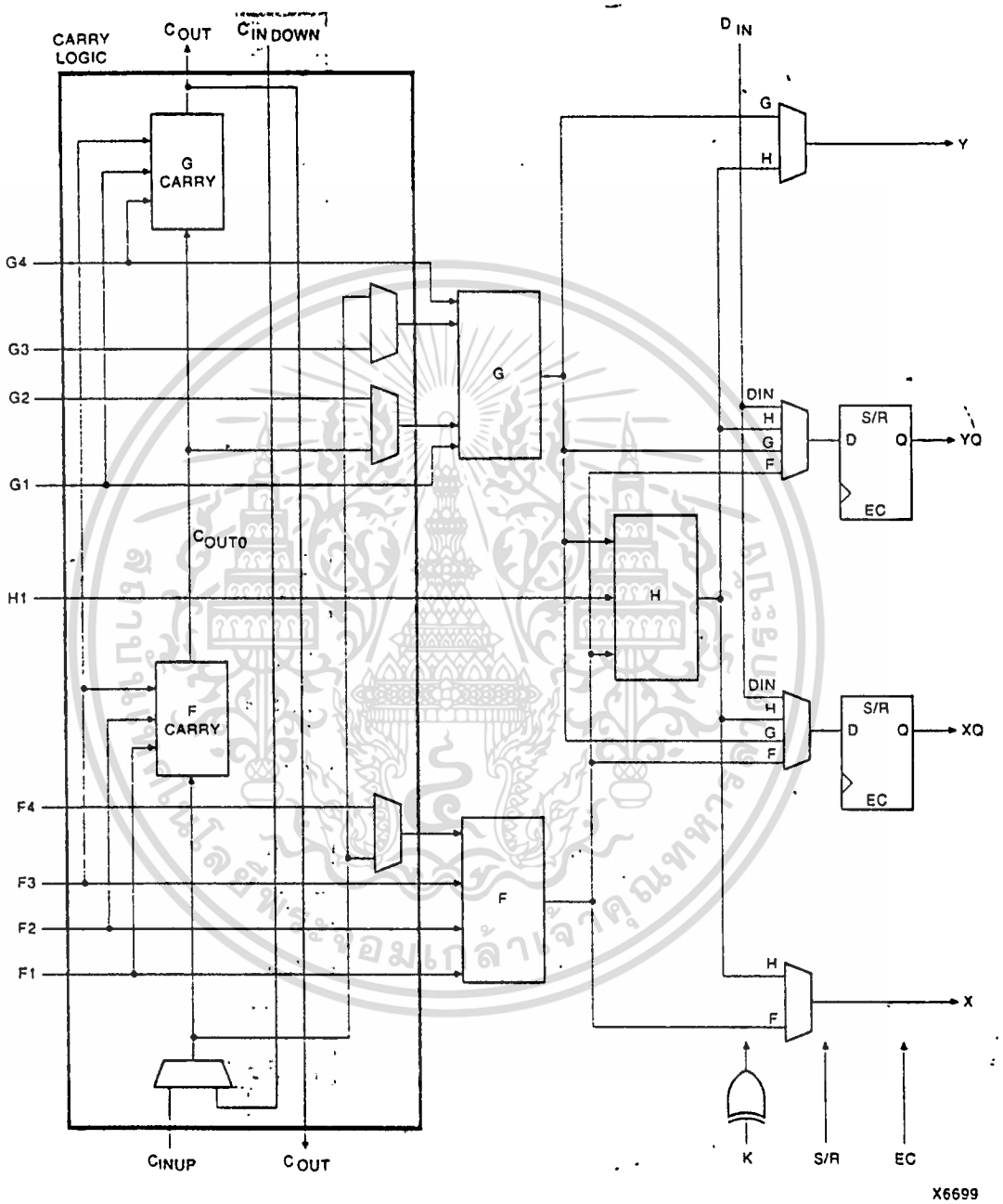


Figure 13: Fast Carry Logic in XC4000E CLB (shaded area not present in XC4000EX)

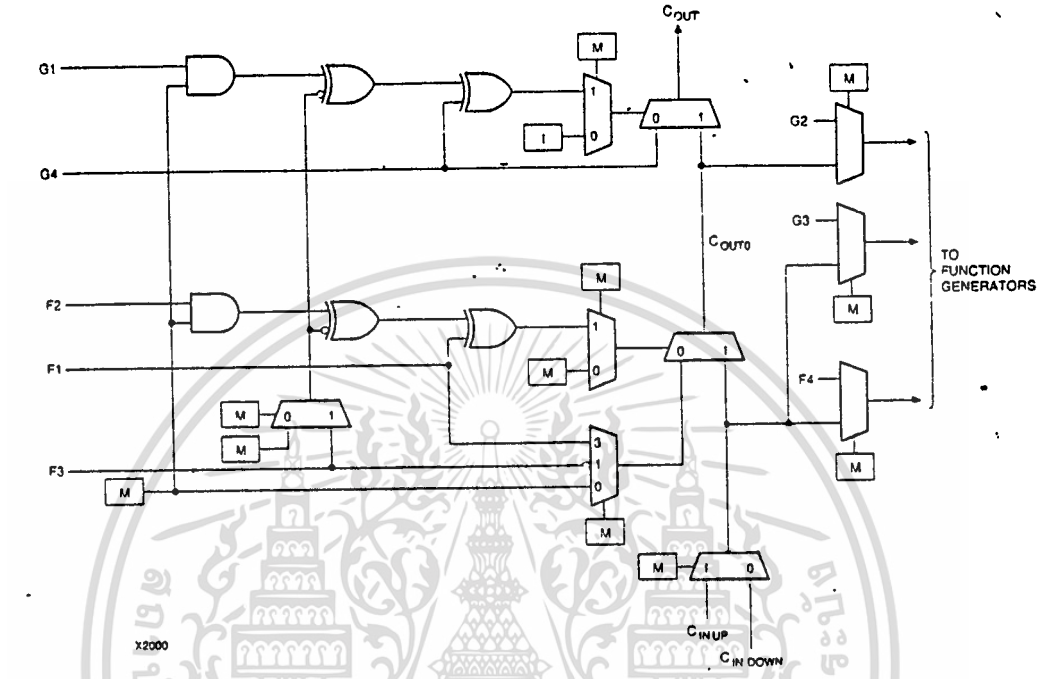


Figure 14: Detail of XC4000E Dedicated Carry Logic

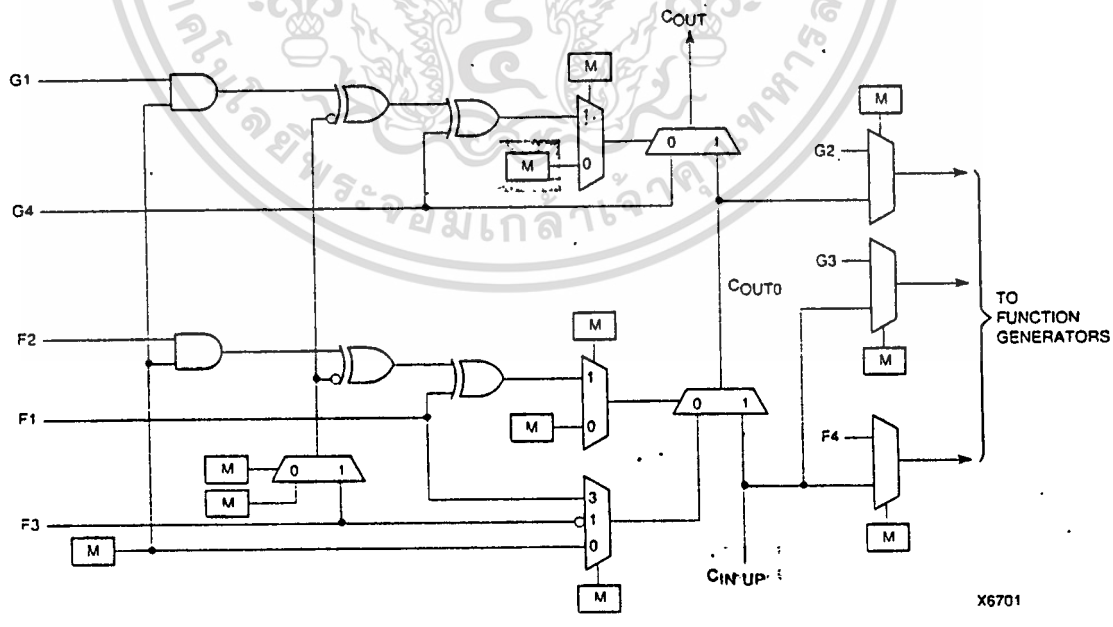


Figure 15: Detail of XC4000EX Dedicated Carry Logic (shaded areas show differences from XC4000E carry logic)

Input/Output Blocks (IOBs)

User-configurable input/output blocks (IOBs) provide the interface between external package pins and the internal logic. Each IOB controls one package pin and can be configured for input, output, or bidirectional signals.

Figure 16 shows a simplified block diagram of the XC4000E IOB. A more complete diagram of the XC4000E IOB can be found in Figure 42 on page 51, in the "Boundary Scan" section. Figure 42 includes the boundary scan logic in the IOB.

Figure 17 shows a simplified block diagram of the XC4000EX IOB. The XC4000EX IOB contains some special features not included in the XC4000E IOB. These features are highlighted in Figure 17, and discussed throughout this section. When XC4000EX special features are discussed, they are clearly identified in the text. Any feature not so identified is present in both XC4000E and XC4000EX devices.

IOB Input Signals

Two paths, labeled I1 and I2 in Figure 16 and Figure 17, bring input signals into the array. Inputs also connect to an input register that can be programmed as either an edge-triggered flip-flop or a level-sensitive latch.

The choice is made by placing the appropriate library symbol. For example, IFD is the basic input flip-flop (rising edge triggered), and ILD is the basic input latch (transparent-High). Variations with inverted clocks are available, and some combinations of latches and flip-flops can be implemented in a single IOB, as described in the *XACT Libraries Guide*.

The inputs can be globally configured for either TTL (1.2V, default) or CMOS thresholds, using an option in the MakeBits program. There is a slight hysteresis of about 300mV. The output levels are also configurable; the two global adjustments of input threshold and output level are independent.

Inputs of the low-voltage devices *must* be configured as CMOS at all times. They can be driven by the outputs of all 5-Volt XC4000-Series devices, provided that the 5-Volt outputs are in TTL mode. They can also be driven by any TTL output that does not exceed 3.7 V. 5-Volt XC3000-family device outputs, for example, are TTL-compatible, but since the output voltage can exceed 3.7 V, they cannot be used to drive an XC4000L or XC4000XL input.

The inputs of XC4000-Series 5-Volt devices can be driven by the outputs of any 3.3-Volt device, if the 5-Volt inputs are in TTL mode.

Supported sources for XC4000-Series device inputs are shown in Table 10.

Table 10: Supported Sources for XC4000-Series Device Inputs

Source	XC4000-Series Inputs		
	3.3 V, CMOS	5 V, TTL	5 V, CMOS
Any device, Vcc = 3.3 V, CMOS outputs	√	√	Unreliable Data
XC4000-Series, Vcc = 5 V, TTL outputs	√	√	
Any device, Vcc = 5 V, TTL outputs (Voh ≤ 3.7 V)	√	√	
Any device, Vcc = 5 V, CMOS outputs	Danger[1]		√

1. Acceptable for XC4000XL if the designated 5-Volt supply pad (V_{TT}) is tied to 5V.

Registered Inputs

The I1 and I2 signals that exit the block can each carry either the direct or registered input signal.

The input and output storage elements in each IOB have a common clock enable input, which, through configuration, can be activated individually for the input or output flip-flop, or both. This clock enable operates exactly like the EC pin on the XC4000-Series CLB. It cannot be inverted within the IOB.

The storage element behavior is shown in Table 11.

Table 11: Input Register Functionality (active rising edge is shown)

Mode	Clock	Clock Enable	D	Q
Power-Up or GSR	X	X	X	SR
Flip-Flop	⎍/⎺	1*	D	D
	0	X	X	Q
Latch	1	1*	X	Q
	0	1*	D	D
Both	X	0	X	Q

Legend:

X	Don't care
⎍/⎺	Rising edge
SR	Set or Reset value. Reset is default.
0*	Input is Low or unconnected (default value)
1*	Input is High or unconnected (default value)

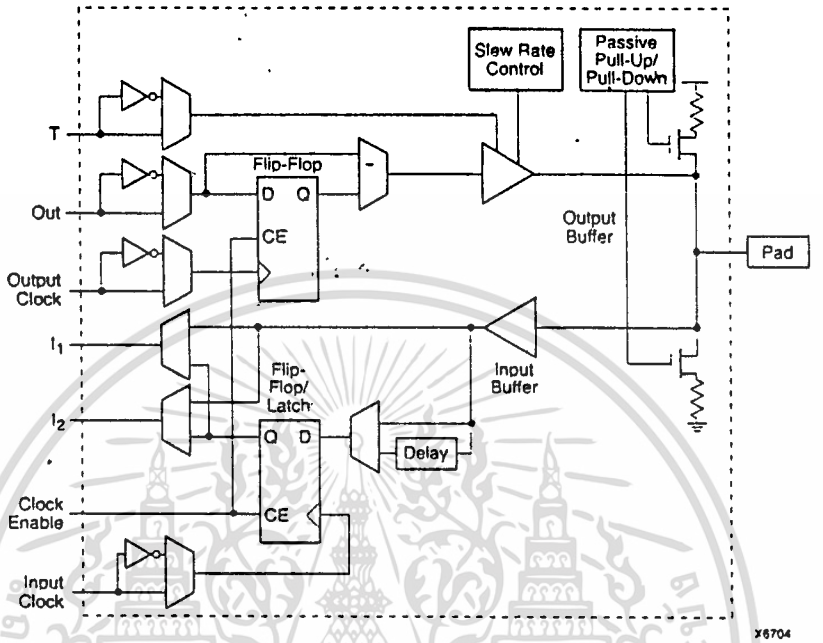


Figure 16: Simplified Block Diagram of XC4000E IOB

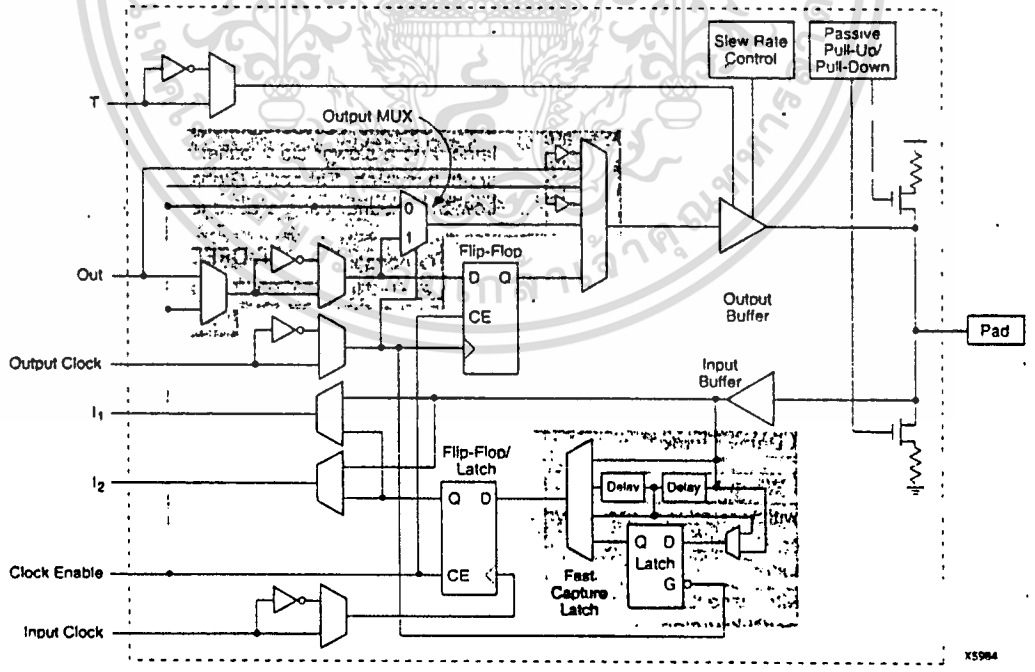


Figure 17: Simplified Block Diagram of XC4000EX IOB (shaded areas indicate differences from XC4000E)

Table 18: Pin Descriptions

Pin Name	I/O During Config.	I/O After Config.	Pin Description
Permanently Dedicated Pins			
VCC	I	I	Eight or more (depending on package) connections to the nominal +5 V supply voltage (+3.3 V for low-voltage devices). All must be connected, and each must be decoupled with a 0.01 - 0.1 µF capacitor to Ground.
GND	I	I	Eight or more (depending on package type) connections to Ground. All must be connected.
CCLK	I or O	I	During configuration, Configuration Clock (CCLK) is an output in Master modes or Asynchronous Peripheral mode, but is an input in Slave mode, Synchronous Peripheral mode, and Express mode. After configuration, CCLK has a weak pull-up resistor and can be selected as the Readback Clock. There is no CCLK High time restriction on XC4000-Series devices, except during Readback. See "Violating the Maximum High and Low Time Specification for the Readback Clock" on page 65 for an explanation of this exception.
DONE	I/O	O	DONE is a bidirectional signal with an optional internal pull-up resistor. As an output, it indicates the completion of the configuration process. As an input, a Low level on DONE can be configured to delay the global logic initialization and the enabling of outputs. The optional pull-up resistor is selected as an option in MakeBits, the XACT*step* program that creates the configuration bitstream. The resistor is included by default.
$\overline{\text{PROGRAM}}$	I	I	$\overline{\text{PROGRAM}}$ is an active Low input that forces the FPGA to clear its configuration memory. It is used to initiate a configuration cycle. When $\overline{\text{PROGRAM}}$ goes High, the FPGA finishes the current clear cycle and executes another complete clear cycle, before it goes into a WAIT state and releases $\overline{\text{INIT}}$. The $\overline{\text{PROGRAM}}$ pin has a permanent weak pull-up, so it need not be externally pulled up to Vcc.
User I/O Pins That Can Have Special Functions			
RDY/$\overline{\text{BUSY}}$	O	I/O	During Peripheral mode configuration, this pin indicates when it is appropriate to write another byte of data into the FPGA. The same status is also available on D7 in Asynchronous Peripheral mode, if a read operation is performed when the device is selected. After configuration, RDY/$\overline{\text{BUSY}}$ is a user-programmable I/O pin. RDY/$\overline{\text{BUSY}}$ is pulled High with a high-impedance pull-up prior to $\overline{\text{INIT}}$ going High.
$\overline{\text{RCLK}}$	O	I/O	During Master Parallel configuration, each change on the A0-A17 outputs (A0 - A21 for XC4000EX) is preceded by a rising edge on $\overline{\text{RCLK}}$, a redundant output signal. $\overline{\text{RCLK}}$ is useful for clocked PROMs. It is rarely used during configuration. After configuration, $\overline{\text{RCLK}}$ is a user-programmable I/O pin.
M0, M1, M2	I	I (M0), O (M1), I (M2)	As Mode inputs, these pins are sampled after $\overline{\text{INIT}}$ goes High to determine the configuration mode to be used. After configuration, M0 and M2 can be used as inputs, and M1 can be used as a 3-state output. These three pins have no associated input or output registers. During configuration, these pins have weak pull-up resistors. For the most popular configuration mode, Slave Serial, the mode pins can thus be left unconnected. The three mode inputs can be individually configured with or without weak pull-up or pull-down resistors. A pull-down resistor value of 4.7 kΩ is recommended. These pins can only be used as inputs or outputs when called out by special schematic definitions. To use these pins, place the library components MD0, MD1, and MD2 instead of the usual pad symbols. Input or output buffers must still be used.

Table 18: Pin Descriptions (Continued)

Pin Name	I/O During Config.	I/O After Config.	Pin Description
TDO	O	O	If boundary scan is used, this pin is the Test Data Output. If boundary scan is not used, this pin is a 3-state output without a register, after configuration is completed. This pin can be user output only when called out by special schematic definitions. To use this pin, place the library component TDO instead of the usual pad symbol. An output buffer must still be used.
TDI, TCK, TMS	I	I/O or I (JTAG)	If boundary scan is used, these pins are Test Data In, Test Clock, and Test Mode Select inputs respectively. They come directly from the pads, bypassing the IOBs. These pins can also be used as inputs to the CLB logic after configuration is completed. If the BSCAN symbol is not placed in the design, all boundary scan functions are inhibited once configuration is completed, and these pins become user-programmable I/O. In this case, they must be called out by special schematic definitions. To use these pins, place the library components TDI, TCK, and TMS instead of the usual pad symbols. Input or output buffers must still be used.
HDC	O	I/O	High During Configuration (HDC) is driven High until the I/O go active. It is available as a control output indicating that configuration is not yet completed. After configuration, HDC is a user-programmable I/O pin.
$\overline{\text{LDC}}$	O	I/O	Low During Configuration ($\overline{\text{LDC}}$) is driven Low until the I/O go active. It is available as a control output indicating that configuration is not yet completed. After configuration, $\overline{\text{LDC}}$ is a user-programmable I/O pin.
$\overline{\text{INIT}}$	I/O	I/O	Before and during configuration, $\overline{\text{INIT}}$ is a bidirectional signal. A 1 kΩ - 10 kΩ external pull-up resistor is recommended. As an active-Low open-drain output, $\overline{\text{INIT}}$ is held Low during the power stabilization and internal clearing of the configuration memory. As an active-Low input, it can be used to hold the FPGA in the internal WAIT state before the start of configuration. Master mode devices stay in a WAIT state an additional 30 to 300 µs after $\overline{\text{INIT}}$ has gone High. During configuration, a Low on this output indicates that a configuration data error has occurred. After the I/O go active, $\overline{\text{INIT}}$ is a user-programmable I/O pin.
PGCK1 - PGCK4 (XC4000E only)	Weak Pull-up	I or I/O	Four Primary Global inputs each drive a dedicated internal global net with short delay and minimal skew. If not used to drive a global buffer, any of these pins is a user-programmable I/O. The PGCK1-PGCK4 pins drive the four Primary Global Buffers. Any input pad symbol connected directly to the input of a BUFGP symbol is automatically placed on one of these pins.
SGCK1 - SGCK4 (XC4000E only)	Weak Pull-up	I or I/O	Four Secondary Global inputs each drive a dedicated internal global net with short delay and minimal skew. These internal global nets can also be driven from internal logic. If not used to drive a global net, any of these pins is a user-programmable I/O pin. The SGCK1-SGCK4 pins provide the shortest path to the four Secondary Global Buffers. Any input pad symbol connected directly to the input of a BUFGS symbol is automatically placed on one of these pins.
GCK1 - GCK8 (XC4000EX only)	Weak Pull-up	I or I/O	Eight inputs can each drive a Global Low-Skew buffer. In addition, each can drive a Global Early buffer. Each pair of global buffers can also be driven from internal logic, but must share an input signal. If not used to drive a global buffer, any of these pins is a user-programmable I/O. Any input pad symbol connected directly to the input of a BUFGLS or BUFGE symbol is automatically placed on one of these pins.
FCLK1 - FCLK4 (XC4000EX only)	Weak Pull-up	I or I/O	Four FCLK inputs can each drive a FastCLK buffer. The FastCLK buffers cannot be driven from internal logic. If not used to drive a global buffer, any of these pins is a user-programmable I/O. Any input pad symbol connected directly to the input of a BUFFCLK symbol is automatically placed on one of these pins.

Table 18: Pin Descriptions (Continued)

Pin Name	I/O During Config.	I/O After Config.	Pin Description
$\overline{CS0}$, CS1, $\overline{WS}$, $\overline{RS}$	I	I/O	These four inputs are used in Asynchronous Peripheral mode. The chip is selected when $\overline{CS0}$ is Low and CS1 is High. While the chip is selected, a Low on Write Strobe ($\overline{WS}$) loads the data present on the D0 - D7 inputs into the internal data buffer. A Low on Read Strobe ($\overline{RS}$) changes D7 into a status output — High if Ready, Low if Busy — and drives D0 - D6 High. In Express mode, CS1 is used as a serial-enable signal for daisy-chaining. $\overline{WS}$ and $\overline{RS}$ should be mutually exclusive, but if both are Low simultaneously, the Write Strobe overrides. After configuration, these are user-programmable I/O pins.
A0 - A17	O	I/O	During Master Parallel configuration, these 18 output pins address the configuration EPROM. After configuration, they are user-programmable I/O pins.
A18 - A21 (XC4000EX only)	O	I/O	During Master Parallel configuration with an XC4000EX master, these 4 output pins add 4 more bits to address the configuration EPROM. After configuration, they are user-programmable I/O pins.
D0 - D7	I	I/O	During Master Parallel and Peripheral configuration, these eight input pins receive configuration data. After configuration, they are user-programmable I/O pins.
DIN	I	I/O	During Slave Serial or Master Serial configuration, DIN is the serial configuration data input receiving data on the rising edge of CCLK. During Parallel configuration, DIN is the D0 input. After configuration, DIN is a user-programmable I/O pin.
DOUT	O	I/O	During configuration in any mode but Express mode, DOUT is the serial configuration data output that can drive the DIN of daisy-chained slave FPGAs. DOUT data changes on the falling edge of CCLK, one-and-a-half CCLK periods after it was received at the DIN input. In Express mode, DOUT is the status output that can drive the CS1 of daisy-chained FPGAs, to enable and disable downstream devices. After configuration, DOUT is a user-programmable I/O pin.
Unrestricted User-Programmable I/O Pins			
I/O	Weak Pull-up	I/O	These pins can be configured to be input and/or output after configuration is completed. Before configuration is completed, these pins have an internal high-value pull-up resistor (50 kΩ - 100 kΩ) that defines the logic level as High.

Configuration Timing

The seven configuration modes are discussed in detail in this section. Timing specifications are included.

Master Serial Mode

In Master Serial mode, the CCLK output of the lead FPGA drives a Xilinx Serial PROM that feeds the FPGA DIN input. Each rising edge of the CCLK output increments the Serial PROM internal address counter. The next data bit is put on the SPROM data output, connected to the FPGA DIN pin. The lead FPGA accepts this data on the subsequent rising CCLK edge.

The lead FPGA then presents the preamble data—and all data that overflows the lead device—on its DOUT pin. There is an internal pipeline delay of 1.5 CCLK periods, which means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

In MakeBits, the user can specify Fast ConfigRate, which, starting several bits into the first frame, increases the CCLK frequency by a factor of eight. The value increases from between 0.5 and 1.25 MHz, to a value between 4 and 10 MHz. (For low-voltage devices, the frequency can be up to 10% lower.) Be sure that the serial PROM and slaves are fast enough to support this data rate. XC2000, XC3000/A, and XC3100A devices do not support the Fast ConfigRate option.

The SPROM CE input can be driven from either $\overline{LDC}$ or DONE. Using $\overline{LDC}$ avoids potential contention on the DIN pin, if this pin is configured as user-I/O, but $\overline{LDC}$ is then restricted to be a permanently High user output after configuration. Using DONE can also avoid contention on DIN, provided the early DONE option is invoked.

Master Serial mode is selected by a <000> on the mode pins (M2, M1, M0).

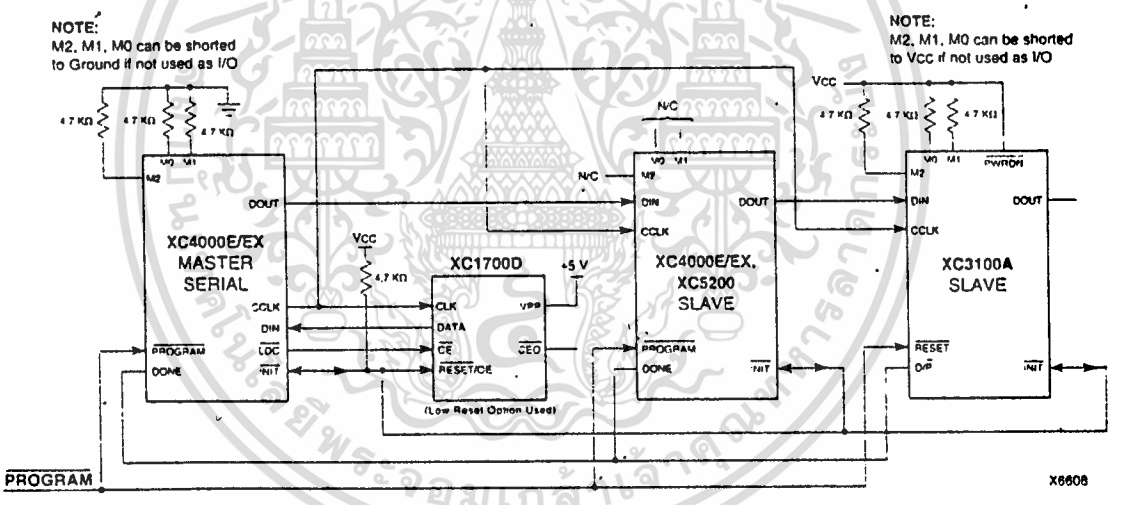
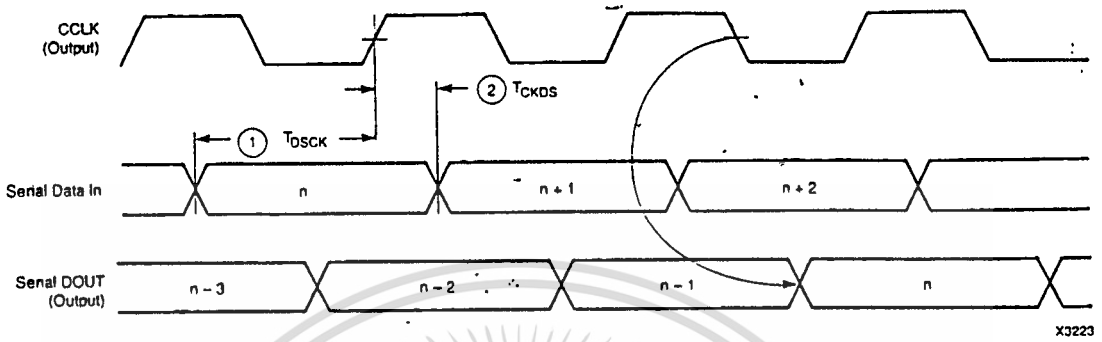


Figure 53: Master Serial Mode Circuit Diagram



	Description		Symbol	Min	Max	Units
CCLK	DIN setup	1	T_{DSCK}	20		ns
	DIN hold	2	T_{CKDS}	0		ns

Notes: 1. At power-up, Vcc must rise from 2.0 V to Vcc min in less than 25 ms, otherwise delay configuration by pulling $\overline{\text{PROGRAM}}$ Low until Vcc is valid.
2. Master Serial mode timing is based on testing in slave mode.

Figure 54: Master Serial Mode Programming Switching Characteristics

Slave Serial Mode

In Slave Serial mode, an external signal drives the CCLK input of the FPGA. The serial configuration bitstream must be available at the DIN input of the lead FPGA a short setup time before each rising CCLK edge.

The lead FPGA then presents the preamble data—and all data that overflows the lead device—on its DOUT pin. There is an internal delay of 0.5 CCLK periods, which means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

Slave Serial mode is selected by a <111> on the mode pins (M2, M1, M0). Slave Serial is the default mode if the mode pins are left unconnected, as they have weak pull-up resistors during configuration.

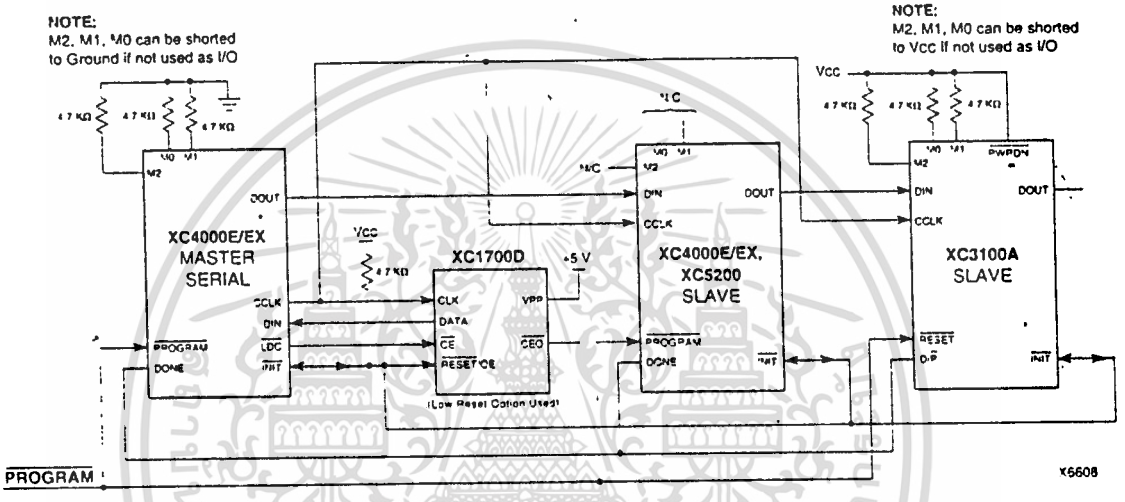
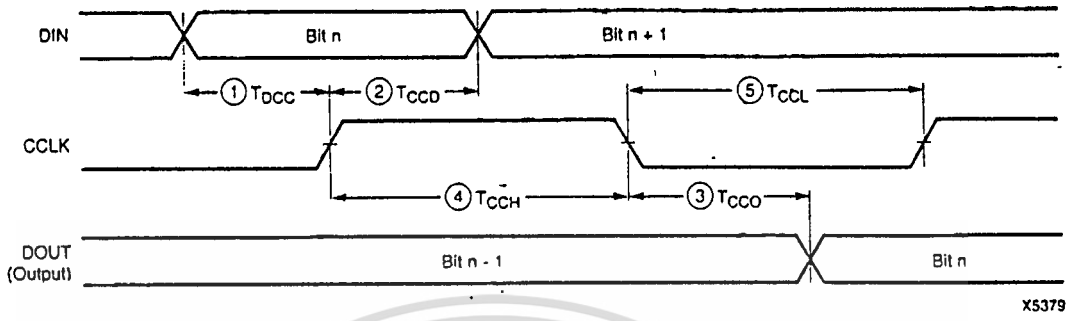


Figure 55: Slave Serial Mode Circuit Diagram



	Description	Symbol		Min	Max	Units
CCLK	DIN setup	1	T_{DCC}	20		ns
	DIN hold	2	T_{CCD}	0		ns
	DIN to DOUT	3	T_{CCO}		30	ns
	High time	4	T_{CCH}	45		ns
	Low time	5	T_{CCL}	45		ns
	Frequency		F_{CC}		10	MHz

Note: Configuration must be delayed until the $\overline{INIT}$ pins of all daisy-chained FPGAs are High.

Figure 56: Slave Serial Mode Programming Switching Characteristics

Master Parallel Modes

In the two Master Parallel modes, the lead FPGA directly addresses an industry-standard byte-wide EPROM, and accepts eight data bits just before incrementing or decrementing the address outputs.

The eight data bits are serialized in the lead FPGA, which then presents the preamble data—and all data that overflows the lead device—on its DOUT pin. There is an internal delay of 1.5 CCLK periods, after the rising CCLK edge that accepts a byte of data (and also changes the EPROM address) until the falling CCLK edge that makes the LSB (D0) of this byte appear at DOUT. This means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

The PROM address pins can be incremented or decremented, depending on the mode pin settings. This option allows the FPGA to share the PROM with a wide variety of microprocessors and microcontrollers. Some processors must boot from the bottom of memory (all zeros) while others must boot from the top. The FPGA is flexible and can load its configuration bitstream from either end of the memory.

Master Parallel Up mode is selected by a <100> on the mode pins (M2, M1, M0). The EPROM addresses start at 00000 and increment.

Master Parallel Down mode is selected by a <110> on the mode pins. The EPROM addresses start at 3FFFF and decrement.

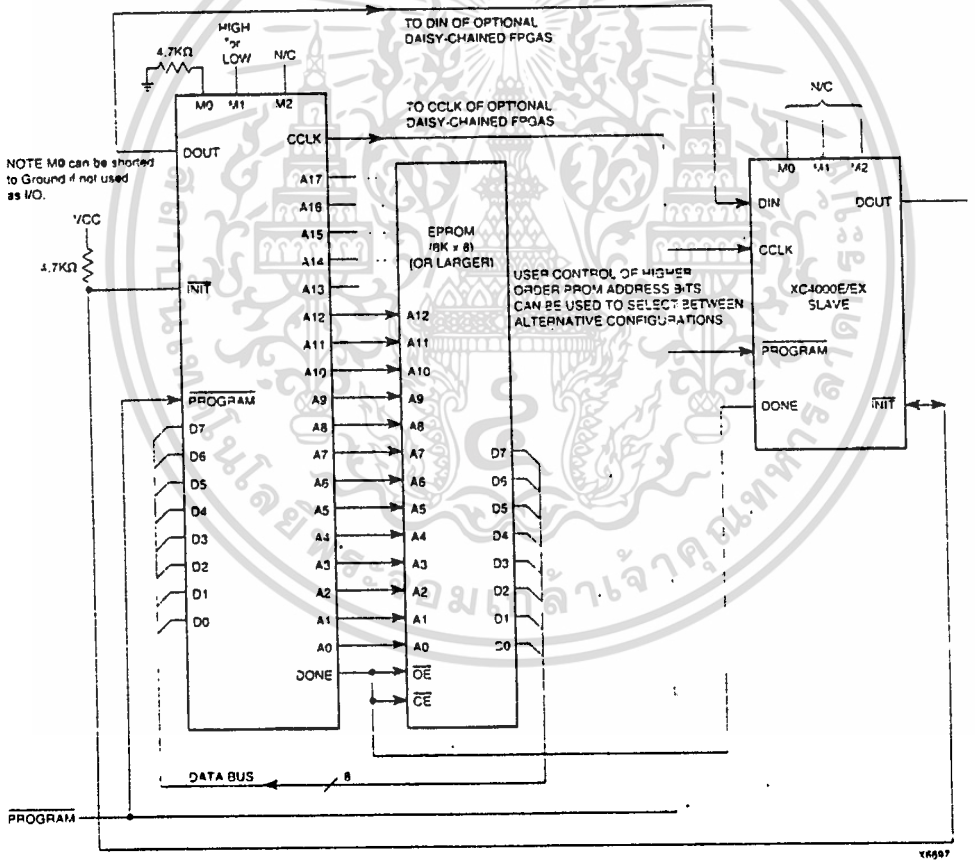


Figure 57: Master Parallel Mode Circuit Diagram

Synchronous Peripheral Mode

Synchronous Peripheral mode can also be considered Slave Parallel mode. An external signal drives the CCLK input(s) of the FPGA(s). The first byte of parallel configuration data must be available at the Data inputs of the lead FPGA a short setup time before the rising CCLK edge. Subsequent data bytes are clocked in on every eighth consecutive rising CCLK edge.

The same CCLK edge that accepts data, also causes the RDY/$\overline{\text{BUSY}}$ output to go High for one CCLK period. The pin name is a misnomer. In Synchronous Peripheral mode it is really an ACKNOWLEDGE signal. Synchronous operation does not require this response, but it is a meaningful signal for test purposes. Note that RDY/$\overline{\text{BUSY}}$ is pulled High with a high-impedance pullup prior to $\overline{\text{INIT}}$ going High.

The lead FPGA serializes the data and presents the preamble data (and all data that overflows the lead device) on its DOUT pin. There is an internal delay of 1.5 CCLK periods, which means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

In order to complete the serial shift operation, 10 additional CCLK rising edges are required after the last data byte has been loaded, plus one more CCLK cycle for each daisy-chained device.

Synchronous Peripheral mode is selected by a <011> on the mode pins (M2, M1, M0).

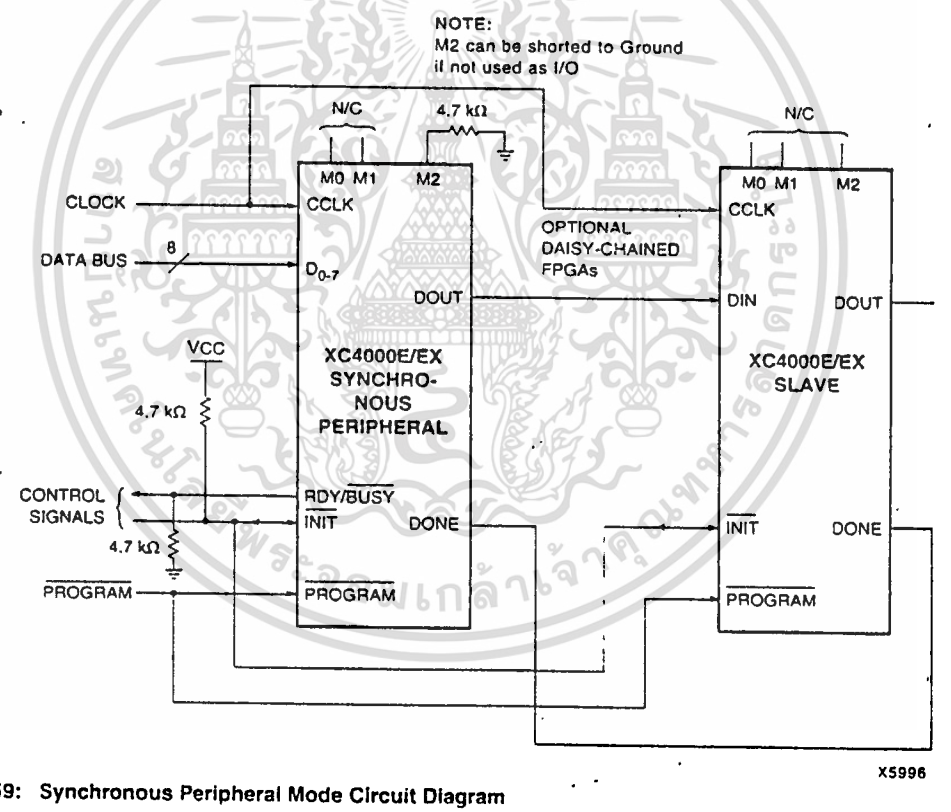
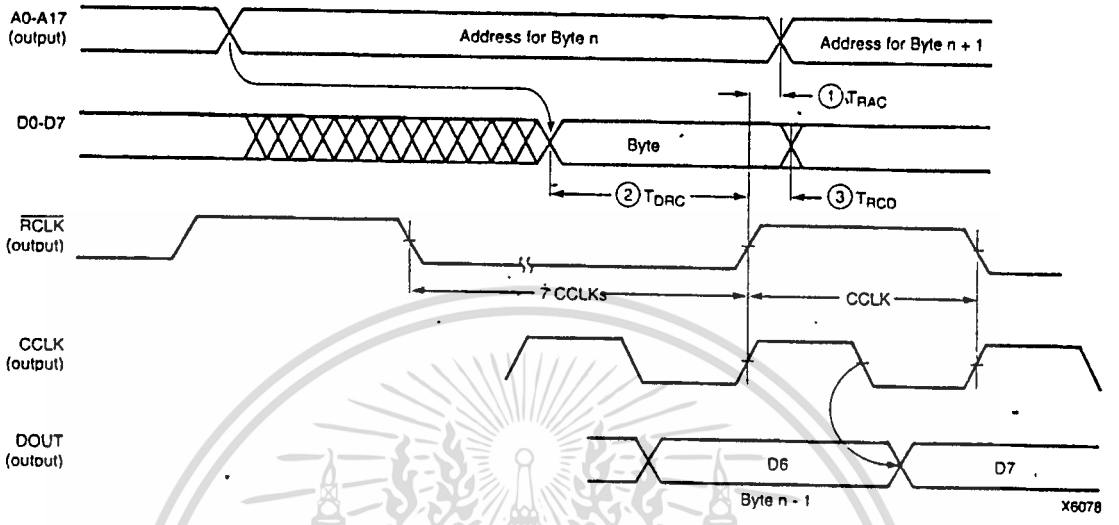


Figure 59: Synchronous Peripheral Mode Circuit Diagram

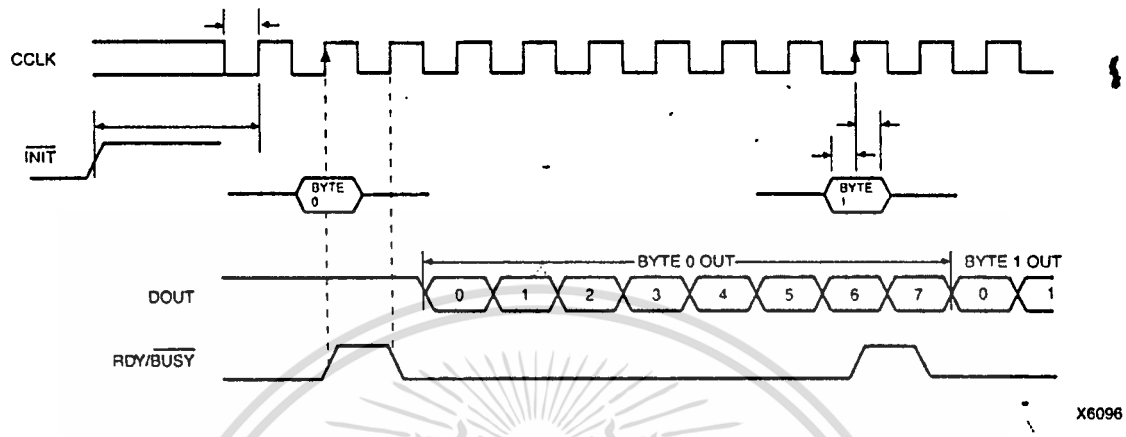


	Description		Symbol	Min	Max	Units
RCLK	Delay to Address valid	1	T_{RAC}	0	200	ns
	Data setup time	2	T_{DRC}	60		ns
	Data hold time	3	T_{RCD}	0		ns

Notes: 1. At power-up, Vcc must rise from 2.0 V to Vcc min in less than 25 ms, otherwise delay configuration by pulling $\overline{\text{PROGRAM}}$ Low until Vcc is valid.
2. The first Data byte is loaded and CCLK starts at the end of the first $\overline{\text{RCLK}}$ active cycle (rising edge).

This timing diagram shows that the EPROM requirements are extremely relaxed. EPROM access time can be longer than 500 ns. EPROM data output has no hold-time requirements.

Figure 58: Master Parallel Mode Programming Switching Characteristics



	Description	Symbol	Min	Max	Units
CCLK	INIT (High) setup time	T_{IC}	5		µs
	D0 - D7 setup time	T_{DC}	60		ns
	D0 - D7 hold time	T_{CD}	0		ns
	CCLK High time	T_{CCH}	50		ns
	CCLK Low time	T_{CCL}	60		ns
	CCLK Frequency	F_{CC}		8	MHz

Notes:
1. Peripheral Synchronous mode can be considered Slave Parallel mode. An external CCLK provides timing, clocking in the first data byte on the second rising edge of CCLK after INIT goes High. Subsequent data bytes are clocked in on every eighth consecutive rising edge of CCLK.
2. The RDY/BUSY line goes High for one CCLK period after data has been clocked in, although synchronous operation does not require such a response.
3. The pin name RDY/BUSY is a misnomer. In Synchronous Peripheral mode this is really an ACKNOWLEDGE signal.
4. Note that data starts to shift out serially on the DOUT pin 0.5 CCLK periods after it was loaded in parallel. Therefore, additional CCLK pulses are clearly required after the last byte has been loaded.

Figure 60: Synchronous Peripheral Mode Programming Switching Characteristics

Asynchronous Peripheral Mode

Write to FPGA

Asynchronous Peripheral mode uses the trailing edge of the logic AND condition of $\overline{WS}$ and $\overline{CS0}$ being Low and $\overline{RS}$ and CS1 being High to accept byte-wide data from a microprocessor bus. In the lead FPGA, this data is loaded into a double-buffered UART-like parallel-to-serial converter and is serially shifted into the internal logic.

The lead FPGA presents the preamble data (and all data that overflows the lead device) on its DOUT pin. The RDY/$\overline{BUSY}$ output from the lead FPGA acts as a handshake signal to the microprocessor. RDY/$\overline{BUSY}$ goes Low when a byte has been received, and goes High again when the byte-wide input buffer has transferred its information into the shift register, and the buffer is ready to receive new data. A new write may be started immediately, as soon as the RDY/$\overline{BUSY}$ output has gone Low, acknowledging receipt of the previous data. Write may not be terminated until RDY/$\overline{BUSY}$ is High again for one CCLK period. Note that RDY/$\overline{BUSY}$ is pulled High with a high-impedance pull-up prior to $\overline{INIT}$ going High.

The length of the $\overline{BUSY}$ signal depends on the activity in the UART. If the shift register was empty when the new byte was received, the $\overline{BUSY}$ signal lasts for only two CCLK periods. If the shift register was still full when the new byte was received, the $\overline{BUSY}$ signal can be as long as nine CCLK periods.

Note that after the last byte has been entered, only seven of its bits are shifted out. CCLK remains High with DOUT equal to bit 6 (the next-to-last bit) of the last byte entered.

The READY/$\overline{BUSY}$ handshake can be ignored if the delay from any one Write to the end of the next Write is guaranteed to be longer than 10 CCLK periods.

Status Read

The logic AND condition of the $\overline{CS0}$, CS1 and $\overline{RS}$ inputs puts the device status on the Data bus.

- D7 High indicates Ready
- D7 Low indicates Busy
- D0 through D6 go unconditionally High

It is mandatory that the whole start-up sequence be started and completed by one byte-wide input. Otherwise, the pins used as Write Strobe or Chip Enable might become active outputs and interfere with the final byte transfer. If this transfer does not occur, the start-up sequence is not completed all the way to the finish (point F in Figure 49 on page 61).

In this case, at worst, the internal reset is not released. At best, Readback and Boundary Scan are inhibited. The length-count value, as generated by MakeBits and MakePROM, ensures that these problems never occur.

Although RDY/$\overline{BUSY}$ is brought out as a separate signal, microprocessors can more easily read this information on one of the data lines. For this purpose, D7 represents the RDY/$\overline{BUSY}$ status when $\overline{RS}$ is Low, $\overline{WS}$ is High, and the two chip select lines are both active.

Asynchronous Peripheral mode is selected by a <101> on the mode pins (M2, M1, M0).

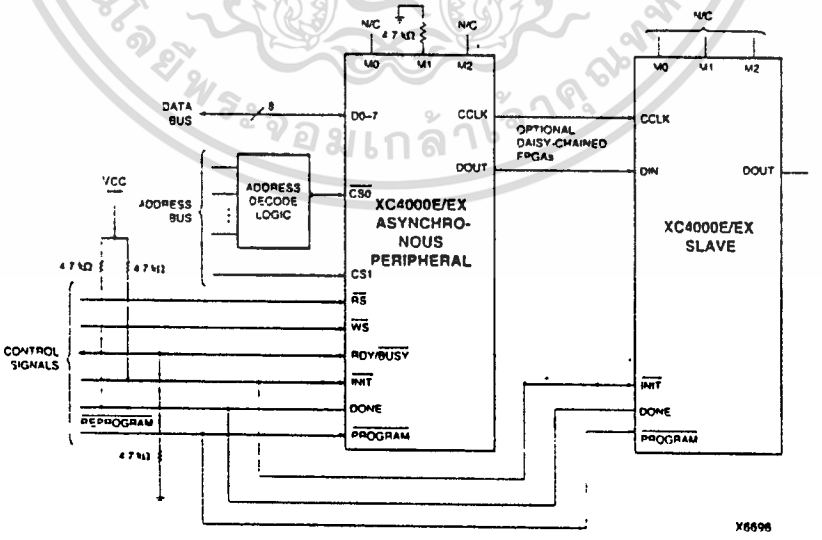
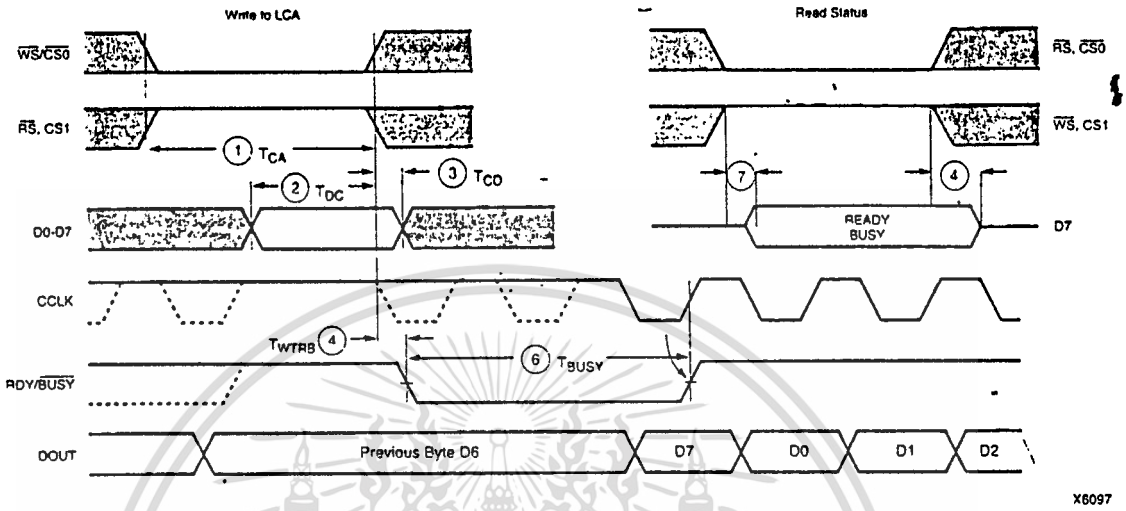


Figure 61: Asynchronous Peripheral Mode Circuit Diagram



	Description	Symbol		Min	Max	Units
Write	Effective Write time ($\overline{CS0}$, $\overline{WS}$=Low; $\overline{RS}$, CS1=High)	1	T_{CA}	100		ns
	DIN setup time	2	T_{DC}	60		ns
	DIN hold time	3	T_{CD}	0		ns
RDY	RDY/$\overline{BUSY}$ delay after end of Write or Read	4	T_{WTRB}		60	ns
	RDY/$\overline{BUSY}$ active after beginning of Read	7			60	ns
	RDY/$\overline{BUSY}$ Low output (Note 4)	6	T_{BUSY}	2	9	CCLK periods

Notes:
1. Configuration must be delayed until the $\overline{INIT}$ pins of all daisy-chained FPGAs are High.
2. The time from the end of $\overline{WS}$ to CCLK cycle for the new byte of data depends on the completion of previous byte processing and the phase of the internal timing generator for CCLK.
3. CCLK and DOUT timing is tested in slave mode.
4. T_{BUSY} indicates that the double-buffered parallel-to-serial converter is not yet ready to receive new data. The shortest T_{BUSY} occurs when a byte is loaded into an empty parallel-to-serial converter. The longest T_{BUSY} occurs when a new word is loaded into the input register before the second-level buffer has started shifting out data.

This timing diagram shows very relaxed requirements. Data need not be held beyond the rising edge of $\overline{WS}$. RDY/$\overline{BUSY}$ will go active within 60 ns after the end of $\overline{WS}$. A new write may be asserted immediately after RDY/$\overline{BUSY}$ goes Low, but write may not be terminated until RDY/$\overline{BUSY}$ has been High for one CCLK period.

Figure 62: Asynchronous Peripheral Mode Programming Switching Characteristics

Express Mode (XC4000EX only)

Express mode is similar to Slave Serial mode, except that data is processed one byte per CCLK cycle instead of one bit per CCLK cycle. An external source is used to drive CCLK, while byte-wide data is loaded directly into the configuration data shift registers. A CCLK frequency of 1 MHz is equivalent to a 8 MHz serial rate, because eight bits of configuration data are loaded per CCLK cycle. Express mode does not support CRC error checking, but does support constant-field error checking.

In Express mode, an external signal drives the CCLK input of the FPGA device. The first byte of parallel configuration data must be available at the D inputs of the FPGA a short setup time before the second rising CCLK edge. Subsequent data bytes are clocked in on each consecutive rising CCLK edge.

Express mode is only supported by the XC4000EX and XC5200 families. It may not be used, therefore, when an XC4000EX or XC5200 device is daisy-chained with devices from other Xilinx families.

If the first device is configured in Express mode, additional devices may be daisy-chained only if every device in the chain is also configured in Express mode. CCLK pins are tied together and D0-D7 pins are tied together for all devices along the chain. A status signal is passed from DOUT to CS1 of successive devices along the chain. The lead device in the chain has its CS1 input tied High (or floating, since there is an internal pullup). Frame data is accepted only when CS1 is High and the device's configuration memory is not already full. The status pin DOUT is pulled Low two internal-oscillator cycles after $\overline{INIT}$ is recognized as High, and remains Low until the device's configuration memory is full. DOUT is then pulled High to signal the next device in the chain to accept the configuration data on the D0-D7 bus.

The DONE pins of all devices in the chain should be tied together, with one or more active internal pull-ups. If a large number of devices are included in the chain, deactivate some of the internal pull-ups, since the Low-driving DONE pin of the last device in the chain must sink the current from all pull-ups in the chain. The DONE pull-up is activated by default. It can be deactivated using a MakeBits option.

XC4000EX devices in Express mode are always synchronized to DONE. The device becomes active after DONE goes High. DONE is an open-drain output. With the DONE pins tied together, therefore, the external DONE signal stays low until all devices are configured, then all devices in the daisy chain become active simultaneously. If the DONE pin of a device is left unconnected, the device becomes active as soon as that device has been configured. XC5200 devices in the chain should be configured as synchronized to DONE (MakeBits option 'CCLK_SYNC or UCLK_SYNC), and their DONE pins wired together with those of the XC4000EX devices.

Express mode must be specified as an option to the MakeBits program, which generates the bitstream. The Express mode bitstream is not compatible with the other six configuration modes.

Express mode is selected by a <010> on the mode pins (M2, M1, M0).

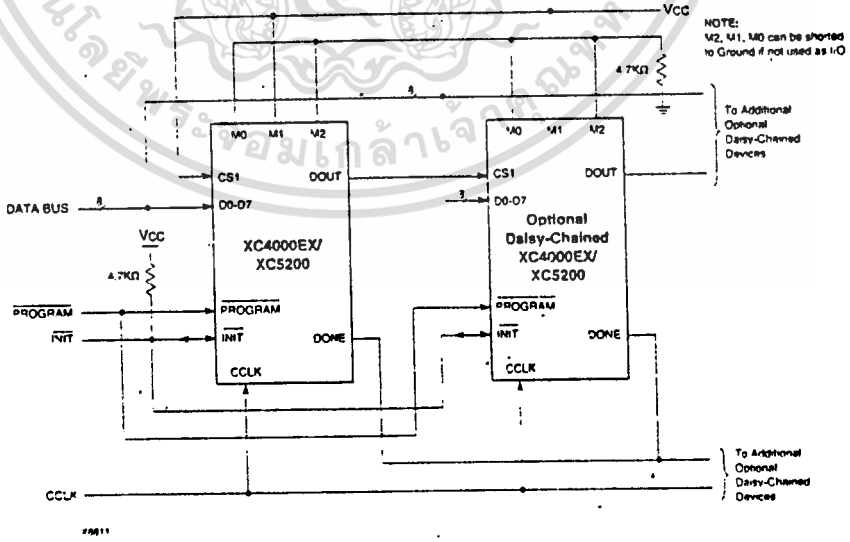
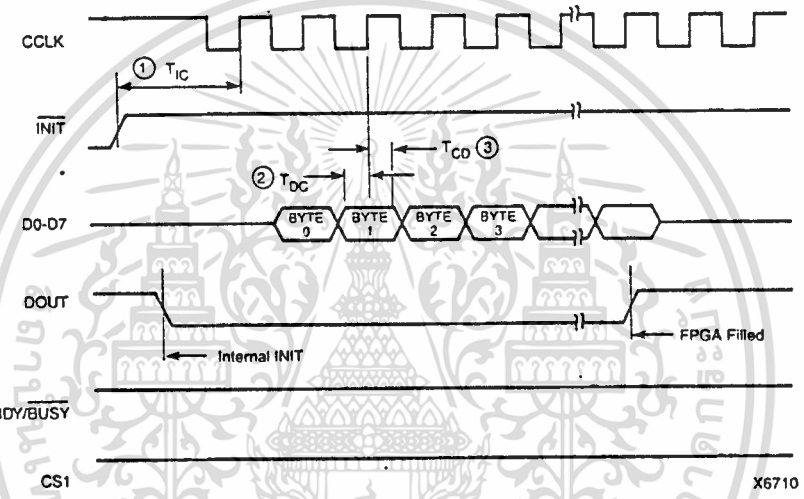


Figure 63: Express Mode Circuit Diagram

	Description	Symbol	Min	Max	Units
CCLK	INIT (High) setup time	T_{IC}	-		µs
	D0 - D7 setup time	T_{DC}	-		ns
	D0 - D7 hold time	T_{CD}	0		ns
	CCLK High time	T_{CCH}	-		ns
	CCLK Low time	T_{CCL}	-		ns
	CCLK Frequency	F_{CC}		-	MHz
			Preliminary		



Note: If not driven by the preceding DOUT, CS1 *must* remain High until the device is fully configured.

Figure 64: Express Mode Programming Switching Characteristics

XC4000E Switching Characteristics

Definition of Terms

In the following tables, some specifications may be designated as Advance or Preliminary. These terms are defined as follows:

Advance: Initial estimates based on simulation and/or extrapolation from other speed grades, devices, or device families. Use as estimates, not for production.

Preliminary: Based on preliminary characterization. Further changes are not expected.

Unmarked: Specifications not identified as either Advance or Preliminary are to be considered Final.[1]

XC4000E Operating Conditions

Symbol	Description		Min	Max	Units
V_{CC}	Supply voltage relative to GND, T_J = -0 °C to +85°C	Commercial	4.75	5.25	V
	Supply voltage relative to GND, T_J = -40°C to +100°C	Industrial	4.5	5.5	V
	Supply voltage relative to GND, T_C = -55°C to +125°C	Military	4.5	5.5	V
V_{IH}	High-level input voltage	TTL inputs	2.0	V_{CC}	V
		CMOS inputs	70%	100%	V_{CC}
V_{IL}	Low-level input voltage	TTL inputs	0	0.8	V
		CMOS inputs	0	20%	V_{CC}
T_{IN}	Input signal transition time (Note 2)			250	ns

Note 1: At junction temperatures above those listed as Operating Conditions, all delay parameters increase by 0.35% per °C.
Note 2: Typical value only. Not tested or characterized.
Note 3: Input and output measurement thresholds for TTL are 1.5 V. Input and output measurement thresholds for CMOS are 2.5 V.

XC4000E DC Characteristics Over Operating Conditions

Symbol	Description		Min	Max	Units
V_{OH}	High-level output voltage @ I_{OH} = -4.0mA, V_{CC} min	TTL outputs	2.4		V
	High-level output voltage @ I_{OH} = -1.0mA, V_{CC} min	CMOS outputs	V_{CC}-0.5		V
V_{OL}	Low-level output voltage @ I_{OL} = 12.0mA, V_{CC} min (Note 1)	TTL outputs		0.4	V
		CMOS outputs		0.4	V
I_{CCO}	Quiescent FPGA supply current (Note 2)	TTL input levels		10	mA
		CMOS input levels		1	mA
I_L	Input or output leakage current		-10	+10	µA
C_{IN}	Input capacitance (sample tested)	PQFP and MQFP packages		10	pF
		Other packages		16	pF
I_{RIN}	Pad pull-up (when selected) @ V_{IN} = 0V (sample tested)		0.02	0.25	mA
I_{RLL}	Horizontal Longline pull-up (when selected) @ logic Low		0.2	2.5	mA

Note 1: With 50% of the outputs simultaneously sinking 12mA, up to a maximum of 64 pins.
Note 2: With no output current loads, no active input or Longline pull-up resistors, all package pins at Vcc or GND, and the FPGA configured with a MakeBits Tie option.

1. Notwithstanding the definition of the above terms, all specifications are subject to change without notice.

XC4000E Absolute Maximum Ratings

Symbol	Description			Units
V_{CC}	Supply voltage relative to GND		-0.5 to +7.0	V
V_{IN}	Input voltage relative to GND (Note 1)		-0.5 to V_{CC} +0.5	V
V_{TS}	Voltage applied to 3-state output (Note 1)		-0.5 to V_{CC} +0.5	V
T_{STG}	Storage temperature (ambient)		-65 to +150	°C
T_{SOL}	Maximum soldering temperature (10 s @ 1/16 in. = 1.5 mm)		+260	°C
T_J	Junction temperature	Ceramic packages	+150	°C
		Plastic packages	+125	°C

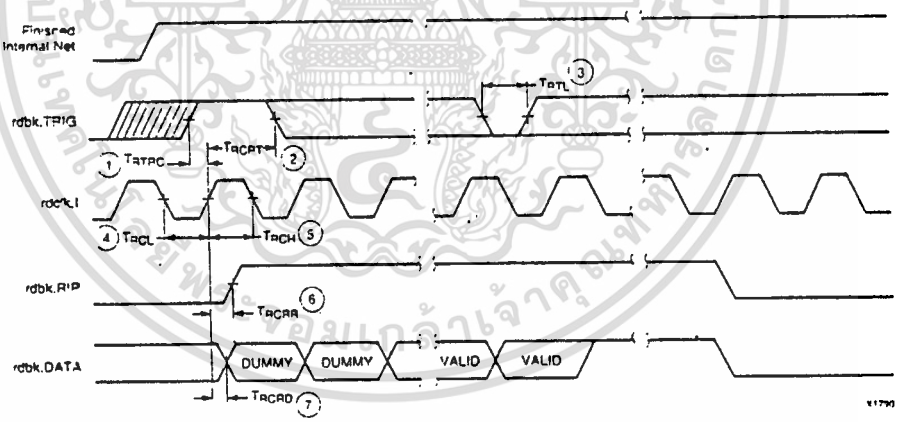
Note 1: Maximum DC overshoot or undershoot above Vcc or below GND must be limited to either 0.5 V or 10 mA, whichever is easier to achieve. During transitions, the device pins may undershoot to -2.0 V or overshoot to Vcc + 2.0 V, provided this over- or undershoot lasts less than 20 ns.

Note 2: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

XC4000E Program Readback Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Internal timing parameters are not measured directly. They are derived from benchmark timing patterns that are taken at device introduction, prior to any process improvements.

The following guidelines reflect worst-case values over the recommended operating conditions.



	Description		Symbol	Min	Max	Units
rdbk.TRIG	rdbk.TRIG setup	1	T_{RTRC}	200	-	ns
	rdbk.TRIG hold	2	T_{RCRT}	50	-	ns
	rdbk.TRIG Low to abort Readback	3	T_{RTL}	100	-	ns
rdclk.1	rdbk.DATA delay	7	T_{RCRD}	-	250	ns
	rdbk.RIP delay	6	T_{RCRR}	-	250	ns
	High time	5	T_{RCH}	250	500	ns
	Low time	4	T_{RCL}	250	500	ns

Note 1: Timing parameters apply to all speed grades.

Note 2: If rdbk.TRIG is High prior to Finished, Finished will trigger the first Readback.

XC4000E Wide Decoder Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Internal timing parameters are not measured directly. They are derived from benchmark timing patterns that are taken at device introduction, prior to any process improvements. For more detailed, more precise, and more up-to-date information, use the values provided by the XACT timing calculator and used in the simulator. These values can be printed in tabular format by running LCA2XNF -S.

The following guidelines reflect worst-case values over the recommended operating conditions.

		Speed Grade	-4	-3	-2		
Description	Symbol	Device	Max	Max	Max		Units
Full length, both pull-ups, inputs from IOB I-pins	T_{WAF}	XC4003E	9.2	5.0	4.3		ns
		XC4005E	9.5	6.0	5.1		ns
		XC4006E	12.0	7.0	6.2		ns
		XC4008E	12.5	8.0	7.0		ns
		XC4010E	15.0	9.0	8.1		ns
		XC4013E	16.0	11.0	9.9		ns
		XC4020E	17.0	13.9	12.5		ns
		XC4025E	18.0	16.9	15.2		ns
Full length, both pull-ups, inputs from internal logic	T_{WAFL}	XC4003E	12.0	7.0	6.0		ns
		XC4005E	12.5	8.0	6.8		ns
		XC4006E	14.0	9.0	7.9		ns
		XC4008E	16.0	10.0	8.8		ns
		XC4010E	18.0	11.0	9.7		ns
		XC4013E	19.0	13.0	11.7		ns
		XC4020E	20.0	15.5	14.0		ns
		XC4025E	21.0	18.9	17.0		ns
Half length, one pull-up, inputs from IOB I-pins	T_{WAO}	XC4003E	10.5	6.0	5.1		ns
		XC4005E	10.5	7.0	6.0		ns
		XC4006E	13.5	8.0	6.8		ns
		XC4008E	14.0	9.0	7.9		ns
		XC4010E	16.0	10.0	8.8		ns
		XC4013E	17.0	12.0	10.8		ns
		XC4020E	18.0	15.0	13.5		ns
		XC4025E	19.0	17.6	15.8		ns
Half length, one pull-up, inputs from internal logic	T_{WAOL}	XC4003E	12.0	8.0	6.8		ns
		XC4005E	12.5	9.0	7.7		ns
		XC4006E	14.0	10.0	8.5		ns
		XC4008E	16.0	11.0	9.4		ns
		XC4010E	18.0	12.0	10.2		ns
		XC4013E	19.0	14.0	11.9		ns
		XC4020E	20.0	16.8	14.3		ns
		XC4025E	21.0	19.6	16.7		ns
					ADVANCE		

Note 1: These values include a minimum load. The values reported by LCA2XNF -S include only a portion of this delay, therefore the values cannot be directly compared. Use XDelay to determine the delay for each destination.

Note 2: These delays are specified from the decoder input to the decoder output. For pin-to-pin delays, add the input delay (T_{PID}) and output delay (T_{OPF} or T_{OPS}), as listed under "IOB Switching Characteristic Guidelines."

XC4000E Global Buffer Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Internal timing parameters are not measured directly. They are derived from benchmark timing patterns that are taken at device introduction, prior to any process improvements. For more detailed, more precise, and more up-to-date information, use the values provided by the XACT timing calculator and used in the simulator. These values can be printed in tabular format by running LCA2XNF -S.

The following guidelines reflect worst-case values over the recommended operating conditions.

		Speed Grade	-4	-3	-2		
Description	Symbol	Device	Max	Max	Max		Units
From pad through Primary buffer, to any clock K	T_{PG}	XC4003E	7.0	4.7	4.0		ns
		XC4005E	7.0	4.7	4.0		ns
		XC4006E	7.5	5.3	4.5		ns
		XC4008E	8.0	6.1	5.2		ns
		XC4010E	11.0	6.3	5.4		ns
		XC4013E	11.5	6.8	5.8		ns
		XC4020E	12.0	7.0	6.2		ns
		XC4025E	12.5	7.2	6.3		ns
From pad through Secondary buffer, to any clock K	T_{SG}	XC4003E	7.5	5.2	4.4		ns
		XC4005E	7.5	5.2	4.4		ns
		XC4006E	8.0	5.8	4.9		ns
		XC4008E	8.5	6.6	5.6		ns
		XC4010E	11.5	6.8	5.8		ns
		XC4013E	12.0	7.3	6.2		ns
		XC4020E	12.5	7.5	6.6		ns
		XC4025E	13.0	7.7	6.8		ns
					ADVANCE		

XC4000E Guaranteed Input and Output Parameters (Pin-to-Pin, TTL I/O)

All values listed below are tested directly, and guaranteed over the operating conditions. The same parameters can also be derived indirectly from the IOB and Global Buffer specifications. The XACT delay calculator uses this indirect method. When there is a discrepancy between the two methods, the values listed below should be used, and the derived values must be ignored. All values are expressed in units of nanoseconds.

Description	Symbol	Device	-4	-3	-2
Global Clock to Output (fast) using OFF	T_{ICKOF} (Max)	XC4003E	12.5	10.2	8.7
		XC4005E	14.0	10.7	9.1
		XC4006E	14.5	10.7	9.1
		XC4008E	15.0	10.8	9.2
		XC4010E	16.0	10.9	9.3
		XC4013E	16.5	11.0	9.4
		XC4020E	17.0	11.0	9.4
		XC4025E	17.0	12.6	10.7
Global Clock to Output (slew-limited) using OFF	T_{ICKO} (Max)	XC4003E	16.5	14.0	11.5
		XC4005E	18.0	14.7	12.0
		XC4006E	18.5	14.7	12.0
		XC4008E	19.0	14.8	12.1
		XC4010E	20.0	14.9	12.2
		XC4013E	20.5	15.0	12.8
		XC4020E	21.0	15.1	12.8
		XC4025E	21.0	15.3	13.0
Input Setup Time, using IFF (no delay)	T_{PSUF} (Min)	XC4003E	2.5	2.3	2.3
		XC4005E	2.0	1.2	1.2
		XC4006E	1.9	1.0	1.0
		XC4008E	1.4	0.6	0.6
		XC4010E	1.0	0.2	0.2
		XC4013E	0.5	0	0
		XC4020E	0	0	0
		XC4025E	0	0	0
Input Hold Time, using IFF (no delay)	T_{PHF} (Min)	XC4003E	4.0	4.0	4.0
		XC4005E	4.6	4.5	4.5
		XC4006E	5.0	4.7	4.7
		XC4008E	6.0	5.1	5.1
		XC4010E	6.0	5.5	5.5
		XC4013E	7.0	6.5	5.5
		XC4020E	7.5	6.7	5.7
		XC4025E	8.0	7.0	5.9
Input Setup Time, using IFF (with delay)	T_{PSU} (Min)	XC4003E	8.5	7.0	6.0
		XC4005E	8.5	7.0	6.0
		XC4006E	8.5	7.0	6.0
		XC4008E	8.5	7.0	6.0
		XC4010E	8.5	7.0	6.0
		XC4013E	8.5	7.0	6.0
		XC4020E	9.5	7.0	6.0
		XC4025E	9.5	7.6	6.5
Input Hold Time, using IFF (with delay)	T_{PH} (Min)	XC4003E	0	0	0
		XC4005E	0	0	0
		XC4006E	0	0	0
		XC4008E	0	0	0
		XC4010E	0	0	0
		XC4013E	0	0	0
		XC4020E	0	0	0
		XC4025E	0	0	0

OFF = Output Flip-Flop IFF = Input Flip-Flop or Latch

ADVANCE

XC4000E Horizontal Longline Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Internal timing parameters are not measured directly. They are derived from benchmark timing patterns that are taken at device introduction, prior to any process improvements. For more detailed, more precise, and more up-to-date information, use the values provided by the XACT timing calculator and used in the simulator. These values can be printed in tabular format by running LCA2XNF -S.

The following guidelines reflect worst-case values over the recommended operating conditions.

		Speed Grade	-4	-3	-2		
Description	Symbol	Device	Max	Max	Max		Units
TBUF driving a Horizontal Longline (LL):	T_{IO1}	XC4003E	5.0	4.2	3.4		ns
		XC4005E	5.0	5.0	4.0		ns
		XC4006E	6.0	5.9	4.7		ns
I going High or Low to LL going High or Low, while T is Low. Buffer is constantly active. (Note1)		XC4008E	7.0	6.3	5.0		ns
		XC4010E	8.0	6.4	5.1		ns
		XC4013E	9.0	7.2	5.7		ns
		XC4020E	10.0	8.2	6.6		ns
		XC4025E	11.0	9.1	7.3		ns
I going Low to LL going from resistive pull-up High to active Low. TBUF configured as open-drain. (Note1)	T_{IO2}	XC4003E	5.0	4.2	3.6		ns
		XC4005E	6.0	5.3	4.5		ns
		XC4006E	7.8	6.4	5.4		ns
		XC4008E	8.1	6.8	5.8		ns
		XC4010E	10.5	6.9	5.9		ns
		XC4013E	11.0	7.7	6.5		ns
		XC4020E	12.0	8.7	7.4		ns
		XC4025E	12.0	9.6	8.2		ns
T going Low to LL going from resistive pull-up or floating High to active Low. TBUF configured as open-drain or active buffer with I = Low. (Note1)	T_{ON}	XC4003E	5.5	4.6	3.9		ns
		XC4005E	7.0	6.0	5.4		ns
		XC4006E	7.5	6.7	5.7		ns
		XC4008E	8.0	7.1	6.0		ns
		XC4010E	8.5	7.3	6.2		ns
		XC4013E	8.7	7.5	6.4		ns
		XC4020E	11.0	8.4	7.1		ns
		XC4025E	11.0	8.4	7.1		ns
T going High to TBUF going inactive, not driving LL	T_{OFF}	All devices	1.8	1.5	1.3		ns
T going High to LL going from Low to High, pulled up by a single resistor. (Note 2)	T_{PUS}	XC4003E	20.0	14.0	11.9		ns
		XC4005E	23.0	16.0	13.6		ns
		XC4006E	25.0	18.0	15.3		ns
		XC4008E	27.0	20.0	17.0		ns
		XC4010E	29.0	22.0	18.7		ns
		XC4013E	32.0	26.0	22.1		ns
		XC4020E	35.0	32.5	27.6		ns
		XC4025E	42.0	39.1	33.2		ns
T going High to LL going from Low to High, pulled up by two resistors. (Note1)	T_{PUF}	XC4003E	9.0	7.0	6.0		ns
		XC4005E	10.0	8.0	6.8		ns
		XC4006E	11.5	9.0	7.7		ns
		XC4008E	12.5	10.0	8.5		ns
		XC4010E	13.5	11.0	9.4		ns
		XC4013E	15.0	13.0	11.0		ns
		XC4020E	16.0	14.8	12.6		ns
		XC4025E	18.0	16.5	14.0		ns
					ADVANCE		

Note 1: These values include a minimum load. The values reported by LCA2XNF -S include only a portion of this delay, therefore the values cannot be directly compared. Use XDelay to determine the delay for each destination.

Note 2: This value includes a minimum load. The value reported by LCA2XNF -S is increased to allow for potentially heavy loading, therefore the values cannot be directly compared. Use XDelay to determine the delay for each destination.

XC4000E IOB Input Switching Characteristic Guidelines (continued)

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Internal timing parameters are not measured directly. They are derived from benchmark timing patterns that are taken at device introduction, prior to any process improvements. For more detailed, more precise, and more up-to-date information, use the values provided by the XACT timing calculator and used in the simulator. These values can be printed in tabular format by running LCA2XNF -S.

The following guidelines reflect worst-case values over the recommended operating conditions. They are expressed in units of nanoseconds and apply to all XC4000E devices unless otherwise noted.

Description		Speed Grade Symbol	Device	-4 Min	-4 Max	-3 Min	-3 Max	-2 Min	-2 Max
Setup Times (TTL Inputs)									
Pad to Clock (IK),	no delay	T_{PICK}	All devices	4.0		2.6		1.7	
	with delay	T_{PICKD}	XC4003E	10.9		8.2		5.5	
			XC4005E	10.9		8.7		5.5	
			XC4006E	10.9		9.2		6.6	
			XC4008E	11.1		9.6		6.9	
			XC4010E	11.3		9.8		7.0	
			XC4013E	11.8		10.2		7.3	
			XC4020E	14.0		11.4		8.2	
			XC4025E	14.0		11.4		8.2	
(CMOS Inputs)									
Pad to Clock (IK),	no delay	T_{PICKC}	All devices	6.0		3.3		2.4	
	with delay	T_{PICKDC}	XC4003E	12.0		8.8		6.2	
			XC4005E	12.0		9.7		6.2	
			XC4006E	12.3		9.9		7.3	
			XC4008E	12.8		10.3		7.6	
			XC4010E	13.0		10.5		7.7	
			XC4013E	13.5		10.9		8.0	
			XC4020E	16.0		12.1		8.9	
			XC4025E	16.0		12.1		8.9	
(TTL or CMOS)									
Clock Enable (EC) to Clock (IK),	no delay	T_{ECIK}	All devices	3.5		2.5		2.0	
	with delay	T_{ECIKD}	XC4003E	10.4		8.1		5.6	
			XC4005E	10.4		8.5		5.6	
			XC4006E	10.4		9.1		6.9	
			XC4008E	10.4		9.5		7.2	
			XC4010E	10.7		9.7		7.3	
			XC4013E	11.1		10.1		7.6	
			XC4020E	14.0		11.3		8.5	
			XC4025E	14.0		11.3		8.5	
Global Set/Reset (Note 3)									
Delay from GSR net through Q to I1, I2		T_{RRI}			12.0		7.8		6.8
GSR width		T_{MRW}		13.0		11.5		11.5	
GSR inactive to first active Clock (IK) edge		T_{MRI}							

ADVANCE

Note 1: Input pad setup and hold times are specified with respect to the internal clock (IK). For setup and hold times with respect to the clock input pin, see the pin-to-pin parameters in the Guaranteed Input and Output Parameters table.

Note 2: Voltage levels of unused pads, bonded or unbonded, must be valid logic levels. Each can be configured with the internal pull-up (default) or pull-down resistor, or configured as a driven output, or can be driven from an external source.

Note 3: Timing is based on the XC4005E. For other devices see the XACT timing calculator.

XC4000E IOB Input Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Internal timing parameters are not measured directly. They are derived from benchmark timing patterns that are taken at device introduction, prior to any process improvements. For more detailed, more precise, and more up-to-date information, use the values provided by the XACT timing calculator and used in the simulator. These values can be printed in tabular format by running LCA2XNF -S.

The following guidelines reflect worst-case values over the recommended operating conditions. They are expressed in units of nanoseconds and apply to all XC4000E devices unless otherwise noted.

	Speed Grade		-4		-3		-2	
Description	Symbol	Device	Min	Max	Min	Max	Min	Max
Propagation Delays								
(TTL Inputs)								
Pad to I1, I2	T_{PID}	All devices		3.0		2.5		2.0
Pad to I1, I2 via transparent latch, no delay	T_{PLI}	All devices		4.8		3.6		3.6
with delay	T_{PDLI}	XC4003E		10.4		9.3		7.0
		XC4005E		10.8		9.6		7.3
		XC4006E		10.8		10.2		7.8
		XC4008E		10.8		10.6		8.1
		XC4010E		11.0		10.8		8.2
		XC4013E		11.4		11.2		8.5
		XC4020E		13.8		12.4		9.5
		XC4025E		13.8		13.7		9.5
(CMOS Inputs)								
Pad to I1, I2	T_{PIDC}	All devices		5.5		4.1		3.7
Pad to I1, I2 via transparent latch, no delay	T_{PLIC}	All devices		8.8		6.8		6.2
with delay	T_{PDLIC}	XC4003E		16.5		12.4		11.0
		XC4005E		16.5		13.2		11.9
		XC4006E		16.8		13.4		12.1
		XC4008E		17.3		13.8		12.4
		XC4010E		17.5		14.0		12.6
		XC4013E		18.0		14.4		13.0
		XC4020E		20.8		15.6		14.0
		XC4025E		20.8		15.6		14.0
(TTL or CMOS)								
Clock (IK) to I1, I2 (flip-flop)	T_{IKRI}	All devices		5.6		2.8		2.8
Clock (IK) to I1, I2 (latch enable, active Low)	T_{IKLI}	All devices		6.2		4.0		3.9
Hold Times (Note 1)								
Pad to Clock (IK), no delay	T_{IKPI}	All devices	0		0		0	
with delay	T_{IKPID}	All devices	0		0		0	
Clock Enable (EC) to Clock (IK), no delay	T_{IKEC}	All devices	1.5		1.5		0.9	
with delay	T_{IKECD}	All devices	0		0		0	
							ADVANCE	

Note 1: Input pad setup and hold times are specified with respect to the internal clock (IK). For setup and hold times with respect to the clock input pin, see the pin-to-pin parameters in the Guaranteed Input and Output Parameters table.

Note 2: Voltage levels of unused pads, bonded or unbonded, must be valid logic levels. Each can be configured with the internal pull-up (default) or pull-down resistor, or configured as a driven output, or can be driven from an external source.

Pin Locations for XC4010E/L Devices

XC4010E/L Pad Name	PC 84	PQ 160	TQ 176	PG 191	PQ/ HQ 208	BG 225	Bndry Scan
VCC	P2	P142	P155	J4	P183	D8	-
I/O (A8)	P3	P143	P156	J3	P184	E8	62
I/O (A9)	P4	P144	P157	J2	P185	B7	65
I/O	-	P145	P158	J1	P186	A7	68
I/O	-	P146	P159	H1	P187	C7	71
I/O	-	-	P160	H2	P188	D7	74
I/O	-	-	P161	H3	P189	E7	77
I/O (A10)	P5	P147	P162	G1	P190	A6	80
I/O (A11)	P6	P148	P163	G2	P191	B6	83
I/O	-	P149	P164	F1	P192	A5	86
I/O	-	P150	P165	E1	P193	B5	89
GND	-	P151	P166	G3	P194	GND*	-
I/O	-	-	-	F2	P195	D6	92
I/O	-	-	P167	D1	P196	C5	95
I/O	-	P152	P168	C1	P197	A4	98
I/O	-	P153	P169	E2	P198	E6	101
I/O (A12)	P7	P154	P170	F3	P199	B4	104
I/O (A13)	P8	P155	P171	D2	P200	D5	107
I/O	-	P156	P172	B1	P201	B3	110
I/O	-	P157	P173	E3	P202	F6	113
I/O (A14)	P9	P158	P174	C2	P203	A2	116
I/O, SGCK1 (A15)	P10	P159	P175	B2	P204	C3	119
VCC	P11	P160	P176	D3	P205	B2	-
GND	P12	P1	P1	D4	P2	A1	-
I/O, PGCK1 (A16)	P13	P2	P2	C3	P4	D4	122
I/O (A17)	P14	P3	P3	C4	P5	B1	125
I/O	-	P4	P4	B3	P6	C2	128
I/O	-	P5	P5	C5	P7	E5	131
I/O, TDI	P15	P6	P6	A2	P8	D3	134
I/O, TCK	P16	P7	P7	B4	P9	C1	137
I/O	-	P8	P8	C6	P10	D2	140
I/O	-	P9	P9	A3	P11	G6	143
I/O	-	-	-	B5	P12	E4	146
I/O	-	-	-	B6	P13	D1	149
GND	-	P10	P10	C7	P14	GND*	-
I/O	-	P11	P11	A4	P15	F5	152
I/O	-	P12	P12	A5	P16	E1	155
I/O, TMS	P17	P13	P13	B7	P17	F4	158
I/O	P18	P14	P14	A6	P18	F3	161
I/O	-	-	P15	C8	P19	G4	164
I/O	-	-	P16	A7	P20	G3	167
I/O	-	P15	P17	B8	P21	G2	170
I/O	-	P16	P18	A8	P22	G1	173

XC4010E/L Pad Name	PC 84	PQ 160	TQ 176	PG 191	PQ/ HQ 208	BG 225	Bndry Scan
I/O	P19	P17	P19	B9	P23	G5	176
I/O	P20	P18	P20	C9	P24	H3	179
GND	P21	P19	P21	D9	P25	H2	-
VCC	P22	P20	P22	D10	P26	H1	-
I/O	P23	P21	P23	C10	P27	H4	182
I/O	P24	P22	P24	B10	P28	H5	185
I/O	-	P23	P25	A9	P29	J2	188
I/O	-	P24	P26	A10	P30	J1	191
I/O	-	-	P27	A11	P31	J3	194
I/O	-	-	P28	C11	P32	J4	197
I/O	P25	P25	P29	B11	P33	K2	200
I/O	P26	P26	P30	A12	P34	K3	203
I/O	-	P27	P31	B12	P35	J6	206
I/O	-	P28	P32	A13	P36	L1	209
GND	-	P29	P33	C12	P37	GND*	-
I/O	-	-	-	B13	P38	L3	212
I/O	-	-	-	A14	P39	M1	215
I/O	-	P30	P34	A15	P40	K5	218
I/O	-	P31	P35	C13	P41	M2	221
I/O	P27	P32	P36	B14	P42	L4	224
I/O	-	P33	P37	A16	P43	N1	227
I/O	-	P34	P38	B15	P44	M3	230
I/O	-	P35	P39	C14	P45	N2	233
I/O	P28	P36	P40	A17	P46	K6	236
I/O, SCGK2	P29	P37	P41	B16	P47	P1	239
O (M1)	P30	P38	P42	C15	P48	N3	242
GND	P31	P39	P43	D15	P49	GND*	-
I (M0)	P32	P40	P44	A18	P50	P2	245
VCC	P33	P41	P45	D16	P55	R1	-
I (M2)	P34	P42	P46	C16	P56	M4	246
I/O, PGCK2	P35	P43	P47	B17	P57	R2	247
I/O (HDC)	P36	P44	P48	E16	P58	P3	250
I/O	-	P45	P49	C17	P59	L5	253
I/O	-	P46	P50	D17	P60	N4	256
I/O	-	P47	P51	B18	P61	R3	259
I/O (LDC)	P37	P48	P52	E17	P62	P4	262
I/O	-	P49	P53	F16	P63	K7	265
I/O	-	P50	P54	C18	P64	M5	268
I/O	-	-	-	D18	P65	R4	271
I/O	-	-	-	F17	P66	N5	274
GND	-	P51	P55	G16	P67	GND*	-
I/O	-	P52	P56	E18	P68	R5	277
I/O	-	P53	P57	F18	P69	M6	280
I/O	P38	P54	P58	G17	P70	N6	283
I/O	P39	P55	P59	G18	P71	P6	286
I/O	-	-	P60	H16	P72	R6	289

XC4010E/L Pad Name	PC 84	PQ 160	TQ 176	PG 191	PQ/ HQ 208	BG 225	Bndry Scan
I/O	-	-	P61	H17	P73	M7	292
I/O	-	P56	P62	H18	P74	R7	295
I/O	-	P57	P63	J18	P75	L7	298
I/O	P40	P58	P64	J17	P76	N8	301
I/O (INIT)	P41	P59	P65	J16	P77	P8	304
VCC	P42	P60	P66	J15	P78	R8	-
GND	P43	P61	P67	K15	P79	M8	-
I/O	P44	P62	P68	K16	P80	L8	307
I/O	P45	P63	P69	K17	P81	P9	310
I/O	-	P64	P70	K18	P82	R9	313
I/O	-	P65	P71	L18	P83	N9	316
I/O	-	-	P72	L17	P84	M9	319
I/O	-	-	P73	L16	P85	L9	322
I/O	P46	P66	P74	M18	P86	N10	325
I/O	P47	P67	P75	M17	P87	K9	328
I/O	-	P68	P76	N18	P88	R11	331
I/O	-	P69	P77	P18	P89	P11	334
GND	-	P70	P78	M16	P90	GND*	-
I/O	-	-	-	N17	P91	R12	337
I/O	-	-	-	R18	P92	L10	340
I/O	-	P71	P79	T18	P93	P12	343
I/O	-	P72	P80	P17	P94	M11	346
I/O	P48	P73	P81	N16	P95	R13	349
I/O	P49	P74	P82	T17	P96	N12	352
I/O	-	P75	P83	R17	P97	P13	355
I/O	-	P76	P84	P16	P98	K10	358
I/O	P50	P77	P85	U18	P99	R14	361
I/O, SGCK3	P51	P78	P86	T16	P100	N13	364
GND	P52	P79	P87	R16	P101	GND*	-
DONE	P53	P80	P88	U17	P103	P14	-
VCC	P54	P81	P89	R15	P106	R15	-
PROGRAM	P55	P82	P90	V18	P108	M12	-
I/O (D7)	P56	P83	P91	T15	P109	P15	367
I/O, PGCK3	P57	P84	P92	U16	P110	N14	370
I/O	-	P85	P93	T14	P111	L11	373
I/O	-	P86	P94	U15	P112	M13	376
I/O (D6)	P58	P87	P95	V17	P113	J10	379
I/O	-	P88	P96	V16	P114	L12	382
I/O	-	P89	P97	T13	P115	M15	385
I/O	-	P90	P98	U14	P116	L13	388
I/O	-	-	-	V15	P117	L14	391
I/O	-	-	-	V14	P118	K11	394
GND	-	P91	P99	T12	P119	GND*	-
I/O	-	P92	P100	U13	P120	K13	397
I/O	-	P93	P101	V13	P121	K14	400
I/O (D5)	P59	P94	P102	U12	P122	K15	403

XC4010E/L Pad Name	PC 84	PQ 160	TQ 176	PG 191	PQ/ HQ 208	BG 225	Bndry Scan
I/O (CS0)	P60	P95	P103	V12	P123	J12	406
I/O	-	-	P104	T11	P124	J13	409
I/O	-	-	P105	U11	P125	J14	412
I/O	-	P96	P106	V11	P126	J15	415
I/O	-	P97	P107	V10	P127	J11	418
I/O (D4)	P61	P98	P108	U10	P128	H13	421
I/O	P62	P99	P109	T10	P129	H14	424
VCC	P63	P100	P110	R10	P130	H15	-
GND	P64	P101	P111	R9	P131	GND*	-
I/O (D3)	P65	P102	P112	T9	P132	H12	427
I/O (RS)	P66	P103	P113	U9	P133	H11	430
I/O	-	P104	P114	V9	P134	G14	433
I/O	-	P105	P115	V8	P135	G15	436
I/O	-	-	P116	U8	P136	G13	439
I/O	-	-	P117	T8	P137	G12	442
I/O (D2)	P67	P106	P118	V7	P138	G11	445
I/O	P68	P107	P119	U7	P139	F15	448
I/O	-	P108	P120	V6	P140	F14	451
I/O	-	P109	P121	U6	P141	F13	454
GND	-	P110	P122	T7	P142	GND*	-
I/O	-	-	-	V5	P143	E13	457
I/O	-	-	-	V4	P144	D15	460
I/O	-	P111	P123	U5	P145	F11	463
I/O	-	P112	P124	T6	P146	D14	466
I/O (D1)	P69	P113	P125	V3	P147	E12	469
I/O (RCLK, RDY/ BUSY)	P70	P114	P126	V2	P148	C15	472
I/O	-	P115	P127	U4	P149	D13	475
I/O	-	P116	P128	T5	P150	C14	478
I/O (D0, DIN)	P71	P117	P129	U3	P151	F10	481
I/O, SGCK4 (DOUT)	P72	P118	P130	T4	P152	B15	484
CCLK	P73	P119	P131	V1	P153	C13	-
VCC	P74	P120	P132	R4	P154	B14	-
O, TDO	P75	P121	P133	U2	P159	A15	0
GND	P76	P122	P134	R3	P160	D12	-
I/O (A0, WS)	P77	P123	P135	T3	P161	A14	2
I/O, PGCK4 (A1)	P78	P124	P136	U1	P162	B13	5
I/O	-	P125	P137	P3	P163	E11	8
I/O	-	P126	P138	R2	P164	C12	11
I/O (CS1, A2)	P79	P127	P139	T2	P165	A13	14
I/O (A3)	P80	P128	P140	N3	P166	B12	17

XC4010E/L Pad Name	PC 84	PQ 160	TQ 176	PG 191	PQ/ HQ 208	BG 225	Bndry Scan
I/O	-	P129	P141	P2	P167	A12	20
I/O	-	P130	P142	T1	P168	C11	23
I/O	-	-	-	R1	P169	B11	26
I/O	-	-	-	N2	P170	E10	29
GND	-	P131	P143	M3	P171	GND*	-
I/O	-	P132	P144	P1	P172	A11	32
I/O	-	P133	P145	N1	P173	D10	35
I/O (A4)	P81	P134	P146	M2	P174	A10	38
I/O (A5)	P82	P135	P147	M1	P175	D9	41
I/O	-	-	P148	L3	P176	C9	44
I/O	-	P136	P149	L2	P177	B9	47
I/O	-	P137	P150	L1	P178	A9	50
I/O	-	P138	P151	K1	P179	E9	53
I/O (A6)	P83	P139	P152	K2	P180	C8	56
I/O (A7)	P84	P140	P153	K3	P181	B8	59
GND	P1	P141	P154	K4	P182	A8	-

4/2/96

* Pads labelled GND* are internally bonded to a Ground plane within the BG225 package. They have no direct connection to any package pin.

Additional Ground (GND) Connections on BG225 Package

GND
F8
G7
G8
G9
H6
H7
H8
H9
H10
J7
J8
J9
K8

2/28/96

Note: The package pins in this table are bonded to an internal Ground plane within the BG225 package. They should all be externally connected to Ground.

Additional No Connect (N.C.) Connections on PQ/HQ208 & BG225 Packages

PQ/HQ208	BG225
P1	A3
P3	B10
P51	C4
P52	C6
P53	C10
P54	D11
P102	E2
P104	E3
P105	E14
P107	E15
P155	F1
P156	F2
P157	F7
P158	F9
P206	F12
P207	G10
P208	J5
	K1
	K4
	K12
	L2
	L6
	L15
	M10
	M14
	N7
	N11
	N15
	P5
	P7
	P10
	R10

3/12/96

Package-Specific Pinout Tables

PC84 Package Pinouts

Pin	XC4003E	XC4005E XC4005L	XC4006E	XC4008E	XC4010E XC4010L
P1	GND	GND	GND	GND	GND
P2	VCC	VCC	VCC	VCC	VCC
P3	I/O (A8)	I/O (A8)	I/O (A8)	I/O (A8)	I/O (A8)
P4	I/O (A9)	I/O (A9)	I/O (A9)	I/O (A9)	I/O (A9)
P5	I/O (A10)	I/O (A10)	I/O (A10)	I/O (A10)	I/O (A10)
P6	I/O (A11)	I/O (A11)	I/O (A11)	I/O (A11)	I/O (A11)
P7	I/O (A12)	I/O (A12)	I/O (A12)	I/O (A12)	I/O (A12)
P8	I/O (A13)	I/O (A13)	I/O (A13)	I/O (A13)	I/O (A13)
P9	I/O (A14)	I/O (A14)	I/O (A14)	I/O (A14)	I/O (A14)
P10	I/O, SGCK1 (A15)	I/O, SGCK1 (A15)	I/O, SGCK1 (A15)	I/O, SGCK1 (A15)	I/O, SGCK1 (A15)
P11	VCC	VCC	VCC	VCC	VCC
P12	GND	GND	GND	GND	GND
P13	I/O, PGCK1 (A16)	I/O, PGCK1 (A16)	I/O, PGCK1 (A16)	I/O, PGCK1 (A16)	I/O, PGCK1 (A16)
P14	I/O (A17)	I/O (A17)	I/O (A17)	I/O (A17)	I/O (A17)
P15	I/O, TDI	I/O, TDI	I/O, TDI	I/O, TDI	I/O, TDI
P16	I/O, TCK	I/O, TCK	I/O, TCK	I/O, TCK	I/O, TCK
P17	I/O, TMS	I/O, TMS	I/O, TMS	I/O, TMS	I/O, TMS
P18	I/O	I/O	I/O	I/O	I/O
P19	I/O	I/O	I/O	I/O	I/O
P20	I/O	I/O	I/O	I/O	I/O
P21	GND	GND	GND	GND	GND
P22	VCC	VCC	VCC	VCC	VCC
P23	I/O	I/O	I/O	I/O	I/O
P24	I/O	I/O	I/O	I/O	I/O
P25	I/O	I/O	I/O	I/O	I/O
P26	I/O	I/O	I/O	I/O	I/O
P27	I/O	I/O	I/O	I/O	I/O
P28	I/O	I/O	I/O	I/O	I/O
P29	I/O, SCGK2	I/O, SCGK2	I/O, SCGK2	I/O, SCGK2	I/O, SCGK2
P30	O (M1)	O (M1)	O (M1)	O (M1)	O (M1)
P31	GND	GND	GND	GND	GND
P32	I (M0)	I (M0)	I (M0)	I (M0)	I (M0)
P33	VCC	VCC	VCC	VCC	VCC
P34	I (M2)	I (M2)	I (M2)	I (M2)	I (M2)
P35	I/O, PGCK2	I/O, PGCK2	I/O, PGCK2	I/O, PGCK2	I/O, PGCK2
P36	I/O (HDC)	I/O (HDC)	I/O (HDC)	I/O (HDC)	I/O (HDC)
P37	I/O ($\overline{LDC}$)	I/O ($\overline{LDC}$)	I/O ($\overline{LDC}$)	I/O ($\overline{LDC}$)	I/O ($\overline{LDC}$)
P38	I/O	I/O	I/O	I/O	I/O
P39	I/O	I/O	I/O	I/O	I/O
P40	I/O	I/O	I/O	I/O	I/O
P41	I/O ($\overline{INIT}$)	I/O ($\overline{INIT}$)	I/O ($\overline{INIT}$)	I/O ($\overline{INIT}$)	I/O ($\overline{INIT}$)
P42	VCC	VCC	VCC	VCC	VCC
P43	GND	GND	GND	GND	GND
P44	I/O	I/O	I/O	I/O	I/O
P45	I/O	I/O	I/O	I/O	I/O
P46	I/O	I/O	I/O	I/O	I/O
P47	I/O	I/O	I/O	I/O	I/O
P48	I/O	I/O	I/O	I/O	I/O
P49	I/O	I/O	I/O	I/O	I/O
P50	I/O	I/O	I/O	I/O	I/O
P51	I/O, SGCK3	I/O, SGCK3	I/O, SGCK3	I/O, SGCK3	I/O, SGCK3
P52	GND	GND	GND	GND	GND
P53	DONE	DONE	DONE	DONE	DONE
P54	VCC	VCC	VCC	VCC	VCC
P55	$\overline{PRO-GRAM}$	$\overline{PRO-GRAM}$	$\overline{PRO-GRAM}$	$\overline{PRO-GRAM}$	$\overline{PRO-GRAM}$
P56	I/O (D7)	I/O (D7)	I/O (D7)	I/O (D7)	I/O (D7)
P57	I/O, PGCK3	I/O, PGCK3	I/O, PGCK3	I/O, PGCK3	I/O, PGCK3
P58	I/O (D6)	I/O (D6)	I/O (D6)	I/O (D6)	I/O (D6)
P59	I/O (D5)	I/O (D5)	I/O (D5)	I/O (D5)	I/O (D5)
P60	I/O ($\overline{CS0}$)	I/O ($\overline{CS0}$)	I/O ($\overline{CS0}$)	I/O ($\overline{CS0}$)	I/O ($\overline{CS0}$)
P61	I/O (D4)	I/O (D4)	I/O (D4)	I/O (D4)	I/O (D4)
P62	I/O	I/O	I/O	I/O	I/O
P63	VCC	VCC	VCC	VCC	VCC
P64	GND	GND	GND	GND	GND
P65	I/O (D3)	I/O (D3)	I/O (D3)	I/O (D3)	I/O (D3)
P66	I/O ($\overline{RS}$)	I/O ($\overline{RS}$)	I/O ($\overline{RS}$)	I/O ($\overline{RS}$)	I/O ($\overline{RS}$)
P67	I/O (D2)	I/O (D2)	I/O (D2)	I/O (D2)	I/O (D2)
P68	I/O	I/O	I/O	I/O	I/O
P69	I/O (D1)	I/O (D1)	I/O (D1)	I/O (D1)	I/O (D1)
P70	I/O ($\overline{RCLK}$, RDY/ $\overline{BUSY}$)	I/O ($\overline{RCLK}$, RDY/ $\overline{BUSY}$)	I/O ($\overline{RCLK}$, RDY/ $\overline{BUSY}$)	I/O ($\overline{RCLK}$, RDY/ $\overline{BUSY}$)	I/O ($\overline{RCLK}$, RDY/ $\overline{BUSY}$)
P71	I/O (D0, DIN)	I/O (D0, DIN)	I/O (D0, DIN)	I/O (D0, DIN)	I/O (D0, DIN)
P72	I/O, SGCK4 (DOUT)	I/O, SGCK4 (DOUT)	I/O, SGCK4 (DOUT)	I/O, SGCK4 (DOUT)	I/O, SGCK4 (DOUT)
P73	CCLK	CCLK	CCLK	CCLK	CCLK
P74	VCC	VCC	VCC	VCC	VCC
P75	O, TDO	O, TDO	O, TDO	O, TDO	O, TDO
P76	GND	GND	GND	GND	GND
P77	I/O (A0, $\overline{WS}$)	I/O (A0, $\overline{WS}$)	I/O (A0, $\overline{WS}$)	I/O (A0, $\overline{WS}$)	I/O (A0, $\overline{WS}$)
P78	I/O, PGCK4 (A1)	I/O, PGCK4 (A1)	I/O, PGCK4 (A1)	I/O, PGCK4 (A1)	I/O, PGCK4 (A1)
P79	I/O (CS1, A2)	I/O (CS1, A2)	I/O (CS1, A2)	I/O (CS1, A2)	I/O (CS1, A2)
P80	I/O (A3)	I/O (A3)	I/O (A3)	I/O (A3)	I/O (A3)
P81	I/O (A4)	I/O (A4)	I/O (A4)	I/O (A4)	I/O (A4)
P82	I/O (A5)	I/O (A5)	I/O (A5)	I/O (A5)	I/O (A5)
P83	I/O (A6)	I/O (A6)	I/O (A6)	I/O (A6)	I/O (A6)
P84	I/O (A7)	I/O (A7)	I/O (A7)	I/O (A7)	I/O (A7)

2/28/96

Product Availability

Table 25 - Table 27 show the planned packages and speed grades for XC4000-Series devices. Call your local sales office for the latest availability information, or see the Xilinx WEBLINX at http://www.xilinx.com for the latest revision of the specifications.

Table 25: Component Availability Chart for XC4000E FPGAs

	Speed Grade	PC 84	PQ 100	VQ 100	PG 120	TQ 144	PG 156	PQ 160	CB 164	PG 191	CB 196	PQ 208	HQ 208	PG 223	BG 225	CB 228	PQ 240	HQ 240	PG 299	HQ 304
XC 4003E	-4	C I	C I	C I	C I															
	-3	C I	C I	C I	C I															
	-2	C	C	C	C															
XC 4005E	-4	C I	C I			C I	C I M B	C I	M B			C I								
	-3	C I	C I			C I	C I	C I				C I								
	-2	C	C			C	C	C				C								
XC 4006E	-4	C I				C I	C I	C I				C I								
	-3	C I				C I	C I	C I				C I								
	-2	C				C	C	C				C								
XC 4008E	-4	C I						C I		C I		C I								
	-3	C I						C I		C I		C I								
	-2	C						C		C		C								
XC 4010E	-4	C I						C I		C I MB	MB	C I	C I		C I					
	-3	C I						C I		C I		C I	C I		C I					
	-2	C						C		C		C	C		C					
XC 4013E	-4							C I				C I	C I	C I M B	C I	M B	C I	C I		
	-3							C I				C I	C I	C I	C I		C I	C I		
	-2							C				C	C	C	C		C	C		
XC 4020E	-4												C I	C I				C I		
	-3												C I	C I				C I		
	-2												C	C				C		
XC 4025E	-4													C I		M B		C I	C I M B	C I
	-3													C I				C I	C I	C I
	-2													C				C	C	C

C = Commercial, T_J = 0° to +85° C
I = Industrial, T_J = -40° to +100° C
M = Mil Temp, T_C = -55° to +125° C
B = MIL-STD-883C Class B, T_C = -55° to +125° C
Shaded device/package combinations are not supported.

Table 26: Component Availability Chart for XC4000EX FPGAs

	Speed Grade	HQ208	HQ240	PG299	HQ304	BG352	PG411	BG432
XC4028EX	-4	C I	C I	C I	C I	C I		
	-3	C	C	C	C	C		
XC4036EX	-4				C I		C I	C I
	-3				C		C	C
XC4044EX	-4						C I	C I
	-3						C	C

C = Commercial, T_J = 0° to +85° C
I = Industrial, T_J = -40° to +100° C
M = Mil Temp, T_C = -55° to +125° C
B = MIL-STD-883C Class B, T_C = -55° to +125° C
Shaded device/package combinations are not supported.

Table 27: Component Availability Chart for XC4000L and XC4000XL FPGAs

	Speed Grade	PC 84	TQ 176	PQ 208	HQ 208	BG 225	PQ 240	HQ 240	PG 299	HQ 304	BG 352	PG 411	BG 432	PG 475
XC4005L	-6	C		C										
	-5	C		C										
	-4													
XC4010L	-6	C	C	C										
	-5	C	C	C										
	-4													
XC4013L	-6			C		C	C							
	-5			C		C	C							
	-4													
XC4028XL					C			C	C	C	C			
XC4036XL										C		C	C	
XC4044XL												C	C	
XC4052XL												C	C	
XC4062XL														C

C = Commercial, T_J = 0° to +85° C
I = Industrial, T_J = -40° to +100° C
M = Mil Temp, T_C = -55° to +125° C
B = MIL-STD-883C Class B, T_C = -55° to +125° C
Shaded device/package combinations are not supported.
Speed grades for the XC4000XL have not yet been determined.

User I/O Per Package

Maximum available user I/O for each device/package combination is shown in Table 28 - Table 30.

Pinout tables for XC4000-Series devices follow. Pinout data is offered in two forms, as device-specific and package-specific tables. Device-specific tables include all packages for each XC4000-Series device. They follow the pad locations around the die, and include boundary scan register locations. Package-specific tables include all XC4000-Series devices available in a given package. These tables are especially useful in determining which pads should be avoided, in case of a future transition to a different device in the same package.

All pinouts defined at the time of publication are included in these tables. Additional information may be available. Call your local sales office or see the Xilinx WEBLINX at http://www.xilinx.com for the latest information.

Table 28: Maximum User I/O for XC4000E Device/Package Combinations

No. of Pins	Package (Code)	XC4003E	XC4005E	XC4006E	XC4008E	XC4010E	XC4013E	XC4020E	XC4025E
Maximum User I/O		80	112	128	144	160	192	224	256
84	PLCC (PC)	61	61	61	61	61			
100	PQFP (PQ)	77	77						
	VQFP (VQ)	77							
120	PGA (PG)	80							
144	TQFP (TQ)		112	113					
156	PGA (PG)		112	125					
160	PQFP (PQ)		112	128	129	129	129		
164	CBFP (CB)		112						
191	PGA (PG)				144	160			
196	CBFP (CB)					160			
208	PQFP (PQ)		112	128	144	160	160		
	HQFP (HQ)					160	160	160	
223	PGA (PG)						192	192	192
225	BGA (BG)					160	192		
228	CBFP (CB)						192		192
240	PQFP (PQ)						192		
	HQFP (HQ)						192	193	193
299	PGA (PG)								256
304	HQFP (HQ)								256

Note: This table includes standard user-programmable I/O. It also includes the TDI, TCK, and TMS pins, which can function as user-programmable I/O if not used for boundary scan. In addition to the I/O listed in this table, the M0 and M2 pins can be used as inputs only; the M1 and TDO pins can be used as outputs only. All of these pins must be called out using special library symbols. The XACT software does not use them by default. (See Table 18 on page 47.)

Table 29: Maximum User I/O for XC4000EX Device/Package Combinations

No. of Pins	Package (Code)	XC4028EX	XC4036EX	XC4044EX
Maximum User I/O		256	288	320
208	HQFP (HQ)	160		
240	HQFP (HQ)	193		
299	PGA (PG)	256		
304	HQFP (HQ)	256	256	
352	BGA (BG)	256		
411	PGA (PG)		288	320
432	BGA (BG)		288	320

Note: This table includes standard user-programmable I/O. It also includes the TDI, TCK, and TMS pins, which can function as user-programmable I/O if not used for boundary scan. In addition to the I/O listed in this table, the M0 and M2 pins can be used as inputs only; the M1 and TDO pins can be used as outputs only. All of these pins must be called out using special library symbols. The XACT software does not use them by default. (See Table 18 on page 47.)

Table 30: Maximum User I/O for XC4000L and XC4000XL Device/Package Combinations

No. of Pins	Package (Code)	XC4005L	XC4010L	XC4013L	XC4028XL	XC4036XL	XC4044XL	XC4052XL	XC4062XL
Maximum User I/O		112	160	192	256	288	320	352	384
84	PLCC (PC)	61	61						
176	TQFP (TQ)		153						
208	PQFP (PQ)	112	160	160					
208	HQFP (HQ)				160				
225	BGA (BG)			192					
240	PQFP (PQ)			192					
240	HQFP (HQ)				193				
299	PGA (PG)				256				
304	HQFP (HQ)				256	256			
352	BGA (BG)				256				
411	PGA (PG)					288	320	352	
432	BGA (BG)					288	320	352	
475	PGA (PG)								384

Note: This table includes standard user-programmable I/O. It also includes the TDI, TCK, and TMS pins, which can function as user-programmable I/O if not used for boundary scan. In addition to the I/O listed in this table, the M0 and M2 pins can be used as inputs only; the M1 and TDO pins can be used as outputs only. All of these pins must be called out using special library symbols. The XACT software does not use them by default. (See Table 18 on page 47.)

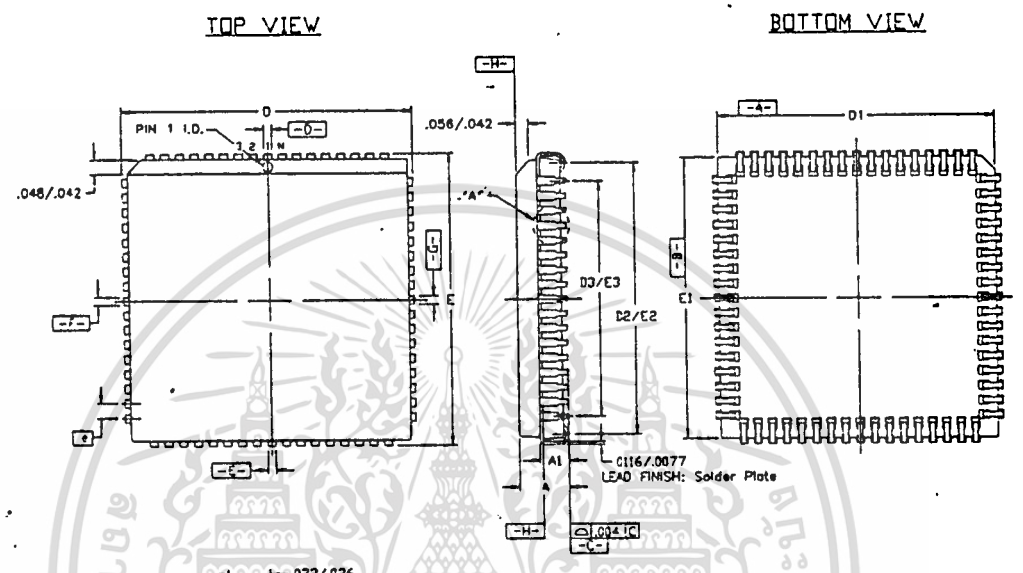
Ordering Information

Example: **XC4013E-3HQ240C**

- **Device Type**
- **Speed Grade**
  - -6
  - -5
  - -4
  - -3
  - -2
- **Temperature Range**
  - C = Commercial (T_J = 0 to +85°C)
  - I = Industrial (T_J = -40 to +100°C)
  - M = Military (T_C = -55 to+125°C)
- **Number of Pins**
- **Package Type**
  - PC = Plastic Lead Chip Carrier
  - PQ = Plastic Quad Flat Pack
  - VQ = Very Thin Quad Flat Pack
  - TQ = Thin Quad Flat Pack
  - BG = Ball Grid Array
  - PG = Ceramic Pin Grid Array
  - HQ = High Heat Dissipation Quad Flat Pack
  - MQ = Metal Quad Flat Pack
  - CB = Top Brazed Ceramic Quad Flat Pack

X6750

PLCC Packages — PC20, PC28, PC44, PC68, PC84



SYMBOL	INCHES									
	MIN.	MAX.	MIN.	MAX.	MIN.	MAX.	MIN.	MAX.	MIN.	MAX.
A	.165	.180	.165	.180	.165	.180	.165	.200	.165	.200
A_1	.090	.120	.099	.110	.090	.120	.090	.130	.090	.130
D/E	.385	.395	.485	.495	.685	.695	.985	.995	1.185	1.195
D_1/E_1	.350	.356	.450	.456	.650	.656	.950	.958	1.150	1.158
D_2/E_2	.290	.330	.390	.430	.590	.630	.890	.930	1.090	1.130
D_3/E_3	.200 REF.		.300 REF.		.500 REF.		.800 REF.		1.000 REF.	
e	.050 BSC		.050 BSC		.050 BSC		.050 BSC		.050 BSC	
N	20		28		44		68		84	

DETAIL 'A'

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1982.
2. DIMENSIONS 'D1' AND 'E1' DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS. MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED .010 PER SIDE.
3. 'N' IS NUMBER OF TERMINALS.
4. CONFORM TO JEDEC MO-047
5. TOP OF PACKAGE MAY BE SMALLER THAN BOTTOM BY .010".

20, 28, 44, 68 and 84-PIN PLCC (PC20 THRU PC84)

บรรณานุกรม

วิบูลย์ ชื่นแขก. ไมโครโปรเซสเซอร์ : สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

บรรจง ปิยะธำรง และ สุรเชษฐ์ ศรีพลกรัง . การออกแบบระบบดิจิตอลโดยใช้ VHDL :
การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 18

ยืน ภู่วรวรรณ และวัฒนา เชียงกูล . ไมโครโปรเซสเซอร์ไมโครคอมพิวเตอร์ :
ซีเอ็ดยูเคชั่น จำกัด

Jean-Michael Berge and Rolan Airiau. Circuit Synthesis with VHDL : Kluwer
Acadamic Publishers.1994

Richard Larry Ukriley. Field Programmable Gate Array(FPGAs) :
Practice-Hall Iinc.1993

Viewlogic Systems Inc. VHDL Modeling : Viewlogic Systems.1994

Xilinx Inc. The Programmable Logic Data Book : Xilinx Inc.1994

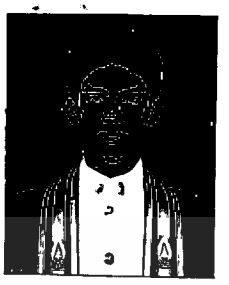
Xilinx Inc. Viewlogic Tutorials : Xilinx Inc.1994

Xilinx Inc. XACT Hardware & Peripherals Guide : Xilinx Inc.1994

Xilinx Inc. XACT Reference Guide. Volume II : Xilinx Inc.1994

Zainalabedin Navabi. VHDL Analysis and Modeling of Digital System :
McGRAW-HILL.1993

ประวัติผู้แต่ง



ชื่อผู้ทำปริญญานิพนธ์	นายจิรวัฒน์ นาคสง
วันเดือนปีเกิด	30 พฤษภาคม 2519
สถานที่เกิด	จังหวัดนครศรีธรรมราช
ภูมิลำเนาเดิม	18/11 ม.7 ต.นบปริง อ.เมือง จ.พังงา (82000)
ที่อยู่ปัจจุบัน	257 หมู่ 1 แขวงลาดกระบัง เขตลาดกระบัง กรุงเทพมหานคร 10520
โทรศัพท์	02-7390277-83 ต่อ 334
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนเมืองพังงา
มัธยมศึกษา	โรงเรียนดีบุกพังงาวิทยายน
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคพังงา
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคนครศรีธรรมราช
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รางวัล	-
ทุนการศึกษา	-
คติพจน์	ทำวันนี้ให้ดีที่สุด

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายนิคม ยอดมณี
วันเดือนปีเกิด	7 ธันวาคม 2515
สถานที่เกิด	จังหวัดสระแก้ว
ภูมิลำเนาเดิม	17 ม.9 ต.บ้านแก้ง อ.เมือง จ.สระแก้ว (27000)
ที่อยู่ปัจจุบัน	35/17 ซ.บริสุทธิ์ ต.หน้าเมือง อ.เมือง จังหวัดฉะเชิงเทรา (24000)
โทรศัพท์	038-512375
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนเขาสิงห์โต
มัธยมศึกษา	โรงเรียนบ้านแก้งวิทยา
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคฉะเชิงเทรา
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคฉะเชิงเทรา
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รางวัล	-
ทุนการศึกษา	-
คติพจน์	ก่อนจะเอาชนะผู้อื่น ต้องเอาชนะตัวเองให้ได้ก่อน

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายเอกรัตน์ นามวงค์
วันเดือนปีเกิด	27 กันยายน 2517
สถานที่เกิด	จังหวัดเชียงใหม่
ภูมิลำเนาเดิม	172/1 หมู่ 1 ต.อุโมงค์ อ.เมือง จ.ลำพูน (51150)
ที่อยู่ปัจจุบัน	257 หมู่ 1 แขวงลาดกระบัง เขตลาดกระบัง
	กรุงเทพมหานคร 10520
โทรศัพท์	02-7390277-83 ต่อ 334
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนมงฟอร์ตวิทยาลัย
มัธยมศึกษา	โรงเรียนมงฟอร์ตวิทยาลัย
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคลำพูน
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคลวิทยาเขตภาคพายัพ
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
	ภาควิชาครุศาสตร์วิศวกรรม
	คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รางวัล	-
ทุนการศึกษา	กรมพัฒนาฝีมือแรงงาน
คติพจน์	ชีวิตคือการเดินทาง เดินจากความมืดมนไปสู่ที่สว่าง

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายอรรถพล กิจปราชญ์
วันเดือนปีเกิด	17 กรกฎาคม 2518
สถานที่เกิด	จังหวัดพระนครศรีอยุธยา
ภูมิลำเนาเดิม	41 หมู่ 3 ต.ปลายกลัด อ.บางซ้าย
	จ. พระนครศรีอยุธยา (13270)
ที่อยู่ปัจจุบัน	333 หมู่ 1 แขวงลาดกระบัง เขตลาดกระบัง
	กรุงเทพมหานคร 10520
โทรศัพท์	02-7390255-58 ต่อ 2114
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนวัดทางหลวง
มัธยมศึกษา	โรงเรียนราษฎร์บำรุงศิลป์
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคพระนครศรีอยุธยา
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคพระนครศรีอยุธยา
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
	ภาควิชาครุศาสตร์วิศวกรรม
	คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รางวัล	-
ทุนการศึกษา	-
คติพจน์	-