การออกแบบวงจรรวมดิจิตอลโดยใช้วีแอชดีแอล

DIGITAL DESIGN USING VHDL

โดย

นาย สุรเชษฐ์ ศรีพลกรัง 35103255

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบวงจรรวมดิจิตอลโดยใช้วีแอชดีแอล

ผู้จัดทำ

1. นาย สุรเชษฐ์ ศรีพลกรัง 35103255

.....อาจารย์ที่ปรึกษา

(อาจารย์ บรรจง ปิยธำรง)

การออกแบบวงจรรวมดิจิตอลโดยใช้วีเอชดีแอล

นาย สุรเชษฐ์ ศรีพลกรัง 35103255
อาจารย์ บรรจง ปิยธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2537

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการพัฒนาวงจรรวมดิจิตอลโดยใช้ภาษา VHDL คำว่า VHDL ย่อมาจาก Vhsic Hardware Description Languge เป็นภาษาคอมพิวเตอร์ระดับสูงใช้ในการเขียนบรรยายฟังก์ชันการทำงานวงจรดิจิตอล โดยที่ตัวภาษาออกแบบมาเพื่อให้บรรยายได้หลาย ๆลักษณะตามความเหมาะสมได้แก่ การอธิบายพฤติกรรมของระบบ (Behavioral), การไหลของข้อมูลในระบบ (Dataflow) จนถึงอธิบายถึงการเชื่อมต่อกันระหว่าง Component ต่าง ๆ จนเป็นโครงสร้างของระบบ (Structural) ด้วยประโยชน์ของภาษานี้ รวมกับความสามารถของระบบที่ช่วยออกแบบทางอิเลคทรอนิกส์แบบอัตโนมัติ (Electronic Design Automation,EDA) ทำให้การออกแบบวงจรดิจิตอลขนาดเล็กจนถึงวงจรรวมขนาดใหญ่เป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ โครงงานนี้ได้ทดลองทำนาฬิกาปลุกดิจิตอล(Digital Alarm Clock)ขึ้นมา 1 ชิ้น โดยกำหนดคุณสมบัติ (Specification)ขึ้นมา เริ่มต้นด้วยการเขียน Source Code เพื่ออธิบายฟังก์ชันการทำงานขึ้นมาด้วยภาษา VHDL จากนั้นทำการคอมพายล์ (Compile), ดีบัก (Debug) และทดสอบการทำงาน (Simulate) จนได้ฟังก์ชันการทำงานตามต้องการ เสร็จแล้วใช้ความสามารถของซอฟแวร์ Synthesis ทำการแปลงโมเดล (Model) ที่อยู่ในรูปของ VHDL Source Code ให้เป็นวงจร Schematic Diagram หลังจากนั้นก็นำวงจร (Schematic Diagram) ไปสร้างฮาร์ดแวร์เป็นต้นแบบใช้งานจริงบน Field Programmable Gate Array (FPGA)

DIGITAL DESIGN USING VHDL

Mr.Surachet Sripolkrang

Mr.Bunjong Piyatamrong Advisor

1994

Abstract

This thesis presents the new trend for developing and modeling the large digital circuit by using VHDL language and CAE tools. VHDL is an acronym for VHSIC Hardware Description Language (VHSIC is Very High Speed Integrated Circuit). It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithm level to the gate level. The Complexity of the digital system being modeled could vary from that of a simple gate to a complete digital electronic system. With high performance CAE tools(Mentor Graphics,Viewlogic,or Neocad) make the development cycle of digital system faster and easier. This project develops the prototype of digital alarm clock. First, We created the VHDL model of digital alarm clock, Compiled and debugged. Until we got a complete VHDL source code for the digital alarm clock. Then, We have to simulate and verify the functional of the source code . Next, By using Synthesis software tools,it converts the compiled VHDL source code to the schematic diagram (digital circuit). With the schematic, We do the setting up on the FPGA (field programmable gate array XC3000,XC4000 family from XILINX) to implement the real working digital design. Finally we got the hardware prototype of the digital alarm clock on FPGA.

สารบัญ

XC3000,XC4000

Logic Cell Array family

บทที่ 1
บทนำ

ปริญญานิพนธ์ฉบับนี้นำเสนอการออกแบบวงจรรวมดิจิตอลโดยใช้ภาษา VHDL โดยใช้คอมพิวเตอร์ช่วยในการออกแบบ Computer Aided Engineering, CAE Tools เพื่อนำเสนอการออกแบบในแนวใหม่ซึ่งสะดวกรวดเร็วและมีประสิทธิภาพ โดยใช้เครื่องมือและทรัพยากรที่มีอยู่ ภาษา VHDL เป็นภาษาบรรยายการทำงานของฮาร์ดแวร์ เพื่อใช้อธิบายการทำงานของวงจรดิจิตอลตัวภาษานั้นประกอบไปด้วยรายละเอียดและส่วนประกอบต่าง ๆ ซึ่งใช้อธิบายพฤติกรรม (Behavioral) หรือโครงสร้าง (Structural) ของระบบ โดยพิจารณาถึงฐานเวลา (Timing) ของระบบเป็นหลักเนื่องจากว่าระบบที่เราจะออกแบบโดย VHDL นั้น ผู้ออกแบบไม่จำเป็นต้องรู้ถึงวงจรภายในว่ามีอะไรต่อกันอย่างไรบ้าง เพียงแต่กำหนด Input,Output,และฟังก์ชั่นการทำงานของระบบเขียนเป็นโปรแกรมแสดงการไหลของข้อมูล(Dataflow),การทำงานของระบบ (Behavioral) หรือโครงสร้างของระบบ(Structural) ขึ้นมา จากนั้นก็ทำการ Compile และ Simulate Model ที่ออกแบบมา เพื่อทำการวิเคราะห์ดูฟังก์ชั่นฐานเวลา (Timing) ของระบบว่าตรงตามต้องการหรือไม่ ถ้ามีการแก้ไขอย่างไรก็ทำการแก้ไข Source Code ใหม่ แล้ว Compile และ Simulate ซ้ำ ๆ จนกว่าจะได้ Model ที่มี Timing ของระบบตามต้องการ ซึ่งจะเห็นว่ามีความง่ายและสะดวกรวดเร็วกว่าเมื่อก่อนมาก เพราะไม่ต้องเสียเวลากับการสร้างวงจรทางฮาร์ดแวร์จริง ๆขึ้นมาทดสอบ ซึ่งทำให้เสียเวลาและค่าใช้จ่ายสูง Model ที่ออกแบบโดยใช้ VHDL สามารถทำการสังเคราะห์ (Synthesis) เพื่อให้ได้ Schematic Diagram และนำเอาไปสร้างเป็นวงจรจริง ๆ ได้ โดยอาจนำเอาไปสร้างเป็น ASIC หรือนำไป Burn ลง EPLD หรือ FPGA เพื่อนำไปใช้งานจริง ๆต่อไป ประโยชน์ของภาษา VHDL ใช้กันมากในการออกแบบ Chip ASIC (Application Specific Integrated Circuit) หรือแม้กระทั่งการออกแบบไมโครโปรเซอร์ในปัจจุบันก็ใช้ภาษา VHDL ในการออกแบบ จะเห็นได้ว่าภาษา VHDL มีประโยชน์อย่างยิ่งในการออกแบบระบบวงจรรวมในปัจจุบันและมี Software หลายตัวซึ่งสนับสนุนภาษา VHDL ตั้งแต่การ Compile , Simulate ,Synthesis ได้แก่ Mentor Graphics , Viewlogic เป็นต้น ประกอบกับคอมพิวเตอร์ระดับเวอร์กสเตชั่น (Work Station) ที่มีระบบกราฟิคที่ดีและการประมวลผลด้วยความเร็วสูงในปัจจุบัน ทำให้ออกแบบระบบอิเลคทรอนิกส์แบบอัตโนมัติ (Electronic Design Automation) ทุกขั้นตอนเป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ Software CAE ที่ใช้ในการพัฒนาโครงงานนี้ขึ้นมาคือ Mentor Graphic ,Viewlogic และ Neocad ตั้งแต่การเขียน source code ,การ compile VHDL source code ,การทดสอบฟังก์ชันการทำงาน เมื่อได้ Source Code ที่สมบูรณ์ ของ Model แล้ว ก็ทำการแปลง Source Code ให้อยู่ในรูป Schematic Diagram เพื่อที่จะไปสร้างบน Field Programmable gate array (FPGA) ซึ่งผู้จัดทำได้รับการแนะนำและความช่วยเหลือเป็นอย่างดีจากอาจารย์ที่ปรึกษาซึ่งท่านได้ชี้แนวทางการทำงานตลอดจนส่งเข้ารับการฝึกอบรมเกี่ยวกับเนื้อหาในการออกแบบ เพื่อให้ผู้

จัดทำได้มีความรู้เพียงพอที่จะพัฒนาระบบขึ้นมาเองได้ โครงงานนี้ได้ทดลองพัฒนานาฬิกาปลุกดิจิตอล(Digital Alarm Clock) โดยกำหนดคุณสมบัติขึ้นมาเองอ้างอิงกับผลิตภัณฑ์ที่มีอยู่แล้วในท้องตลาดเพื่อให้ได้งานที่สามารถพัฒนาขึ้นมาเองได้และมีประสิทธิภาพเหมือนกัน ตลอดภาคการศึกษาผู้จัดทำได้ศึกษาเนื้อหามากมายเกี่ยวกับตัวภาษา VHDL , ซอฟแวร์ CAE Tools ที่ใช้ในการพัฒนาได้แก่ Mentor Graphics, Viewlogic, Neocad และ XACT Development ซึ่งล้วนเป็นสภาพแวดล้อมในการพัฒนาแบบใหม่ ผู้จัดทำยอมรับว่าต้องมีความผิดพลาดและไม่สมบูรณ์อยู่ในตัวรายงานและโครงงานซึ่งผู้เขียนเองยินดีรับคำติชมจากท่านผู้อ่านทุกประการ เพื่อที่จะได้ปรับปรุงแก้ไขให้ดีขึ้นต่อไป ในที่นี้ขอขอบคุณท่านอาจารย์บรรจง ปิยธำรง อาจารย์ที่ปรึกษา ซึ่งได้ชี้แนวทางในการทำงานและจัดหาเครื่องมือต่าง ๆให้ได้ใช้จนพัฒนางานขึ้นมาได้ หวังว่ารายงานเล่มนี้จะเป็นประโยชน์ต่อท่านผู้ที่สนใจเพื่อเป็นแนวทางประดิษฐ์สิ่งใหม่ ๆขึ้นมาต่อไป

ผู้จัดทำ

บทที่ 2
การออกแบบวงจรรวม ASIC

บทนำ

นับตั้งแต่มีการประดิษฐ์ทรานซิสเตอร์เมื่อกว่าสามสิบปีที่ผ่านมา อุปกรณ์อิเลคทรอนิคส์ได้เข้ามามีบทบาทามากมายต่อการดำเนินชีวิตของมนุษย์ และเมื่อเทคโนโลยีไมโครอิเลคทรอนิคส์ก้าวหน้ามาถึงระดับที่สามสามารถรวมเอาวงจรทรานซิสเตอร์ จำนวนนับแสนนับล้านตัวบรรจุไว้ในอุปกรณ์ขนาดจิ๋ว ที่เรียกกันว่าวงจรรวม (Integrated Circuit) หรือ IC อุปกรณ์มหัศจรรย์ขนาดจิ๋วนี้ ได้กลายเป็นส่วนประกอบที่สำคัญของอุปกรณ์และเครื่องมือเครื่องใช้เกือบทุกชนิด ในชีวิตประจำวันของเราอาทิ เครื่องคิดเลข โทรทัศน์ เครื่องซักผ้า วิดีโอเกมส์ คอมพิวเตอร์ เป็นต้น วงจรรวมจึงนับเป็นสิ่งประดิษฐ์ที่สำคัญ ยิ่งอย่างหนึ่งของมนุษยชาติ จนกระทั่งบางคนถือว่าวงจรรวมเป็นการปฏิวัติอุตสาหกรรมครั้งใหม่

ในระยะที่สินค้าอิเลคทรอนิกส์ยังเป็นสิ่งประดิษฐ์ที่มีความซับซ้อนไม่มากนักผู้ผลิตจะเริ่มต้นจากการนำวงจรรวมมาตรฐาน ที่ผลิตจากบริษัทผู้ผลิตวงจรรวมรายใหญ่ เช่น INTEL Motorola Hitachi มาประกอบลงบนแผ่นวงจรพิมพ์ (Printed Circuit Board) แล้วนำไปรวมกับอุปกรณ์อื่น ๆสร้างขึ้นเป็นสินค้าอิเลคทรอนิกส์จัดจำหน่ายต่อไป วงจรมาตรฐานที่ผลิตออกจำหน่ายทั่วไปนี้เรียกว่า Standard Product IC (SPIC) ตัวอย่างวงจรรวม SPIC ที่รู้จักกันดีได้แก่ วงจรตระกูล TTL 74/5400 วงจรรวม CMOS 4000 เป็นต้น

เทคโนโลยีสารกึ่งตัวนำที่พัฒนาขึ้น ทำให้สามารถเพิ่มวงจรจำนวนทรานซิสเตอร์ที่บรรจุในวงจรรวมขึ้นอย่างรวดเร็วจากเพียงไม่กี่สิบตัว ในวงจรรวมระดับ Small Scale Integrated , SSI) ในราว ค.ศ. 1961 มาเป็นหลายร้อยตัวในระดับ Medium Scale Integration (MSI) ภายในห้าปีถัดมาจำนวนวงจรทรานซิสเตอร์ที่บรรจุในวงจรรวมเพิ่มขึ้นหลายพันตัว กลายเป็นวงจรระดับ Large Scale Integration (LSI) ราวค.ศ. 1971 ในปัจจุบันเราสามารถบรรจุวงจรทรานซิสเตอร์ลงในวงจรรวมได้นับหมื่นตัวที่เรียกกันว่าวงจรรวมขนาดใหญ่มาก (Very Large Scale Integration, VLSI) การที่มนุษย์สามารถสร้างวงจรขนาดใหญ่มากขึ้นได้ทำให้สามารถสร้างสรรค์สินค้าอิเลคทรอนิกส์ที่มีความสลับซับซ้อนมากขึ้นได้อย่างที่ไม่เคยปรากฏมาก่อน เช่น เครื่องคิดเลข เครื่องไมโครคอมพิวเตอร์ เป็นต้น ทิศทางความก้าวหน้าของเทคโนโลยีทำให้เชื่อได้ว่าในอนาคตอันใกล้นี้มนุษย์สามารถสร้างวงจรรวมที่บรรจุทรานซิสเตอร์ได้หลายล้านตัว ดังแสดงให้เห็นในวิวัฒนาการในตารางที่ 2.1

ปี ค.ศ.1947		1961	1966	1971	1980	1985	1990	
เทคโนโลยี	ประดิษฐ์ทรานซิสเตอร์	อุปกรณ์ดีสครีท	SSI	MSI	LSI	VLSI	ULSI	GLSI
จำนวนทรานซิสเตอร์/ชิพของผลิตภัณฑ์เชิงพานิชย์โดยประมาณ	1	1	10	100 -1000	1000-20000	20000-500000	>500,000	> 10 ล้านตัว
ตัวอย่างของผลิตภัณฑ์	-	จังชั่นทรานซิสเตอร์ไดโอด	เกทฟลิปฟลอป	ตัวนับ, มัลติเพล็กเซอร์	ไมโครโปรเซอร์แบบ 8 บิท ROM, RAM	ไมโครโปรเซอร์แบบ 16 บิท, 32 บิท	ไมโครโปรเซอร์แบบพิเศษ อุปกรณ์ประมวลแบบ Real time	

1. ULSI Ultra large - scale integration
2. GLSI Giant large - scale integration

ตาราง 2.1

ต่อมาเมื่อสินค้าอิเลคทรอนิคส์มีความสลับซับซ้อนมากขึ้น พร้อมกับการแข่งขันในตลาดมีความรุนแรงมากขึ้น ผู้ผลิตสินค้าอิเลคทรอนิกส์จึงต้องหาทางเพิ่มประสิทธิภาพสินค้าของตนเองควบคู่ไปกับการลดต้นทุนการผลิต และป้องกันการลอกเลียนแบบสินค้าของตน หนทางหนึ่งที่ทำได้คือ ลดจำนวนของวงจรรวมในแผ่นวงจรพิมพ์ให้น้อยลง โดยรวมเอาวงจรรวมมาตรฐาน SPIC หลายตัวเข้าเป็นวงจรรวมตัวเดียวกันหรือในบางครั้งจะต้องมีการดัดแปลงวงจรนี้ให้มีรายละเอียดและคุณสมบัติแตกต่างไปจากวงจรรวม SPIC ที่มีจำหน่ายโดยทั่วไป จากแนวความคิดนี้เอง ทำให้เกิดวงจรรวมอีกกลุ่มหนึ่งที่ออกแบบและผลิตให้มีคุณสมบัติตรงตามความต้องการของผู้ใช้วงจรกลุ่มนี้เราเรียกว่า Application Specific IC หรือ ASIC

ในขณะที่วงจรรวม SPIC ผลิตจำหน่ายโดยบริษัทขนาดใหญ่ในลักษณะผลิตคราวละมาก ๆ(Mass Production) นับแสนนับล้านตัว โดยมีต้นทุนการผลิตและออกแบบสูงมาก วงจรรวม ASIC สามารถผลิตได้ทั้งในลักษณะเฉพาะลูกค้า (CUSTOM Made) จำนวนน้อย ๆไปจนกระทั่งผลิตคราวละนับแสนตัว และถึงแม้ว่าต้นทุนต่อตัวจะสูงกว่าวงจรรวม SPIC แต่การลงทุนทั้งหมดจะต่ำกว่า จึงมีบริษัทที่ผลิตวงจรรวมขนาดใหญ่และเล็กเป็นจำนวนมากให้บริการออกแบบและเจือสาร วงจรรวม ASIC อาทิ VLSI Technology,Hitachi,AWA

ข้อดีของการใช้วงจรรวม ASIC เมื่อเปรียบเทียบกับการใช้วงจรรวม SPIC ในสินค้าอิเลคทรอนิคส์ชนิดเดียวกันพอสรุปได้ดังนี้

1. ลดต้นทุนการผลิตซึ่งเห็นได้จากการที่ขนาดของแผ่นวงจรพิมพ์มีขนาดเล็กลงราคาจึงถูกลงด้วย

2. ลดพลังงานไฟฟ้าสูญเสียที่เกิดขึ้น

3. เพิ่มความเชื่อถือได้ของวงจรอิเลคทรอนิกส์เนื่องจากมีอุปกรณ์น้อยลง

4. ป้องกันการลอกเลียนแบบสินค้า เนื่องจากการลอกเลียนแบบวงจรรวม ASIC ทำได้ยากกว่าวงจรรวม SPIC มาก

5. เพิ่มประสิทธิภาพการแข่งขันในตลาดสินค้า เนื่องจากลดเวลาการออกแบบลงได้มากและดัดแปลงให้สินค้ามีคุณสมบัติพิเศษต่าง ๆตามความต้องการของลูกค้าได้

จากข้อดีข้างต้นทำให้วงจรรวม ASIC มีบทบาทอย่างมากมายในวงการอุตสาหกรรมอิเลคทรอนิกส์ของประเทศอุตสาหกรรมใหม่ โดยเฉพาะไต้หวัน ฮ่องกง และสิงคโปร์ ดังนั้นการที่ประเทศไทยจะได้เรียนรู้และนำเทคโนโลยีของวงจรรวม ASIC มาใช้งานจึงเป็นประโยชน์อย่างยิ่ง

ในบทนี้จะแนะนำให้ผู้อ่านทราบถึงการจำแนกประเภทของวงจรรวม ASIC การใช้คอมพิวเตอร์ในการออกแบบวงจรรวม การส่งไปเจือสารและข้อพิจารณาในการเลือกใช้วงจรรวม ASIC

การจำแนกประเภทของวงจรรวม ASIC

วงจรรวมประเภท ASIC อาจจำแนกได้ตามลักษณะผู้ออกแบบวงจร(ผู้ใช้หรือผู้ผลิต) และ ตามลักษณะทางโปรแกรม

ในเชิงพานิชย์ เรามักจำแนกวงจรรวม ASIC ตามลักษณะผู้ออกแบบเป็น 2 ประเภท ดังนี้

1. Full Custom ASIC

วงจรรวม ASIC ประเภทนี้ ลักษณะการออกแบบคล้ายคลึงกับวงจรรวม SPIC เพียงแต่ผู้ใช้เป็นผู้ออกแบบแทนที่ผู้ผลิตจะเป็นออกแบบ ผู้ใช้ต้องออกแบบวงจรรวม ASIC แบบ Full Custom ตั้งแต่ระดับทรานซิสเตอร์ จนกระทั่งรวมเป็นวงจรขนาดใหญ่มากที่ทำงานได้ตามความต้องการ การออกแบบวงจรรวมประเภทนี้ต้องใช้ผู้ออกแบบที่มีความสามารถ ทักษะ และประสบการณ์สูง เนื่องจากเป็นการออกแบบโครงสร้างและมิติในขั้นงานซิลิคอนโดยตรง (Silicon Geometric Shape) ระบบคอมพิวเตอร์ช่วยออกแบบ (CAD) จึงมีความสลับซับซ้อนมาก ทำให้ต้องใช้เวลานับเป็นเดือนในการเรียนรู้และออกแบบวงจรรวมประเภทนี้

2. Semi-Custom ASIC

วงจรรวม ASIC ประเภทนี้ ผู้ผลิตได้รวบรวมวงจรพื้นฐานต่างๆ ไว้ในวงจรรวม ASIC ตัวเดียวกัน ซึ่งผู้ใช้จะเป็นผู้ออกแบบเชื่อมโยงวงจรพื้นฐานที่มีอยู่นี้เข้าเป็นวงจรขนาดใหญ่มากที่ทำงานได้ตามความต้องการ โดยใช้คอมพิวเตอร์เป็นเครื่องมือช่วยในการออกแบบ วงจรรวม ASIC ประเภทนี้มีบทบาทมากในอุตสาหกรรมอิเล็กทรอนิกส์ของประเทศอุตสาหกรรมใหม่ ประเทศไทยมีความพร้อมทั้งด้านบุคลากรและเครื่องมือในการออกแบบวงจรรวม ASIC ประเภทนี้ จึงเป็นสิ่งที่น่าจะให้ความสนใจเป็นพิเศษ

วงจรรวมASIC แบบ Semi-Custom

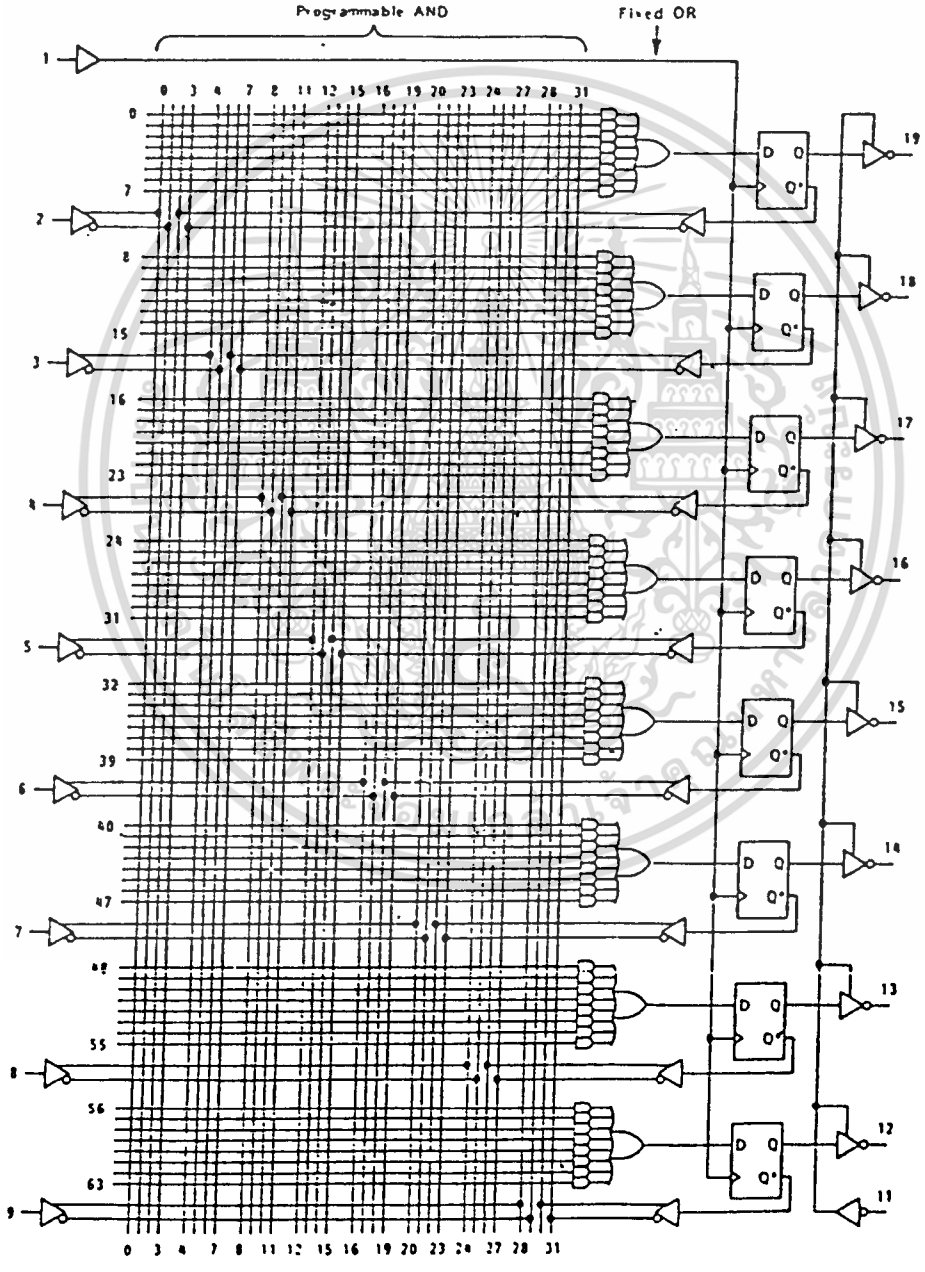
วงจรรวม ASIC แบบ Semi-Custom จำแนกออกเป็น 3 กลุ่มใหญ่ๆ ได้แก่

1. Programmable Logic Devices (PLD)

วงจรรวม ASIC กลุ่มนี้ ผู้ผลิตได้บรรจุพื้นฐานประเภทเกท (Gate) แบบต่างๆเช่น AND OR INVERTER ไว้ในวงจรรวมตัวเดียวกัน ผู้ใช้สามารถออกแบบเชื่อมโยงวงจรประเภทนี้เข้าเป็นวงจรเชิงเลขขนาดใหญ่ที่ทำงานตามความต้องการได้ และเมื่อออกแบบเสร็จแล้วก็สามารถผลิตขึ้นใช้ได้เองโดยไม่ต้องสั่งให้ผู้ผลิตทำการเจือสารแต่อย่างใด ระยะเวลานับจากการออกแบบจนกระทั่งผลิตขึ้นใช้งานจึงสั้นที่สุดเมื่อเทียบกับวงจรรวม ASIC กลุ่มอื่น จึงเหมาะสมที่จะพัฒนาต้นแบบหรือผลิตสินค้าที่มีจำนวนขายนับเป็นสิบเป็นร้อยตัวเท่านั้น วงจรรวม PLD แบ่งออกเป็น 3 กลุ่มย่อย ได้แก่

1.1 Programmable Array Logic (PAL)

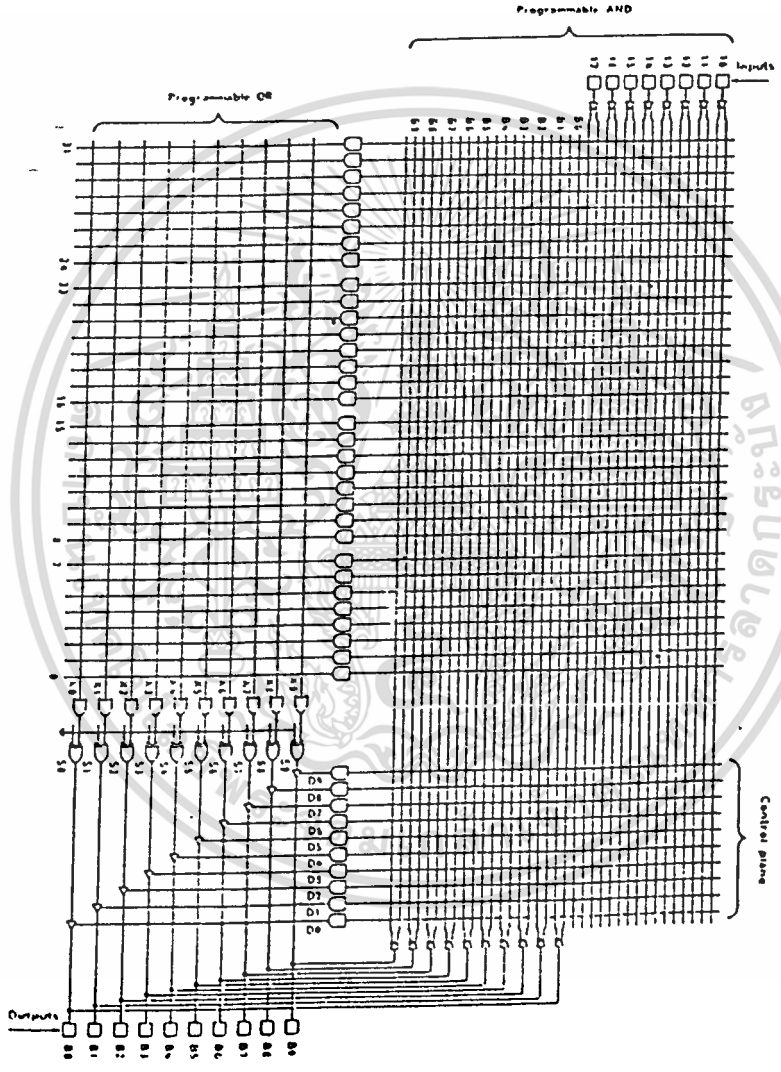
วงจรรวม PAL แต่ละตัวจะประกอบด้วย วงจรเกทแบบ AND ที่เลือกต่อสายขาอินพุทได้ตามต้องการ กับวงจรเกทแบบ OR ที่ต่อสายไว้คงที่แล้ว ดังแสดงในรูปที่ 1.1 ผู้ใช้สามารถเลือกต่อสายอินพุทให้วงจรรวม PAL ตัวนี้ทำงานตามความต้องการได้ และจากลักษณะนี้เองทำให้บางคนเรียกวงจร PAL ว่าเป็นวงจรรวมที่มี Fixed OR และ Programmable AND วงจรรวม PAL นี้เริ่มผลิตขึ้นประมาณ ค.ศ. 1970 โดยบริษัท MMI ประเทศสหรัฐอเมริกา



รูปที่ 1.1 วงจรภาย PAL

1.2 Programmable Logic Array (PLA)

วงจรรวม ASIC กลุ่มนี้ แต่ละตัวจะประกอบด้วย วงจรเกทแบบ AND และ OR ที่ผู้ใช้เลือกต่อสายขาอินพุทได้ตามต้องการ ดังแสดงในรูปที่ 1.2 บางคนจึงเรียกว่าเป็นวงจรรวมที่มี Programmable AND และ OR วงจรรวมPLA นี้เริ่มผลิตขึ้นประมาณ ค.ศ. 1975 โดยบริษัท Signetics สหรัฐอเมริกาวงจรรวม PLA ตัวหนึ่งจะทดแทนวงจร SPIC ขนาดเล็ก (SSI) ได้ประมาณ 15-20 ตัว

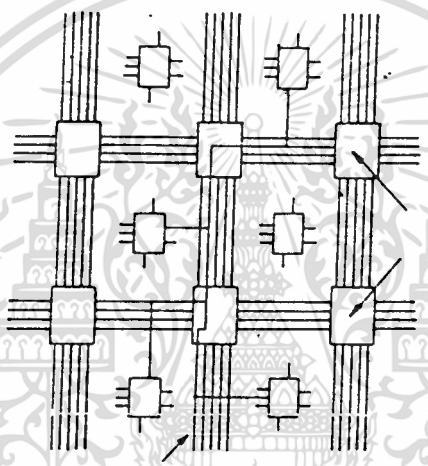


รูปที่ 1.2 วงจรรวม PLA

อนึ่ง ผู้ผลิต PLA บางรายจะเรียกวงจร PLA ของตนว่าเป็น Programmable Gate Array และผู้ผลิตวงจรรวม PLA บางรายก็เรียกวงจรรวม PAL ของตนว่า Programmable Logic Array ดังนั้นผู้ใช้จำเป็นต้องพิจารณารายละเอียดคุณสมบัติของวงจร ASIC นั้นโดยละเอียด ก่อนการเลือกซื้อมาใช้งาน

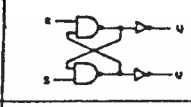
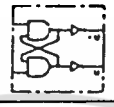
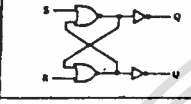
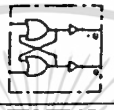
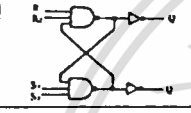
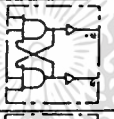
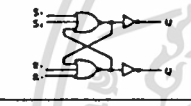
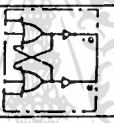
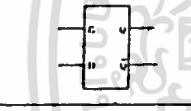
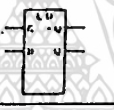
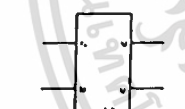
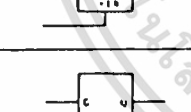
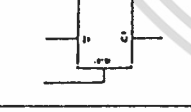
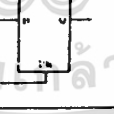
1.3 Programmable gate Array (PGA)

วงจรรวม ASIC กลุ่มนี้ เริ่มผลิตขึ้นประมาณปี ค.ศ. 1986 โดยบริษัท Xilink สหรัฐอเมริกาและบางครั้งจะรู้จักกันในชื่อว่า Logic cell Array (LCA) วงจรรวม LCA นี้ประกอบด้วยหน่วยวงจรพื้นฐานที่ผู้ใช้สามารถกำหนดหน้าที่การทำงานได้ตามต้องการ เรียกว่า Configurable Logic block (CLB) ผู้ใช้สามารถเลือกต่อสายเชื่อมโยงระหว่าง CLB แต่ละหน่วยเพื่อให้เป็นวงจรขนาดใหญ่ที่ทำงานตามที่ต้องการได้ ดังแสดงในรูปที่ 1.3 เมื่อออกแบบเรียบร้อยแล้วผู้ใช้สามารถโปรแกรมวงจรรวม LCA นี้ขึ้นใช้งานได้ด้วยตนเอง ไม่จำเป็นต้องส่งไปเจือสารที่โรงงานผู้ผลิตแต่อย่างใด วงจร LCA นี้สามาถถใช้กับวงจรเชิงเลขขนาดตั้งแต่ 100 เกท ขึ้นไปจนกระทั่งนับเป็นพันเกทก็ได้



รูปที่ 1.3 การเชื่อมต่อ CLB ภายในวงจรรวม LCA

LATCHES (with Scan Function)

Macrocell: Function / Function Name	Equivalent Circuit	Symbol	Delay: Input Name	Output Name	t_{PLH} (ns)	t_{PHL} (ns)
$\overline{RS}$-Latch LRS0			$\overline{S}$	+Q	2.7	–
			R		2.5	2.6
			$\overline{S}$	−Q	2.5	2.6
			R		2.7	–
RS-Latch LRS3			S	+Q	2.4	2.7
			R		–	3.3
			S	−Q	–	3.3
			R		2.4	2.7
R2S2-Latch LR2S20			S	+Q	2.7	–
			R		2.6	2.6
			S	−Q	2.6	2.6
			R		2.7	–
R2S2-Latch LR2S23			S	+Q	2.4	3.1
			R		–	3.7
			S	−Q	–	3.7
			R		2.4	3.1
D-Latch LD			G	+Q	3.2	2.9
			D		3.2	2.9
			G	−Q	2.6	2.9
			D		2.6	2.9
D-Latch with CLR LDC1			G	+Q	3.5	3.4
			CL		2.3	2.4
			D		3.5	3.4
			G	−Q	3.1	3.2
			CL		2.1	2.0
			D		3.1	3.2
D-Latch with PRE LDP1			G	+Q	3.3	3.2
			PR		2.4	2.5
			D		3.3	3.2
			G	−Q	2.9	3.0
			PR		2.2	2.1
			D		2.9	3.0
D-Latch with $\overline{PRE}$/ $\overline{CLR}$ LDPC3			G	+Q	3.8	3.5
			$\overline{PR}$		2.9	2.8
			$\overline{CL}$		2.3	2.4
			D		3.8	3.5
			G	−Q	3.2	3.5
			$\overline{PR}$		2.5	2.6
			$\overline{CL}$		2.1	2.0
			D		3.2	3.5

Macrocell: Function / Function Name	Input Load Unit Load Count	Normalized Load Factor	Clamp Level When Used	Symbol No.	Delay: Input Name	Output Name	t_{PLH} (ns)	R_{LH}	t_{PHL} (ns)	R_{HL}
$\overline{RS}$-Latch LRS0	8	1	@	A3	$\overline{S}$	+Q	2.7	1.0	–	0.6
			@		R		2.5		2.6	
					$\overline{S}$	−Q	2.5	1.0	2.6	0.6
					R		2.7		–	
RS-Latch LRS3	8	1	#	A3	S	+Q	2.4	1.0	2.7	0.6
			#		R		–		3.3	
					S	−Q	–	1.0	3.3	0.6
					R		2.4		2.7	
R2S2-Latch LR2S20	9	1	@	A4	S	+Q	2.7	1.0	–	0.6
			@		R		2.6		2.6	
			@		S	−Q	2.6	1.0	2.6	0.6
			@		R		2.7		–	
R2S2-Latch LR2S23	9	1	#	A4	S	+Q	2.4	1.0	3.1	0.6
			#		R		–		3.7	
			#		S	−Q	–	1.0	3.7	0.6
			#		R		2.4		3.1	
D-Latch LD	5	1	@	C	G	+Q	3.2	1.2	2.9	0.9
		1	@		D		3.2		2.9	
					G	−Q	2.6	1.2	2.9	0.9
					D		2.6		2.9	
D-Latch with CLR LDC1	6	1	@	C	G	+Q	3.5	1.2	3.4	0.9
		1	@		CL		2.3		2.4	
					D		3.5		3.4	
					G	−Q	3.1	1.2	3.2	0.9
		1	#		CL		2.1		2.0	
					D		3.1		3.2	
D-Latch with PRE LDP1	6	1	@	C	G	+Q	3.3	1.2	3.2	0.9
		1	@		PR		2.4		2.5	
					D		3.3		3.2	
					G	−Q	2.9	1.2	3.0	0.9
		1	#		PR		2.2		2.1	
					D		2.9		3.0	
D-Latch with $\overline{PRE}$/ $\overline{CLR}$ LDPC3	7	1	@	C	G	+Q	3.8	1.2	3.5	0.9
		1	@		$\overline{PR}$		2.9		2.8	
					$\overline{CL}$		2.3		2.4	
					D		3.8		3.5	
					G	−Q	3.2	1.2	3.5	0.9
		1	#		$\overline{PR}$		2.5		2.6	
		1	#		$\overline{CL}$		2.1		2.0	
					D		3.2		3.5	

รูปที่ 1.4 ตัวอย่าง Macrocell

2. Gate Array

วงจร ASIC ประเภทนี้ ผู้ผลิตจะบรรจุวงจรพื้นฐานประเภทเกท (Gate) แบบต่างๆ ไว้ในวงจรรวมเดียวกัน และวงจรเกทบางส่วนจะต่อสายภายในไว้เป็นกลุ่มวงจรเชิงเลขที่ใช้งานกันทั่วไป ได้แก่ วงจร Latch วงจร Decoder วงจร Register เป็นต้น กลุ่มวงจรเชิงเลขที่รวบรวมบรรจุไว้ในวงจร Gate Array นี้เรียกว่า Macrocell Logic-cell ดังรูปที่ 1.4 ผู้ใช้มีหน้าที่ออกแบบต่อสายเชื่อมโยงระหว่าง Macrocell แต่ละตัวเข้าด้วยกันให้เป็นวงจรเชิงเลขขนาดใหญ่ที่ทำงานได้ตามต้องการ แล้วส่งข้อมูลการออกแบบนี้ไปให้ผู้ผลิตทำการเจือสารต่อไป การออกแบบวงจรรวม Gate Array นี้ ผู้ใช้ต้องมีข้อมูลเกี่ยวกับตัววงจรรวม Gate Array และ Macrocell ที่บรรจุอยู่ภายในวงจรรวมตัวนั้น ซึ่งตามปกติผู้ผลิตจะส่งมาให้โดยไม่คิดค่าใช้จ่าย เนื่องจากวงจรรวมประเภทนี้ต้องเจือสารโดยผู้ผลิตหลังการออกแบบทำให้มีต้นทุนการผลิตสูงคิดเป็นเงินหลายแสนบาทในการผลิตแต่ละครั้ง จึบเหมาะสำหรับการออกแบบเชิงพาณิชย์ที่มีปริมาณผลิตตั้งแต่หนึ่งพันตัวขึ้นไป ในปัจจุบันมีผู้ผลิตหลายรายที่ให้บริการเจือสารวงจรรวม Gate Array เช่น Hitachi,NEC,Texas Instruments เป็นต้น

3. Standard Cell

วงจรรวม ASIC ประเภทนี้ ผู้ใช้เป็นผู้เลือกกลุ่มวงจรทำหน้าที่ต่างๆ เช่น เกท, ฟลิปฟลอบตัวนับหน่วยความจำ ไมโครโปรเซสเซอร์ จากแฟ้มข้อมูลคอมพิวเตอร์เป็นเครื่องมือช่วยในการออกแบบ กลุ่มวงจรนี้เรียกว่า Standard cell กลุ่มวงจร Standard cell นี้ ผู้ผลิตจะทำการออกแบบให้ใช้พื้นที่ซิลิคอนอย่างเป็นประโยชน์สูงสุด และเจือสารทดสอบคุณสมบัติว่าตรงตามที่ออกแบบไว้จริงก่อนแจกจ่ายไปให้กับลูกค้า กลุ่มวงจร Standard cell ส่วนใหญ่จะคล้ายคลึงกับวงจรรวม SPIC ที่ใช้กันมากอาทิ วงจรรวมตระกูล 7400 ว่งจร รวมตระกูล 4000 ดังนั้นแทนที่ผู้ใช้จะเลือกวงจรรวม SPIC หลายตัวมาประกอบลงบนแผงวงจรพิมพ์ ผู้ใช้จะเลือกกลุ่มวงจร Standard cell จากแฟ้มข้อมูลคอมพิวเตอร์ของผู้ผลิต แล้วนำมาออกแบบเชื่อมโยงกันบนแผนซิลิคอนอย่างเต็มที่ ในขณะที่วงจรรวม Gate Array นั้น พื้นที่ซิลิคอนบางส่วนจะสูญเสียไปโดยไร้ประโยชน์ การออกแบบจึงอาศัยความชำนาญแบะประสบการณ์ใกล้เคียงกับวงจรรวม ASIC แบบ Full custom เวลาในการออกแบบโดยเฉลี่ยนับเป็นเดือน นอกจากนั้นแล้วเวลาที่ต้องใช้ในการเจือสารจะยาวกว่าวงจรรวม Gate Array สองถึงสามเท่า และต้นทุนการผลิตจะสูงนับแสนนับล้านบาทต่อการผลิตแต่ละครั้ง วงจรรวม Standard cell จึงเหมาะสมกับการออกแบบเชิงพาณิชย์ที่มีปริมาณการผลิตสูงนับพันนับหมื่นตัวในปัจจุบันมีผู้ผลิตหลายรายที่ให้บริการเจือสารวงจรรวม Standard cell เช่น NEC,AWA

อนึ่ง ปัจจุบันมีหน่วยงานในสหรัฐอเมริกาและในยุโรป ให้บริการเจือสารวงจรรวม Standard cell และ Full custom ในราคาประหยัด วงเงินตั้งแต่ไม่กี่หมื่นบาทจนถึงเกือบหนึ่งล้านบาท ดยรวบรวมวงจรของผู้ใช้หลายคนเจือสารเข้าไว้ในวงจรรวมตัวเดียวกัน วงจรรวมที่ผลิตในลักษณะนี้เรียกว่า Multi-project chip อย่างไรก็ตาม วงจรรวม Multi-project chip นี้ มุ่งประโยชน์ทางการศึกษาและการค้นคว้าวิจัยเป็นสำคัญ

การแบ่งประเภทวงจรรวม ASIC ตามลักษณะการโปรแกรม

บางครั้งเราจะจำแนกประเภทวงจรรวม ASIC ตามลักษณะการโปรแกรม ได้เป็น 2 ประเภท ได้แก่

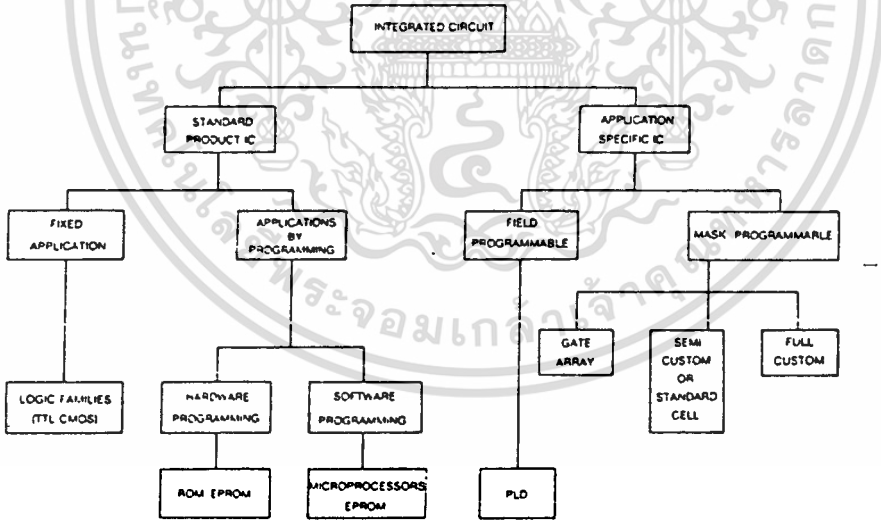
1. Field Programmable ASIC

วงจรรวม ASIC กลุ่มนี้ ผู้ใช้สามารถออกแบบ และโปรแกรมการทำงานได้ด้วยตนเองโดยมิต้องใช้บริการเจือสารจากผู้ผลิต ตัวอย่างวงจรรวมกลุ่มนี้ได้แก่ Programmable Logic Devices (PLD)

2. Mask Programmable ASIC

วงจรรวม ASIC กลุ่มนี้ หลังจากผู้ใช้ออกแบบตามความต้องการแล้ว ต้องส่งให้ผู้ผลิตทำการเจือสาร (Fabrication) ตัวอย่างวงจรรวมกลุ่มนี้ได้แก่ Gate Array, Standard cell

รูปที่ 1.5 จะแสดงให้เห็นการจำแนกวงจรรวม SPIC และ ASIC ตามแนวทางนี้ ซึ่งการจำแนกประเภทวงจรรวมนี้จะแตกต่างกันไปตามแนวทางที่ผู้จำแนกยึดเป็นหลัก เช่น ลักษณะการผลิตลักษณะการติดต่อระหว่างโรงงานกับผู้ใช้ลักษณะการออกแบบ



รูปที่ 1.5 วงจรรวมประเภทต่าง ๆ

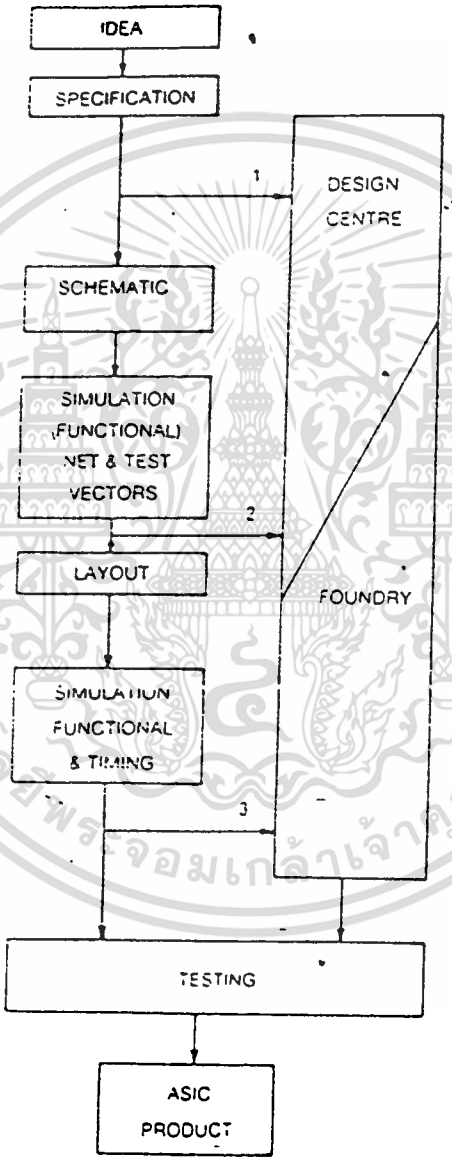
บทที่ 3
การใช้คอมพิวเตอร์ช่วยในการออกแบบวงจรรวม

การใช้คอมพิวเตอร์ช่วยออกแบบวงจรรวม
(Computer Aided Design/Computer Aided Enginnering,CAD/CAE)

การพัฒนาของระบบคอมพิวเตอร์ ตั้งแต่เริ่มต้นมาจนถึงปัจจุบัน มีวัตถุประสงค์ใหญ่เพื่อใช้เป็นเครื่องมือช่วยการทำงานต่าง ๆ ของมนุษย์ คอมพิวเตอร์ขนาดเล็กในปัจจุบันมีความสามารถทัดเทียมกับคอมพิวเตอร์ขนาดใหญ่มากในอดีต การนำเอาคอมพิวเตอร์ขนาดเล็กมาช่วยใช้งานก็สามารถใช้งานได้ซับซ้อนมีประสิทธิภาพสูงเพียงพอ คอมพิวเตอร์ส่วนบุคคลและคอมพิวเตอร์เอ็นจีเนียริ่งเวอร์คสเตชั่นได้เข้ามามีบทบาทใช้ช่วยงานด้านต่าง ๆ ในการออกแบบวงจรรวมก็เช่นกัน ได้มีการสร้างโปรแกรมช่วยออกแบบวงจรรวมขนาดใหญ่มาก (VLSI CAD TOOLS) โดยเฉพาะวงจรรวม ASIC ได้มีบริษัทผู้ผลิตซอฟท์แวร์ช่วยออกแบบวงจรรวมหลายรายซึ่งทำงานระบบคอมพิวเตอร์ตั้งแต่ระดับไมโครคอมพิวเตอร์ มินิคอมพิวเตอร์ และเอ็นจิเนียริ่งเวิคร์สเตชั่น ดังนั้น การออกแบบวงจรรวมในปัจจุบันจึงมักใช้ระบบคอมพิวเตอร์ช่วยออกแบบเป็นส่วนใหญ่ และโรงงานผลิตวงจรรวม ASIC ในปัจจุบันก็สามารถรับข้อมูลเพื่อผลิตวงจรรวมจากการออกแบบบนคอมพิวเตอร์ได้เช่นกัน

ระบบคอมพิวเตอร์โดยทั่วไปจะประกอบด้วย 2 ส่วน คือ ฮาร์ดแวร์และซอฟท์แวร์ ในระบบคอมพิวเตอร์ช่วยออกแบบวงจรรวม ทางด้านฮาร์ดแวร์อาจใช้คอมพิวเตอร์ที่เพิ่มเติมความสามารถในการแสดงกราฟฟิคที่มีความละเอียดและแสดงสีได้ และอุปกรณ์รับตำแหน่ง เช่น เมาท์หรือดิจิไตเซอร์ อาจใช้โปรเซสเซอร์ช่วยในการคำนวณ (Math Coprocessor) ร่วม และอาจรวมถึงระบบเนทเวอร์คหรือเครือข่ายเพื่อที่จะทำให้ผู้ออกแบบสามารถทำงานเป็นทีมโดยเชื่อมโยงแบบผ่านระบบเครือข่ายของคอมพิวเตอร์ได้ด้วย คอมพิวเตอร์ที่นิยมนำมาใช้เป็นเครื่องมือออกแบบวงจรรวมอย่างมากในปัจจุบันเป็นระบบคอมพิวเตอร์ที่มีความสมบูรณ์

ซอฟท์แวร์คอมพิวเตอร์ช่วยในการออกแบบวงจรรวม ASIC ต้องอาศัยคอมพิวเตอร์ที่มีขีดความสามารถในการทำงานสูงควบคู่ไปกับความสามารถแสดงภาพกราฟิคที่มีความละเอียดสูงและมีหน่วยความจำสำรองขนาดใหญ่พอสมควร ซึ่งได้แก่ คอมพิวเตอร์แบบ Engineering Workstation เครื่อง Workstation ที่รู้จักกันดีในประเทศไทยได้แก่ SUN,HP,DEC และ Apollo ซอฟท์แวร์ออกแบบที่มีเสียงได้แก่ Mentor Graphic,Daisy,Viewlogic อย่างไรก็ตามในปัจจุบันมีซอฟแวร์ออกแบบ ASIC ลายตัวที่ทำงานบนเครื่องคอมพิวเตอร์ส่วนบุคคล โดยเฉพาะอย่างยิ่ง ซอฟท์แวร์ออกแบบวงจรรวม PLD ซึ่งนอกจากจะช่วยออกแบบแล้วยังช่วยโปรแกรมวงจรรวม PLD ให้ทำงานตามต้องการได้อีกด้วย การออกแบบวงจรรวม ASIC โดยใช้คอมพิวเตอร์ ประกอบด้วย 3 ขั้นตอนที่สำคัญดังแสดงในรูป ดังนี้



รูปที่ 1.6 การออกแบบวงจรรวม ASIC

1.Design Entry

ผู้ออกแบบเริ่มต้นด้วยการกำหนดรายละเอียดหน้าที่การทำงานของวงจรรวมตามที่ตนต้องการ Functional Specification) แล้วจึงแปลงรายละเอียดนี้ให้อยู่ในรูปแบบที่เข้าใจได้ซึ่งอาจจะเป็นภาษารรยายฮาร์ดแวร์(Hardware Description Language หรือ HDL) หรือภาพวงจรไฟฟ้า (Schematic) โดยทั่วไปแล้วจะได้แฟ้มข้อมูลชื่อ Net-List จากการป้อนภาพวงจรไฟฟ้า (Schematic capture) เข้าสู่คอมพิวเตอร์ และความสัมพันธ์ระหว่างอินพุทและเอาท์พุทที่เกิดจากอินพุทนั้น จะสร้างเป็นแฟ้มข้อมูลชื่อ Test Vector นำไปใช้งานในขั้นต่อไป

2. Simulation

ซอฟท์แวร์ออกแบบวงจรรวมจะเปลี่ยนข้อมูลในแฟ้มข้อมูล Net-list ให้เป็นแฟ้มข้อมูลที่สามารถจำลองแบบการทำงานได้ การจำลองแบบการทำงานนี้จะใช้ตรวจสอบความถูกต้องในการทำงานของวงจรที่ออกแบบตามข้อมูล Test Vector ที่ผู้ออกแบบกำหนดไว้ โดยพิจารณาจากตารางเวลา Timing diagram) และการจำลองความผิดพลาดที่เกิดขึ้น การจำลองแบบการทำงานในขั้นตอนนี้ ผู้ออกแบบลองจำเป็นต้องมีข้อมูลคุณสมบัติทางไฟฟ้าของวงจรรวมที่คาดว่าจะใช้จากผู้ผลิต จึงจะได้ผลสมบรูณ์

3. Physical Layout

หากผลการจำลองแบบการทำงานเป็นไปตามที่คาดหมายไว้ ขั้นตอนต่อไปจะเป็นการออกแบบวงจรรวมเพื่อการผลิตโดยตรง กล่าวคือ ออกแบบส่วนประกอบสำคัญที่เรียกว่า หน้ากาก (Mask) ของวงจรรวม หน้าที่หลักที่ต้องกระทำมี 2 ประการคือ วางตำแหน่งกลุ่มวงจรต่างๆ ให้เหมาะสม Placement) และต่อสายเชื่อมโยงกลุ่มวงจรต่างๆเข้าด้วยกัน (Routing) ถ้าซอฟท์แวร์ที่ใช้ในขั้นตอนนี้ช่วยในการวาดภาพหน้ากากโดยตรง จะเรียกว่าเป็นแบบ Geometry layout แต่ถ้าเป็นการสร้างสัญญลักษณ์แทนส่วนประกอบต่างๆ ของวงจรรวม แล้วแปลงสัญญลักษณ์ดังกล่าวให้กลายเป็นภาพหน้าการอีกต่อหนึ่งจะเรียกว่าเป็นแบบ Symbolic layout ซอฟท์แวร์แบบแรกต้องใช้ผู้ออกแบบที่มีความรู้ด้านวงจรรวมมากกว่าแบบหลังซึ่งวิศวกรคแมพิวเตอร์หรือวิศวกรไฟฟ้าที่พอมีความรู้ด้านวงจรรวมสามารถใช้งานได้ ข้อมูลคอมพิวเตอร์ที่ได้จากขั้นตอนนี้จะอยู่ในรูปแบบมาตรฐานที่ผู้ผลิตยอมรับได้แก่ CIF (CalTech Intermediate Fom) หรือ Calma GDS-2 หลังจากการออกแบบ Physical layout แล้ว จะมีการจำลองแบบการทำงานอีกครั้งเพื่อตรวจสอบการทำงาน ก่อนส่งแฟ้มข้อมูลการออกแบบไปทำการเจือสารนำมาใช้งานต่อไป

อนึ่ง จะพิจารณาได้ว่าวงจรรวมประเภท Mark Programmable ASIC เท่านั้น ที่ต้องการทำการออกแบบครบถ้วนทั้งสามขั้นตอน ส่วนวงจรรวมประเภท Field Programmable ASIC นั้น ใช้ในงานได้ทันที

การเจือสารวงจรรวม ASIC

วงจรประเภท Mask Programmable ASIC นั้นเมื่อออกแบบเสร็จแล้ว ต้องส่งให้ผู้ผลิตทำการเจือสารผลิตออกใช้งานต่อไป ซึ่งผู้ผลิตจะคิดค่าบริการที่เรียกว่า Non Return Engineering (NRE) charge เพิ่มเติมจากค่าใช้จ่ายในการเจือสาร ค่า NRE นี้เป็นค่าจ้างเตรียมการสำหรับการเจือสารวงจรรวมที่ผู้ผลิตออกแบบส่งมาและจะเปลี่ยนแปลงไปตามสภาพความพร้อมในการเจือสารที่ผู้ออกแบบได้ดำเนินการไปแล้ว ตามปกติผู้ออกแบบสามารถส่งวงจรรวมที่ออกแบบมาให้ผู้ผลิตเจือสารได้ 3 ระดับ ดังนี้

ระดับที่ 1ผู้ออกแบบเพียงแต่ส่งภาพวงจรที่ออกแบบและ Test vector ให้กับผู้ผลิต โดยอาจอยู่บนแผ่นกระดาษ หรือตัวกลางคอมพิวเตอร์อื่นๆ ก็ได้ ในลักษณะนี้ผู้ออกแบบเป็นเพียงผู้กำหนดหน้าที่ของวงจรรวมที่ต้องการภาระการออกแบบวงจรรวมเกือบทั้งหมดเป็นของผู้ผลิต ค่า NRE ในระดับนี้จึงสูงที่สุด แต่ความเสี่ยงของผู้ใช้น้อยที่สุดเป็นวิธีที่เหมาะสมผู้ใช้ที่ไม่คุ้นเคยกับการออกแบบวงจรรวม

ระดับที่ 2 ผู้ออกแบบจะป้อนข้อมูลวงจรเข้าในซอฟท์แวร์ออกแบบวงจรรวมที่ผู้ผลิตยอมรับพร้อมทำการจำลองแบบการทำงานตรวจสอบความถูกต้องในการทำงานจนกระทั่งได้ผลถูกต้องตาที่ออกแบบไว้ ผู้ออกแบบจะส่งแฟ้มข้อมูลวงจรนี้พร้อมผลการจำลองแบบการทำงาน ไปให้ผู้ผลิตทำการเจือสารผลิตขึ้นใช้งาน ในระดับนี้ผู้ออกแบบจะรับผิดชอบออกแบบในระดับวงจร ส่วนผู้ผลิตจะทำหน้าที่ออกแบบระดับซิลิคอน ค่า NRE ระดับนี้จะลดลงจากระดับแรกประมาณ 30 % การส่งไปเจือสารระดับนี้น่าจะเหมาะสมที่สุดกับประเทศไทยในขณะนี้

ระดับที่ 3 ผู้ออกแบบจะออกแบบหน้าการของวงจรรวมโดยตรง ผู้ผลิตเพียงแต่รับข้อมูลที่อยู่ในรูปแบบมาตรฐาน เช่น CIF หรือ GDS-2 ไปทำการสร้างหน้ากากของวงจรรวมแล้วผลิตใช้งานต่อไปถึงแม้ว่าการส่งไปเจือสารระดับนี้จะเสียค่า NRE ต่ำที่สุด แต่ก็มีเพียงผู้ผลิตไม่กี่รายที่ยอมรับการเจือสารในระดับนี้ส่วนหนึ่งจะเป็นบริการเพื่อการศึกษาและวิจัย เช่น โครงการ MPC ของ AWA โครงการ MOSIS โครงการ ES II

คอมพิวเตอร์เอ็นจิเนียริ่งเวอร์คสเตชั่น (Engineering Work Station)

ระบบคอมพิวเตอร์เอ็นจิเนียริ่งเวร์คสเตชั่น(Engineering Workstation) เป็นระบบคอมพิวเตอร์ที่มีความสามารถสูงเทียบเท่ามินิคอมพิวเตอร์ แต่มีขนาดพอ ๆ กับคอมพิวเตอร์ส่วนบุคคลเน้นการใช้งานคนเดียวต่อ 1 เครื่อง ในปัจจุบันได้มีการนำระบบเอ็นจิเนียริ่งเวร์คสเตชั่นเข้ามาใช้กับงานที่ต้องการระบบคอมพิวเตอร์ที่มีประสิทธิภาพสูง ๆ เช่น งานออกแบบต่าง ๆ (Computer Aided Design,CAD) งานด้านวิศวกรรม(Computer Aided Enginnering , CAE) หรือแม้แต่การทำงานด้านกราฟฟิค (Computer Graphic) เช่น ในงานโฆษณา เป็นต้น ความสามารถและคุณสมบัติของเอ็นจิเนียริ่งเวร์คสเตชั่น เริ่มตั้งแต่ความสามารถของซีพียูในระดับตั้งแต่ 32 บิทขึ้นไป เช่น 68020,RISC PROCESSOR มีซีพียูช่วยคำนวณ มีความเร็วในการทำคำสั่งตั้งแต่ 1 ล้านคำสั่งต่อวินาที (Million Instructions Per Second,MIPS) เวร์คสเตชั่นปัจจุบันความสามารถในการทำคำสั่งสูงถึง 27 MIPS ต้องสามารถทำงานหลายงานพร้อมกัน (Multitasking) ได้ มีการจัดหน่วยความจำเสมือน (Virtual Memory) หน่วยความจำหลักของเวร์คสเตชั่นมีขนาดตั้งแต่ 4 เมกกะไบต์ขึ้นไป อุปกรณ์สำรองข้อมูล (Storage) คือฮาร์ดดิสก์ที่มีขนาดตั้งแต่ 70 เมกกะไบต์ถึงกิกะไบต์ ความสามารถด้านกราฟฟิคมักจะใช้จอภาพขนาด 15 - 19 นิ้ว มีความละเอียดของภาพตั้งแต่ 1024 x 800 จุด (Pixels) แต่ละจุดสามารถแสดงสีได้ตั้งแต่ 16 สีขึ้นไป มีความเร็วในการแสดงภาพกราฟฟิค 2 มิติแบบเวคเตอร์ตั้งแต่ 5000 เวคเตอร์ต่อวินาทีขึ้นไป และถ้ามีวงจรช่วยด้านกราฟฟิค (Graphic Accelator) จะสามารถแสดงกราฟฟิคได้ด้วยความเร็วตั้งแต่ 40,000 เวคเตอร์ต่อวินาทีขึ้นไป ระบบคอมพิวเตอร์เอ็นจิเนียริ่งเวร์คสเตชั่นโดยปกติจะเน้นการใช้งาน 1 คนต่อ 1 เครื่อง แต่ความสามารถในการต่อระบบเครือข่ายซึ่งเป็นจุดหลักอีกประการหนึ่งของเวร์คสเตชั่นเพื่อใช้อุปกรณ์หรือเนื้อที่เก็บโปรแกรมและข้อมูลร่วมกันทำให้สามารถทำงานเป็นกลุ่ม (Workgroup)ได้ดี เครือข่ายที่ใช้เป็นเครือข่ายระยะใกล้ คือ เครือข่ายท้องถิ่น (Local Area Network) ที่มีความเร็วในการติดต่อผ่านระบบเนทเวร์คในระดับ 10 ล้านบิทต่อวินาที (Megabit per Seconds)

ซอฟท์แวร์ปฏิบัติการบนเวร์คสเตชั่น มักจะใช้โปรแกรมระบบปฏิบัติการยูนิกส์ (UNIX) เนื่องจากเป็นโปรแกรมระบบปฏิบัติการที่สามารถทำงานได้หลายงานพร้อมกันหรือเป็นระบบปฏิบัติงานหลายผู้ใช้งานก็ได้ และมักจะมีโปรแกรมจัดระบบหน้าต่าง (WINDOW) เพื่อให้สามารถแบ่งแยกแสดงงานได้หลายงานพร้อม ๆ กันบนจอภาพ 19 นิ้ว ในปัจจุบันได้มีการค้นคิดมาตรฐานการติดต่อระหว่างคอมพิวเตอร์การจัดการหน้าต่าง และการแสดกราฟฟิคบนระบบปฏิบัติการ UNIX ที่มีชื่อเรียกว่า X-WINDOW เพื่อให้คอมพิวเตอร์ที่ใช้ระบบปฏิบัติการ UNIX สามารถติดต่อกันได้เป็นมาตรฐานเดียวกันและก็ได้มีผู้ที่นำเอาระบบ X - WINDOW เข้ามาใช้กับเวร์คสเตชั่นแล้ว

เอ็นจิเนียริ่งเวร์คสเตชั่นในปัจจุบันมีผลิตมากมาย เช่น SUN MICROSYSTEM ผู้ผลิตระบบเอ็นจิเนียริ่งเวร์คสเตชั่น SUN ได้ผลิต SUN 3 , SUN 4 , รุ่นต่าง ๆ ใช้ไมโครโปรเซสเซอร์ 68020 , 68030 เป็นซีพียู และSPARC PROCESSOR ซึ่งออกแบบและผลิตโดยเฉพาะโดย SUN MICROSYSTEM เอง, HP 9000 SERIES ของ HEWLETT PACKARD ใช้ซีพียู 68020 หน่วยความจำ 4 เมกกะไบต์ ทำงานที่ความถี่ 16.67 เมกะเฮิร์ท จอภาพ 16 หรือ 19 นิ้ว แสดงสี 256 สี 1280 x 1042 จุด, APOLLO DOMAIN อนุกรม 3000, 4000 , 10000 ใช้ซีพียู 68020, 68030 หน่วยความจำ 2 - 4 เมกะไบต์ ระบบเนทเวร์คแบบ Token Ring ใช้โปรแกรมระบบปฏิบัติการ UNIX ,VAX STATION ของ DIGITAL , IBM อนุกรม 6000 ซึ่งใช้ RISC Processor เป็นต้น

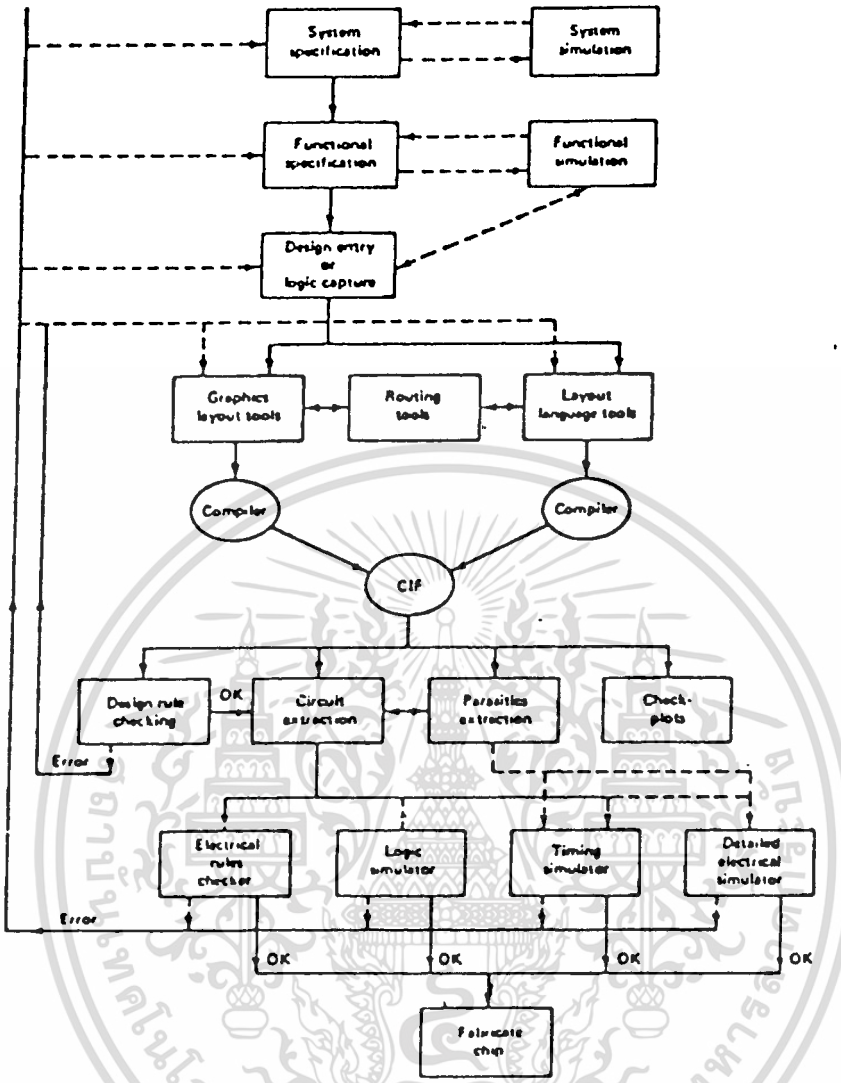
จากความสามารถและประสิทธิภาพที่สูงยิ่งของเอ็นจิเนียริ่งเวร์คสเตชั่น ซึ่งเหมาะสมต่อการใช้งานช่วยออกแบบและงานด้านวิศวกรรม ทำให้มีผู้ผลิตซอฟท์แวร์ช่วยออกแบบวงจรรวมที่ทำงานบนเอ็นจิเนียริ่งเวร์คสเตชั่นหลายราย เช่น MENTOR GRAPHICS , DAZIX , HILO , CALMA เป็นต้น รวมทั้งซอฟท์แวร์ช่วยออกแบบที่มหาวิทยาลัยต่าง ๆ สร้างขึ้นก็มักจะสร้างบนเอ็นจิเนียริ่งเวร์คสเตชั่นเช่นกัน

ซอฟท์แวร์ช่วยออกแบบวงจรรวมขนาดใหญ่มาก

ระบบของซอฟท์แวร์ที่ใช้ช่วยออกแบบวงจรรวมขนาดใหญ่มาก (VLSI CAD TOOLS) มีระดับของการออกแบบหลายระดับ ตั้งแต่ระดับของการออกแบบการทำงานในส่วนย่อยของระบบ การออกแบบในระดับล่างสุดคือ การนำทรานซิสเตอร์มาต่อขึ้นเป็นวงจรลอจิกขนาดเล็ก เช่น เกท, ฟลิปฟลอปบ หรืออาจแบ่งเป็นการออกแบบในระดับเซลล์ (Logic Cell) การออกแบบในระดับโมดุล (Function Module)

ผังของระบบซอฟท์แวร์ช่วยออกแบบวงจรรวมขนาดใหญ่ โดยทั่วไปแสดงดังในรูปที่ 3.2 ซึ่งสามารถแบ่งเป็นส่วนใหญ่ ๆ ได้ 2 ส่วนคือ

- ส่วนของการป้อนแบบ, วางผัง และสร้างหน้ากาก
- ส่วนของการวิเคราะห์ คือ การตรวจสอบความถูกต้องและจำลองการทำงาน



รูปที่ 4.2 ผังโดยทั่วไปของซอฟท์แวร์ช่วยออกแบบวงจรรวมขนาดใหญ่มาก

ขั้นตอนของการออกแบบวงจรรวมขนาดใหญ่มากจากผังรูปที่ 3.2 จะเริ่มจากการระบุการทำงานของระบบใหญ่ (System Specification) ซึ่งจะมีโปรแกรมช่วยจำลองการทำงานของระบบ (System Simulation) เพื่อดูว่าสิ่งที่ระบุไปนั้นถูกต้องหรือไม่ ระบบที่เราได้ใส่ข้อระบุวิธีการทำงานจะประกอบไปด้วยบล็อกหรือส่วนย่อยของระบบ ในแต่ละส่วนย่อยนั้นจะถูกนำมากำหนดคุณสมบัติเป็น

ฟังก์ชั่น (Function Specification) และจำลองการทำงานของแต่ละส่วนย่อย รวมทั้งส่วนรวมของระบบที่ประกอบจากส่วนย่อยต่าง ๆ (Function Simulation)

ลักษณะการระบุการทำงานของระบบหรือส่วนย่อยในระบบนั้นมักจะบุเป็นภาษาซึ่งบรรยายถึงคุณสมบัติของระบบ ภาษาดังกล่าวจะเป็นภาษาที่มีโครงสร้างคล้ายกับภาษาโปรแกรมชั้นสูง (High Level Language) และอาจมีโปรแกรมที่ติดต่อกับผู้ออกแบบโดยใช้กราฟฟิคเพื่อให้ผู้ใช้ป้อนภาพของระบบการแบ่งระบบเป็นส่วนย่อย เมื่อทดสอบจำลองการทำงานได้ถูกต้องก็จะแบ่งการออกแบบของส่วนต่าง ๆ ลงไปในระดับของรายละเอียดของวงจรจนถึงระดับของหน้ากาก ซึ่งขึ้นกับระดับของระบบออกแบบและชนิดของวงจรรวมที่จะออกแบบด้วย เช่น การออกแบบวงจรรวม Full Custom ผู้ออกแบบจะลงไปถึงส่วนของหน้ากากวงจรรวม ประกอบด้วยการจัดวางชั้นต่าง ๆ ของสารกึ่งตัวนำ การออกแบบวงจรรวมโดยใช้เซลล์มาตรฐาน ผู้ออกแบบเพียงแต่ออกแบบถึงระดับวงจรที่ประกอบด้วยอุปกรณ์มาตรฐาน ซึ่งในปัจจุบันมีตั้งแต่ระดับเกทขึ้นไปจนถึงโปรเซสเซอร์หรือวงจรคอนโทรลเล่อร์ เช่น ซีพียู Z - 80 ตัวควบคุมจอภาพ (CRT Controller) 6845 เป็นต้น การออกแบบเพื่อใช้กับอุปกรณ์ PLD ก็จะออกแบบถึงระดับวงจรที่ประกอบด้วยเกทหรือฟลิปฟลอบ ฯลฯ ขั้นตอนการแยกย่อยถึงระดับวงจรนี้คือ Design Entry หรือ Logic Capture หรือ Schematic Capture ซึ่งวิธีการระบุถึงการต่อวงจรต่าง ๆ นั้นมี 2 รูปแบบ คือ การต่อวงจรโดยใช้ภาพกราฟฟิคแสดงสัญลักษณ์ของวงจร ใช้ภาษาระบุหรือป้อนเป็นข้อความซึ่งแสดงถึงการใช้อุปกรณ์ต่าง ๆ การต่อเชื่อมสัญญาณของอุปกรณ์ต่าง ๆ เข้าด้วยกัน จากวงจรที่ป้อนเข้าไปนี้ก็สามารถนำไปจำลองการทำงานแต่ละส่วนย่อย เพื่อเปรียบเทียบทดสอบดูว่าตรงกับที่ต้องการหรือไม่

การระบุหรือป้อนข้อมูลในระดับวงจรหรือระดับหน้ากาก จะเป็นส่วนย่อยที่สุดซึ่งสามารถนำไปสร้างเป็นข้อมูลมาตรฐานที่ใช้ผลิตเป็นวงจรได้ โดยจะมีโปรแกรมแปลงจากข้อมูลหรือNet list ออกไปข้อมูลมาตรฐานที่ใช้ในการผลิตวงจรรวม CIF,Calma ซึ่งเป็นข้อมูลมาตรฐานในการผลิจวงจรรวม Full custom หรือ JEDEC ซึ่งเป็นข้อมูลมาตรฐานเพื่อผลิตวงจรรวม PLD แต่ถ้าเป็นวงจรรวมที่ใช้เซลล์มาตรฐานก็อาจใช้ข้อมูลวงจรเพื่อส่งไปยังโรงงานผลิตได้เลย ในกรณีของวงจรรวม Full custom ซึ่งผู้ออกแบบจะต้องออกแบบถึงระดับหน้ากากของวงจรรวมนั้น เครื่องมือที่ใช้ในการออกแบบหน้ากากโดยใช้ภาพกราฟฟิคก็ยังแบ่งเป็น 2 วิธี คือ ออกแบบแผนภาพของหน้ากากจริง ๆ (Geometry Layout)หรือใช้สัญลักษณ์แสดงอุปกรณ์ในระดับสารกึ่งตัวนำ (Symbolic layout) ในการออกแบบวิธีแรกนั้น ผู้ออกแบบจะต้องมีความรู้และความชำนาญเกี่ยวกับกฎการออกแบบเป็นอย่างดีจะต้องระวังและสร้างภาพหน้ากากที่ถูกต้อง ส่วนการออกแบบในวิธีหลังคือการใช้สัญลักษณ์นั้น ผู้ออกแบบเพียงแต่ใช้สัญลักษณ์แทนแผนภาพซึ่งจะประกอบด้วยสัญลักษณ์ต่างๆ คือ มอสทรานซิสเตอร์ (MOS transistor) ทางเดินของสารชั้นต่างๆ (wire)การเจาะเชื่อมแบบต่างๆ (cut) กรอบของผนังเพื่อสร้างข้างใต้ที่ต่างชนิดกับชนิดของแผ่นวงจรรวม (well) จะมีโปรแกรมที่จัดการแปลงสัญลักษณ์เป็นแผนภาพของหน้ากากซึ่งจะดูแลการจัดให้ตรงตามกฎการออกแบบด้วย

นอกจากนี้การออกแบบวงจรรวม Full custom ยังมีขั้นตอนอื่นๆ อีกคือ การตรวจสอบหน้ากากตามกฎของการออกแบบ (Desihn Rule Checking, DRC)กฎการออกแบบนี้จะต่างกันไปขึ้นกับเทคโนโลยี(Technology file) ของโรงงานที่ต้องการส่งไปผลิตด้วยโปรแกรมตรวจสอบกฎการออกแบบจะรายงานข้อผิดพลาดถ้าหน้ากากที่ออกแบบผิดจากกฎการออกแบบ ถ้ามีข้อผิดพลาดดังกล่าวแล้วส่งไปผลิตอาจได้วงจรรวมที่มีคุณสมบัติผิดจากที่ต้องการได้ มีการตรวจสอบกฎทางไฟฟ้า (Electrical Rule Checker)เช่น การเชื่อมต่อสัญญาณครบถ้วนทุกจุดหรือไม่ มีจุดใดในวงจรถูกปล่อยลอยหรือไม่ หรือมีจุดใดที่อาจเกิดการลัดวงจรได้บ้าง จุดใดอาจเกิดปรากฏการณ์ latch up ได้บ้าง ฯลฯ

การจำลองการทำงานจากข้อมูลหน้ากากซึ่งจะต้องผ่านการแปลงจากข้อมูลหน้ากากเพื่อสร้างข้อมูลคุณสมบัติของวงจรและคุณสมบัติทางไฟฟ้า เช่น การเป็นสวิตช์ทางเดินสัญญาณ ค่าความเหนี่ยวนำ การเก็บประจุ การต้านทาน การนำกระแส ระหว่างจุดต่างๆ ในวงจร(Circuit extraction, parasitics extraction) แล้วจึงนำมาจำลองการทำงานซึ่งมี 2 ลักษณะ คือ การจำลองสถานะทางลอจิกของสัญญาณออกเทียบกับสัญญาณเข้า (Logic Simulation) ซึ่งจะสนใจเฉพาะค่าทางลอจิกที่เกิดขึ้นจากวงจรเท่านั้น และการจำลองทางเวลา (Timming Simulation) ซึ่งจะสนใจเรื่องของการดีเลย์สัญญาณ อันเป็นผลเนื่องมาจากค่าความเหนี่ยวนำ ความต้านทาน ค่าการเก็บประจุภายในวงจรสามารถใช้โปรแกรมจำลองแบบวงจรอนาลอกได้และโปรแกรมที่นิยมนำมาใช้คือ โปรแกรม SPICE ที่สร้างขึ้นโดย University of California at Berkeley ตั้งแต่ปี 1970 และต่อมาก็มีผู้นำไปดัดแปลงและใช้ชื่อทางการค้าใหม่แต่มักจะยังคงชื่อ SPICE ไว้ในชื่อใหม่ เช่นPSPICE ,MGSPICE (Mentor Graphics), DESPICE (Daisy,Cadenentic Inc)เป็นต้น ภาพกากที่ผ่านการทดสอบและจำลองการทำงานแล้วสามารถนำ ส่งไปผลิตวงจรรวมในโรงงานผลิตวงจรรวมต่อไปได้

ในหัวข้อต่อๆ ไปในบทนี้ จะได้กล่าวถึงระบบคอมพิวเตอร์ช่วยออกแบบวงจรที่มีจำหน่ายและมีใช้อยู่หลายๆ แบบ ทั้งที่เป็นระบบทางการค้าและที่ใช้สำหรับงานวิจัยพัฒนาที่มีในประเทศไทย

ระบบคอมพิวเตอร์ช่วยออกแบบวงจรรวม Mentor Graphics

Mentor Graphics เป็นบริษัทผู้ผลิตและจำหน่ายระบบช่วยออกแบบวงจรอิเลคทรอนิคส์ (Electronics Design Automation) ซึ่งมีซอฟท์แวร์ช่วยออกแบบที่สมบูรณ์มากระบบหนึ่ง ระบบฮาร์ดแวร์ที่ Mentor Graphics ใช้เป็นเครื่องคอมพิวเตอร์ Engineering Workstation ที่มีความสามารถสูง พร้อมที่จะใช้ออกแบบเป็นกลุ่มเป็นอย่างดีในส่วนของการออกแบบวงจรรวม Mentor Graphics ได้แบ่งระบบช่วยออกเป็น 3 ระบบคือ

- **Chip** Station เป็นระบบออกแบบวงจรรวม Full Custom ซึ่งสามารถออกแบบวงจรได้ในระดับฟังก์ชัน(Function Block) โดยใช้ภาษาบรรยายฮาร์ดแวร์ การป้อนวงจรภายนอกแต่ละ Block ใช้ในการจำลองการทำงานของระบบเพื่อทดสอบความถูกต้อง การออกแบบจัดวางหน้ากาก (Floor Planing) ซึ่งจัดวางแต่ละ block ของวงจรเข้าไปถึงระดับการป้อนหน้ากากในระดับล่างสุดซึ่งมีโปรแกรมตรวจสอบกฎการออกแบบ(DRC) จำลองการทำงานทั้งลอจิกและอนาลอก
- **Cell** Station ระบบการออกแบบวงจรรวมโดยใช้เซลล์มาตรฐาน ซึ่งอาจจะต้องมี Library ของเซลล์จากโรงงานที่ต้องการส่งไปเจือสารผลิตวงจรรวม Library ของ Cell ต่างๆจะเก็บข้อมูลหน้ากากของวงจรรวมในระดับกายภาพ คุณสมบัติทางไฟฟ้า ข้อมูลสำหรับจำลองการทำงาน ซึ่งทางโรงงานผู้เจือสารมักจะจ่ายให้แก่ผู้ออกแบบโดยไม่คิดมูลค่า
- **Gate** Station ระบบออกแบบวงจรรวม Gate Array มีขั้นตอนการออกแบบตั้งแต่การป้อนวงจรการจำลองเพื่อทดสอบการทำงาน แล้วนำวงจรนั้นกำหนดลงไปบน Gate Array โดยกำหนดขนาดหรืออุปกรณ์ Gate Array ที่ต้องการใช้การต่อสัญญาณออกภายนอก การวางตำแหน่งวงจรและเชื่อมโยงทางเดินของสัญญาณภายในโดยอัตโนมัติ การออกแบบสามารถเลือกอุปกรณ์เกทอาเรย์ได้หลายขนาดหลายรุ่นจากหลายบริษัทผู้ผลิต

แบบวงจรรวมที่ออกแบบจากระบบช่วยออกแบบของ Mentor Graphics สามารถนำไปผลิตได้ในโรงงานหลายแห่ง เช่น Hitachi,Motorola,NEC รายละเอียดการส่งแบบวงจรรวมไปเจือสาร

ในปัจจุบันระบบออกแบบวงจรรวมของ Mentor Graphics มีใช้งานอยู่ ณ ศูนย์ออกแบบวงจรรวมของมหาวิทยาลัยหลายแห่ง ภายใต้โครงการวิจัยการออกแบบวงจรรวมขนาดใหญ่มากของศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ กระทรวงวิทยาศาสตร์เทคโนโลยีและการพลังงาน

บทที่ 4
ภาษา VHDL

บทนี้จะแนะนำประวัติความเป็นมาอย่างย่อ ๆ ของภาษา VHDL (VHSIC Hardware Description Langauge) ว่ามีความเป็นมาอย่างไร

ภาษา VHDL

VHDL ย่อมาจากคำว่า VHSIC Hardware Description Langauge (VHSIC ย่อมาจาก Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Langauge) ซึ่งใช้อธิบายการทำงานของระบบ Digital Hardware สามารถใช้อธิบายฟังก์ชันการทำงานได้หลาย ๆ ระดับ ตั้งแต่ระดับ Block Level จนถึงระดับ Gate Level ความซับซ้อนของระบบสามารถจะเขียนได้ตั้งแต่ระดับ Gate Level ประกอบกันจนเป็นระบบที่สมบูรณ์ รูปแบบของภาษา VHDL นั้น จะประกอบไปด้วย 2 ส่วนใหญ่ ๆได้แก่ ส่วนของ Sequencial Langauge และ Concurrent Langauge การโปรแกรมด้วยภาษา VHDL สามารถจะเขียนได้ทั้ง 2 รูปแบบรวมกัน เพราะในการทำงานของระบบใด ๆย่อมจะมีการทำงานในแบบ Sequential และ Concurrent อยู่รวมกัน นอกจากนี้ตัวภาษา VHDL ยังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อย ๆ เข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้ ตัวภาษา VHDL นอกจากจะกำหนดรูปแบบไวยากรณ์(Syntax)ของตัวภาษาแล้ว ยังมีการตรวจสอบความหมายของตัวภาษาว่าจะทำการ Simulation ได้หรือไม่ เพราะว่าโปรแกรมที่เขียนโดย VHDL ต้องผ่านการ Simulation เพื่อตรวจสอบดูการทำงาน ฉะนั้นในการ Compile จะมีการตรวจสอบทั้ง Syntax และ Simulation Semantics อย่างไรก็ตามตัวภาษานั้นถึงจะมีความซับซ้อนมากมายในรูปแบบและกฎเกณฑ์ของภาษาแต่การเรียนรู้เพียงบางส่วนของตัวภาษาก็สามารถนำมาใช้งานได้โดยไม่จำเป็นจะต้องศึกษารายละเอียดทั้งหมด เนื่องจากตัวภาษา VHDL ออกแบบมาให้ใช้ออกแบบได้ตั้งแต่วงจรที่มีเล็กจนถึงวงจรที่มีขนาดใหญ่และซับซ้อน

ประวัติความเป็นมาของภาษา VHDL

ความต้องการภาษานี้เริ่มจากโครงการ VHSIC ของ Department OF Defence ของสหรัฐอเมริกาเนื่องจากมีบริษัทที่สร้าง VHSIC Chip หลายบริษัทได้ร่วมโครงการที่จะพัฒนา ในขณะนั้นหลาย ๆ บริษัทใช้ภาษา VHDL ซึ่งแตกต่างกัน ในการที่จะอธิบายการทำงาน Chip ของตน ด้วยเหตุนี้ทำให้เกิดความแตกต่าง แต่ละบริษัทไม่สามารถแลกเปลี่ยนเทคโนโลยี่ให้กันและกันได้ทำให้ DOD เกิดปัญหาในการที่จะพัฒนาและซ่อมบำรุงในภายหลัง จึงเกิดความต้องการภาษา VHDL ซึ่งเป็นมาตราฐานในการที่จะอธิบายถึงตัว Design นั้น ๆ ดังนั้น DOD จึงมอบให้บริษัท IBM , TEXAS INSTRUMENT และ INTERMETICS 3 บริษัทแรกร่วมกันพัฒนาและกำหนดมาตราฐานของ VHDL ขึ้นมาในปี 1983 หลังจากนั้น VHDL VERTION 7.2 ได้ทำการพัฒนาและออกเผยแพร่ต่อสาธารณะชนในปี 1985 ได้รับความสนใจเป็นอย่างมากในอุตสาหกรรม โดยเฉพาะอย่างยิ่งบริษัทที่ทำ VHSIC CHIP จากผลสำเร็จนี้ทำให้เกิดมาตราฐาน IEEE ของ VHDL ในปี 1986 ภายหลังจากนั้นก็มีการพัฒนาขยายขีดความสามารถของตัวภาษา VHDL เพิ่มขึ้นจากในวงการอุตสาหกรรม มหาวิทยาลัย, และ DOD ก็มีการปรับปรุงและจดมาตราฐานใหม่ IEEE ในปี 1987 อีกครั้ง ซึ่งเป็นที่รู้จักกันในชื่อของ IEEE STD 1076 - 1987 หลังจาก กันยายน 1988 บริษัทใด ๆ ที่ทำการพัฒนา ASIC CHIP ใน Department Of Defence ของอเมริกาต้องส่งตัว VHDL Model พร้อมกับ TEST BENCH ตามมาตราฐานที่ได้กำหนดเอาไว้

ความสามารถของภาษา VHDL
(CAPABILITY)

หัวข้อดังต่อไปนี้คือความสามารถหลัก ๆ ของตัวภาษา VHDL

- ตัวภาษา VHDL สามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิต CHIP กับ ผู้ออกแบบ (CAD Tools)
- ใช้เป็นการสื่อกลางในการแลกเปลี่ยนสื่อสารระหว่าง CAE และ CAD Tools เช่นตัวภาษา Source Code ของ VHDL สามารถใช้ Compile โดยใช้ Compiler & Simulator ได้หลายตัวแตกต่างกัน
- ภาษาวีเอชดีแอล สนับสนุนการออกแบบ Top Down Design และ Bottom up Design หรือผสมกันทั้ง 2 แบบ
- ตัวภาษาวีเอชดีแอลเป็น Generic คือไม่อิงเทคโนโลยี่อันใดอันหนึ่ง (แต่สามารถอิงเทคโนโลยี่ใดก็ได้และในขณะเดียวกัน ก็สามารถสนับสนุนหลาย ๆ เทคโนโลยี่)
- สนับสนุนการออกแบบทั้งระบบ Synchronuos และ Asynchronous
- สนับสนุนการออกแบบระบบ Digital ในหลาย ๆ เทคนิค เช่น Finite State Machine , Algorithmic หรือ Boolean Equation
- ตัวภาษา VHDL สามารถอ่านและทำความเข้าใจได้โดยมนุษย์ (Human Readable)
- ภาษา VHDL เป็นมาตราฐานรับรองโดย IEEE และ ANSI ทำให้ Model ที่ออกแบบโดย VHDL สามารถเคลื่อนย้าย(Portable) ไปยังระบบใด ๆ ก็ได้และสามารถนำกลับมาใช้ใหม่ได้ (Reuse)
- ภาษา VHDL สนับสนุนรูปแบบการเขียนถึง 3 รูปแบบ ได้แก่ Behavioral Stye,Structural Stye, DataFlow Stye หรือสามารถ เขียนรวมกันทั้ง 3 รูปแบบ (Mixed Style)
- สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของ Component,Function Procedure และ Package
- ไม่จำเป็นต้องศึกษา Software Simulator เพราะ Simulation Model สามารถเขียนได้โดยใช้ภาษา VHDL เช่นกัน
- สามารถเขียน Model ได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของ Model (ขึ้นอยู่กับ Software Tools)
- สามารถอธิบาย Parameter ที่เกี่ยวกับฟังก์ชั่นทางด้านเวลา เช่น Propagation Delay, Min-Max Delay,Setup, Holding Time, Spike Detection สามารถอธิบายได้ภายในตัวภาษา
- GENERICS ช่วยให้เราสามารถสร้าง Parameter ของ Design

- Model ที่สร้างด้วยภาษา VHDL นั้น ไม่เพียงแต่จะอธิบายฟังก์ชั่นการทำงานเท่านั้น แต่ยังสามารถอธิบายถึงรายละเอียดของตัว Model เช่น Total Area และ Speed ของ Model
- ภาษา VHDL เป็นมาตรฐานใช้โดยบริษัทและผู้ออกแบบหลาย ๆ แห่ง ฉะนั้นจึงเป็นการง่ายที่จะทำความเข้าใจ ถึงแม้ว่าจะมาจากแหล่งต่าง ๆ
- Model ที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะว่าตัวแปลภาษาได้ตรวจสอบไวยากรณ์ทางด้าน Simulation Semantic ไว้ด้วย
- การอธิบาย Model ด้วย Behavioral Style สามารถ Synthesis ไปเป็นระดับ Gate - Level ได้ถ้าทำตามกฎของ Synthesis Guideline
- มีความสามารถที่ให้เราออกแบบ Data ชนิดใหม่ ๆ ได้ทำให้ VHDL Model เป็นการออกแบบในระดับสูง ที่ไม่ต้องคำนึงถึงว่าจะสร้างตัว Model นั้นขึ้นมาได้อย่างไร

หลักการสร้างโมเดลโดยภาษา VHDL

(General VHDL Modelling Principles)

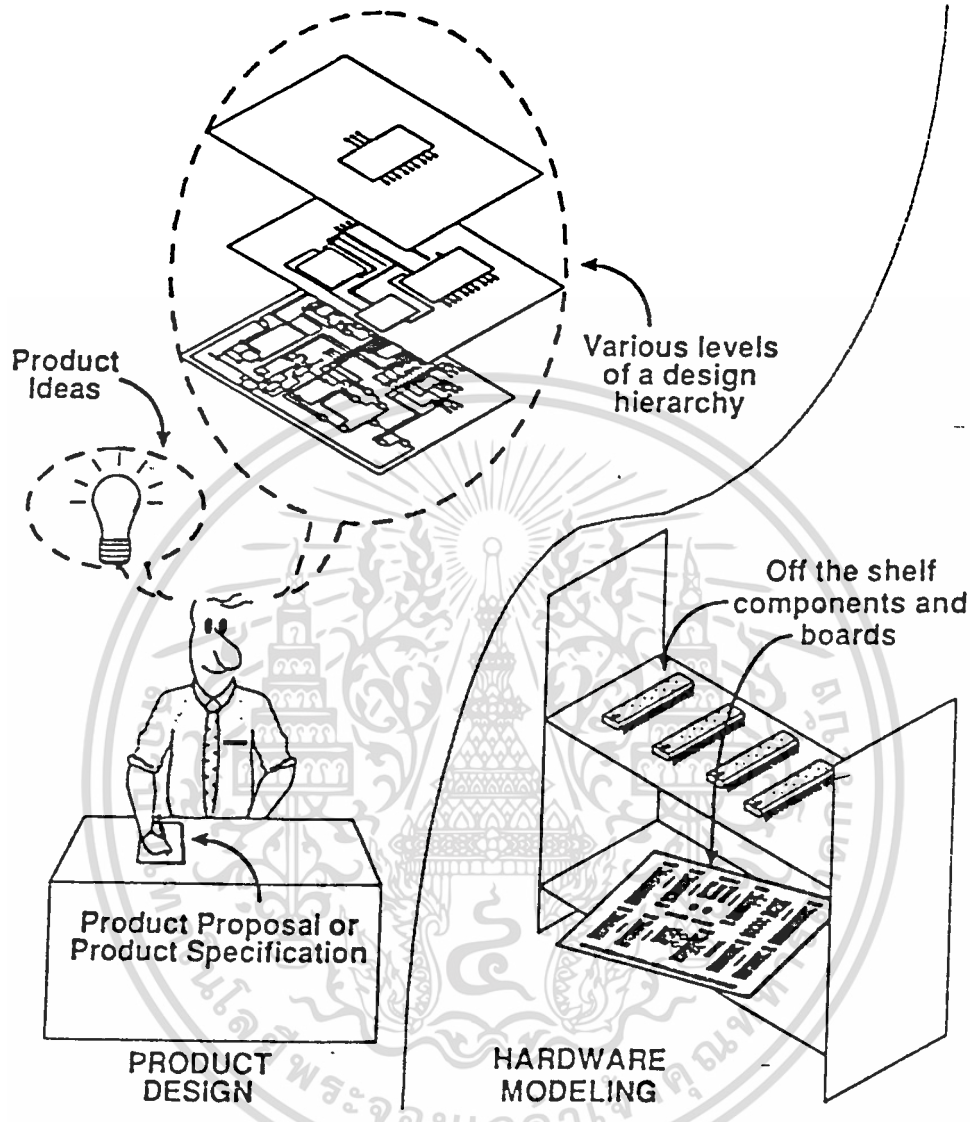


Figure 1-1. Various Things You Can Describe With VHDL

VHDL เป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้ (Human Readable)ซึ่งช่วยในการสร้างและออกแบบวงจรรวมดิจิตอล(Digital Hardware System,Circuit Board) และ Component ต่าง ๆ อาจใช้อธิบายระบบทั้งระบบหรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของ Component Block จากนั้นก็ทำการจำลองการทำงาน (Simulate)โดยที่ Design นั้นยังไม่ได้สร้างขึ้นจริงหรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังจากจำลองการทำงานจนได้ตามที่ต้องการจึงนำไปทำการ Synthesis เพื่อให้ได้วงจร Gate Level ต่อไป

ประโยชน์จริง ๆ ของการใช้ VHDL เป็น Design Tools แทนการสร้าง Prototype ขึ้นมาจริงแบบเมื่อก่อน ก็คือเราสามารถอธิบาย Product Idea, Product Proposal ,Product Speciification เป็นลักษณะในรูปของ Text จากนั้นก็นำไป Compile เพื่อดู Timing การทำงาน จากนั้นก็ทำการ Refine แก้ไขจนกว่าจะได้ Specification ตามต้องการ เมื่อ Product ได้ผลตามที่ต้องการแล้วจึงนำไปเข้าสู่การ Synthesis เพื่อให้ได้ Gate Level Schematic แล้วนำไปสร้างเป็น Prototype จริงต่อไป ซึ่ง Prototype ที่สร้างนั้นทำงานได้จริงเพราะได้ทำการ Simulate มาเรียบร้อยแล้ว เป็นการลดเวลาและค่าใช้จ่ายในการสร้าง Prototype ได้มาก

เนื่องจากว่า VHDL Language เป็นภาษาที่มีประสิทธิภาพสูง เราจึงใช้ VHDL อธิบายฮาร์ดแวร์ เพราะว่ามีข้อดี 2 ประการ คือ

1.เข้าใจได้ง่าย (Easy To Understand)

2.สามารถแก้ไขได้ง่าย (Modificable)

การเข้าใจได้ง่ายมีประโยชน์ต่อใครก็ได้ซึ่งมีความจำเป็นที่จะต้องอ่าน Code ที่ได้ออกแบบมาล้วโดยไม่จำเป็นต้องให้ผู้ออกแบบมาอธิบายให้ฟัง ตัวภาษา VHDL อธิบายการทำงานภายในตัวอยู่แล้ว ส่วนอีกประการหนึ่งก็คือความต้องการในการเปลี่ยนแปลง Hardware ที่ได้ออกแบบแล้วก็คือว่าหลังจากทดสอบแล้วพบข้อที่ผิดพลาดซึ่งต้องแก้ไขหรือว่าระหว่างพัฒนามีการเปลี่ยนแปลงความต้องการของระบบหรือต้องการเพิ่มการทำงานบางส่วนลงไป ในกรณีอื่น ๆ ที่ VHDL มีประโยชน์ก็คือตัวภาษานั้นสนับสนุนหลักการต่าง ๆ ให้เขียนแก้ไขและบำรุงรักษาวงจรดิจิตอลที่มีความซับซ้อนเป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ ซึ่งหลักการตัวที่กล่าวมีดังนี้

1. Top Down Design
2. Modularity
3. Abstraction
4. Information Hiding
5. Uniformity

ซึ่งจะอธิบายประโยชน์ของหลักการต่าง ๆ ในหัวข้อต่อไปและแสดงให้เห็นว่า VHDL นั้นช่วยในการพัฒนาวงจรดิจิตอลขนาดใหญ่และซับซ้อนนั้น ให้อ่านเข้าใจได้ง่ายและแก้ไขได้ง่ายอย่างไร

1.Top Down Design

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน เช่น ASIC (Application Specific Integrated Circuit) วิศวกรหรือผู้ออกแบบมักจะมอง Design ให้อยู่ในรูปของของ Block Diagram เสียก่อน **ก่อนที่จะย่อย** Design ให้ลึกถึงรายละเอียดต่อไป ซึ่ง VHDL นั้นอนุญาติให้

- **อธิบายการทำงาน**ของแต่ละ Block
- **วิเคราะห์การทำงาน** (Analyze)
- **จัดการแก้ไข**และปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ ก่อนที่จะทำการออกแบบให้ละเอียดลึกลงไปในขั้นตอนต่อไป การแก้ไขในขั้นตอนนี้จะทำให้ลดค่าใช้จ่ายกว่าการไปแก้ไขในช่วงของการพัฒนาในระดับสร้าง Silicon CHIP

2. Modularity

Modularity คือหลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ออกเป็นส่วนย่อยเล็ก ๆ ลงไปซึ่งปรกติการทำงานของฮาร์ดแวร์ใหญ่ต้องประกอบด้วยฮาร์ดแวร์ส่วนย่อย ๆ ลงไป ดัง Figure 1 - 2 แสดง Design ซึ่งแสดงวงจรทั้งหมดในรูป ๆ เดียว (Flatten Design) หลังจากนั้นตัดออกเป็นส่วนย่อย ๆ เล็กลงมา เมื่อเราออกแบบโดยใช้ VHDL หน้าที่การทำงานของแต่ละส่วนสามารถอธิบายได้โดยได้ Module ของ Code (คล้าย Function หรือ Procedure) ซึ่งแสดงการทำงานของส่วนย่อยนั้นอย่างชัดเจน ซึ่งการแยก Design ใหญ่ ๆ ออกเป็นส่วนย่อย ๆ นี้ทำให้ง่ายต่อการจัดการและง่ายต่อการทำความเข้าใจ

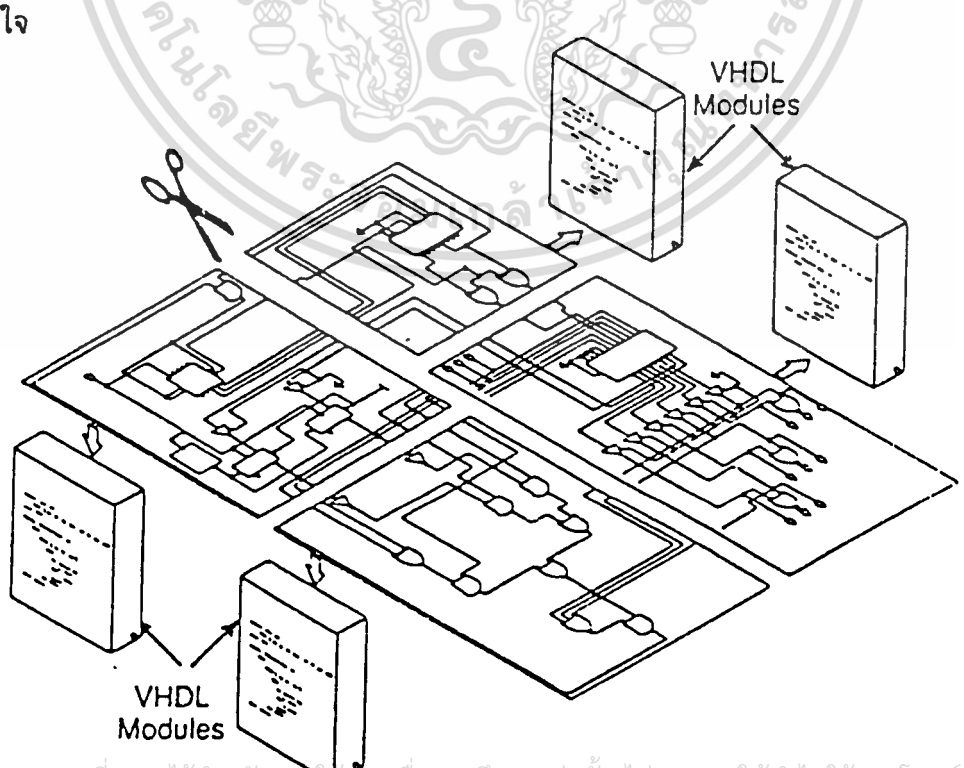


Figure 1-2. Flat-Level Partitioning of a Hardware Design

ตัวภาษา VHDL ประกอบขึ้นมาด้วย Language Building Block ซึ่งประกอบไปด้วย 75 Reserved Word และมากกว่า 200 Combination words รูปใน Figure 1-3 แสดงในเห็นว่า VHDL แต่ละ Module นั้นประกอบด้วย Language Building Block อะไรบ้าง Figure 1-3 อธิบายการทำงานของ NAND Gate

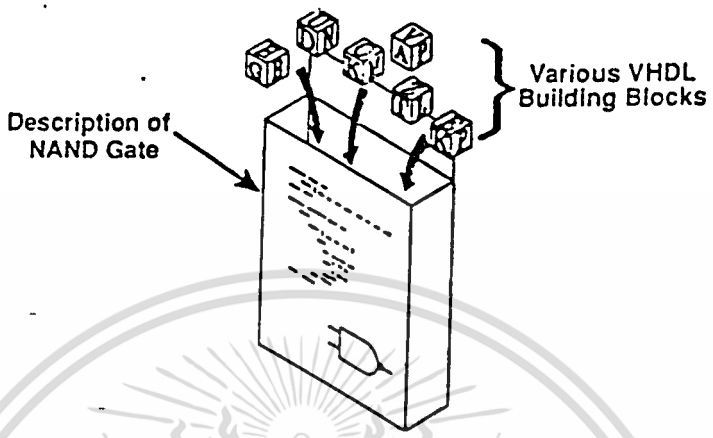


Figure 1-3. A Hardware Module Created from VHDL Building Blocks

จากรูป 1-4 แสดง Hierarchy Method โดยการแยกส่วน Design ออกเป็นส่วนย่อย ๆ ส่วนบนสุดอธิบายการทำงานของ Shifter ส่วนล่าง ๆ ลงมาคือการแยกส่วนของ Shifter ออกเป็น Flip - Flop จาก Flip - Flop แยกเป็น Nand Gate ภายใน Shifter ได้อธิบายการทำงานโดยเป็นการทำงานโดยใช้การต่อกันของ Flip - Flop ในระดับต่ำลงมา Flip - Flop ก็เกิดจากการใช้ Nand Gateต่อกัน 2 ตัว ในระดับลึกต่ำลงมาอีกก็เป็น Nand Gate ซึ่ง Nand Gate ก็มีการอธิบายการทำงานอยู่ภายใน ซึ่งแต่ละ Module จะมีคำอธิบายการทำงานในตัวของมันเองอยู่แล้ว คำอธิบายภายในแต่ละ Module มีไว้เพื่อใช้ในการเชื่อมต่อกับ Module อื่น ๆ ณ ระดับ High Level SHIPTER อธิบายการเชื่อมต่อไว้อย่างดีทำให้สามารถใช้ Flip - Flop Module ได้ส่วน Flip -Flop Module ก็อธิบายการเชื่อมต่อไว้อย่างดีทำให้สามารถเชื่อมต่อกับ Nand Gate ในระดับล่างสุดได้

ประโยชน์อย่างหนึ่งของการแยกส่วน Flip - Flop และ Nand Gate ออกจากกันเนื่องจากว่าทำให้ง่ายในการที่จะใช้ Nand Gate ตัวนี้ใน High Level Design ตัวอื่น ๆ ทำให้นำออกใช้ได้อีก (REUSEABLE) และลดการซับซ้อนในการใช้อุปกรณ์ส่ง เป็นการง่ายที่จะแก้ไขการทำงานของ SHIPTER โดยปราศจากการแก้ไข Flip - Flop และ Nand Gate จากประโยชน์ที่ได้ของ MODULARITY นี้ทำให้ Design ที่เราออกแบบนั้นง่ายต่อการเข้าใจและแก้ไขได้เสมอ

3. Abstraction

คำนิยามของ Design จะอธิบายการทำงานของตัว Design มากกว่าที่จะอธิบายถึงว่าจะพัฒนาตัว Design นั้นอย่างไร หลักการนี้จะมีความสัมพันธ์อย่างใกล้ชิดกับหลักการ Modularity ในรูป 1 - 4 Flip - Flopเป็นนิยามในการใช้ Nand Gate และ SHIPTER นิยามการใช้ Flip - Flop

รูป 1 - 5 แสดงอีกวิธีการหนึ่งซึ่งแสดงถึงการอธิบายการทำงานของ Design โดยใช้ VHDL ในหลาย ๆระดับของการนิยาม ROM (READ ONLY MEMORY) อธิบายโดยใช้ภาษาระดับสูง High Level แสดงถึง Address ต่าง ๆ ซึ่งเก็บ DATA ไว้ใน Address นั้น ๆ ที่ระดับนี้ไม่ต้องสนใจถึง Address Line, Data Line หรือ Control Line เราสามารถเพ่งจุดสนใจไปที่ขนาดของ DATA โดยไม่ต้องคิดถึงสัญญาณควบคุมต่าง ๆ มากมายภายใน เพราะว่าส่วนนั้นจะถูกจัดการเองในระดับต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ในการจัดการสัญญาณภายในทุกเส้นภายในการที่จะอ่านข้อมูลหรือ Program ข้อมูลใน ROM ถ้าต้องการเปลี่ยนค่าข้อมูลภายใน ROM เราควรขึ้นมาแก้ไขในระดับที่สูงขึ้นมา (High Level) จะทำให้ง่ายกว่าในการที่จะควบคุมสัญญาณภายใน ซึ่งเราจะเห็นว่าในแต่ละระดับมีความเหมาะสมแตกต่างกันไปและตรงจุดนี้เองทำให้ Design ที่เราออกแบบง่ายต่อการแก้ไขโดยการใช้ประโยชน์ของ Abstraction

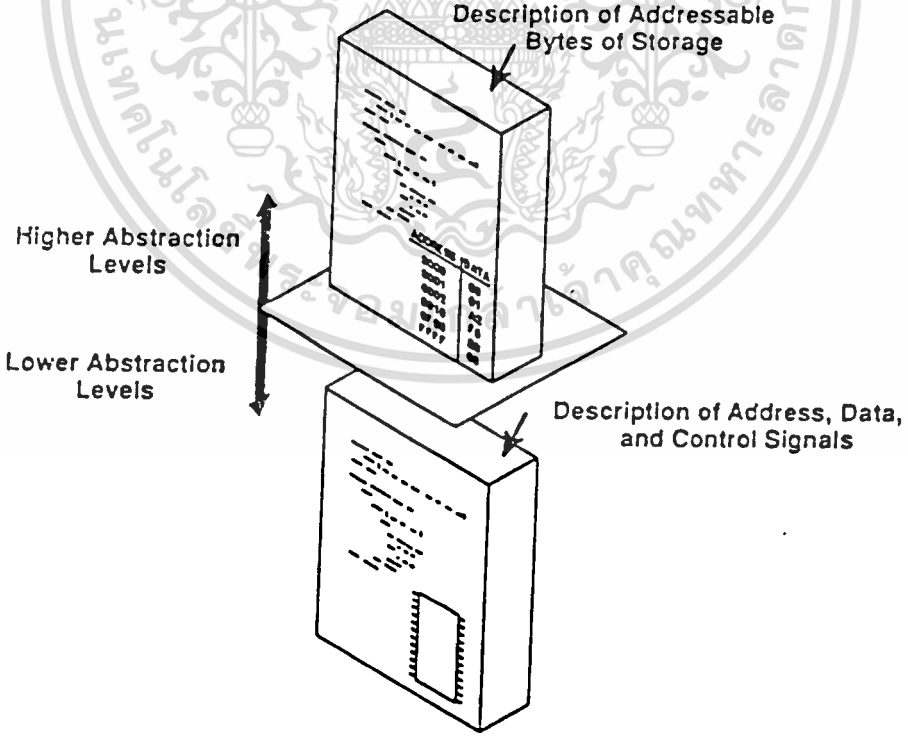


Figure 1-5. Applying Abstraction to a ROM Description

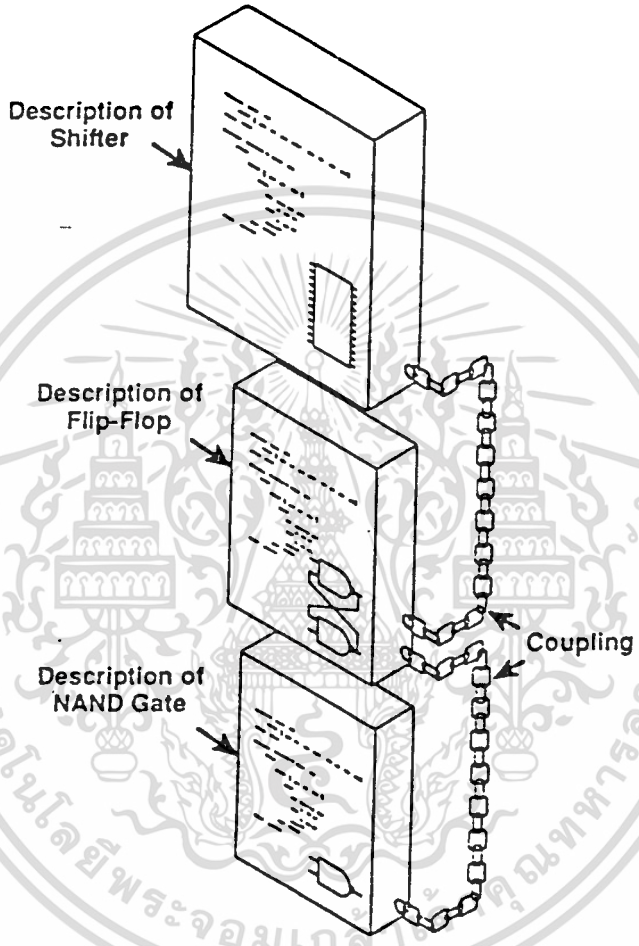


Figure 1-4. Hierarchical Partioning of a VHDL Shifter Description

4.Information Hiding

เมื่อเราทำการเขียน VHDL Code ขึ้นมาเพื่ออธิบายการทำงานของฮาร์ดแวร์ตัวหนึ่ง บางครั้งเราอาจต้องการที่จะซ่อนรายละเอียดการพัฒนา Module นั้น ๆ โดยไม่ต้องการให้ส่วน Module อื่น ๆ รู้การทำงานภายใน Information Hiding มีประโยชน์ก็คือทำให้ VHDL Design นั้นสามารถจัดการได้ง่ายและสามารถอ่านและทำความเข้าใจได้ง่ายกว่า หลักการนี้จะใช้สนับสนุนหลักการ Abstraction คือจะสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่า Design นั้นจะถูกสร้างขึ้นมาอย่างไรมีวงจรอย่างไรบ้างเป็นต้น การซ่อนรายละเอียดภายใน Module ทำให้ความสนใจของผู้ออกแบบสนใจไปในส่วนที่สำคัญมากกว่า ในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ ในรูป Figure 1-4

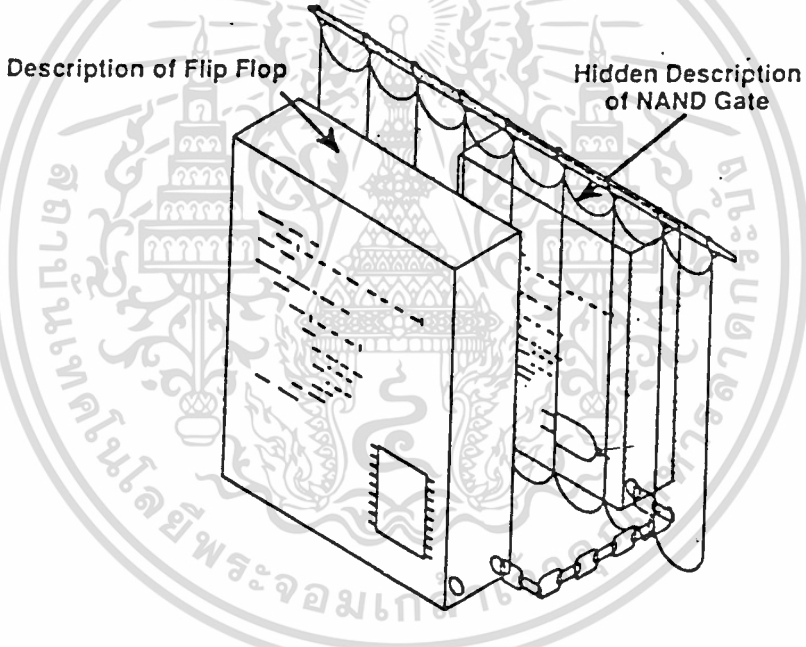


Figure 1-6. Hiding Unessential Details of NAND Gate Level

คำอธิบายของ Nand Gate นั้นจะถูกปิดบังเอาไว้จากคนที่เขียนอธิบาย Flip - Flop ดูรูป (1 - 6) คนที่เขียนอธิบายการทำงานของ Flip - Flop ไม่ต้องสนใจเลยว่า Nand Gate จะทำงานอย่างไรจะต่อกันภายในอย่างไร โดย Nand Gate สามารถเขียนขึ้นมาแล้วคอมไพล์เก็บไว้ใน LIBRARY ผู้ที่ออกแบบ Flip Flop ระดับสูงขึ้นมาเพียงแต่ต้องรู้ว่าจะเชื่อมต่อ Input / Output ของ Nand Gate มาใช้งานได้อย่างไร โดยไม่ต้องสนใจว่า Nand Gate จะถูกสร้างและพัฒนามาอย่างไร และประโยชน์อีกอย่างของ Information Hiding ก็คือป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย VHDL Model ไปยังที่อื่น ๆ เช่น ส่งไปให้อีกบริษัทใช้พัฒนาร่วมกับ VHDL อื่น ๆ โดยเป็นการแจกจ่าย อาจส่งไปแค่ VHDL ที่คอมไพล์แล้วไม่ต้องส่งตัว Source Code ไปทำให้เราป้องกันทรัพย์สินทางปัญญาได้ในอีกระดับหนึ่ง

5.Uniformility

จากหลักการต่าง ๆที่กล่าวมา ได้แก่ Modularity,Abstraction,Information Hiding Uniformility เป็นหลักการอีกอย่างที่ช่วยในการอธิบายฮาร์ดแวร์อีกด้วยภาษา VHDL นั้น อ่านได้ง่ายขึ้น Uniformility หมายถึงการสร้าง Module ของ Code ในลักษณะคล้ายกันโดยใช้ตัวภาษา VHDL BUIDING Block ทำให้เกิดการเขียน Code ที่ดีอย่างเช่น มีการใช้ย่อหน้า มีการใช้ Comment อธิบาย เป็นต้น ทำให้การพัฒนา Module อ่านและทำการเข้าใจง่าย

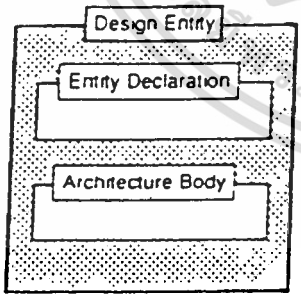
รูปแบบพื้นฐานของภาษา

Primary Language Abstraction

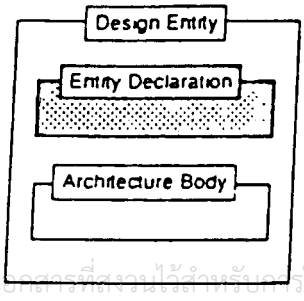
ระหว่างการออกแบบ นักออกแบบมักจะพยายามย่อยแตก วงจรออกเป็นส่วนย่อยๆ เอามาเพื่อที่จะจัดการได้ง่ายขึ้น ภาษา VHDL สนับสนุนการแยกส่วนฮาร์ดแวร์และทำให้ง่ายที่จะเขียนพฤติกรรมการทำงานของแต่ละส่วนย่อยๆ นั้น บางที Unit ย่อยๆ สามารถใช้ได้ในหลายๆ ส่วนของวงจรที่เราจะออกแบบ หรือแม้กระทั่งนำไปใช้ได้ใน Design อื่นๆ รูปแบบพื้นฐานของภาษา VHDL ในการสร้าง Hardware Model ก็คือ Design Entity ซึ่ง Design Entity สามารถแทน Cell,Chip,board หรือ Subsystem ได้

Design Entity ประกอบด้วย 2 ส่วนใหญ่คือ ส่วนของ Entity Declaration และส่วนของ Architecture Body

Entity Declaration และ Architecture Body เป็น 2 ส่วนหลักที่สำคัญของภาษา VHDL Languge Library , Library คือ ส่วนของ Hardware Description หรือ โมเดล(Model) ซึ่งสามารถจะอยู่ใน Design File ใดๆหรือถูก Compile อยู่ใน Design File หลายๆไฟล์ที่แยกจากกัน(Package Declaration,Package Body,Library Unit เช่นกัน) ความสามารถอันนี้ทำให้นักออกแบบสามารถจัดวางวงจรให้เป็นโมดูล (Module) เขียนอธิบายแต่ละโมดูล (Module) จากนั้นก็ Compile แต่ละ Entity แยกจากกัน โดยมี Architecture Body ของแต่ละ Entity ที่แตกต่างกันออกไป การประกาศ Entity Declaration เป็นการกำหนดการเชื่อมต่อ(Interface) ระหว่าง Design Entity กับวงจรส่วนอื่นๆภายนอก โครงสร้างของ Entity Declaration แสดงดังตัวอย่างต่อไปนี้

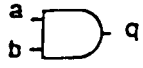



```
Entity indentifier is
        Entity_header
        --(generic and/or port clause)
        Entity_declarative_part
        --(Declarations for subprograms,
        --(types,Signal,...)
begin
        Entity_Statement_part
end identifier;
```



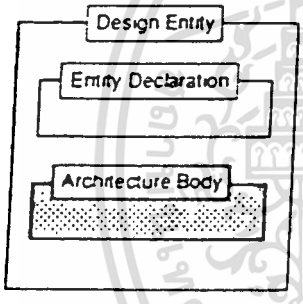
Entity Identifier คือ ชื่อของ Entity ที่เรากำหนดขึ้น แต่ละ Design Entity รับข้อมูลจากภายนอกโดย Port Mode In และส่งข้อมูลออกไปภายนอกโดย Port Mode Out ดูรูป Figure 2-2

ตัวอย่างดังต่อไปนี้แสดง Entity Declaration (รวม Port Clause ด้วย)ของ AND Gate 2 Input




```
ENTITY and2 IS
        PORT (a,b: IN bit,
              c: OUT bit);
END and2;
```

Architecture Body อธิบายความสัมพันธ์ระหว่าง Input และ Output ของ Design Entity . โครงสร้างของ Architecture แสดงดังต่อไปนี้




```
Architecture identifier of Entity_name is
        Architecture_declarative_part
begin
        Architecture_Statement_part
end identifier;
```

Identifier และ Entity name คือ words ที่คุณจะต้องเขียนไว้ใน VHDL Code สิ่งสำคัญที่จะต้องคำนึงถึงก็คือ ชื่อของ Entity name ใน Architechture Body จะต้องสัมพันธ์กัน ดังแสดงในรูป Figure 2-3

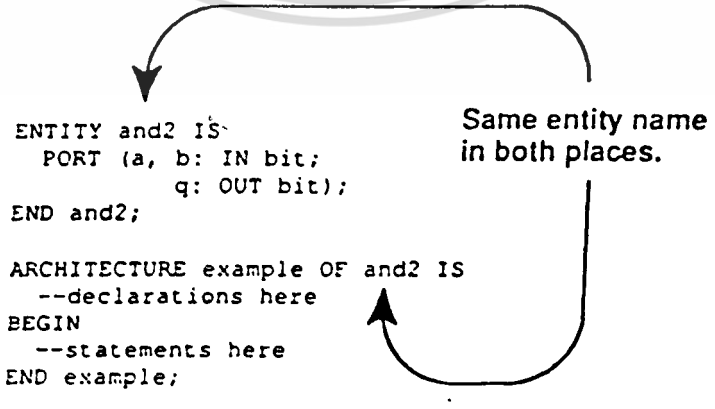


Figure 2-3. Entity Name Usage in Entity Declaration and Architecture Body

นักออกแบบกำหนดพฤติกรรมการทำงานหรือโครงสร้างของ Design Entity ใน Architecture Body โดยใช้วิธีการเขียนอธิบายโดยใช้ภาษา VHDL (Design Description Methods) Figure 2-4และ Design Entityซึ่งมี Architecture Body มากกว่า 1 Architecture โดยนักออกแบบจะต้องเขียน EntityDeclaration (ชื่อควรเป็น"Trfc_lc")แล้วก็ Compile หลังจากนั้นจึงเขียนและCompile ส่วนของ Behavioral Description ของวงจรซึ่ง Architecture name ควรเป็น "Behav" ดังแสดงที่มุมล่างด้านขวาของ Architecture Body 1 ในรูป Figure 2-4 เมื่อพอใจกับการทำงาน Behavioral ของวงจรที่ออกแบบแล้ว (ที่ระดับ High-Abstraction) เราสามารถที่จะเขียน Architecture Body ขึ้นมาอีกเพื่อเป็นการทดสอบ การทำงานของวงจร ณ.ที่ระดับ Lower-Abstration Architecture 2 (Figure2-4) แสดงโครงสร้างและรายละเอียดการไหลของข้อมูล (Data Flow) ซึ่งมี Architecture name คือ "DFlow" หลังจากนั้นเราจึงทำการ Simiulate Designที่ระดับนี้จากนั้นทำการแก้ไขปรับปรุงจนได้ผลการทำงานที่น่าพอใจ

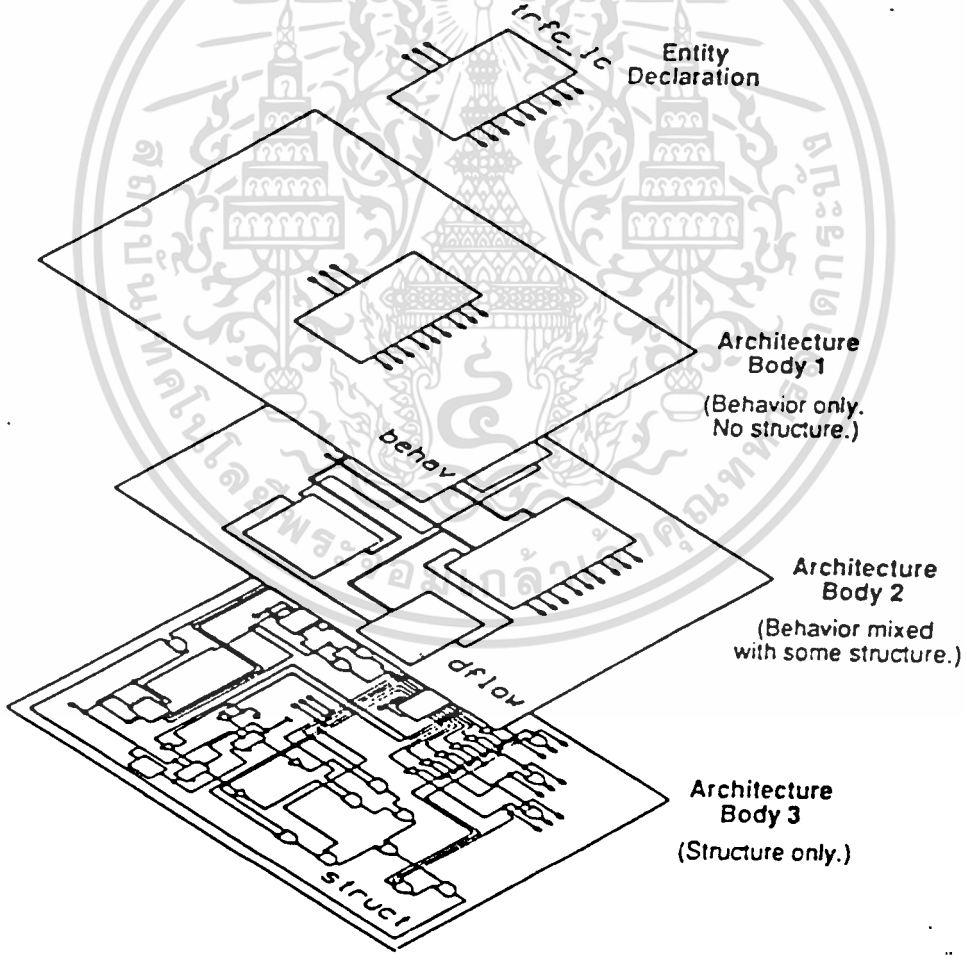


Figure 2-4. Multiple Architecture Bodies for One Entity Declaration

ระดับการเขียนอธิบายด้วยภาษา VHDL ขั้นต่ำสุด (Lowerst Abstraction Level) คือการเขียนในแบบโครงสร้าง Structural Description ซึ่งเป็นการอธิบายการทำงานของวงจร ณ.ระดับ Component Level Architecture Body 3 Figure 2-4 แสดงให้เห็นถึงการอธิบายในระดับ Structural ชื่อ Architecture นี้คือ "Struct" ในการใช้วิธีการอธิบายที่แตกต่างกัน 3 วิธีนี้ ทำให้เราสามารถพัฒนาออกแบบ Design 1 ตัว โดยใช้วิธีการ Top-Down Design ในแต่ละ Abstraction Level จะถูกเขียนและเก็บไว้ใน Design File แยกจากกัน

วิธีการเขียนอธิบายในรูปแบบต่าง ๆ

Design Description Methodes

VHDL เป็นวิธีการเขียนอธิบายการทำงานของฮาร์ดแวร์ในลักษณะของ Texual Format แทนที่จะใช้ Schematic Diagram เหมือนเมื่อก่อน หัวข้อต่าง ๆ ดังนี้คือ วิธีการเขียนอธิบาย VHDL ในหลาย ๆ วิธี เพื่อที่จะอธิบาย Hardware Architecture

- **Structural Description Method** อธิบายตัว Design ในรูปแบบของการเชื่อมต่อ Component ต่าง ๆ เข้าด้วยกัน
- **Behavioral Description Method** อธิบายฟังก์ชันการทำงานของ Hardware Design ในรูปแบบของ Circuit,Signal ที่ตอบสนองกับสัญญาณที่รับเข้ามาจากภายนอก . พฤติกรรมการทำงาน (Hardware Behavior) จะถูกอธิบายด้วยอัลกอริทึม (Algorithm) โดยไม่ได้ว่าจะสร้างขึ้นมาอย่างไร
- **Data Flow Description Method** มีความคล้ายคลึงกับ Register-Tranfer Language วิธีการนี้อธิบายฟังก์ชันการทำงานของ Design โดยการกำหนดการไหล (Flow)ของข้อมูลจากInputหรือ RegisterไปยังตัวOutputหรือ Register อีกตัววิธีการทั้ง 3แบบที่ใช้อธิบายArchitectureของ Handware สามารถอธิบายร่วมกันโดยใช้ทั้ง 3 แบบ ต่อ 1 Design ก็ได้

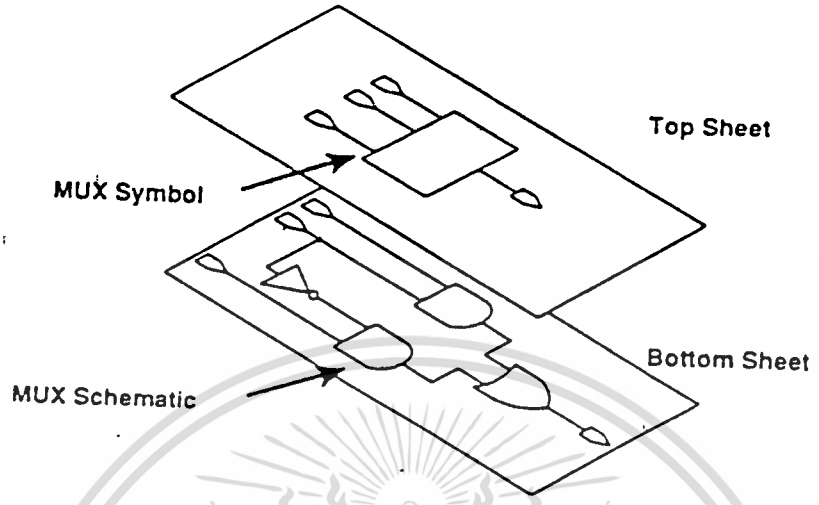


Figure 2-6. A Schematic Editor Hierarchical Design of a Multiplexer

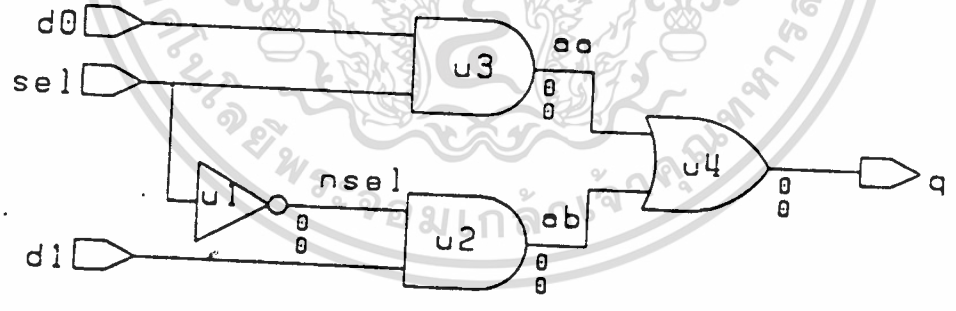


Figure 2-7. Gate-Level Representation of Two-Input Multiplexer

Figure 2-8 แสดง VHDL Structural Description ของ Multiplex 2 Input โดย VHDL Code สามารถเขียน Comment โดยการเขียน Double Dash (--) ข้อความหรืออักษรใด ๆที่อยู่หลังจากเครื่องหมายจะถูกมองว่าเป็น Comment ไม่สนใจโดย Compiler (บรรทัด 1,2,5,7,17,19 ถึง 21,24 ในรูป Figure 2-8) comment ทำให้ Code อ่านง่าย

```
1 ENTITY mux IS                          -- entity Declaration
2   PORT (d0,d1,sel : IN bit; q: OUT bit); -- port clause
3 END mux;
4
5                          -- architecture Body
6 ARCHITECTURE struct OF mux IS
7   COMPONENT  and2
8     PORT(a,b:IN bit;c :OUT bit);
9   END COMPONENT;
10  COMPONENT or2
11    PORT (a,b:IN bit;c:OUT bit);
12  END COMPONENT;
13  COMPONENT inv
14    PORT(a:IN bit;c:OUT bit);
15  END COMPONENT;
16
17  SIGNAL aa,ab,nsel:bit;       -- Signal Declaration
18
19  FOR u1      :inv  USE ENTITY WORK.invrt(behav); -- config.
20  FOR u2,u3   :and2 USE ENTITY WORK.and_gt(dFlow);-- specif.
21  FOR u4      :or2  USE ENTITY WORK.or_gt(archl); --
22
23  BEGIN
24     u1:inv  PORT MAP (sel,nsel); --architecture Statement part
25     u2:and2 PORT MAP (nsel,d1,ab);
26     u3:and2 PORT MAP (d0,sel,aa);
27     u4:or2  PORT MAP (aa,ab,q);
28 END struct;
```

Figure 2-8. Code of Structural Description For a Multiplexer

Multiplexer 2 Input แสดงใน Figure 2-8 เป็นวงจรพื้นฐาน Entity Declaration อยู่ที่ด้านบน บรรทัด 1 ถึง 3 กำหนดการ Interface Design Entity และสภาพแวดล้อมหรือวงจรอื่นๆ ภายนอก

Entity Declaration มี Port Clause ซึ่งบอก Input Channels (d0,d1 และ sel) และ Output Chanels (q) สัญญาณของแต่ Channel ถูกกำหนดให้เป็น Bit คือมีสถานะเป็น 0 กับ 1 ซึ่ง Entity Declaration นี้สามารถเปรียบเทียบกับ MUX Symbol ใน Schematic รูป Figure 2-6 ได้

Architecture Body ในรูป Figure 2-8 (บรรทัดที่ 6 ถึง 28) อธิบายความสัมพันธ์ระหว่าง Design Entity Input และ Output ในลักษณะของโครงสร้าง Architecture นี้สามารถทำงานได้เหมือนกับ Schematic ดังแสดงในรูป Figure 2-6 Component หลายๆตัว (AND2,OR2 และ INV)ที่ประกอบรวมกันจนเป็น MUX Design Entity ในรูป Figure 2-8 ถูกประกาศไว้ในส่วนของ Architecture Declaration (บรรทัด 7 ถึง 15) Signal (aa,bb และ nsel) ถูกประกาศไว้ใน Architecture Body (บรรทัดที่ 17) ด้วยเช่นกัน เพื่อที่จะแทน Output ของ AND Gate 2 ตัว (U2 และ U3) และ Inverter (U1)

Configuration Specification ในบรรทัดที่ 19 ถึง 21 เชื่อมเอา Component แต่ละตัวที่ประกาศไว้เข้ามายัง Design Entity เพื่อบอก Design Entity ว่า Component แต่ละตัวนี้ทำงานอย่างไร ยกตัวอย่าง เช่น Component U1 ในบรรทัดที่ 24รูป Figure 2-8 ถูก Bound มายัง Architecture Body ที่ชื่อ "Behav" สำหรับ Design Entity ที่เรียกว่า Invrt

Architecture Statement Part (บรรทัดที่ 24 ถึง 27) อธิบายการเชื่อมต่อ (Connection) ระหว่าง Component ที่ประกอบด้วยใน Design Unit ในส่วนนี้จะมีการประกาศการใช้ Component Figure 2-9 แสดงให้เห็นว่า Schematic Sheet ที่มี MUX Symbol มีความเกี่ยวข้องกับ VHDL Structural Description อย่างไร แทนที่จะใช้การลากเส้นใน Schematic Sheet VHDL Structure Description กำหนดการเชื่อมต่อภายในของ Component

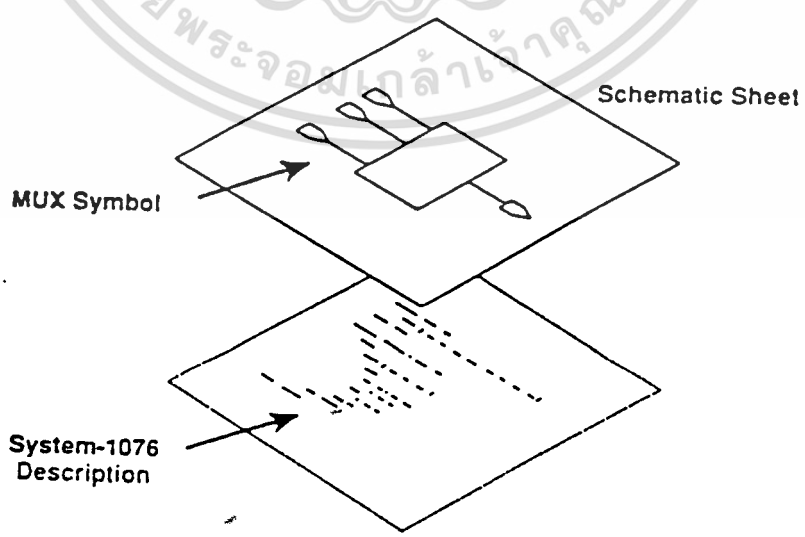


Figure 2-9. Two-Input Multiplexer with Associated Structural Description

Structural Description

ในส่วนนี้จะอธิบายในส่วนของภาษาเพียงบางส่วนในการที่จะแสดง VHDL Structural Description โดยใช้ตัวอย่างคือ Multiplexer 2 Input จะไม่ได้อธิบายโครงสร้างของภาษาในส่วนนี้ทั้งหมด เพียงต้องการยกตัวอย่างให้เห็นเท่านั้น รายละเอียดทั้งหมดให้ดูได้ที่ VHDL Reference Manual

VHDL Structural Description เป็นวิธีการอธิบายตัว Hardware Design ที่คล้ายกับแสดงโดยใช้ Schematic Diagram เพราะว่าแสดงให้เห็นถึงการเชื่อมต่อของ Component ตัวอย่างต่าง ๆที่จะแสดงให้เห็นในส่วนต่อไปนี้ เป็นการเปรียบเทียบวงจรง่าย ๆ 1 วงจร ซึ่งแทนด้วย VHDL และแทนด้วย Schematic Diagram ว่าเป็นอย่างไร

รูป Figure 2-5 แสดงสัญลักษณ์(Symbol) ของ Multiplxer 2 Input วงจร MUX นี้เป็นการออกแบบในลักษณะของ Hierarchical Design (Figure 2-6) ซึ่งวงจรในระดับล่างสุดเป็น Schematic Diagram หรืออยู่ในรูปแบบของการอธิบายถึงการเชื่อมต่อภายใน(Description) ดังรูป Figure 2-7 หมายเหตุจะสังเกตุเห็นว่าชื่อ Pin ใน MUX Symbol ในรูป Figure 2-5 จะตรงกับชื่อ Net ของ Input/Output ของ Schematic ในรูป Figure 2-7

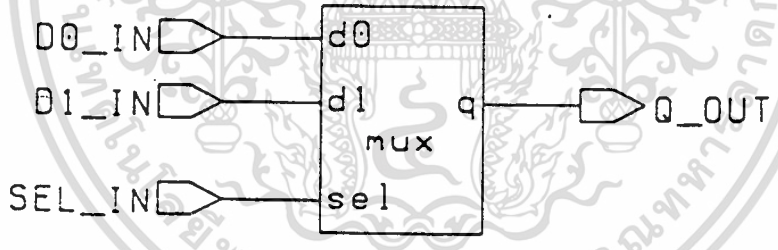


Figure 2-5. Symbol Representation of Two-Input Multiplexer

Behavioral Description

ในส่วนนี้จะเป็นการอธิบายถึงส่วนสำคัญของภาษาส่วนหนึ่งนั้นคือBehavioral Descriptionโดยใช้วงจรซึ่งได้ยกตัวอย่างมาแล้วก่อนหน้านี้ คือ MUX และ 4 Bit Shifter หลังจากอ่านส่วนที่ได้อธิบายไปก่อนหน้านี้ คือ Structural Description เราสามารถเปรียบเทียบกันได้ระหว่างการอธิบายด้วย Behaveoral กับแบบ Structural Methodว่าแตกต่างกัน

VHDL Behavioral Description แทนฟังก์ชันการทำงานของ Design ในรูปแบบของ วงจรและสัญญาณที่ตอบสนองต่อการกระตุ้นการกระตุ้นจากภายนอก ใน Design แสดงในรูป Figure 2-6 ถึง Figure 2-9 พฤติกรรมการทำงานของ MUX ถูกกำหนดด้วยการเชื่อมต่อระหว่าง Inverter AND Gate และ OR Gate ซึ่งฟังก์ชันของ Gate แต่ละตัวนี้เป็นที่เข้าใจกันดีอยู่แล้ว ใน Design ที่มีความซับซ้อนมากขึ้น Component U1ถึง U4 ในรูป 2-8 ต้องสามารถแทนด้วย Entity ที่มีฟังก์ชันการทำงานที่ซับซ้อน อย่างเช่น Central Processing Unit หรือ Bus Controller เมื่อฟังก์ชันการทำงานแต่ละ Component ด้วย Behaviral Description

VHDL Description แสดงในรูป 2-9 แสดง Behaviral Description แทน ที่จะเป็นแบบ Structure Description เหมือนหัวข้อก่อนหน้านี้ ในครั้งนี้เราสามารถวาง MUX Symbol ลงใน Schematic sheet แต่ในที่นี้เราใช้ Behavioral ของ Component ระหว่างการทำ Circuit Simulation รูป Figure 2-10 แสดง VHDL Code ซึ่ง กำหนดการทำงานของ MUX ในรูปแบบ Behavioral

Behavioral Description ในรูป Figure 2-10 และ Structure Description ในรูป Figure 2-8 ทั้งคู่มี Entity Declaration และ Architecture Body เช่นกัน ในทางปฏิบัติเราไม่จำเป็นจะต้องเขียน 2 Architecture Body ไว้ใน File เดียวกัน (ซึ่งสามารถทำได้) เราควรเขียน Entity Declaration ไว้ที่ Fileใดที่หนึ่งแล้วเขียน Behavioral Description ไว้อีก File หนึ่ง และ Structure Description ไว้อีก File เช่นกัน ในการออกแบบจริง ๆ หลังจากการ Write และ Compile Entity Declaration เสร็จเรียบร้อยแล้ว ขั้นตอนต่อไปควรเขียน Behavioral Architecture เป็นขั้นตอนต่อไปในการที่จะทดสอบการทำงานของวงจรโดยรวมทั้งหมด หลังจากนั้น เราจึงทำการเขียน Simulate และ Refine ฟังก์ชันการทำงานของ Modelจนกว่าจะทำงานได้ถูกต้อง จึงจะทำการเขียน Structural Architecture จากนั้นก็เปลี่ยน Structural Description เข้าไปแทนที่ Behaviral Description เพื่อที่จะทำการ Simulate ดูอีกครั้ง

```
1  ENTITY mux IS                    -- entity Declaration
2    PORT(d0,d1,sel : IN bit; q: OUT bit);  -- port clause
3  END max;
4
5  ARCHITECTURE behav OF mux IS
6  BEGIN
7      f1:                          -- Process Statement
8      PROCESS (d0,d1,sel)          -- sensitivity list
9      BEGIN
10         IF sel = '0' THEN        -- Process Statement part
11           q <= d1;
12         ELSE
13           q <= d0;
14         END IF;
15     END PROCESSS f1;
16 END behav;
```

Figure 2-10. Code of Behavioral Description For a Multiplexer

Behavioral Description Mode มีประโยชน์ต่อการกำเนิดสัญญาณ Input ขึ้นมาทดสอบ VHDL Model ในส่วนอื่นๆ เมื่อต้องการทำ Simulation ยกตัวอย่างเช่น เราต้องการออกแบบ Traffic Ligth Controller โดยใช้ Structural Description และเราต้องการจะทดสอบ Model นั้น โดยที่ Traffic Light Contoller นั้น Input จะต้องรับจาก Sensor ที่ต่อเข้ามา สำหรับการ Simulation เราต้องเขียน Behavioral Description Model ขึ้นมาเพื่อทำจำลองสัญญาณที่ออกจาก Sensor (แทนการสร้างจริงๆ)

ข้อแตกต่างที่เห็นได้ชัดระหว่าง Structural และ Behavioral Description ของ MUX คือ Architecture Body ดังรูป Figure 2-10 มี Process Statement ซึ่งแทนการทำงานที่เป็นอิสระกำหนดพฤติกรรมการทำงานของฮาร์ดแวร์ หรือส่วนใดส่วนหนึ่งของ Design รูปแบบของการเขียน Process Statement แสดงดังต่อไปนี้

Process Statement.....label:

```
Process (sensitivity_list)
        Process_declarative_part
begin
        Process_Statement_part
end Process label;
```

Process Statementใน Figure 2-10 Process Label f1 ตามด้วย เครื่องหมาย Colon; (Line 7) Process Label เป็น Optional แต่มีประโยชน์ในการที่จะช่วย แยกให้เห็นความแตกต่าง ๆ Process หลายตัวให้ Design ใหญ่ ๆ

ตามหลังคำสงวน (Reserver Word) Process คือ Option Sensitivity List (อยู่ภายในวงเล็บ) Sensitity List ในรูป Figure 2-10 (บรรทัดที่ 8) ประกอบไปด้วยสัญญาณ (d0,d1 sel) ระหว่างการทำ Simulation ถ้าสัญญาณใดสัญญาณหนึ่งใน Sensitivity List เกิดการเปลี่ยนแปลง State, Process จะทำการ Execute ทันที ในตัวอย่าง MUX เมื่อไรก็ตามที่สัญญาณ d0,d1 และ sel เปลี่ยน StateProcess f1 จะ Execute และ State ของ Output ก็จะเป็นไปตามเช่นกัน แต่ละ Process ใน VHDL Design จะ Execute 1 ครั้งระหว่างการทำ Initialize VHDL Hardware Model

หัวใจสำคัญของ Process Statement ในรูป Figure 2-10 คือ If Statement ที่อยู่ภายใน Process Statement Part รูปแบบพื้นฐานของการเขียน If Statement แสดงดังข้างล่าง

```
if Statement..... if condition then
                      sequence_of_Statements
                  elsif condition then
                      sequence_of_Statements
                  else
                      sequence_of_Statements
                  end if;
```

VHDL if Statement จะถูกตีความคล้าย ๆ กับประโยคในภาษาอังกฤษ ยกตัวอย่างจากประโยคข้างล่าง

if the traffec light is green then proceed across the intersection or else (if the traffic light is not green) remain stopped

ในประโยคนี้จะอยู่ในสภาวะทำงานก็ต่อเมื่อไฟ traffic light เป็นสีเขียว ก่อนที่คำสั่งต่าง ๆจะถูก execute "else" Statement จะเป็นทางเลือกอีกทางหนึ่งเมื่อสถานะอื่นที่ traffic light ไม่เป็นสีเขียวตรงตามเงื่อนไข

if-Statement Figure2-10 (บรรทัดที่ 10 ถึง 14) สามารถเขียนได้ใหม่ดังรูปต่อไปนี้

if Signal sel(select) is equal to 0 ,then assign the value of the Waveform on Signal d1 to target Signal q or else assign the value of the Waveform on Signal d0 to target Signal q.

เมื่อใดก็ตามที่สภาวะภายใน if conditionหรือ else conditionในตัวอย่าง ได้รับสภาพความตรงตามเงื่อนไข (sel='0' หรือสภาวะตรงข้ามของ sel ซึ่งไม่เท่ากับ '0') target Signal q จะถูก modified ตามความเหมาะสมขึ้นอยู่กับ Signal Assignment Statement รูปแบบพื้นฐานของ Signal Assignment Assignment มีดังนี้

Signal Assignment Statement : target (=transport Waveform;)

(Note transport Optional)

Signal Assignment Statement ประโยคแรกใน Figure 2-10 คือ

q <= d1;

ประโยคนี้จะทำการกำหนดสัญญาณ d1 ให้แก่สัญญาณq (Optional transport) ไม่ได้ใช้ในตัวอย่างนี้ Signal Assignment delimeter ประกอบด้วยตัวอักษรพิเศษ 2 ตัวคือ บางครั้งเรียกว่า compound delimeter ประโยค Signal Assignment Statement อันที่สองคือ

q <= d0;

จะทำการ assign Waveform ให้กับสัญญาณ q การใช้งานอีกอย่างของ compound delimeter คือใช้เป็น relational operator "น้อยกว่าหรือเท่ากับ" เช่นประโยคตัวอย่างดังนี้

IF Z <= '1' Then

บทสรุป Signal Assignment delimeter <= ใช้สำหรับ assign ค่าที่อยู่ทางขวามือให้กับสัญญาณที่อยู่ด้านซ้ายมือ และ delimeter ตัวเดียวกันนั้นก็ใช้ในการทำ relational operator เปรียบเทียบ "น้อยกว่าหรือเท่ากับ" ใช้ในการเปรียบเทียบ condition ใน if Statement อีกตัวอย่างในรูป Figure 2-11 แสดงการเขียน VHDL Behavioral Description ของ 4 bit Shifter เพื่อแสดงให้เห็นว่า accurate และ succinet ของ VHDL Description เป็นอย่างไรเมื่อเปรียบเทียบกับรูปแบบของ textual Description ดังนี้

the four-bit Shifter has four input Data lines,four outputData lines, and two control lines. When both control lines are low, the input levels are passed directly to the corresponding output. When control line 0 is high and control line is low,output line 0 is low;input line 0 is passed to output line1;input line 1 is passed to output line 2;and input line 2 is passed to output line 3.
When control line 0 is low and control 0; input line 2 is passed to output line 1;input line 3 is passed to output line 2;and output line 3 is low. When both control lines are high,input line 0 is passed to both output line 0 and line 1; input line 1 is passed to output line 2; and input line 2 is passed to putput line 3.

```
1  ENTITY  Shifter IS                          -- entity Declaration
2   PORT ( shftin     : IN  bit_vector(0 to 3);   -- port clause
3          shftout    : OUT bit_vector(0 to 3);
4          shftctl    : IN  bit_vector(0 to 1));
5  END Shifter;
6
7  ARCHITECTURE behav OF Shifter IS    -- architecture Body
8  BEGIN
9      f2:                              -- process statement
10     PROCESS(shftin,shftctl)
11       VARIABLE shifted : bit_vector(0 to 3) -- process Declaration part
12     BEGIN
13       CASE  shftctl IS
14            WHEN "00" => shifted := shftin;
15            WHEN "01" => shifted := shftin(1 to 3) & '0';
16            WHEN "10" => shifted := '0' & shftin(0 to 2);
17            WHEN "11" => shifted := shftin(0) & shftin(0 to 2);
18       END CASE;
19       shftout <= shifted AFTER 10 ns;
20     END PROCESS f2;
21 END behav;
```

Figure 2-11. Code Example of Behavioral Description for a Shifter

Structural and begavioral Description summary

สรุป Structural และ Behavioral Description ที่ได้กล่าวมาแล้วย่อๆ ดังนี้

VHDL structure Description เป็นการกำหนดการเชื่อมต่อ component ต่างๆ ในวงจรที่เราสร้างขึ้นส่วน Behavioral Description เป็นการอธิบายถึงอัลกอริทึม (Algorithm) ของวงจรและการทำงานของ Input/Output ภายในว่ามีการตอบสนองอย่างไรต่อสัญญาณภายนอก Design entity เป็น Unit พื้นฐานของ Hardware Description ใช้แทนการทำงานของ Cell,Chip,Board หรือ Subsystem ทั้ง Structural และ Behavioral Description มีการประกาศ Design Entity โดยใช้ Entity Declaration ส่วน Architecture Body ของแต่ละ Entity มีไว้เพื่ออธิบายความสัมพันธ์ระหว่าง Input & Output ของ Entity นั้นๆ Structural และ Behavioral Description มีความแตกต่างกันอย่างเห็นได้ชัดในส่วนของ Architecture Body ดังแสดงไว้ในรูป Figure 2-20 เปรียบเทียบตัวอย่าง MUX Architecture Body ของ Structural Description (ส่วนบนของรูป Figure 2-20) มีส่วนของ Architecture Statement part ซึ่งอธิบายการเชื่อมต่อของ Component ที่อยู่ใน Design Entity ส่วน Architecture Body ของ Behavioral Description (ส่วนล่างรูป Figure 2-20) มีส่วนของ Process Statement ซึ่งอธิบายการทำงานของ Design Entity นั้นๆ

```
1 ENTITY mux IS -- STRUCTURAL ---------------- entity declaration
2  PORT (d0,d1,sel : IN bit; q: OUT bit); -- port clause
3 END mux;
4
5                    -- architecture body
6 ARCHITECTURE struct OF mux IS
7  COMPONENT and2
8    PORT(a,b:IN bit;c :OUT bit);
9  END COMPONENT;
10 COMPONENT or2
11   PORT (a,b:IN bit;c:OUT bit);
12 END COMPONENT;
13 COMPONENT inv
14   PORT(a:IN bit;c:OUT bit);
15 END COMPONENT;
16
```

```
17 SIGNAL aa,ab,nsel:bit;      -- signal declaration
18
19 FOR u1      :inv  USE ENTITY WORK.invrt(behav); -- config.
20 FOR u2,u3   :and2 USE ENTITY WORK.and_gt(dflow);-- specif.
21 FOR u4      :or2  USE ENTITY WORK.or_gt(archl); --
22
23 BEGIN
24     u1:inv  PORT MAP (sel,nsel); --architecture statement part
25     u2:and2 PORT MAP (nsel,d1,ab);
26     u3:and2 PORT MAP (d0,sel,aa);
27     u4:or2  PORT MAP (aa,ab,q);
28 END struct;
```

```
1  ENTITY mux IS -----BEHAVIORAL--------------- entity declaration
2   PORT(d0,d1,sel : IN bit; q: OUT bit);  -- port clause
3  END max;
4
5  ARCHITECTURE behav OF mux IS
6  BEGIN
7     f1:                          -- process statement
8     PROCESS (d0,d1,sel)          -- sensitivity list
9     BEGIN
10        IF sel = '0' THEN        -- process statement part
11          q <= d1;
12        ELSE
13          q <= d0;
14        END IF;
15    END PROCESSS f1;
16 END behav;
```

Figure 2-20

Data-Flow Description

VHDL Data Flow Description และ register tranfer language มีความคล้ายคลึงกันคือทั้งคู่อธิบายการทำงานของ Design โดยกำหนดการไหล (Flow) Information จาก Input หนึ่ง หรือ register หนึ่งไปยัง Output หรืออีกregister หนึ่ง DataFlow และ Behavioral Description มีความคล้ายคลึงกันคือทั้งคู่ใช้ Process เพื่อที่จะอธิบายฟังก์ชันการทำงานของวงจร Behavioral description ใช้จำนวน Process น้อยกว่าโดยภายในแต่ละ Process ใช้ลักษณะของ sequential Signal Assignment หลายๆคำสั่งภายใน Process ในทางตรงกันข้าม DataFlow descripti ใช้จำนวนของ Concurrent Signal Assignment มาก Concurrent Statement ใช้ใน DataFlow Description ประกอบด้วย

- Block statment
- Concurrent procedure call
- Concurrent assertation Statement
- Concurrent Signal Assignment Statement

นอกจากนี้ Process Statement,genetate Statement และ component instamtiation Statement เป็น Concurrent Statement ด้วยเหมือนกัน ซึ่งโครงสร้างเหล่านี้ไม่ค่อยพบในการอธิบายแบบ Data Flow Description

Concurrent Statement กำหนดการเชื่อมต่อ Processes Blocks ซึ่งทั้งหมดรวมๆกันจะอธิบายการทำงานของ Design Concurrent Statement ทำการ executeแบบ asynchrounous ร่วมไปกับ Concurrent Statement อื่นๆ

รูป Figure 2-21 แสดงให้เห็นรูปแบบการเขียนอธิบาย MUX โดยวิธี DataFlow Description ซึ่งก่อนหน้านี้ใช้ Behavioral และ Structural Description เขียนอธิบายมาแล้วตัวอย่างนี้ง่ายเกินไปที่จะแสดงให้เห็นถึงประโยชน์จริงๆของ DataFlow Description เพราะว่ามีความคล้ายคลึงกับ Behavioral Description ในรูป Figure 2-10 มาก โดยทั้งสองตัวอย่างใช้ Process Statement (แสดงด้วย Concurrent Statement ในรูป 2-21 บรรทัด 8 ถึง 10 เพื่อกำหนด Signal behavior

```
1  ENTITY mux IS
2    PORT(d0,d1,sel : IN bit ; q: OUT bit);   -- port clause
3  END mux;
4
5
6  ARCHITECTURE data_flow OF mux IS
7  BEGIN
8       csl:
9            q <= d1 WHEN sel = '0' ELSE -- conditional signal assignment
10                d0;
11 END data_flow;
```

Figure 2-21

Data Flow Description ประกอบไปด้วย Entity Declaration (บรรทัดที่1 ถึง 3) เช่นเดียวกันกับ ตัวอย่างใน Structural และ Behavioral Description ที่อธิบายมาแล้วก่อนหน้านี้ ส่วนของ Architecture Body ประดอบไปด้วย Concurrent Signal Assignment Statement ซึ่งทำหน้าที่เทียบเท่ากับ Process Statement (ซึ่งมีความหมายเหมือนกัน) รูปแบบของการเขียน Concurrent Signal Assignment แสดงดังต่อไปนี้

```
concurrent signal    label: conditional_signal_assignment
assignment statement.....
                     label: selected_signal_assignment
```

ในรูป Figure 2-21 (บรรทัดที่ 9 และ 10) Conditional Signal Assignment ทำหน้าที่เป็น Signal Assignment (q<d1หรือ q<d0) ขึ้นอยู่กับสถานะที่กำหนดใน Conditional Waveform รูปแบบของการทำ Conditional Waveform มีดังนี้

```
conditional signal      target <= options conditional_waveform;
assignment

conditional waveforms   waveform when condition else
                        -- ;
```

waveform when condition else

waveform

Conditional Signal Assignment แทนการทำงานของ Process Statement ที่ใช้ if Statement ในการเปลี่ยนแปลงลักษณะของสัญญาณ (Optional guarded transport)ไม่ใช้แสดงให้เห็นในตัวอย่าง) เพื่อการเปรียบเทียบ Behavioral Description ของ 4 bit Shifter รูป Figure 2-11 แสดงให้เห็นอีกครั้งใน Figure 2-22 (ด้านบนของรูป) พร้อมทั้ง Data Flow Description ซึ่งอธิบาย 4 bit Shifter เช่นกัน(ด้านล่างรูป)

ข้อแตกต่างซึ่งเห็นได้ชัดระหว่างการอธิบายโดยใช้ 2 วิธีคือ 4 Process Statement ถูกแสดงเป็นนัย ๆใน DataFlow Description และ 4 Conditional Signal Assignment (บรรทัดที่ 9 ถึง 20) ในวิธี Behavioral Description มี 1 Process Statement ใช้อย่างเห็นได้ชัด(บรรทัดที่ 9 ถึง 20)

ตัวอย่างของ DataFlow Description ในรูป Figure 2-22 มีการใช้ Entity Declaration เช่นเดียวกับ Behavioral Description (บรรทัดที่ 1ถึง 5)Architecture Body ใน DataFlow Description ใช้ Concurrent Signal Assignment ซึ่งประกอบด้วย 4 condetional Signal Assignment มีหนึ่งสำหรับแต่ละ Element ของ shiftout array (Concurrent Signal Assignment Statement ไม่ได้ใช้ Optional label เหมือนกับที่ใช้ในรูป 2-21 บรรทัดที่ 8)

```
1  ENTITY  shifter IS --BEHAVIORAL---------------- entity declaration
2    PORT ( shftin    : IN  bit_vector(0 to 3);   -- port clause
3           shftout   : OUT bit_vector(0 to 3);
4           shftctl   : IN  bit_vector(0 to 1));
5  END shifter;
6
7  ARCHITECTURE behav OF shifter IS   -- architecture body
8  BEGIN
9      f2:                                  -- process statement
10     PROCESS(shftin,shftctl)
11       VARIABLE shifted : bit_vector(0 to 3) -- process declaration part
12     BEGIN
13       CASE  shftctl IS
14           WHEN "00" => shifted := shftin;
15           WHEN "01" => shifted := shftin(1 to 3) & '0';
16           WHEN "10" => shifted := '0' & shftin(0 to 2);
```

```
17            WHEN "11" => shifted := shftin(0) & shftin(0 to 2);
18        END CASE;
19        shftout <= shifted AFTER 10 ns;
20     END PROCESS f2;
21 END behav;
```

```
1  ENTITY  shifter IS --DATA FLOW---------------- entity declaration
2   PORT ( shftin      : IN  bit_vector(0 to 3);   -- port clause
3          shftout     : OUT bit_vector(0 to 3);
4          shftctl     : IN  bit_vector(0 to 1));
5  END shifter;
6
7  ARCHITECTURE data_flow OF shifter IS  -- architecture body
8  BEGIN
9     shftout(3)  <= '0'       AFTER 10 ns WHEN shftctl = "01" ELSE
10                  shftin(3)  AFTER 10 ns WHEN shftctl = "00" ELSE
11                  shftin(2)  AFTER 10 ns ; -- end cond. sig assign.1
12    shftout(2)  <= shftin(3)  AFTER 10 ns WHEN shftctl = "01" ELSE
13                  shftin(2)  AFTER 10 ns WHEN shftctl = "00" ELSE
14                  shftin(1)  AFTER 10 ns ; -- end cond. sig assign.2
15    shftout(1)  <= shftin(2)  AFTER 10 ns WHEN shftctl = "01" ELSE
16                  shftin(1)  AFTER 10 ns WHEN shftctl = "00" ELSE
17                  shftin(0)  AFTER 10 ns ; -- end cond. sig assign.
18    shftout(0)  <= shftin(1)  AFTER 10 ns WHEN shftctl = "01" ELSE
19                  '0'        AFTER 10 ns WHEN shftctl = "10" ELSE
20                  shftin(0)  AFTER 10 ns ; -- end cond. sig assign.4
21  END data_flow;
```

Figure 2-22

สรุป

- ภาษา VHDL คือตัวภาษาที่ออกแบบมาเฉพาะเพื่อใช้อธิบายการทำงานของฮาร์ดแวร์ให้อยู่ในรูปที่สามารถอ่านทำความเข้าใจได้ (Human Readable) สามารถอธิบายได้ถึงการจัดระบบและการทำงานของวงจรดิจิตอล,วงจรระดับ Board และอุปกรณ์ต่าง ๆ
- เหตุผลที่ทำให้ภาษา VHDL ใช้ในการออกแบบและจำลองการทำงานของ Product ตัวหนึ่งซึ่งยังไม่ได้สร้างจริง ๆ เพื่อดูการทำงานก่อนลงมือสร้าง หรืออาจใช้เป็นตัวแทนแนวคิดใน Product นั้น ๆ มีดังนี้

  1.ภาษาVHDL อนุญาตให้เราออกแบบ,จำลองการทำงานและทดสอบระบบโดยใช้รูปแบบของภาษาระดับสูงจนถึงระดับ Gate Level

  2. ภาษา VHDL ถ้าเราเขียนตามรูปแบบของ VHDL Synthesis Guide จะทำให้เราสามารถใช้ VHDL Code นั้นไปทำการสร้างวงจรได้โดยใช้ VHDL Synthesis Tools

  3. เพราะว่าภาษา VHDL เป็นภาษาที่กำหนดเป็นมาตรฐาน IEEE 1076 - 1987 IEEE Standard VHDL Reference Manual , วิศวกรหรือผู้ออกแบบสามารถใช้ภาษานี้ในการพัฒนาได้เหมือนกันลดปัญหาความเข้ากันไม่ได้ลงไป

- VHDL มีคุณสมบัติที่ดีที่ทำให้เราสามารถเขียนและแก้ไขวงจรดิจิตอลที่มีขนาดใหญ่และซับซ้อนได้อย่างสะดวกรวดเร็วและมีประสิทธิภาพ ดังนี้

1.Top Down Design วิธีการนี้ให้เราสามารถอธิบายการทำงานของระบบได้ในลักษณะของ Block ใหญ่ ๆ จากนั้นทำการวิเคราะห์จำลองการทำงานและแก้ไขให้ได้คุณสมบัติตามที่เราต้องการ ณ ระดับ Block ก่อนที่จะลงลึกในระดับต่ำต่อไป

2.Modularity วิธการที่แยกส่วน (หรือการประกอบส่วนย่อย ๆ ขึ้นมา) วงจรที่เราออกแบบออกเป็นส่วนย่อย ๆ เล็ก ๆ ออกมา

3. Abstraction รายละเอียดใน Module ซึ่งอธิบายการทำงานของ Module มากกว่าที่จะอธิบายถึงการพัฒนาและการสร้าง Module นั้น

4. Information Hiding การพัฒนา Module ใหม่จาก Module อื่น ๆที่สร้างขึ้นมาแล้ว

5. Uniformility สร้าง Module โดยใช้ตัวภาษา VHDL Building Blocks

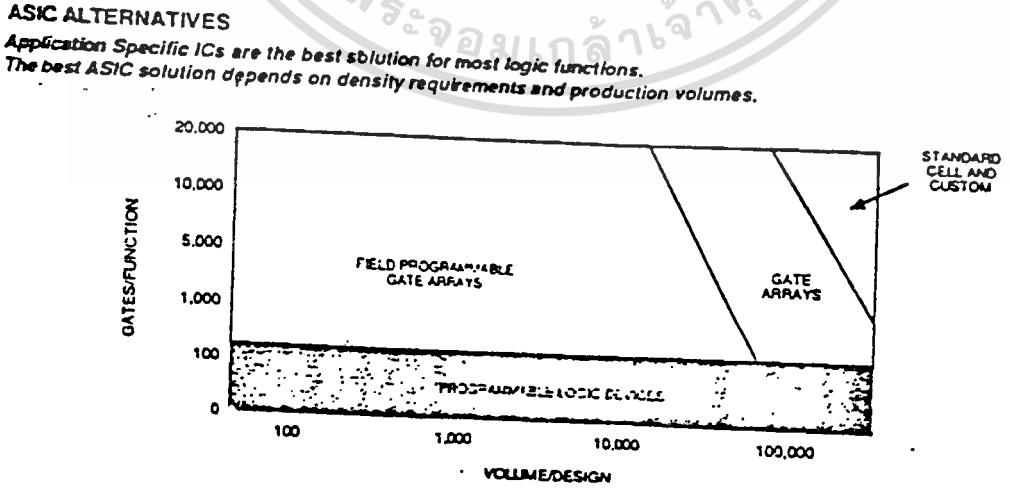
บทที่ 5

Field Programmable Gate Array(FPGA)

รู้จักกับ Programmable Gate Array

ความก้าวหน้าของอุตสาหกรรมอิเลคทรอนิกส์ในปัจจุบันทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่าง ๆมากมายซึ่งทำให้เกิดการลดราคาค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาด ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้น เห็นได้ชัดจากเทคโนโลยีไมโครเปรเซสเซอร์และหน่วยความจำ ทุก ๆครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างวงจรรวม VLSI และ IC standard มากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวน Logic Function ที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตให้ขนาดมาก ๆ และการผลิตวงจรรวม ASIC (Application Specific Integrated Circuit)

Field Programable Gate Arrays (FPGA) เป็น ASIC Chip ที่มีปริมาณความหนาแน่นของ Gate สูงสามารถจะกำหนดฟังก์ชันการทำงานได้ตามความต้องการของผู้ใช้ (User) FPGA ได้รวบรวมข้อดีทั้งหมดของการทำ CUSTOM VLSI มารวมไว้ทั้งหมดได้แก่ การออกแบบการผลิต,ลดเวลาที่จะส่งตัวผลิตภัณฑ์ออกตลาด ซึ่งเป็นประโยชน์ต่อการผลิตวงจรรวมเป็นอย่างมาก นักออกแบบเพียงแต่กำหนดฟังก์ชันการทำงานของวงจร ดังนั้นการออกแบบวงจรโดยใช้ FPGA สามารถออกแบบและทดสอบภายในเวลาเพียง 2-3 วันเท่านั้นตรงข้ามกับการออกแบบโดยใช้ Custom Gate Array ซึ่งใช้เวลาหลายอาทิตย์ การเปลี่ยนแปลงแก้ไขแบบก็เช่นเดียวกัน จากผลประโยชน์ของ FPGA ซึ่งที่ได้กล่าวมา ทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ลดความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์,ลดค่า NRE (Nonrecurring engineering Cost) ลงไปด้วย



Field Programmable Gate Array (FPGA)

คล้าย ๆกับ Gate Array ทั่วไป FPGA ไม่ต้องมีค่าใช้จ่ายคงที่และไม่มีค่า Fabrication เพราะว่า FPGA ทุกค่ามีโครงสร้างที่เหมือนกันใช้จ่ายในการผลิตมีลักษณะ คล้าย ๆกับการผลิต IC standard ที่มีปริมาณมาก ๆโดยทั่วไป

Programmable Logic Device (PLD)

มีลักษณะ PLDs โดยส่วนใหญ่จะใช้แทนที่ SSI/MSI device ได้ประมาณ 5-10 ตัวในวงจร จะมีเหมาะที่สุดเมื่อใช้ทำ ASIC ที่มีเกทจำนวน 200-300 เกท ภายในPLD จะประกอบด้วยวงจรของ AND-OR Gate เป็นพื้นฐาน ข้อจำกัดของตัว PLD ก็คือจำนวนของฟลิปฟลอป (flip-flop) ,จำนวนของ อินพุท เอาท์พุท และการเชื่อมต่อภายใน

Standard Cell & Custom IC

การผลิตแบบนี้ ต้องการหน้ากาก Mark สำหรับทุก Layer ในการผลิตทำให้ต้องมีค่าใช้จ่ายพิเศษ และมีความล่าช้าในการพัฒนา แต่ผลดีคือให้ต้นทุนต่อหน่วยต่ำที่สุดเหมาะสำหรับการผลิตให้จำนวนมาก ๆ standard Cell ให้ประโยชน์ในการ high level building blocks

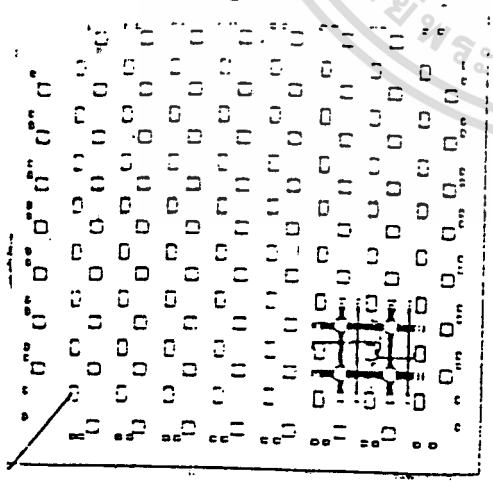
Gate Array

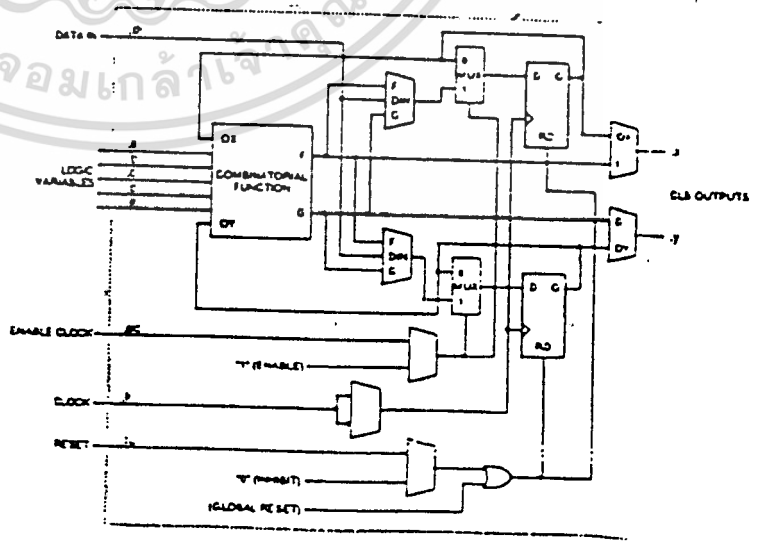
เราใช้ Gate Array สร้างวงจรลอจิกโดยการเชื่อมต่อทรานซิสเตอร์และ Gate ต่าง ๆเข้าด้วยกันจนได้ฟังก์ชั่นการทำงานที่ตามที่ต้องการซึ่งเป็นขั้นตอนสุดท้ายของกระบวนการผลิต Gate Array มีความหนาแน่นของ Gate ประมาณ 100,000 Gate หรือมากกว่านั้น ใช้ประโยชน์ประมาณ 80-90% สำหรับอุปกรณ์เล็ก ๆ และ 40-60% สำหรับอุปกรณ์ใหญ่ ลักษณะไม่เหมือน IC standard Products, Gate-Array มีค่าใช้จ่าย fixed Cost และ Product Cost การใช้งาน Gate Array จะประหยัดที่สุดเมื่อจำนวนการผลิตสูงมากจนทำให้ค่าใช้จ่ายคงที่หายไป

Programmable Gate Array Architecture

- Logic Cell Array (LCA) เป็นสิ่งประดิษฐ์ของบริษัท XILINX มีสถาปัตยกรรมภายในคล้าย ๆ Gate Array โดยทั่วไป ภายในมีลักษณะเป็นเมตริกซ์ของ Logic Block และล้อมรอบไปด้วย I/O Interface Block, การเชื่อมต่อระหว่าง CLB และ IOB ทำได้โดยผ่าน Channel ที่ว่างพาดผ่านระหว่าง Row และ Column มีการทำงาน เหมือนกับ Microprocessor ตัว LCA จะทำงานได้ต้องใช้ Program-driven Logic device หน้าที่ของ CLB,IOB แต่ละตัว,การเชื่อมต่อภายใน (interconnection) ถูกกำหนดไว้ใน Configuration program หรือเก็บไว้ใน EPROM ภายใน LCA โปรแกรมจะถูก Load เข้าสู่ LCA เมื่อมีการจ่ายไฟ (Power-up) , โดยทางคำสั่ง (command) ซึ่งเป็นส่วนหนึ่งของการทำ System initialization

ประสิทธิภาพของ LCA กำหนดโดย ความของ Logic ส่วนประกอบหน่วยความจำและการโปรแกรมการเชื่อมต่อต่าง ๆ ความเร็วของ System Clock rate ถูกกำหนดด้วย toggle flip-flop สำหรับการประยุกต์ใช้โดยทั่วไปจะอยู่ที่ประมาณ 1/3 ถึง 1/2 ของ maximum toggle Gate





Configuration Logic Block(CLB)

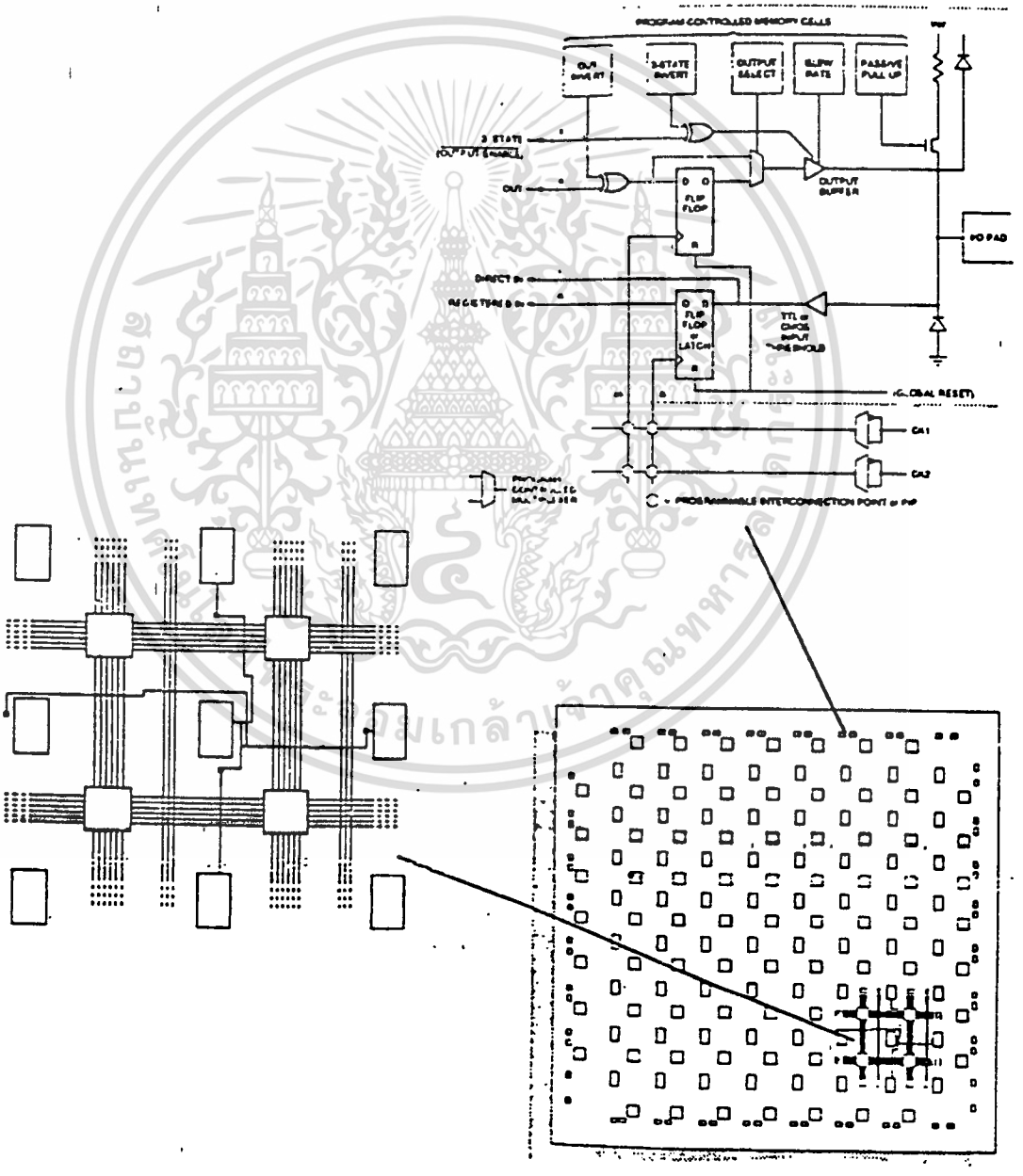
หัวใจหลักภายใน LCA คือเมตริกซ์ของ CLB หลายๆตัว CLB แต่ละตัวประกอบด้วย Programmable combination Logic และ Storage register ส่วนของวงจร Combinatorial Logic section สามารถใช้สร้างวงจรทางด้าน Boolean Function ของ Input ส่วน Register รับค่าจากส่วน Combination หรือ โดยตรงจาก CLB Output สามารถขับวงจร Combination Logic โดยตรงผ่านทาง Feedback path

Input/Output Block(IOB)

เป็นส่วนติดต่อกับวงจรภายนอกของ LCA ได้สร้างมาจาก Programmable Input/Output device (IOBs) IOB แต่ละตัวสามารถโปรแกรมได้อย่างอิสระโดยจะให้เป็น Input/Output แบบ 3 State หรือ I/O แบบสองทิศทางก็ได้โดย Input สามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณ TTL และ CMOS threshold ของ IOB แต่ละตัวมี flip-flop สามารถใช้เป็น Buffer สำหรับ Input หรือ Output

Interconnect

ความยืดหยุ่นของการใช้ LCA มาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรม Resource ต่าง ๆ ที่อยู่ภายในเข้าด้วยกันการที่จะควบคุมการเชื่อมตัวระหว่างจุดสองจุดภายใน Chip เหมือนกับ Gate Array ทั่ว ๆไป การเชื่อมต่อภายใน LCA ประกอบด้วย Network 2 ทิศทางคือทาง Vertical และ horizontal หรือ(Row และ Column)ซึ่งวางอยู่ระหว่าง CLB Programmable switch จะทำการเชื่อมต่อ Input และ Output ของ IOBs และ CLBs ที่จุดต่อร่วมระหว่าง Row กับ Column สามารถ switch Signal จาก path ไปยังส่วนอื่น ๆ เส้น long line จะลากผ่านตลอด Chip เพื่อแก้ปัญหาเรื่อง critical Signal ด้วย Minimum delay หรือ skew ที่เกิดขึ้น



XC3000

Logic Cell Array family

คุณลักษณะ (Features)

- ประสิทธิภาพสูงใช้งานได้ที่ความถี่ 70-,100- 125 Mhz ขึ้นไป
  - I/O Function
  - Digital Logic Function
  - Interconnection
- มีสถาปัตยกรรมภายในที่ยืดหยุ่น
  - มีจำนวน Gate ภายในตำนวน 2,000 ถึง 9,000 Gate
  - เพิ่มความสามารถพิเศษของ Register และ I/O
  - มีค่า fan-out สูง
  - มี 3 State bus ภายใน
  - ทำงานกับสัญญาณ TTL และ CMOS
  - มี Oscillator amplifier ในตัว
- เป็นผลิตภัณฑ์มาตรฐาน
  - ใช้พลังงานต่ำ ,เป็น CMOS ,ใช้เทคโนโลยี Static Memory
  - ประสิทธิภาพเทียบเท่ากับการใช้ TTL SSI/MIS
  - ผ่านการทดสอบจากโรงงานแล้ว 100%
  - สามารถเลือก Configuration Mode ได้
  - มี Development software ช่วยในการพัฒนา (XACT Development software)

รายละเอียด (description)

LCA ตระกูล XC3000 Logic Cell Array (LCA) family ประกอบไปด้วยวงจรลอจิกที่มีความหนาแน่นและประสิทธิภาพสูง ความยืดหยุ่น,และการโปรแกรมได้ User Array architecture เกิดขึ้นมาจาก Configuration program ที่อยู่ภายในและส่วนประกอบที่สามารถปรับเปลี่ยนเพื่อให้ได้วงจรตามต้องการได้ 3 แบบ คือ IOB, Array of CLB, เชื่อมต่อภายในตามความต้องการของผู้ใช้ Logic Function และ การเชื่อมต่อภายในถูกกำหนดด้วย โปรแกรมที่อยู่ในหน่วยความจำ ภายในตัว LCA โปรแกรมนี้ Load เข้าสู่ Memory ได้ในหลาย ๆ รูปแบบตามความเหมาะสมที่ใช้งาน โปรแกรมถูกบรรจุอยู่ใน EPROM หรือ ROM ภายนอกหรืออาจอยู่บน floppy disk , Hardisk ก็ได้ภายใน Chip มีส่วนพิเศษที่ช่วยในการ Load โปรแกรมแบบอัตโนมัติเมื่อจ่ายไฟเข้าระบบ (Power-up) LCA ตระกูล XC3000 นี้มีหลายแบบให้เลือกตามความเหมาะสมตามขนาด, อุณหภูมิ,รูปแบบของตัวถัง,อุณหภูมิใช้งาน เป็นต้น

Basic Array	Logic capacity (gates)	Configurable Logic Blocks	Max User I/Os	No. of PADs	Program Data (Bits)
XC3020	2000	64	64	74	14,779
XC3030	3000	100	80	98	22,176
XC3042	4200	144	96	118	30,784
XC3064	6400	224	120	140	46,064
XC3090	9000	320	144	166	64,160

สถาปัตยกรรมของ LCA (Logic Cell Array Architecture) ~

I/O Block ที่โปรแกรมการทำงานได้ เรียงล้อมรอบตัว CLB อยู่รอบนอกมีไว้เพื่อเชื่อมต่อกับ Package Pin ชุดของ CLB ทำหน้าที่เป็นวงจรใดก็ได้ตามต้องการแล้วแต่ผู้ใช้จะโปรแกรมการเชื่อมต่อภายในระหว่าง Resource ที่มีอยู่การส่งผ่านลอจิก ระหว่าง Block ต่าง ๆ เปรียบเสมือนสัญญาณต่าง ๆที่ส่งผ่านระหว่างอุปกรณ์ในรูปแบบ MSI/SSI package

Block Logic Function พัฒนาในรูปแบบของ Program Look-up table หน้าที่การทำงานพิเศษสร้างได้โดยการใช้ Program controlled multiplexer การเชื่อมต่อระหว่าง Block สร้างขึ้นได้โดยการใช้ Metal segment ที่เชื่อมต่อกัน ฟังก์ชั่นการทำงานของ LCA ถูกกำหนดโดย Configuration program จะถูก Load เข้าไปใน Internal Configuration Memory Cell เมื่อ Power ถูกจ่ายให้ LCA หรืออาจทำได้โดยการส่งทางคำสั่ง (command) นอกจากนี้ในตัว LCA มีสัญญาณควบคุมที่สามารถเลือกลักษณะการปรแกรมตัวเองได้ Program Data อาจ Load ได้ในลักษณะ Serial หรือ Parallel ได้

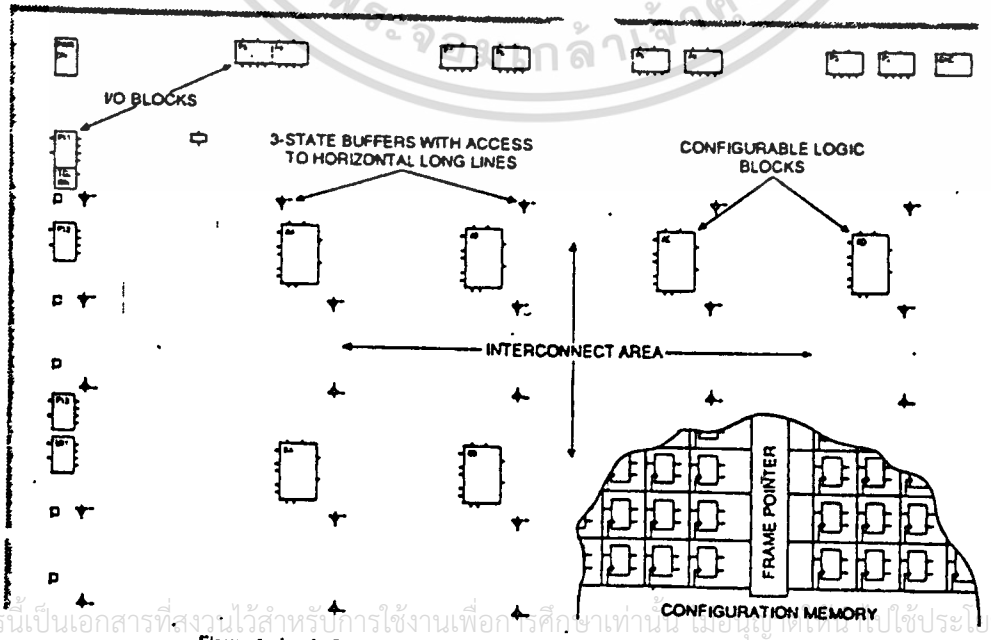


Figure 1. Logic Cell Array Structure. It consists of a perimeter of programmable

Configuration_Memory

static Memory Cell ภายใน LCA ถูกออกแบบเป็นอย่างดีมีความเชื่อถือได้สูงและป้องกันสัญญาณรบกวนได้ดี ความแน่นอนของ Memory ภายใน LCA ซึ่งได้รับการออกแบบเป็นอย่างดีนี้ ทำงานได้แม้ภายใต้สภาวะที่เลวร้าย ถ้าเทียบกับการโปรแกรมในแบบอื่น ๆ static Memory ให้ความเชื่อถือได้ดี,มีความหนาแน่นสูง,มีประสิทธิภาพสูงและสามารถทดสอบได้ จาก Figure 2 Memory Cell พื้นฐานประกอบด้วย inverter CMOS 2ตัว กับทรานซิสเตอร์ ใช้สำหรับอ่านและเขียนข้อมูลลง Memory Cell ซึ่งจะถูกเขียนเฉพาะเมื่อช่วง Configuration และถูกอ่านเฉพาะเมื่อตอน Readback เท่านั้น ระหว่างการใช้งานปกติ pass transistor จะอยู่ในสภาวะ off ตลอดเวลาทำให้ไม่มีผลต่อการทำงานของ Memory Cell ซึ่งตรงจุดนี้เป็นข้อแตกต่างที่เห็นอย่างได้ชัดจาก Memory ทั่วไป ซึ่งถูกอ่านและเขียนอยู่ตลอดเวลา Memory Cell Output Q และ Q ใช้ระดับ GND และ VCC เพื่อทำให้เกิดความแน่นอนและควบคุมได้โดยตรง comparative Load และ sense amplify ทำให้การทำงานมีความเที่ยงตรงสูงเนื่องจากโครงสร้างของ Configuration Memory Cell เป็นอย่างที่กล่าวมาทำให้ค่า LCA ไม่มีการได้รับผลกระทบใด ๆ เกิดขึ้นที่เกิดจากการเปลี่ยนแปลงของ Power-supply อย่างรุนแรง ,การแพร่กระจายของรังสี alpha ที่มีความเข้มสูง ในการทดสอบความแน่นอนในการทำงานไม่พบข้อผิดพลาดใด ๆ ที่เกิดขึ้น การถ่ายเทเอา Configuration Data ลงไปที่ LCA ทำได้ 2 ทางคือ ทาง Serial 2 วิธี และทาง Byte-wide 3 วิธี Configuration Logic ภายในจัดการแยก bit stream information ที่ได้รับจาก Development software เพื่อที่จะเก็บใน Memory ได้อย่างถูกต้องซึ่งทำให้การโปรแกรม LCA เป็นไปได้ในหลาย ๆลักษณะเช่น Synchronous, Serial หรือ Daisy chain ได้

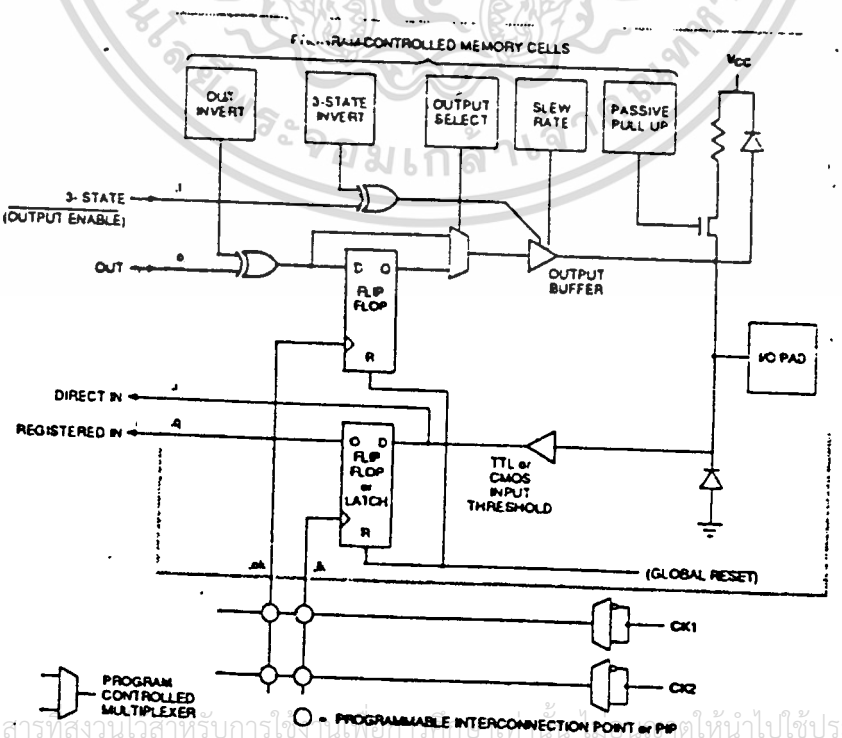


Figure 3. Input/Output Block. Each IOB includes input and output storage elements and I/O options selected by

I/O_Block

IOB แต่ละตัวดัง Figure 3 จะเป็นตัวจัดการเชื่อมต่อ Package Pin ภายนอกกับ Logic ซึ่งอยู่ภายใน IOB ประกอบด้วย register และ Direct Input path ภายใน IOB ยังมี Programmable 3 State Buffer ซึ่งถูกใช้ได้โดย register หรือ Direct Output Signal ขบวนการทำ Configuration สามารถเลือกให้ IOB ทำงาน เป็นแบบ Inversion ได้ การควบคุม slew rate หลังการเพิ่มค่าให้เป็น high impedance โดยการ pull up Resistor ได้ โดยที่วงจรภาค Input ยังมี Input clamping diode เพื่อเป็นการป้องกัน electro-static และยังมีวงจรที่ป้องกัน latch-up อันเกิดจาก Input current ส่วนของ Input Buffer ของ IOB มีการทำ threshold detection เพื่อที่จะแปลงสัญญาณจากภายนอกก่อนที่จะผ่าน Pin package เข้ามายัง Internal Logic ภายใน global Input Buffer threshold ของ IOB สามารถโปรแกรมให้รับรู้ที่ระดับสัญญาณ TTL และ CMOS I/O storage element ถูก reset ระหว่างขบวนการ Configuration หรือขณะที่ขา RESET มีสัญญาณ Active low เข้ามา สำหรับการใช้งานน่าเชื่อถือของภาค Input จะต้องมี termination time น้อยกว่า 100 ms และไม่ควรปล่อยให้ลอย (floating) การปล่อยลอยใน CMOS level อาจก่อให้เกิดการ oscillation ได้ ซึ่งอาจทำให้เกิดสัญญาณรบกวน (noise) ให้แก่ระบบ IOB ยังสามารถโปรแกรมให้ต่อกับ high impedance pull-up Resistor ซึ่งช่วยได้เมื่อ User I/O ขาใดไม่ได้ต่อใช้งาน ถึงแม้ว่าภายใน LCA จะมีวงจรซึ่งป้องกัน electrostatic discharge การจับต้อง LCA ควรทำด้วยความระมัดระวัง flip-flop loop delay ของ IOB มีค่าประมาณ 3 ns ค่า delay สั้นๆ ทำให้การใช้งานภายใน Asynchronous Clock มีประสิทธิภาพดี และเป็นการลดค่าเอาความไม่แน่นอน (metastable)ให้น้อยที่สุด

IOB Output Buffer มีลักษณะเป็น CMOS compatible 4 ma source or sink ขับสัญญาณ high out CMOS หรือ TTL compatible ขา IOB Pin [I] สามารถควบคุมการทำงานของ Output ได้ Configuration program bit สามารถกำหนดการทำงานของ IOB ให้เป็นไปได้ในหลายๆ ลักษณะ เช่น optimal Output register,logical Signal inversion และ 3 State,slew rate control ของ Output

Program controlled Memory Cell ดัง Figure 3 ควบคุมลักษณะต่างๆดังต่อไปนี้

- Logic inversion of Output ถูกควบคุมโดย 1 Configuration bit แต่ละ IOB
- Logic 3-State control ของแต่ละ IOB Output Buffer ถูกกำหนดโดย Configuration program bit ซึ่งจะเป็นตัวปิดเปิด Buffer หรือเลือกการเชื่อมต่อของ 3-State interconnection (IOB Pin_t) ถ้า IOB control Signal เป็น high, logic1 จะทำให้ Buffer disable ถ้าเป็น low จะ enable Buffer

- Direct or register Output เลือกได้แต่ละ IOB register ใช้สัญญาณขอบหน้า trig
- เพิ่มค่า Output transition speed
- high impedance pull-up Resistor ใช้สำหรับป้องกันการ floating เมื่อไม่ได้ใช้ขานั้นทำงาน

สรุปการทำงาน I/O

- Inputs
  - Direct
  - flip-flop latch
  - CMOS/TTL threshold
  - pull-up Resistor/open circuit
- Outputs
  - Direct/Resistor
  - inverted/not
  - 3 State/on/off
  - full speed/slew limit
  - 3 State/Output enable

Configuration_Lgic_Block_(CLB)

ชุดของ Configuration Logic Block (CLB) ที่ต่ออยู่เป็น Array จะเป็นองค์ประกอบหลักในการสร้าง User Logic ขึ้นมา CLB จะจัดวางอยู่ในลักษณะของ matrix ล้อมรอบด้วย IOB ตัว LCA ตระกูล XC3000 มี 64 Logic Block จัดวางอยู่ในรูป 8 Row * 8 Column ซอฟแวร์ XACT Development system มีหน้าที่แปลง Configuration แล้วทำการ Load ลง Internal Memory เพื่อกำหนดการทำงานและการเชื่อมต่อภายในของ CLB ในรูปแบบต่าง ๆ CLB แต่ละตัวมีส่วนของ Combinatorial Logic , flip-flop 2 ตัว และส่วนควบคุมภายใน (Internal control section) ดัง Figure 4 และ Figure 5 Logic Input [a,b,c,d และ e] ขา common Clock [k] 3 ขา Asynchronous Direct reset Input [rd]; และขา enable Clock [ec] ทั้งหมดสามารถถูกไปรวมให้เชื่อมต่อกับ CLB ข้างเคียงแต่ละ CLB มี 2 Output [x และ y] ซึ่งสามารถต่อเข้ากับ Interconnect Network Resource

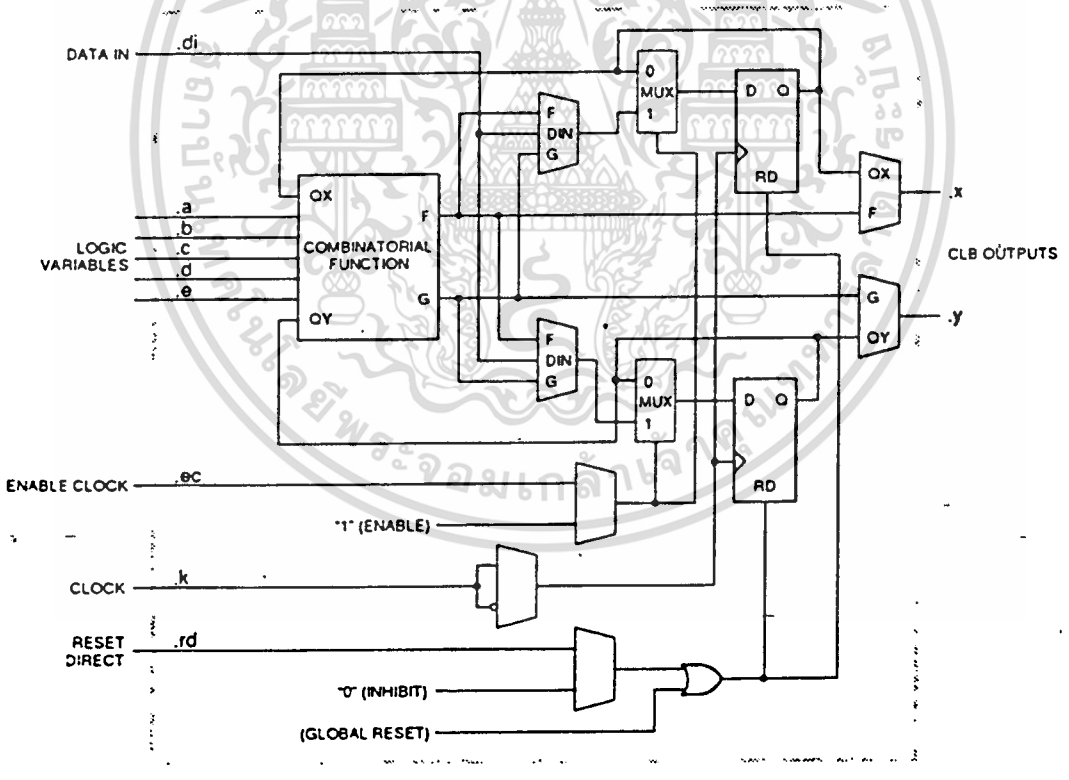


Figure 4. Configurable Logic Block. Each CLB includes a combinatorial logic section, two flip-flops and a program memory controlled multiplexer selection of function.

It has: five logic variable inputs .a, .b, .c, .d and .e.
a direct data in .di
an enable clock .ec
a clock (invertible) .k
an asynchronous reset .rd
two outputs .x and .y

Data Input ของ flip-flop ที่อยู่ภายใน CLB ได้รับจาก Function F และ g ซึ่งเป็น Output ของ Combinatorial Logic section หรือ Block Input,datain [di] flip-flop ทั้ง 2 ตัว ใน CLB แต่ละตัวมีขาใช้ขา Asynchronous ร่วมกันซึ่งเมื่อ enabled และได้รับ Logic high จะมีความสำคัญเหนือกว่า Clock Inputs flip-flop ทุกตัว ใน CLB ถูก resetโดย Active low Input ขา RESET หรือระหว่างการทำ Configuration นอกจากนี้ flip-flop ยังใช้ขา enable Clock [ec] ร่วมกันเมื่อขานี้มี Logic low จะคงสภาวะเดิมเอาไว้ไม่สนใจ Clock และ Data Input ที่รับเข้ามาจากส่วน Combinatorial Logic section ซึ่ง User สามารถจะ enable สัญญาณควบคุมได้และเลือกการเชื่อมต่อเหล่านี้ได้ นอกจากนี้ User ยังสามารถเลือก Clock Input (k) ตลอดจน sense ภายใน CLB ได้อีกด้วย การ routing ที่ยืดหยุ่นสามารถทำได้กับ CLB แต่ละตัว ส่วนของ Combinatorial Logic section ของ Logic Block ใช้ 32 ต่อ 1 look-up table ในการที่จะ implement 1 boolean Function ตัวแปรสามารถเลือกได้จาก 5 Logic Input และ flip-flop ภายใน 2 ตัว ใช้เลือก table address Input propagation delay ของ Combinatorial Logic section เป็นอิสระจาก Logic Function และมีการป้องกัน spike free สำหรับ Signal Input variable ด้วยวิธีการนี้สามารถสร้าง Logic Function 2 ส่วนมี 4 Input variable ที่เป็นอิสระต่อกันดังรูปที่ 5a หรือ Signal Function ดังรูป 5d หรือ Function พิเศษ ที่ใช้ 7 variable ดังรูป 5c บางส่วนของ partial Function ที่มี 6 ตัวแปรหรือ 7 ตัวแปร สามารถสร้างได้โดยใช้ขา Input variable (e) ในการที่จะเลือกระหว่าง 2 Function ที่มี 4 variable สำหรับ 2 Function ที่มี 4 variable

5a Combinatorial Logic option FG จะสร้างฟังก์ชัน 2 ฟังก์ชั่นที่มีตัวแปร 4 ตัวแต่ละฟังก์ชัน ตัวแปร A ต้องใช้ร่วมกันระหว่าง 2 Function ตัวแปรตัวที่ 2 หรือ 3 สามารถเป็นตัวเลือกอะไรก็
ได้ b,c,qx หรือ qy ตัวแปรตัวที่ 4 จะเป็น D หรือ E อะไรก็ได้แล้วแต่จะเลือก

5b Combinatorial Logic option จะสร้างฟังก์ชั่นอะไรก็ได้ที่มี 5 ตัวแปร a,d,e และ เลือกมาอีก 2 ตัวจาก b,c,qx,qy

5c Combinatorial Logic option E จะเลือกระหว่าง 2 ฟังก์ชันที่ใช้ 4 ตัวแปร โดยทั้งคู่มีขา Input ใช้ร่วมคือ A,D และตัวแปรอะไรก็ได้จาก b,c,qx,qy มาอีก 2 ตัว option 3 สามารถจะสร้างฟังก์ชั่นที่ต้องการตัวแปร 6 ถึง 7 ตัวได้

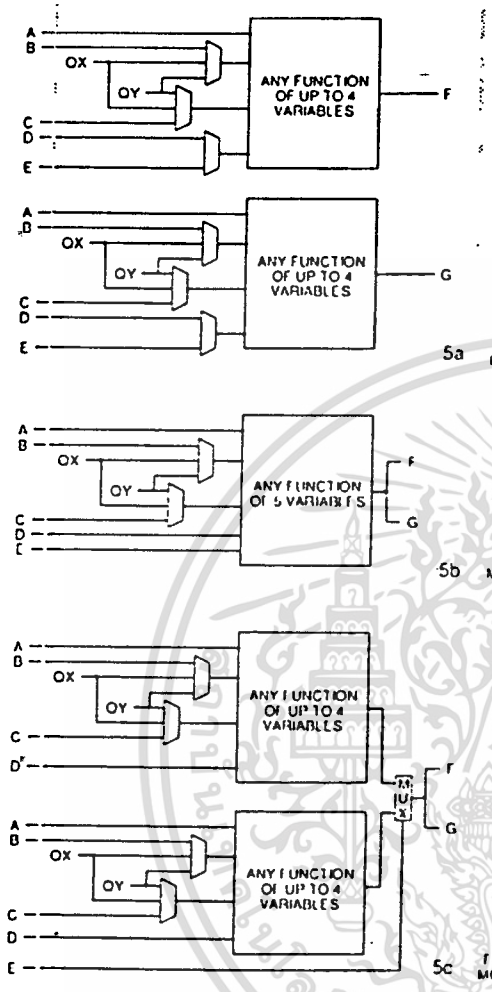


Figure 5

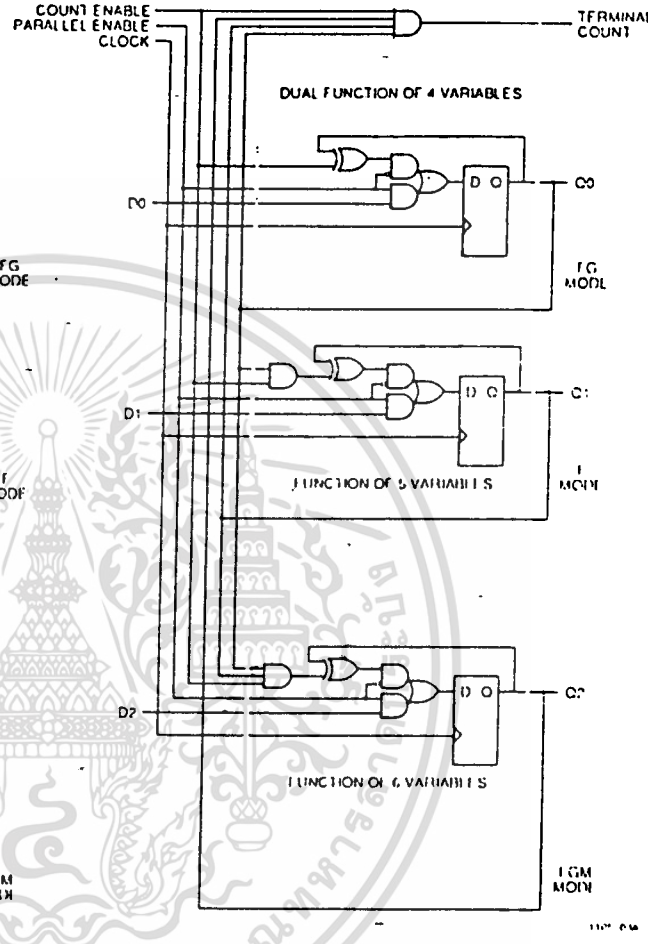


Figure 6. C8BCP Macro. TheC8BCP macro (modulo-8 binary counter with parallel enable and clock enable) uses one combinatorial logic block of each option.

การเชื่อมต่อภายในที่สามารถโปรแกรมได้ (Programmable Interconnect)

Resource ภายใน FPGA ที่ช่วยอำนวยความสะดวกในการเชื่อมต่อ Logic Cell Array ต่างๆ เข้าด้วยกันได้แก่ทางเดิน (Path) ที่ช่วยในการเชื่อมโยง Input และ Output ของ I/O Block เข้ากับ Logic Block ให้กลายเป็น Logic Network การเชื่อมต่อภายในระหว่าง Block ประกอบไปด้วย Metal Segment จำนวน 2 Layer ออกแบบพิเศษโดยใช้ Pass Transistor ซึ่งถูกควบคุมโดย Configuration bit โดยจะเปลี่ยนรูปแบบให้เป็นการเชื่อมต่อตามที่ต้องการเพื่อให้ได้ Network ตามต้องการ รูปที่ 7 แสดงการเชื่อมต่อ Routing Net ซอฟแวร์ Software Development ช่วยจัดการในเรื่อง Automatic Routing โดยอัตโนมัติและยังมี Option พิเศษในการทำ Interactive เพื่อใช้ในการทำ Design Optimization Input ของ IOB เป็นลักษณะ Multiplexed สามารถโปรแกรมเลือก Input Network จาก Segment ตัวข้างๆได้ รูปที่ 8 และการ Routing ในการที่จะ Access Logic Block Input variable, control Input และ Block Output Resource metal 3 ชนิด อำนวยความสะดวกในการที่จะเชื่อมต่อ Networkเข้าไว้ในหลายรูปแบบตามความต้องการ ดังนี้

- General Purpose Interconnect
- Direct Connection
- Long line (multiplex busses และ Wide AND Gate)

General purpose Interconnect

การเชื่อมต่อแบบ General Purpose Interconnect ดังแสดงใน Figure 9 ประกอบด้วย Grid Metal Segment ในแนวนอน 5 เส้น และพาดทางแนวขวาง 5 เส้น อยู่ระหว่าง Row และ Column ของ CLB และ IOB แต่ละ Segment เป็น high หรือ width ของ Logic Block switching matrix จะต่ออยู่ที่ปลายของแต่ละ Segment ทั้งในแนวนอนและในแนวตั้งโดยที่เราสามารถจะโปรแกรมการเชื่อมต่อภายในระหว่าง Grid segment กับ Row และ Column Switch ที่ไม่ได้ถูกโปรแกรมจะอยู่ในหภาพที่ไม่เป็นตัวนำใดๆ ทั้งสิ้น การเชื่อมต่อภายใน Switching Matrix สามารถทำได้ทั้งแบบ Automatic หรือโดย Manualโดยใช้ Edinet ในการที่จะเลือกคู่เชื่อมต่อตามต้องการ รูปที่ 10 แสดงรูปแบบการเชื่อมต่อต่างๆภายใน Switching Matrix Buffer แบบพิเศษเพื่อออกแบบขึ้นมาใช้ใน General Interconnect zone ช่วยให้เกิดการ Isolation และ Restoration เพื่อที่จะปรับปรุงคุณภาพของสัญญาณที่วิ่งผ่าน Net ที่มีขนาดยาวๆ

Interconnection Buffer ช่วยส่งให้สัญญาณแรงขึ้นก่อนที่จะเดินทางภายใน General Interconnect Segment Bi-directional Buffer นี้จะอยู่ในตำแหน่งถัดจาก Switching Matrix ด้านขวาบน PIP อันอื่นๆที่อยู่ถัดจาก Matrices สามารถถูกใช้งานเข้าสู่หรือออกจาก Long line Software Development จะเป็นตัวเลือกทิศทางของ Buffer ตามทิศทางการเชื่อมต่อภายใน Network Resource โดยอัตโนมัติ การคำนวณการหน่วงเวลาในแต่ละ Path ที่เลือกถูกคำนวณอย่างอัตโนมัติโดย Development software

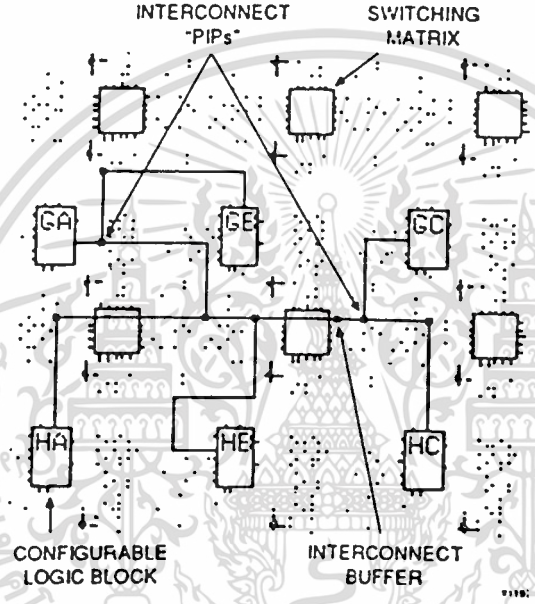


Figure 7. An XACT view of routing resources used to form a typical interconnection network from CLB GA.

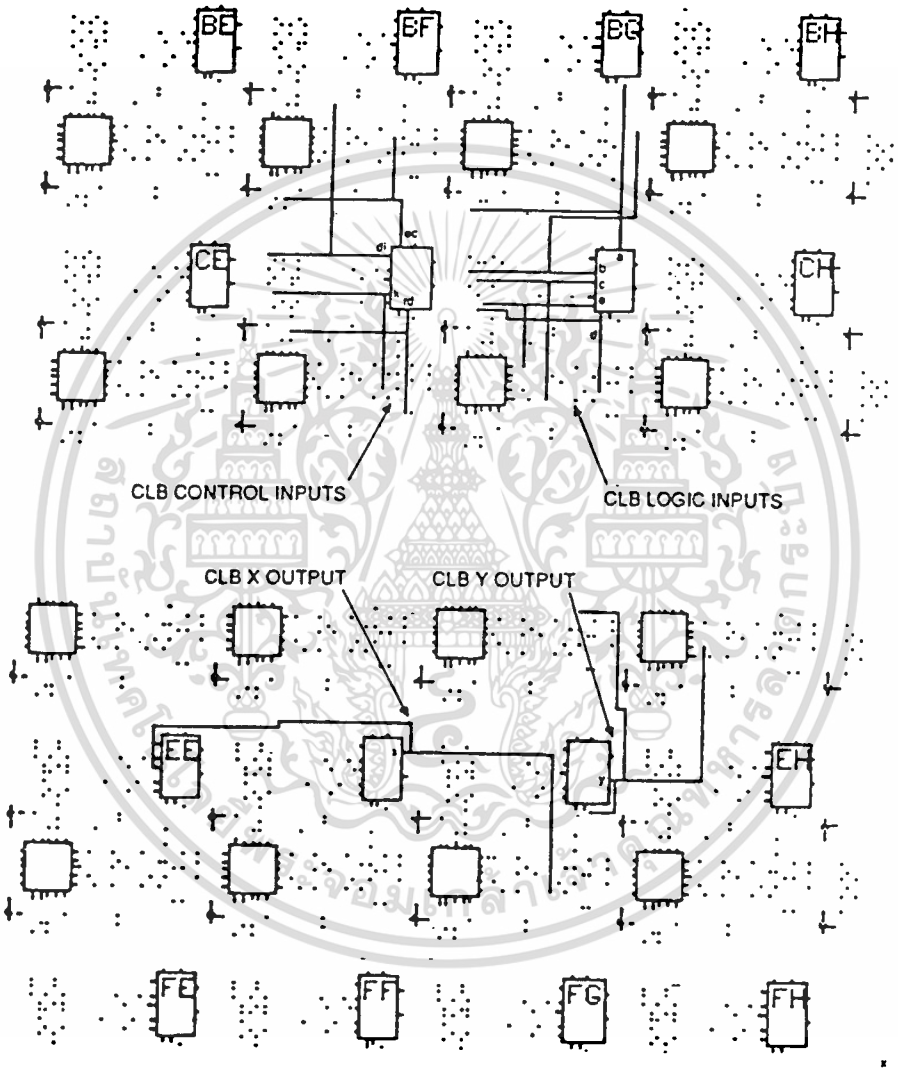


Figure 8. XACT Development System Locations of interconnect access, CLB control inputs, logic inputs and outputs. The dot pattern represents the available programmable interconnection points (PIPs).

Direct Interconnect

การเชื่อมต่อโดยตรง (Direct Interconnect) ดังแสดงใน Figure 11 แสดงให้เห็นถึงการเชื่อมต่อที่มีประสิทธิภาพที่สุดในการเชื่อมต่อ Logic และ I/O ที่อยู่ติดกันเข้าด้วยกัน สัญญาณที่เดินทางระหว่าง Block หนึ่งไปยังอีก Block หนึ่งจะมี Propagation น้อยที่สุด และไม่ได้ใช้ Resource ของ General Interconnect เลย สำหรับแต่ละ CLB ขา Output (X) จะต่อเข้าโดยตรงกับขา Input (B) ของ CLB ที่อยู่ทางด้านขวา และขา Input (C) ของ CLB ที่อยู่ด้านซ้าย ขา Output (y) อาจจะถูกต่อเข้าโดยตรง เพื่อที่จะไปขับขา Input(D) ที่อยู่ใน Block ด้านบน และขา Input (A) ขา Block ที่อยู่ด้านล่าง การเชื่อมต่อโดยตรงนี้ควรใช้ในกรณีที่ต้องการความเร็วในการทำงานสูงสุด Maximum Speed) Direct connect ระหว่าง IOB และ CLB แสดงดังใน Figure 12

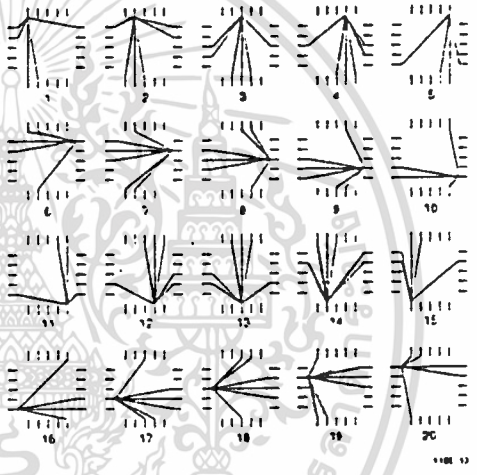


Figure 10. Switch Matrix Interconnection Options for Each Pin. Switch matrices on the edges are different. Use Show Matrix menu option in XACT

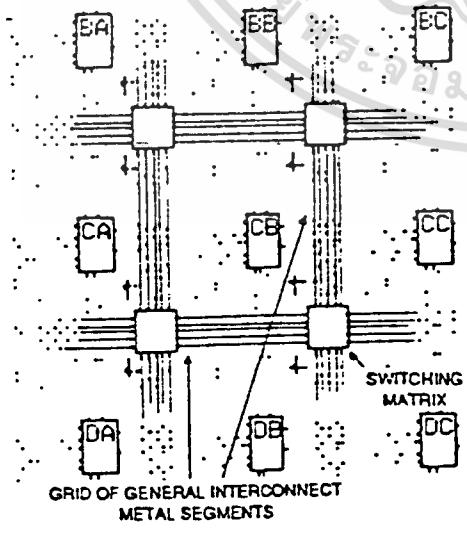


Figure 9. LCA General-Purpose Interconnect. Composed of a grid of metal segments which may be interconnected through switch matrices to form networks for CLB and IOB inputs and outputs.

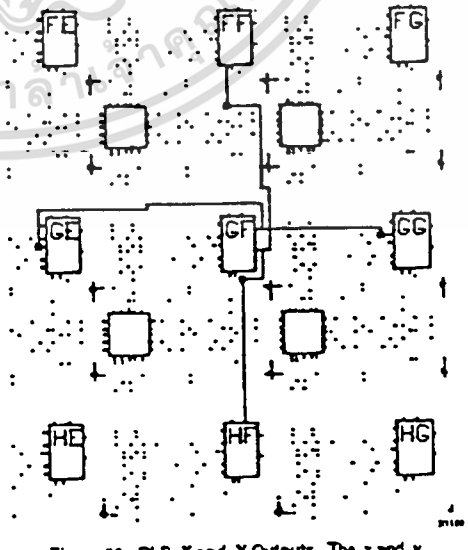


Figure 11. CLB .X and .Y Outputs. The .x and .y outputs of each CLB have single contact, direct access to inputs of adjacent CLBs.

Interconnection Buffer ช่วยส่งให้สัญญาณแรงขึ้นก่อนที่จะเดินทางภายใน General Interconnect Segment Bi-directional Buffer นี้จะอยู่ในตำแหน่งถัดจาก Switching Matrix ด้านขวาบน PIP อันอื่นๆที่อยู่ถัดจาก Matrices สามารถถูกใช้งานเข้าสู่หรือออกจาก Long line Software Development จะเป็นตัวเลือกทิศทางของ Buffer ตามทิศทางการเชื่อมต่อภายใน Network Resource โดยอัตโนมัติ การคำนวณการหน่วงเวลาในแต่ละ Path ที่เลือกถูกคำนวณอย่างอัตโนมัติโดย Development software

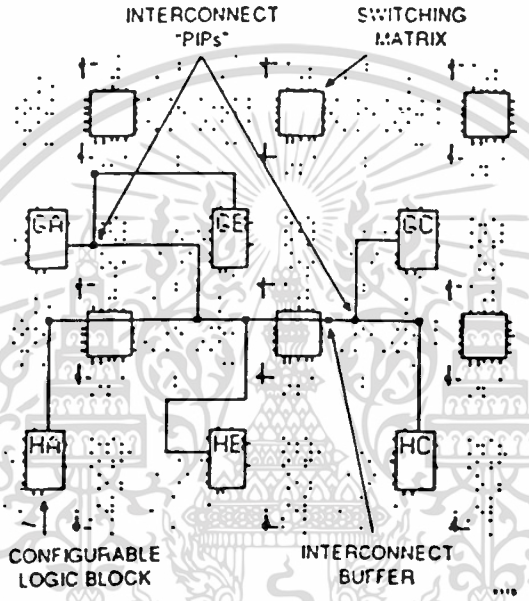


Figure 7. An XACT view of routing resources used to form a typical interconnection network from CLB GA.

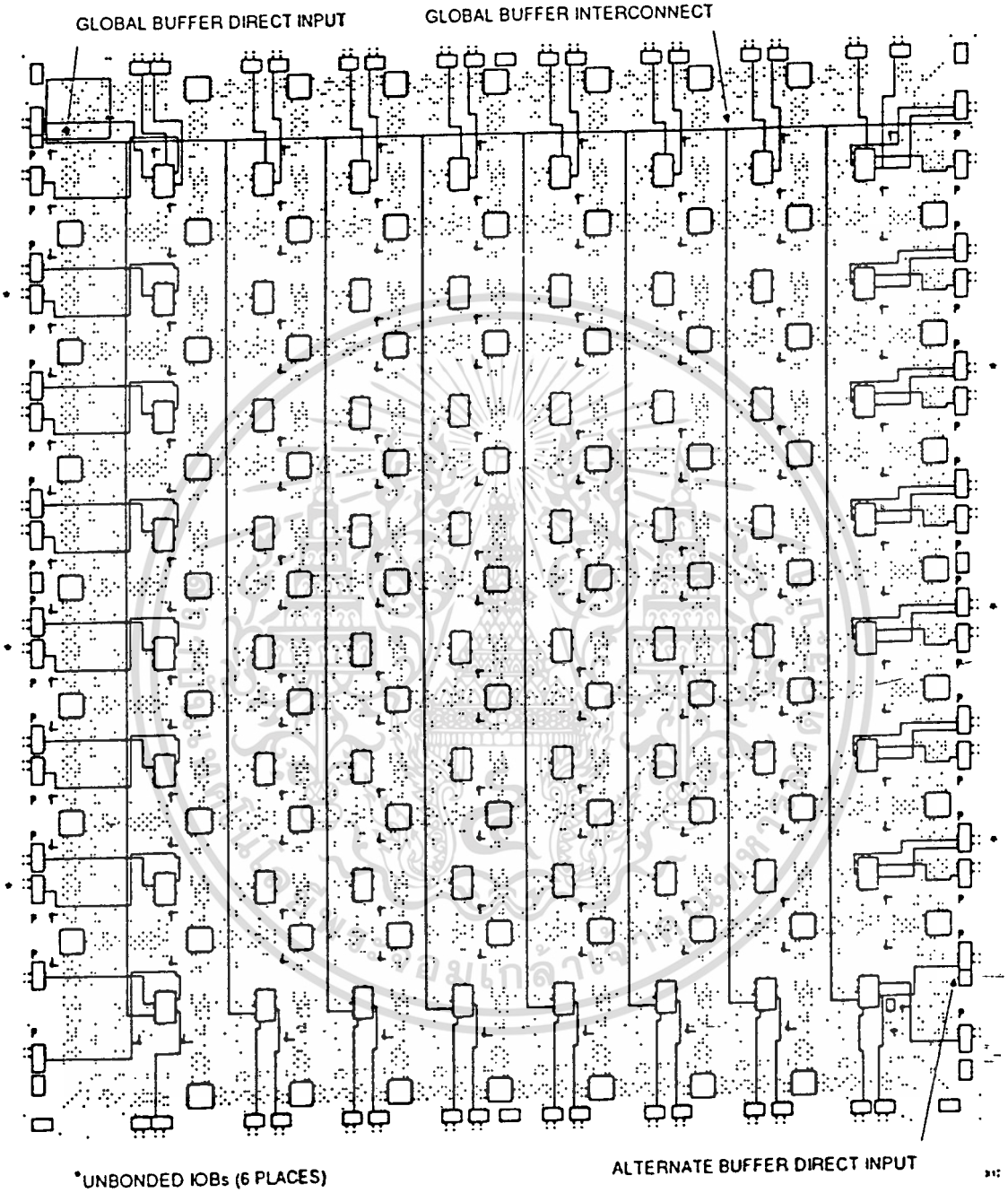


Figure 12. X3020 Die-Edge IOBs. The X3020 die-edge IOBs are provided with direct access to adjacent CLBs.

Internal_Busses

3 State Buffer 1คู่ วางอยู่ในตำแหน่งที่ถัดจาก CLB ยอมให้ Logic ออกไปที่ Horizontal Long line การทำงาน Logic ที่ 3 State Buffer ทำให้สามารถทำงานแบบ Multiplex Function ได้ 3 State Buffer สามารถควบคุมให้สัญญาณเข้าไปยัง Horizontal Long Line ได้โดยส่งสัญญาณ Logic low เข้าไปยังขาควบคุม ดัง Figure 15a การใช้งานต้องหลีกเลี่ยงการ Drive สัญญาณหลายสัญญาณที่มี Logic ตรงข้ามกัน พยายามใช้สัญญาณควบคุม 3 State Input ด้วยสัญญาณ Logic เดียวกันกับที่ Drive ที่ขา Input ถ้าปรากฏ Logic high ที่ขาทั้งสองของ Buffer Input จะทำให้เกิดสภาวะ high Impedance ซึ่งแสดงให้เห็นว่าไม่มีการขัดแย้ง (Conflict) ระหว่าง Logic ทั้งสอง Logic low จะทำให้ Buffer สามารถขับ long line ดูจากรูป 15b pull up register มีไว้ที่ปลายแต่ละด้านของ Long line เพื่อกำหนดให้ Output มีค่า High เพื่อ Buffer ทั้งหมดไม่ได้ต่อใช้งาน เมื่อ Data เข้ามายัง Input สัญญาณควบคุม 3 State ต่างหากที่ Control line ในรูปนี้จะอยู่ในรูป Multiplexed (3-State busses) ในกรณีนี้จะต้องระมัดระวังเป็นพิเศษเพื่อป้องกันการขัดแย้งระหว่าง control line หลายๆ เส้นเรื่อง conflict level รูปที่16 แสดง 3 State Buffer long line และ Pull-up resister

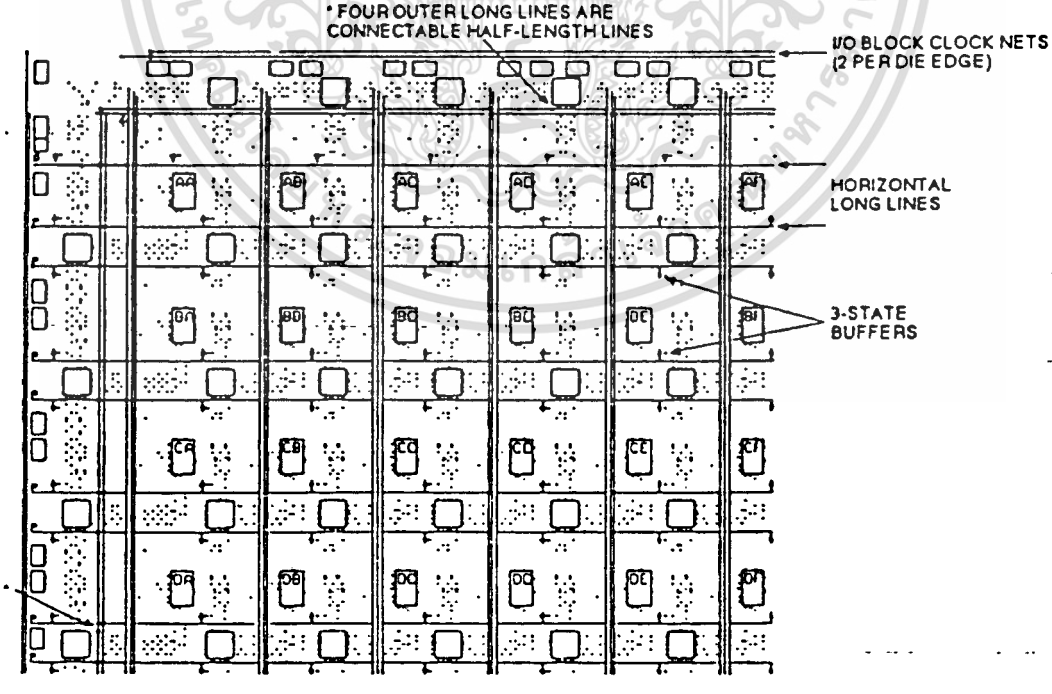


Figure 14. Programmable Interconnection of Long Lines. This is provided at the edges of the routing area. Three-state buffers allow the use of horizontal long lines to form on-chip wired-AND and multiplexed buses. The left two non-clock vertical long lines per column (except XC3020) and the outer perimeter long lines may be programmed as connectible half-length lines.

Crystal_Oscillator

รูปที่ 16 แสดงถึงตำแหน่งของ Internal high speed inverting amplifier ซึ่งสามารถนำมาใช้เป็น Crystal Oscillator ภายใน Chip ได้ ซึ่งมีใช้งานร่วมกับ Auxiliary Buffer ซึ่งอยู่มุมล่างของ Chip เมื่อ Oscillator ถูกโปรแกรม จะต่อเข้าเป็นแหล่งกำเนิดสัญญาณ (Signal Source) IOB พิเศษ 2 ตัว จะถูกโปรแกรมให้เป็น Oscillator amplifier ดัง Figure17 ร่วมกับอุปกรณ์ Oscillator จากภายนอกวงจร Oscillator จะต้อง Active ก่อนที่ขบวนการทำ Configuration จะสมบูรณ์เพื่อที่จะทำให้ Oscillator นั้นอยู่ในสภาวะคงที่ การเชื่อมต่อภายในจะต่อให้จริงๆ ก็ต่อเมื่อขบวนการ Configuration เสร็จสิ้น ใน Figure 17 Feedback Resistor R1 ระหว่าง Output และ Input Bias amplifier ที่ค่า Threshold ค่า R ตัวนั้นควรจะมีค่าใหญ่พอที่จะลด Load ไว้จะเกิดแก่ Crystal ให้น้อยที่สุด วงจร Inversion ของ amplifier ร่วมกับ R-C Network และวงจร AT-Cut Serial Resonant กำเนิดสัญญาณ 360 องศา shift Resistor R2 ต่ออนุกรมกันเพื่อเพิ่ม Output Impedance ให้กับ amplifier เมื่อตรงการควบคุม Phase shift , matching Crystal resistance หรือจำกัด Amplitude ให้ swing อยู่ในขอบเขตที่กำหนด Voltage ที่ Feedback กลับมาเกินสามารถแก้ไขให้มีค่าถูกต้องได้โดยใช้อัตราส่วน C2/C1 Amplifier ถูกออกแบบมาเพื่อให้ใช้กับความถี่ตั้งแต่ 1 MHz จนถึง 1/2 ของความถี่ CLB Toggle frequency เมื่อไม่มีการใช้งาน Oscillator invertor IOBs ที่ใช้สามารถถูกใช้งาน User I/O ได้ตามปกติ

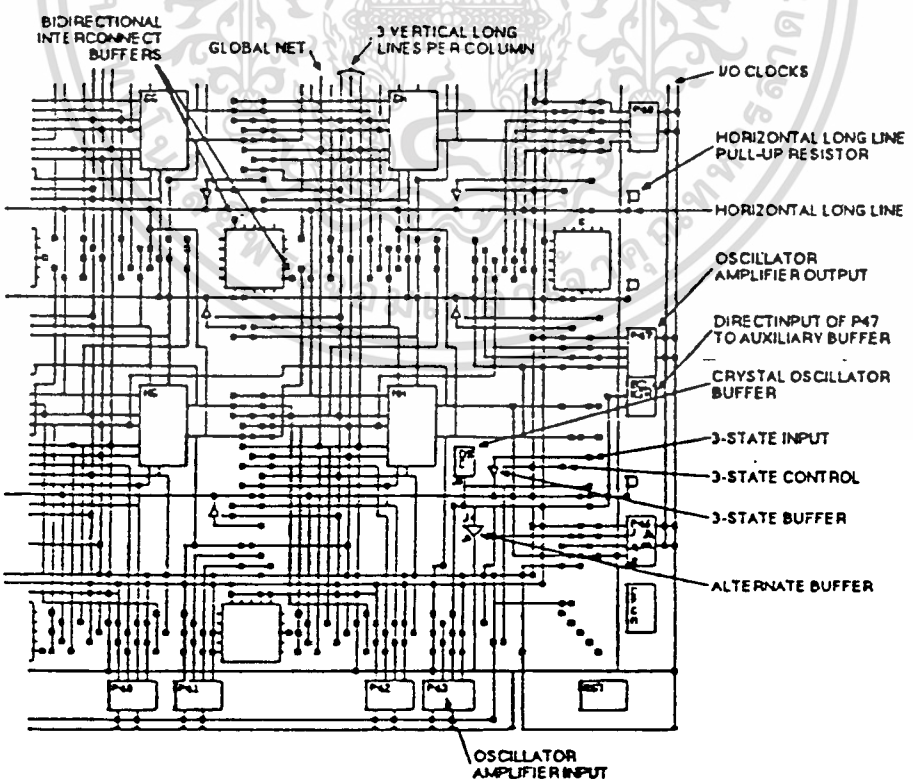


Figure 16. XACT Development System. An extra large view of possible interconnections in the lower right corner of the XC3020.

การโปรแกรม (Programming)

การเริ่มต้น (Initialization phase)

วงจรภายใน Power-on-reset จะถูกกระตุ้นให้ทำงานเมื่อมีการจ่ายไฟให้กับ LCA เมื่อ Vcc เพิ่มขึ้นถึงขีดที่ LCA จะทำงานได้มีค่า อยู่ในย่าน 2.5-3 v Programmable I/O Output Buffer จะถูก disable และ high impedance pull-up Resistor จะทำให้เป็นขา User I/O Pin วงจร Time-out delay จะถูกรีเซ็ตให้ทำงานเพื่อรอให้ Power supply คงที่ก่อน วงจร initialization State time-out (ประมาณ 11-33 Ms) ถูกกำหนดโดย 14 bit counter ภายใน LCA เอง จากตารางที่ 1 จะเห็นว่าเราสามารถเลือกได้โดยขา M0,M1 และ M2 ใน Master Configuration Mode LCA จะเป็นตัวกำเนิด Configuration Clock (CCLK) การเริ่มที่จะทำ Configuration ใน Peripheral หรือ Slave Mode จะต้องหน่วงเวลาให้นานพอที่จะทำให้ช่วงเวลาการ Initialization เสร็จสิ้นไป Mode line ใน LCA ใน Master Configuration Mode จะยืดเวลาการ Initialization State ออกไป โดยใช้ Delay time 4 ค่า ตั้งแต่ 43-130 mS เพื่อยืนยันว่า daisy-chain Slave devices ทั้งหมด ถึงแม้ว่า Master จะเร็วและ Slave จะช้ากว่ามาก Figure 18 แสดงถึง State sequence ขั้นตอนสุดท้ายของ Initialization LCA จะเข้าสู่ Clear State หลังจากการ clear Configuration Memory เสร็จเรียบร้อยแล้ว ขา INIT ซึ่ง Active low จะแสดงให้รู้ว่าการ Initialization State ได้เสร็จสิ้นแล้ว หลังจากนั้น LCA ตรวจสอบสัญญาณ Low ที่ขา RESET ก่อนที่จะอ่านค่าจาก Mode lines เพื่อเข้าสู่ Configuration State

M0	M1	M2	Clock	Mode	Data
0	0	0	Active	Master	Bit Serial
0	0	1	Active	Master	Bytewide addr = 0000 up
0	1	0	-	reserved	
0	1	1	Active	Master	
1	0	0	-	reserved	
1	0	1	Passive	Peripheral	
1	1	0	-	reserved	
1	1	1	Passive	Slave	

Long_line

long line จะเป็นเส้นทางเดินของสัญญาณที่ไม่ผ่าน switch matrices และมีจุดประสงค์หลักที่ช่วยให้สัญญาณที่จะต้องเดินทางผ่านระยะทางไกลๆ หรือช่วยให้ Signal มี Minimum skew rate ระหว่างปลายทางหลายๆ จุด Long line ดังแสดงใน Figure 13 ลากในแนวนอนและแนวตั้งในส่วนสูงและความกว้างของ Interconnect area ในแต่ละ Interconnect Column มี 3 เส้นในแนวตั้ง และในแต่ละ Interconnect Row จะมีเส้น Long line 2 เส้นพาดในแนวนอน สัญญาณ พิเศษถูกวางตำแหน่งอยู่ถัดจาก Switching Long line สามารถถูกขับได้ด้วย Logic Block หรือ I/O Block Output ทีละ Column คุณสมบัติอันนี้เองทำให้มีค่า skew rate ที่ต่ำเอาไว้สำหรับไปขับ Clock การเชื่อมต่อแบบ Interconnection Long line แสดงใน Figure 14 Buffer ที่อยู่มุมด้านซ้ายของ LCA Chip ขับ Global Net ซึ่งว่างอยู่สำหรับขา Input(K) ของ Logic Block การใช้ Global Buffer สำหรับสัญญาณ Clock ทำให้ skew rate free มีค่า fan-out สูงและ synchronous Clock ใช้ได้ในทุกๆ IOB และ CLB Configuration bit สำหรับ K Input สามารถเลือกให้เป็น Global line หรือเป็น Routing Resource ใน สถานะ แบบ Flip-flop CMOS แบบความเร็วสูง

Buffer ที่อยู่ในตำแหน่งมุมด้านล่างของชุด Array มีหน้าที่ขับ horizontal long line สามารถโปรแกรม drive Vertical long line ได้ Buffer ที่ตัวมี low skew และ high fan out Network ที่ถูกสร้างขึ้นโดย Alternate Buffer Long line นี้สามารถเลือกให้ต่อเพื่อขับขา (K) Input ของ Logic Block

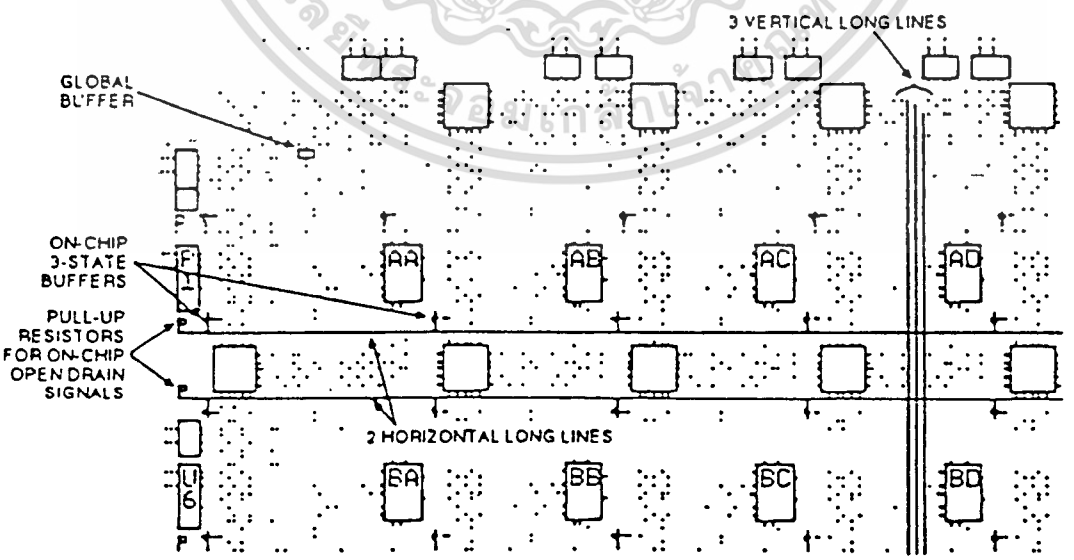


Figure 13. Horizontal and Vertical Long Lines. These long lines provide high fan-out, low-skew signal distribution in each row and column. The global buffer in the upper left die corner drives a common line throughout the LCA.

Configuration Data

Configuration Data เป็นตัวกำหนดการเชื่อมต่อ Resource ภายใน LCA เพื่อทำให้ LCA ทำงานเป็นหน้าที่ตามที่ต้องการ ซึ่ง Data สามารถ Load จากภายนอกเมื่อจ่ายไฟให้ LCA หรือทางการสั่งงานโดยสัญญาณคำสั่ง(Command)มีรูปแบบในการโปรแกรมหลายรูปแบบให้เลือกตามความเหมาะสม รูปแบบแต่ละอย่างเลือกได้โดยขา select Mode Pin โดยขาถูกอ่านเข้ามาก่อนที่จะเริ่ม Configuration time รูปแบบของ Data อ่านจะอยู่ในรูป Serial หรือ Byte parale ขึ้นอยู่กับ Configuration Mode

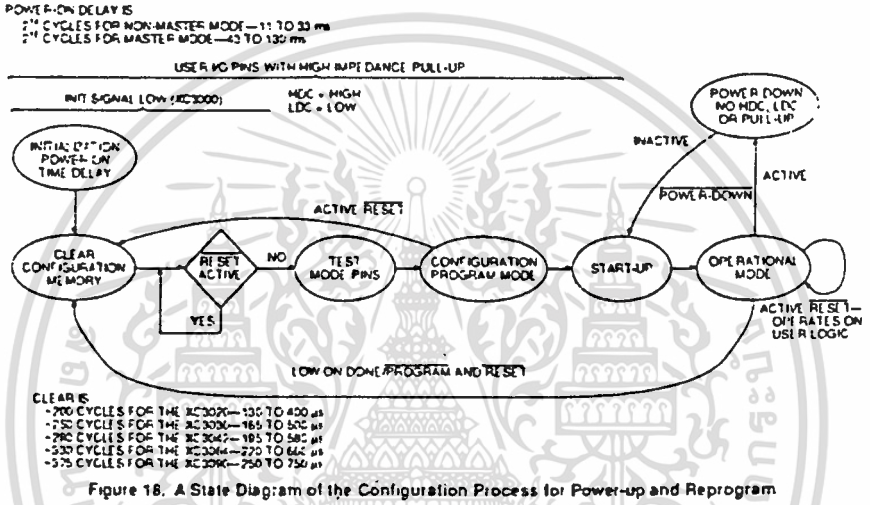
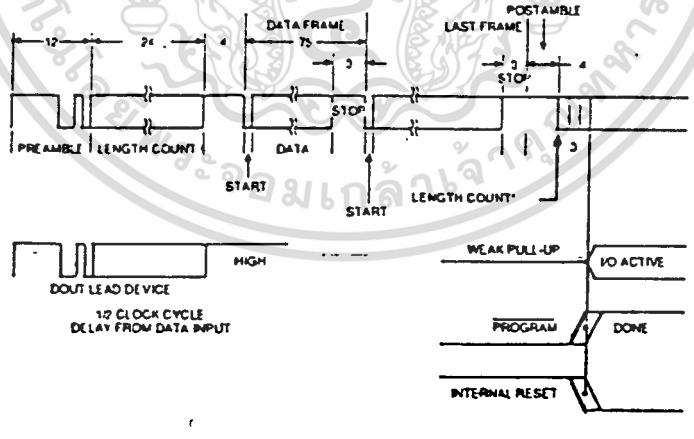


Figure 18. A State Diagram of the Configuration Process for Power-up and Reprogram



* The configuration data consists of a composite 40-bit preamble-length count, followed by one or more concatenated LCA programs, separated by 4-bit postambles. An additional final postamble bit is added for each slave device and the result rounded up to a byte boundary. The length count is two less than the number of resulting bits.

Timing of the assertion of DONE and termination of the INTERNAL RESET may each be programmed to occur one cycle before or after the I/O outputs become active.

Figure 20. Configuration and Start-up of One or More LCAs.

Configuration bit stream เริ่มต้นด้วย high preamble bit, 4bit preamble code และ 24 bit length count เมื่อกระบวนการ Configuration เริ่มต้น counter ใน LCA จะเซ็ทเป็นศูนย์ และเริ่มที่จะนับไปจนเท่ากับจำนวน Clock ของ Configuration cycle แต่ละ frame ของ Configuration Data ที่ส่งไปให้ LCA ภายใน LCA จะจัดการแปลงให้เป็น Data word แต่ละ word ก็จะ Load แบบ Parallel เข้าไว้ในConfiguration Memory Array การ Load Configuration Data จะสิ้นสุดเมื่อ current length count เท่ากับ Load length cout และ จำนวน Configuration Data ที่ต้องการได้ถูกเขียนลงไปแล้ว Internal User flip-flop จะถูก reset ระหว่างการ Configuration ขาที่ใช้ในการแสดงการ Configuration คือ high during Configuration (HDC) และ low during Configuration LDC และขา DONE/PROG อาจใช้ลักษณะสัญญาณภายนอกระหว่างการ Configuration ใน Master Mode จะใช้ LDC ที่ Active low เพื่อใช้ในการ Enable EPROM Chip หลังจาก Configuration Data ชุดสุดท้ายถูก Loaded เรียบร้อยแล้วค่า Length count และ Compare count เปรียบเทียบว่าเท่ากันเรียบร้อย User I/O Pin ก็จะเริ่ม Active

Master Mode

ในการทำงานแบบ Master Mode ตัว LCA จะ Load Configuration Data จากหน่วยความจำภายนอกเข้ามาโดยอัตโนมัติ มี Mode ที่แตกต่างกัน 8 Mode ใช้ Internal timing ภายในจ่ายให้ Configuration Clock (CCLK) เพื่อที่จะเป็นฐานเวลาในการนำเอา Data ที่เข้ามาทาง Serial Master Mode รับเอา Configuration Data ผ่านเข้ามาทางขา Data in (DIN) จาก synchronous source เช่น Xilink Serial Configuration PROM ,Parallel Master Low and Master high Mode รับเอา Parallel Data มาจาก D0-D7 โดยสัมพันธ์กับ address 16 bit ที่กำหนดโดย LCA Figure 22 แสดงตัวอย่างการต่อ Parallel Mode LCA เริ่มต้นที่ address 0000 และเพิ่มขึ้นเรื่อยๆ สำหรับ Master low ส่วน Master High จะเริ่มที่ FFFF และจะลดค่าลงมาเรื่อยๆ 2 Mode ที่สร้างขึ้นมาเพื่อให้ใช้งานได้กับ Microprocessors ที่เริ่มต้นทำงานในลักษณะต่างๆ กัน สำหรับทั้ง Master high และ Master Low ข้อมูล (Data Byte) จะถูกอ่านแบบขนานทุกๆ Read Clock (RCLK) และส่งเข้าไปภายในแบบ Serial โดย Configuration Clock

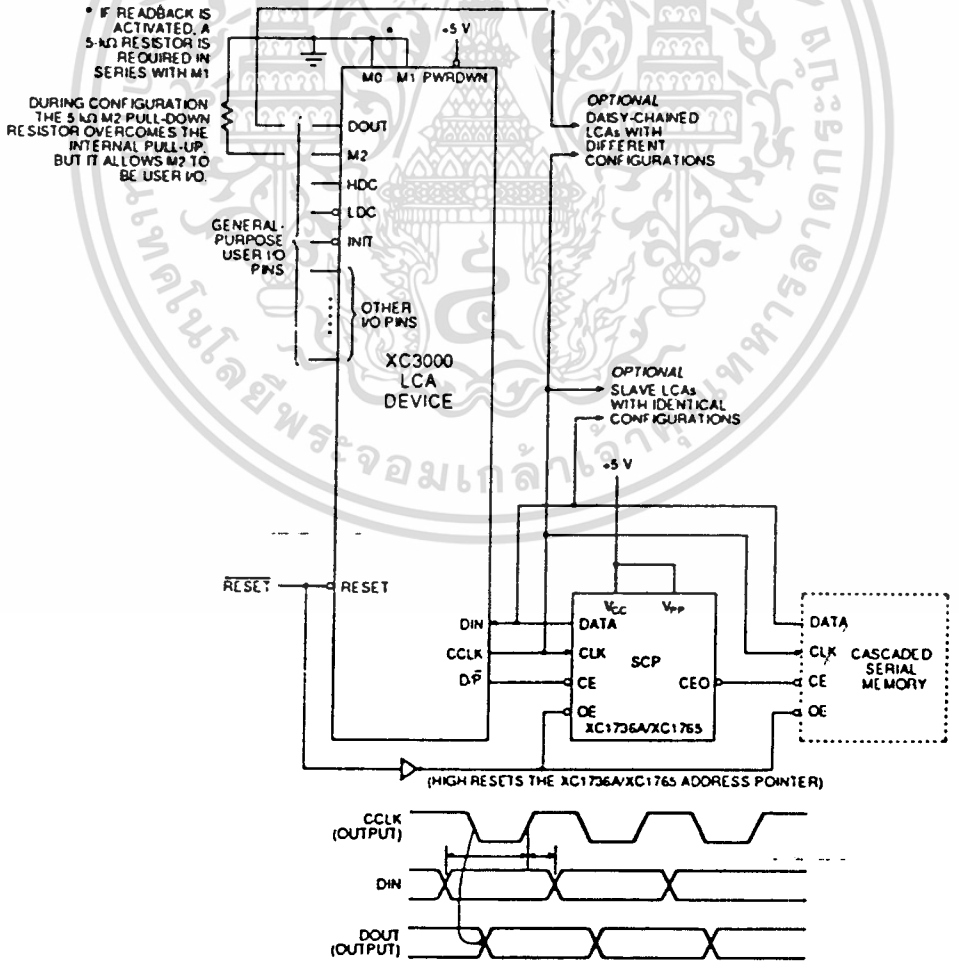


Figure 21. Master Serial Mode. The one-time-programmable XC1736A/XC1765 Serial Configuration PROM supports automatic loading of configuration programs up to 36K/64K bits. Multiple devices can be cascaded to support additional LCAs. An early D/P inhibits the PROM data output a CCLK cycle before the LCA I/Os become active.

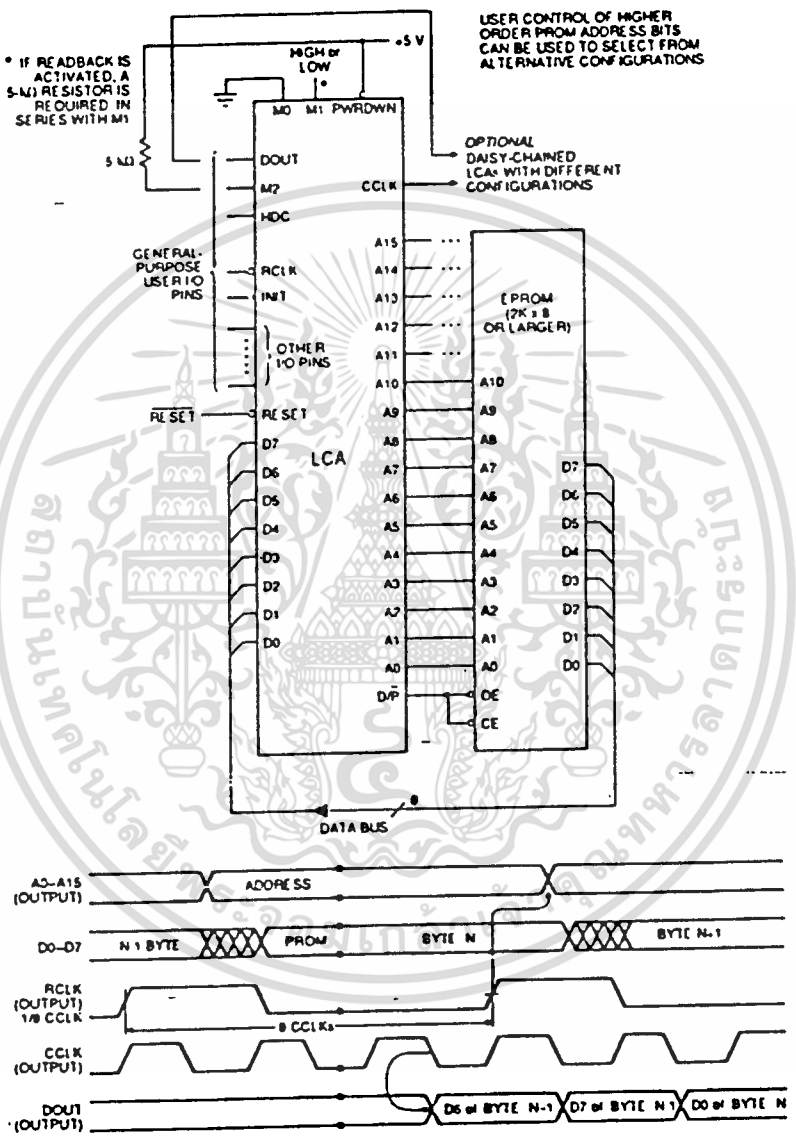


Figure 22. Master Parallel Mode. Configuration data are loaded automatically from an external byte wide PROM. An early D/P inhibits the PROM outputs a CCLK cycle before the LCA I/Os become active.

Peripheral Mode

Peripheral Mode เป็นการส่งผ่านข้อมูลโดยผ่านการ Load แบบ Byte wide ลักษณะคล้ายๆ Microprocessors interface ดัง Figure 23 แสดงการต่อแบบ peripheral Write Cycle Processor จะถูก Decode จาก สัญญาณ writ strobe (WS) ซึ่ง Active low และ 2 สัญญาณ Active low (CS0,CS1) และสัญญาณ Active high (CS2) ถ้าสัญญาณต่างๆ เหล่านี้ไม่ได้ถูกจัดตาม Logic ที่ต้องการจะต้องต่อกับระดับสัญญาณที่ต้องการ LCA จะรับข้อมูล 1 Byte ผ่านทางขา D0-D7 ทุกๆ write cycle แต่ละ Byte จะถูก Load เข้า Buffer Register จากนั้น LCA จะกำเนิดสัญญาณ Configuration Clock จากวงจรกำเนิดฐานเวลาภายใน และทำการส่งออกแบบ Parallel ข้อมูลที่รับเข้ามาออกไปยัง Slave ที่ต่ออยู่แบบอนุกรมทางขา Data Out (DAT) ขา Output Logic high ที่ขา READY/BUSY แสดงการ Load แต่ละ Byte เสร็จพร้อมที่จะ Load Byte ใหม่ต่อไปคล้ายกับ Master Mode Peripheral Mode อาจใช้เป็นแบบ Lead device หรือ Daisy-chain device

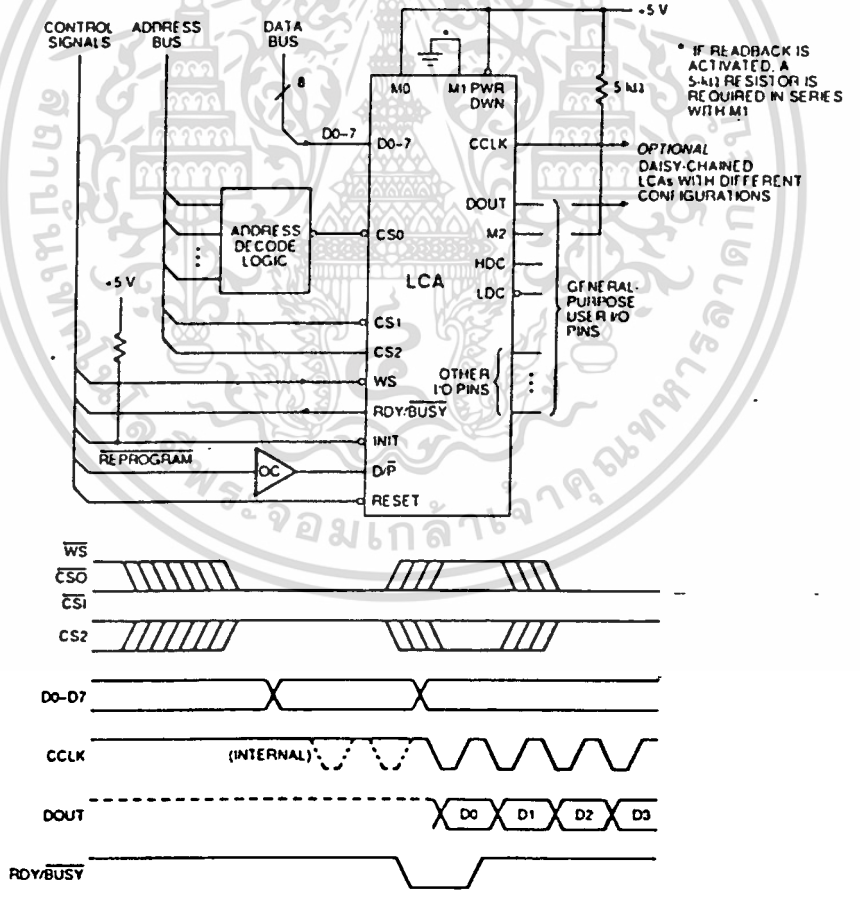


Figure 23. Peripheral Mode. Configuration data are loaded using a byte-wide data bus from a microprocessor

Slave_Mode

เป็นรูปแบบการ Load Configuration Data อย่างหนึ่ง โดย Data จะส่งออกในรูปแบบอนุกรม Serial โดยมีการ Synchronization Input Clock ตัวเดียวลักษณะการต่อ Slave Mode จะอยู่ในรูปของ Daisy-chain โดยที่ Data in ของ LCA จะต่อกับ Data Out ของ LCA ที่ต่ออยู่ก่อนหน้านี้ Clock ที่ให้ในระบบอาจมาจากอุปกรณ์หลักใน Peripheral หรือ Master Mode Data อาจส่งมาจากไมโครโปรเซเซอร์หรือวงจรที่สร้างขึ้นมาเป็นพิเศษก็ได้

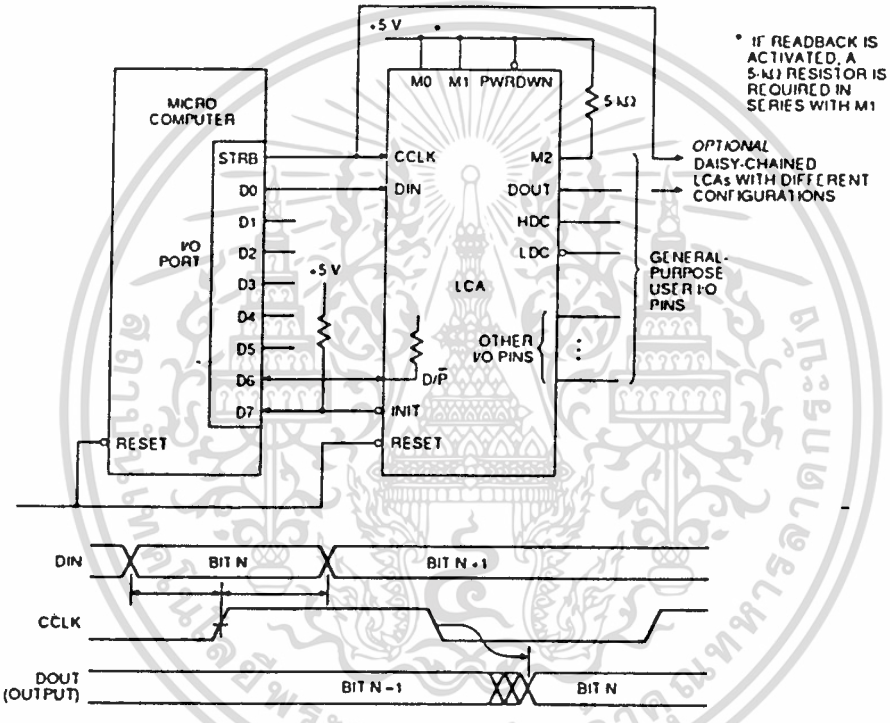


Figure 24. Slave Mode. Bit-serial configuration data are read at rising edge of the CCLK. Data on DOUT are provided on the falling edge of CCLK.

Daisy-chain

XACT Development ใช้สำหรับสร้างสัญญาณต่างๆ ซึ่งจะรวมเป็น Configuration ของ LCA แต่ละตัวใน Daisy-Chin ซึ่งประกอบไปด้วย preamble, length count สำหรับจำนวน Bitsteram ทั้งหมดที่ต้องสร้างขึ้นบวกกับจำนวน Bit พิเศษใน Device แบบ Daisy-Chain หลังจากที่ Load preamble และ length ไปยัง Daisy-Chain แล้ว Load Device LCA ตัวหลักก็จะเริ่ม Load เอา Configuration เข้าไปก่อนโดยจะให้ขา Data Out เป็น High จน Load Device, Load bitstream ครบหลังจากนั้น Bitstream Data ก็จะผ่านออกไปยังขา Data out ซึ่งเมื่อก่อนเป็น Hiht อยู่ออกไปยัง LCA ตัวต่อไป อุปกรณ์ Load Device ยังได้กำเนิด Configuration Clock (CCLK) ด้วย เพื่อทำการ Synchronization ที่จะส่งไปยัง Down stream LCA Data จะถูกอ่านเข้าไปที่ขา Pin ทั้งขอบหน้าของ CCLK และ Shift ไปยัง DOUT ที่ขอบหลังของ CCLK

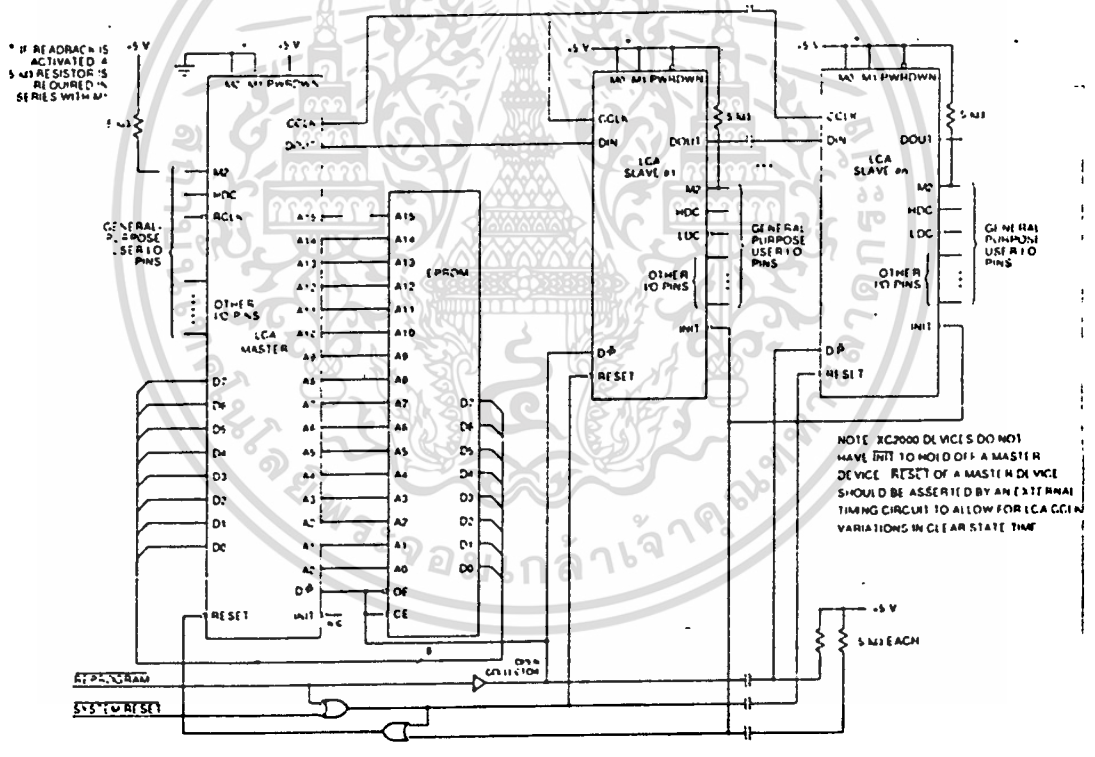


Figure 25. Master Mode Configuration with Daisy Chained Slave Mode Devices.
All are configured from the common EPROM source. The Slave mode device INIT signals delay the Master device configuration until they are initialized. A well defined termination of SYSTEM RESET is needed when controlling multiple LCAs.

Special Configuration Function

ภายใน Configuration Data นอกจากจะประกอบด้วย User Logic Function และการเชื่อมต่อภายในแล้วยังประกอบด้วย ส่วนของข้อมูลที่ใช้ควบคุมฟังก์ชันพิเศษอื่นๆ นอกจากนั้นอีก ได้แก่ ฟังก์ชัน ดังต่อไปนี้

- Input Threshold
- Readback disable
- DONE Pull-up Resistor
- RESET timing
- Oscillator frequency

ฟังก์ชันข้างต้นถูกควบคุมโดย Configuration Data bit ที่ถูกสร้างขึ้นโดย XACT Development system software เรียกขบวนการนี้ว่า bitstream generator process

Input Threshold

ก่อนที่บวนการ Configuration จะสิ้นสุด LCA ทุกตัวจะมี Input threshold ของ TTL compatible แต่หลังจากขบวนการทำ Configuration สิ้นสุดลงแล้ว Input Threshold จะเป็น CMOS หรือ TTL นั้นก็จะขึ้นอยู่กับการโปรแกรม การใช้ TTL compatible ต้องใช้ supply เพิ่มขึ้นมาช่วยสำหรับ Threshold shifting มีข้อยกเว้นคือ Threshold ของขา PWRDWN และ Direct Clock จะต้องเป็น CMOS เสมอก่อนที่ขบวนการ Configuration จะสิ้นสุด User I/O Pin แต่ละขาทุกขาจะมีค่าเป็น high Impedance pull-up ข้อมูลใน Configuration Program สามารถทำให้ IOB ถูก Pull-up Resistor เพื่อให้เป็น Input Load ป้องกัน Input อยู่ในสภาวะลอย (Floating) เมื่อขา User I/O นั้นไม่ได้ใช้งาน

Readback

ข้อมูลภายใน LCA สามารถจะอ่านออกมา (Readback) ได้ ถ้าตอนโปรแกรม Bitstream ได้กำหนด Option ในการทำ Readback เอาไว้ การ Readback มีเอาไว้เพื่อทำการตรวจสอบความถูกต้องของ Configuration ที่โปรแกรมเข้าไป หรือใช้สำหรับตรวจสอบ Internal Logic ขณะที่ทำการ Debug ทางเลือก (Option) ในการทำ Bitstream Readback มี 3 แบบ ได้แก่

- Never หมายถึง ห้ามไม่ให้มีการ Readback
- One-time หมายถึง ห้ามไม่ให้ Readback หลังจากฟังก์ชัน Readback ไปแล้ว 1 ครั้ง
- One-Command หมายถึง ใช้คำสั่ง Readback ได้ตามต้องการไม่มีข้อจำกัด

ขบวนการ Readback สามารถกระทำได้โดยไม่ต้องใช้ขา User I/O Pin ใดๆ มีเพียงแต่ขา M0,M1 และ CCLK เท่านั้นที่ใช้การเริ่มขบวนการ Readback ทำได้โดยการส่งสัญญาณขอบหน้า (low to high) ไปที่ขา M0/RTRIG (Read Trigger) ขา CCLK Input จะต้องขับด้วยสัญญาณ Logic จากภายนอกในการที่จะ Readback Configuration Data ออกมาได้ ขอบหน้าของสัญญาณ CCLK 3 ลูกแรกจะส่ง Dummy Data ออก ขอบหน้าของสัญญาณรูปต่อไป-จะเริ่มส่ง Data ออกมาที่ขา M1/RDATA (Read Data) อย่าลืมจำสัญญาณจะมาในรูป Invert ตลอด Logic 0 ใน Configuration จะมีค่าเป็น 1 ใน Readback Logic 1 ก็เช่นกัน และแต่ละ Frame ของการ Readback จะมี start bit 1 bit มี Logic 1 แต่ไม่เหมือนกับ Configuration Data และจะมี 1 Stop bit (มี Logic เป็น 0 Bit ที่ 3 ของ Dummy Data ที่กล่าวมาข้างต้นอาจกล่าวได้ว่าเป็น Start bit ของ Frame แรก Data frame ทุก frame จะต้องถูกอ่านออกมาจนหมดก่อนที่จะสิ้นสุดขบวนการ Readback หลังจากนั้นจึงทำการคืน Mode select และ CCLK Pin กลับไปยังสภาวะปกติ

ขบวนการ Readback จะรวมเอาสภาวะปัจจุบัน (Current State) ของ Internal Logic Block Storage element และสภาวะของ I และ O ของ IOB แต่ละตัว ข้อมูลเหล่านี้จะถูกรวมเข้ากับ Configuration bit ในตำแหน่งที่ไม่ได้ใช้ข้อมูลเหล่านี้เป็นประโยชน์ต่อ XACT Development system ในการที่จะตรวจสอบสภาพ การทำงานของ Logic ภายในตัว LCA

Reprogram

LCA Configuration Memory สามารถเขียนเข้าไปใหม่อีกครั้งได้ ขณะที่กำลังทำงานอยู่ในระบบ ในการเริ่มต้นที่จะทำการ Reprogram ทำได้โดยให้สัญญาณขอบหน้า (low to high) เข้าไปที่ขา DONE/PROG ในการที่จะลด Noise ที่จะ เข้ามา สัญญานขอบหน้าที่เข้ามาจะถูกหารโดย Internal timing generation ก่อนประมาณ 2 cycle เพื่อให้แน่ใจ เมื่อการ Reprogram เริ่มต้นขึ้น ขา USER Programmable I/O จะถูก Disable ให้ต่อเข้ากับ high Impedance Pull-up Resistor LCA จะกลับเข้าสู่สภาวะ Clear State และ Configuration Memory จะถูก Clear ก่อนที่จะทำการ Reprogram

การ Reprogram โดยส่วนใหญ่จะกระทำโดยให้สัญญาณ Low เข้าที่ขา DONE/PROG เมื่อ LCA รับรู้สัญญาณ low จนกว่าขบวนการ Reprogram จะเสร็จสิ้น

Done Pull-up

DONE/PROG เป็นขาที่มีลักษณะ open-drain I/O Pin แสดงว่า LCA นั้นอยู่ในสภาวะกำลังทำงาน (operation State)การ pull-up Internal Resistor สามารถทำได้โดยใช้ XACT Development system ในการทำ Makebit ขา DONE/PROG ของ LCA หลายตัวที่ต่อแบบ Daisy-chain สามารถต่อร่วมกันได้เพื่อให้ทำการ Reprogram พร้อมกันทั้งหมด

DONE Timing

ช่วงเวลา(timing) ของ DONE Status Signal สามารถควบคุมได้โดยการเลือกที่ Makebit program ในการที่จะให้ CCLK เกิดขึ้นก่อนหรือหลังเพื่อให้ Output activate (ดู Figure 20) ความสะดวกในการควบคุมนี้มีประโยชน์ต่อการควบคุม external Function เช่นไป enable PROM หรือหน่วงเวลาระบบให้อยู่ใน Wait State

Reset Timing

เหมือนกับ DONE Timing ช่วงเวลาที่จะ Release สัญญาณ Reset ภายใน สามารถควบคุมได้โดยใช้ Makebit program ในการที่จะใช้ CCLK เกิดขึ้นก่อนหรือหลังที่จะให้ Output activate (ดังรูปที่20) ขา Reset นี้จะรักษาสภาวะ User Programmable flip-flop เอาไว้และจะ Latch ให้ค่าเป็น Logic low ระหว่างการทำ Configuration

ประสิทธิภาพ (Performance)

Device Performance

LCA ที่มีประสิทธิภาพที่ดีขึ้นอยู่กับขบวนการผลิต คล้ายกับขบวนการผลิต CMOS high-speed static Memory ประสิทธิภาพของ Device อาจจะวัดได้จากหน่วยของ Minimum Propagation delay ของ Logic element ที่จริงแล้ว toggle frequency ของ flip-flop จะเป็นตัววัดประสิทธิภาพโดยรวมของ LCA การวัดประสิทธิภาพของ toggle performance แสดงการต่อดังใน Figure 26 flip-flop Output Q จะถูกย้อนกลับไปยัง Combinatorial Logic เพื่อให้เป็นสัญญาณ Q เพื่อที่จะเป็น Toggle flip-flop

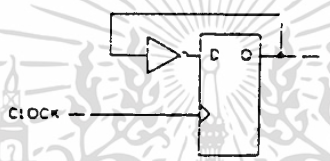


Figure 26. Toggle Flip-Flop. This is used to characterize device performance.

ประสิทธิภาพของ LCAที่แท้จริงถูกกำหนดโดยเวลาของ Critical Path ซึ่งรวมทั้ง fixed timing ของ Logic และ storage element ใน Path นั้น timing ที่มีความสัมพันธ์กับการ Route Network ค่า Timing Worst-case จะแสดงใน Performance Data เพื่อบอกให้ User ทราบ จะได้ใช้ความสามารถของ device ได้สูงสุด XCAT Development จะคำนวณหาค่า Worst case path โดยใช้ค่า actual Impedance และ Loading Information Figure 27 แสดงรูปแบบการเชื่อมต่อที่กำหนด system performance ในลักษณะต่าง ๆ

Logic_Block_Performance

ประสิทธิภาพของ Logic Block แสดงให้เห็นได้จาก Propagation time จาก interconnectpoint ที่ Input ของ conbinatorial Logic ถึง Output ของ Block ใน Interconnect area ส่วนประสิทธิภาพของ Combinatorial เป็นอิสระจาก special Logic Function เพราะว่าใช้การพัฒนาแบบ table look-up Timing จะแตกต่างเมื่อส่วนของ Combinatorial Logic ต่อกับส่วนของ storage element สำหรับส่วนของ Combinatorial Logic Function ที่ขับ Data Input ของ storage element ค่า critical timing คือ Data setup ที่มีความเกี่ยวข้องกับ Clock edge ใน Flip-Flop element ค่า delay จากแหล่งกำเนิด Clock ถึง Output ของ Logic black เป็นค่า critical timing ของสัญญาณผลิตขึ้นโดย storage element loading ของ Logic Block จะถูกจำกัดเกิดขึ้นเมื่อต่อเข้ากับ Network ขนาดใหญ่ๆเท่านั้น ประสิทธิภาพในเรื่องความเร็วของ Logic Block ขึ้นอยู่การทำงานของ supply voltage และอุณหภูมิ (Figure 29)

Interconnect_Performance

ประสิทธิภาพของการเชื่อมต่อขึ้นอยู่กับการ routing Resource ที่มีอยู่ภายในในการที่จะสร้างทางเดินของสัญญาณภายใน LCA ดังที่กล่าวมาแล้วข้างต้นในเรื่องของProgrammable Interconnect Direct Interconnect จะเป็นการเชื่อมต่อที่เร็วที่สุด ส่วน Long Line ซึ่งเป็น Single metal segment นั้นจะให้ค่าความต้านทานต่ำสุดปลายถึงปลายแต่จะให้ค่า high capacitance สัญญาณที่ขับผ่าน Programmable switch จะมีค่า Impedance ของ switch รวมเข้าไปด้วยนอกจากค่าความต้านทานธรรมดาประสิทธิภาพของ General purpose Interconnect นั้นขึ้นอยู่กับจำนวน switch และจำนวน segment ที่ใช้ จำนวนของ Buffer แบบbidirection และจำนวน loading บนทางเดินของสัญญาณทั้งหมดภายใน path นั้น ในการคำนวณ worst-case ใน general Interconnect โดย XACT Development software นั้นรายงาน element ทั้งหมด

อธิบายหน้าที่การทำงานแต่ละขา Pin description

กลุ่มขาที่ใช้งานแบบถาวรใช้เป็น I/O ของ User ไม่ได้

VCC

มีจำนวน 2 ถึง 8 ขา ขึ้นอยู่กับขนาดของตัวถังต่อเข้ากับไฟ +5v โดยทุกขาจะต้องต่อให้ครบ

GND

มีจำนวน 2 ถึง 8 ขา ขึ้นอยู่กับชนิดของตัวถัง ต่อกับ GND ทุกขาจะต้องต่อให้ครบ

PWRDWN

ถ้ามี Logic low ที่ขานี้ (เป็น CMOS Input) จะหยุดการทำงานภายใน FPGA ทั้งหมด แต่ยังคงสภาพการถูกโปรแกรมเอาไว้ flip-flop ทุกตัวทุก reset Output ทุกตัวอยู่ในสภาวะ 3 State Input ทุกขาถูกให้เป็น Logic high เป็นอิสระจากสัญญาณ Input ที่ต่อเข้ามา ที่ขา PWRDWN เป็น Logic 0 Vcc อาจจะลดค่ามาอยู่ที่ประมาณ >2.3v เมื่อ PWRDWN กลับเข้าสู่สภาวะ Logic อีกครั้ง LCA จะเริ่มต้นทำงานเมื่อขา DONE เป็น LOW ในช่วงเวลา 2 cycle ของ Clock ภายใน 1 MHz ระหว่างการ Configuration ขา PWRDWN จะต้องเป็น high ถ้าไม่ใช้ให้ต่อเข้ากับ vcc

RESET

เป็นขา Input ที่ Row ซึ่งมีหน้าที่ 3 อย่างคือ ก่อนที่จะมีการ Configuration ต่อ LCA ขา RESET จะถูกทำให้เป็น Low เป็นหน่วงเวลาการ Configuration ออกไป วงจรจะตรวจสอบระดับของไฟ Vcc และ time out cycle เสียก่อน เมื่อ time-out cycle และ reset ได้จบสิ้นลง ระดับสัญญาณที่ขา M ทั้ง 3 จะถูกอ่านเข้ามาเพื่อทำการเลือกโหมดในการทำ Configuration

CCLK

ระหว่างการทำ Configuration Configuration clk จะเป็น Output เมื่ออยู่ใน Master Mode หรือ peripheral Mode แต่จะเป็น Input เมื่ออยู่ใน Slave Mode ระหว่างการทำ readblock CCLK จะเป็น Clock Input สำหรับเลื่อน Configuration Data ของ LCA แต่ละตัว CCLK ขับวงจรอิเล็คทรอนิกส์ภายในแบบ Dynamic ใน LCA Low time อาจเกิดขึ้นได้อย่าให้เกิน 2-3 microsecond เมื่อให้เป็น Input CCLK จะต้องต่อกับ high เอาไว้ Internal pull-up resister จะรักษาระดับ high เอาไว้เมื่อ Pin ไม่ได้ถูกต่อ

DONE/PROG (D/P)

DONE เป็นขา open drain Output ถูก Configuration ทั้งที่มีหรือไม่มี Internal pull-up register เมื่อขบวนการ Configuration สิ้นสุด วงจรภายใน LCA จะอ่านในแบบ synchronous ขา DONE ถูกโปรแกรมให้เป็น high 1 cycle ทั้งหลังหรือก่อนที่ Output ต่าง ๆจะ Active หลังจากที่ LCA ได้ถูกโปรแกรมไปครั้งหนึ่ง ถ้ามีการเปลี่ยนระดับสัญญาณจาก high to low ทำให้ LCA เริ่มกระบวนการ Reconfiguration ตัวเองขึ้นมาใหม่

M0/RTRIG

ขณะอยู่ใน Mode 0 ขา Input ขานี้ ร่วมกับขา M1,M2 ถูกอ่านค่าเข้าไปก่อนเพื่อกำหนด Mode ที่จะเริ่มในการทำกระบวนการ Configuration ถ้ามีการเปลี่ยนแปลงระดับของสัญญาณจาก Low-to-high ทั้งนี้หลังจากที่กระบวนการ Configuration ได้เสร็จสิ้นไปแล้ว จะทำให้เกิดกระบวนการ Readtrigger และเริ่มต้นกระบวนการ Readback Configuration Data ภายใน LCA ออกมา ใช้ CCLK เป็น Clock เลือก option ให้การ readblock ได้เมื่อตอนสร้าง bitstream

M1/RDATA

ขณะที่อยู่ใน Mode1 ขา Input ขานี้ร่วมกับขา M1,M2 จะถูกอ่านค่าเข้าไปเพื่อกำหนด Mode ในการที่จะเริ่มกระบวนการ Configuration ถ้าขบวนการไม่มีการใช้ Readback M1 สามารถต่อลงกราวด์ หรือ vcc ได้โดยตรง ถ้าจะมีการใช้ขบวนการ Readback M1 จะต้องต่อกับความต้านทาน 5K ลงกราวด์หรือ Vcc เพื่อใช้งานของ RDATA Output ขานี้จะ Active low read Data หลังจากขบวนการ Configuration เสร็จสิ้นขาจะเป็น Output สำหรับ Readback Data

ขาที่ใช้นำหน้าที่พิเศษและใช้ทำเป็น User I/O Pin ได้

M2

ระหว่างกระบวนการ Configuration ขา Input ขานี้จะมี Pull-up resister อยู่ ทำงานร่วมกับขา M0,M1 จะถูกอ่านค่าก่อนที่จะเริ่มทำการเลือก Mode ในการที่จะเริ่มกระบวนการ Configuration หลังจากกระบวนการ Configuration ผ่านไปแล้ว ขานี้ถูกใช้เป็น User I/O ได้

HDC

ขานี้จะมีสภาวะเป็น high ระหว่างการทำ Configuration เพื่อบอกให้รู้ว่าขณะนี้ยังทำกระบวนการ Configuration ยังไม่สิ้นสุด หลังจากการ Configuration แล้วขานี้จะถูกใช้เป็น User I/O Pin

LDC

ขานี้จะมีสภาวะเป็น Low ระหว่างการทำ Configuration เพื่อบอกให้รู้ว่าขณะนี้ยังทำการ Configuration ยังไม่เสร็จ หลังจากการทำ Configuration ขานี้จะถูกใช้เป็น User I/O Pin ได้ LDC นี้มีประโยชน์โดยเฉพาะอย่างยิ่งใน Master Mode ในการที่จะเอาสัญญาณ Low นี้ไป enable EPROM Configuration แต่หลังจาก program comgiguration เสร็จแล้วขานี้จะต้องถูกโปรแกรมให้มีค่าเป็น high

INIT

เป็นขา Output ที่จะ Active low ระหว่างที่มีการทำ Power initialization microprocessor ในลักษณะของ wire and ใน Slave Mode และ hold-off Signal ใน Master Mode หลังจาก Configuration เสร็จขานี้ใช้เป็น User I/O Pin ได้

BCLKIN

เป็นขา CMOS Direct level Clock Input Buffer ที่มุมล่างด้านขวา

XTL1

ขานี้ใช้สำหรับ User I/O ขานี้ถูกใช้งานเป็นoutputamplifier สำหรับขับ crystal จากวงจรภายนอก

XTL2

ขา User I/O ขานี้ถูกใช้งานเป็น Input ของ amplifier ในการที่จะต่อกับ crystal ภายนอก

CS0,CS1,CS2,WS

ขา Input ที่ 4 ขา เป็นชุดของสัญญาณ 3 ขาจะ Active ที่ Logic high,อีก 1 เส้น Active ที่ low ใช้สำหรับกำหนด Configuration Data peripheral Mode

RCLK

ระหว่างการทำ Master Parallel Mode Configuration RCLK จะกำเนิดสัญญาณ 'read' สำหรับ external dynamic Memory ภายนอก หลังจาก Configuration เสร็จเรียบร้อยขานี้จะถูกใช้เป็น User I/O Pin

RDY/BUSY

ระหว่าง peripheral Parallel Mode Configuration ขานี้ใช้สำหรับแสดงว่า Chip พร้อมที่จะรับข้อมูลในไบต์ต่อไปที่จะเขียนลงสู่ตัวมัน หลังจากกระบวนการ Configuration เรียบร้อยขานี้จะใช้เป็น User I/O Pin

D0-D7

เป็นกลุ่มของสัญญาณสำหรับส่งผ่านข้อมูลใน Mode parale Configuration Byte ใน Master Parallel และ peripheral Mode หลังจากกระบวนการ Configuration เสร็จสิ้นขานี้จะถูกใช้เป็น I/O Pin

A0-A15

ระหว่าง Master parale Mode ขาสัญญาณ 16 ขานี้จะให้ Output address สำหรับ Configuration EPROM หลังจากขาเหล่านี้ถูกใช้เป็น User I/O Pin ได้

DIN

ระหว่างการทำ Configuration ใน Slave หรือ Master Serial Configuration ขานี้ใช้สำหรับเป็นขา Data 0 Input หลังจากกระบวนการ Configuration เสร็จสิ้นขานี้จะเป็น User I/O Pin

DOUT

ระหว่างกระบวนการ Configuration ขานี้ใช้สำหรับเป็น Output สำหรับ Serial Data ส่งไปยัง Pin Serial Data Input ของ Slave ตัวต่อไปในคุณสมบัติของ daisy-chain หลังจากกระบวนการ Configuration เสร็จสิ้นขานี้ใช้เป็น User I/O

TCLKIN

เป็นขา Direct CMOS-level Input สำหรับ global Clock Buffer ขานี้ใช้เป็น Use I/O Pin ได้ อย่างไรก็ตาม TCLKIN เหมาะที่จะใช้สำหรับ Input global Clock และ global Clock ควรจะเป็น Clock หลักให้ Chip ปกติ Pin นี้จะใช้สำหรับเป็น Clock Input สำหรับ Chip Aurestricted User I/O pins I/O Pin อาจจะถูก program ให้เป็น Input และ Output ตาม Configuration ขา nrestricted และขาพิเศษดังที่ได้กล่าวมามี weak-pull-up resister 50K ซึ่งจะเป็น Active เมื่อ Power-up และยังคง Active จนกว่าจะงบการ Configuration

ขา User I/O Pin

I/O

ขา I/O Pin ทุกๆขาจะถูกโปรแกรมให้เป็น Input หรือ Output ตามต้องการ ขึ้นอยู่กับ Configuration Data ที่โปรแกรมเข้ามา แต่ละขาจะมี Weak Pull-up Resistor ค่า 50k ถึง 100k อยู่ จะ Active เมื่อ LCA ได้รับการจ่ายไฟ (Power up) และยังคงอยู่สภาพเต็มจนกว่ากระบวนการ Configuration จะสิ้นสุด

บทที่ 6
การสร้างและพัฒนาโครงงาน

โครงงานที่ทดลองสร้าง

ชิ้นงานที่นำเสนอคือ นาฬิกาปลุกดิจิตอล(Digital Alarm-Clock)

อธิบายลักษณะโดยทั่วไป (General Description)

ชิ้นงานชิ้นนี้นำเสนอการสร้างและออกแบบนาฬิกาปลุก (Digital Alarm Clock) โดยออกแบบให้มีฟังก์ชั่นการทำงานให้คล้ายกับ IC เบอร์ MM 5387 AA ของ NATIONAL SEMICONDUCTOR โดยใช้ภาษา VHDL ในการออกแบบ ซึ่งลักษณะโดยทั่วไปสามารถอธิบายได้โดยตัวภาษา VHDL Module Digital Alarm Clock ที่สร้างขึ้นนี้ สามารถแสดงค่าได้หลาย ๆ ฟังก์ชั่น ได้แก่ แสดงเวลา , แสดงเวลาปลุก , แสดงหลักวินาที , แสดงเวลาเปิด - ปิด อุปกรณ์ไฟฟ้า เป็นต้น โดยสามารถต่อเข้าโดยตรงกับ LED 7 SEGMENT มีเอ้าท์พุทพิเศษสามารถตั้งเวลาเปิด - ปิด อุปกรณ์ไฟฟ้าได้นานถึง 59 นาที การแสดงเวลานั้นสามารถเลือกได้ทั้งแบบ 12 Hour Format หรือ 24 Hour Format สัญญาณนาฬิกา Input สามารถเลือกได้ 2 ค่า คือ 50 Hz และ 60 Hz

คุณสมบัติ (Specification)

- ใช้งานที่ความถี่ 50 Hz หรือ 60 Hz
- แสดงเวลาได้ 2 รูปแบบ 12 หรือ 24 Hour Format
- AM/PM Output หลักหน้าเป็นศูนย์จะดับ (12 Hour Format)
- ตั้งเวลาปลุกได้ตลอด 24 ชม.
- Counter ทุกตัวสามารถ Reset ได้
- ขาควบคุม Fast และ Slow Setting
- ต่อ LED 7 SEGMENT ได้โดยตรง
- หยุดปลุกชั่วขณะเป็นเวลา 9 นาที
- ตั้งเวลาปิดเปิดอุปกรณ์ไฟฟ้าได้ในช่วง 1 - 59 นาที

การประยุกต์ใช้งาน (Applications)

- Alarm Clock
- Desk Clock
- Clock Radio
- Automobile Clock
- StopWatch
- Industrials Clock
- Portable Clock
- Photography Clock
- Industrial Timer
- Applilance Timer
- Sequencial Controller

อธิบายการทำงานแต่ละขา (Function Description)

- ขา 50 OR 60 Hz INPUT : เป็นสัญญาณในฐานเวลาให้แก่นาฬิกาปลุก โดยสัญญาณจะเป็น SQUARE WAVE เข้ามาสัญญาณนี้จะถูกหารเพื่อให้ได้ TIME BASE 1 Hz ป้อนวงจรภายในต่อไป
- ขา 50 OR 60 Hz SELECT INPUT : เป็นขาที่เอาไว้เลือกค่าความถี่ Input ที่ใช้งานโดยถ้าขานี้มี LOGIC " 1 " หมายถึงเลือกใช้สัญญาณ 60 Hz ถ้า LOGIC " 0 " เลือกใช้สัญญาณ 50 Hz
- ขา DISPLAY MODE & SELECT INPUT : เลือกลักษณะการแสดงผลถ้าไม่มีการเลือกใด ๆ ค่าที่ แสดงคือ เวลาปรกติ (Time Mode) ถ้าต้องการให้แสดง Mode ได้ก็ต่อขานั้นลง Vss ค่าต่าง ๆ ที่ต้องการก็จะแสดงผลที่ LED ตามต้องการ ลำดับความสำคัญในการแสดงที่ตารางที่ 1 ในกรณีที่กด 2 Mode พร้อมกัน Mode ที่ลำดับความสำคัญสูงก็จะแสดงก่อน
- ขา TIME SETTING INPUT : ขา FAST และ SLOW มีไว้สำหรับตั้งเวลาต่าง ๆ ตามต้องการ เช่น เวลาปลุก , ช่วงเวลาเปิดปิดอุปกรณ์ , เวลาปรกติ เป็นต้น ซึ่งการใช้งาน Fast กับ Slow นี้จะต้องใช้งานร่วมกับ Mode ที่เลือกแสดงอยู่ด้วย รายละเอียดแสดงดังตารางที่ 2

- ขา 12 OR 24 Hour SELECT Input : เลือกรูปแบบการแสดงเวลาปรกติ จะให้แสดง 12 Hour Format หรือ 24 Hour Format
- ขา ALARM OPERATION AND OUTPUT : ขานี้จะทำงานให้ LOGIC " 1 " เมื่อเวลาที่ตั้งเอาไว้ที่ Alarm ตรงกับเวลาจริง (REAL TIME) โดย Output จะค้างประมาณ 59 นาที สามารถให้สัญญาณหยุดชั่วคราวได้โดยการกด SNOOZE SWITCH เพื่อให้หยุดปลุกชั่วคราว ประมาณ 8 - 9 นาที จากนั้นสัญญาณจะทำงานอีกครั้ง
- ขา SNOOZE ALARM INPUT : ขาที่เป็นขาอินพุตเพื่อให้ Alarm OUT หยุดปลุกชั่วขณะ เป็นเวลา 8 - 9 นาที สามารถกดซ้ำได้ในระหว่างช่วงเวลา 59 นาที ที่มีการปลุก
- ขา ALARM OFF : ขานี้เป็นขาอินพุต เพื่อหยุดการปลุก เพื่อให้วงจรพร้อมที่จะรับอินพุตจากวงจร COMPARATOR เพื่อทำการปลุกครั้งต่อไป ถ้าต้องการไม่ให้มีการปลุกเลยให้ต่อขานี้ลง Vss ตลอด
- ขา SLEEP TIMER AND OUTPUT : ขานี้เป็นขา Output สามารถตั้งเวลาให้ทำงานได้ภายในช่วงเวลา 1 - 59 นาที ประโยชน์คือเอาไว้เปิดปิด - อุปกรณ์ต่าง ๆ เป็นช่วงเวลาตามเวลาที่ตั้งเอาไว้จะเริ่มทำงานพร้อมกับ ALARM OUTPUT ถ้าต้องการหยุด SLEEP OUTPUT ก่อนที่จะให้หมดตามเวลาให้ กด SNOOZE SWITCH ขา SLEEP OUTPUT ก็จะถูก Reset พร้อมจะทำงานในครั้งต่อไป

ตารางแสดงลักษณะการแสดงผล

Select Display Mode	DIGIT No. 1	DIGIT No. 2	DIGIT No. 3	DIGIT No. 4
Time Display	10 'S Of Hour	Hour	10'S Of Minute	Minute
Second Display	Blank	Minutes	10'S Of Second	Second
Alarm Display	10'S Of Hour	Hour	10'S Of Minute	Minute
Sleep Display	Blank	Blank	10'S Of Minute	Minute

ตารางแสดงการตั้งเวลา

Selected Display Mode	Control Input	Control Function
Time	Slow	ค่าเวลาเพิ่มขึ้นด้วยอัตรา 2 Hz
	Fast	ค่าเวลาเพิ่มขึ้นด้วยอัตรา 60 Hz
	Both	ค่าเวลาเพิ่มขึ้นด้วยอัตรา 60 Hz
Alarm	Slow	ค่าเวลาเพิ่มขึ้นด้วยอัตรา 2 Hz
	Fast	ค่าเวลาเพิ่มขึ้นด้วยอัตรา 60 Hz
	Both	เวลาปลุกรีเซ็ทไปยัง 12:00AM(12 HourFormat)
		เวลาปลุกรีเซ็ทไปยัง 00:00 (24 HourFormat)
Second	Slow	หยุดเวลาไม่ให้นับ (HOLD)
	Fast	รีเซ็ทหลักวินาทีเป็นศูนย์โดยไม่ทดไปยังหลักนาที
	Both	เวลา TIME Reset ไปที่เวลา 12: 00 AM
		หรือ 00 : 00 (24 HourFormat)
Sleep	Slow	ลดค่าเวลาลงด้วยอัตรา 2 Hz
	Fast	ลดค่าเวลาลงด้วยอัตรา 60 Hz
	Both	ลดค่าเวลาลงด้วยอัตรา 60 Hz

อธิบายการทำงานแต่ละ BLOCK

1.CONVERTER ทำหน้าที่เลือกรูปแบบการแสดงผล,แปลงข้อมูลให้อยู่ในรูป7 SEGMENT อาจจะเลือกการแสดงผลแบบ 12 HourFormat หรือ 24 HourFormat ได้

2.COMPARATOR ทำหน้าที่เปรียบเทียบเวลาที่ตั้งไว้กับเวลาที่เดินจริง เมื่อเท่ากันก็จะส่งสัญญาณไปให้ภาคควบคุมต่อไป

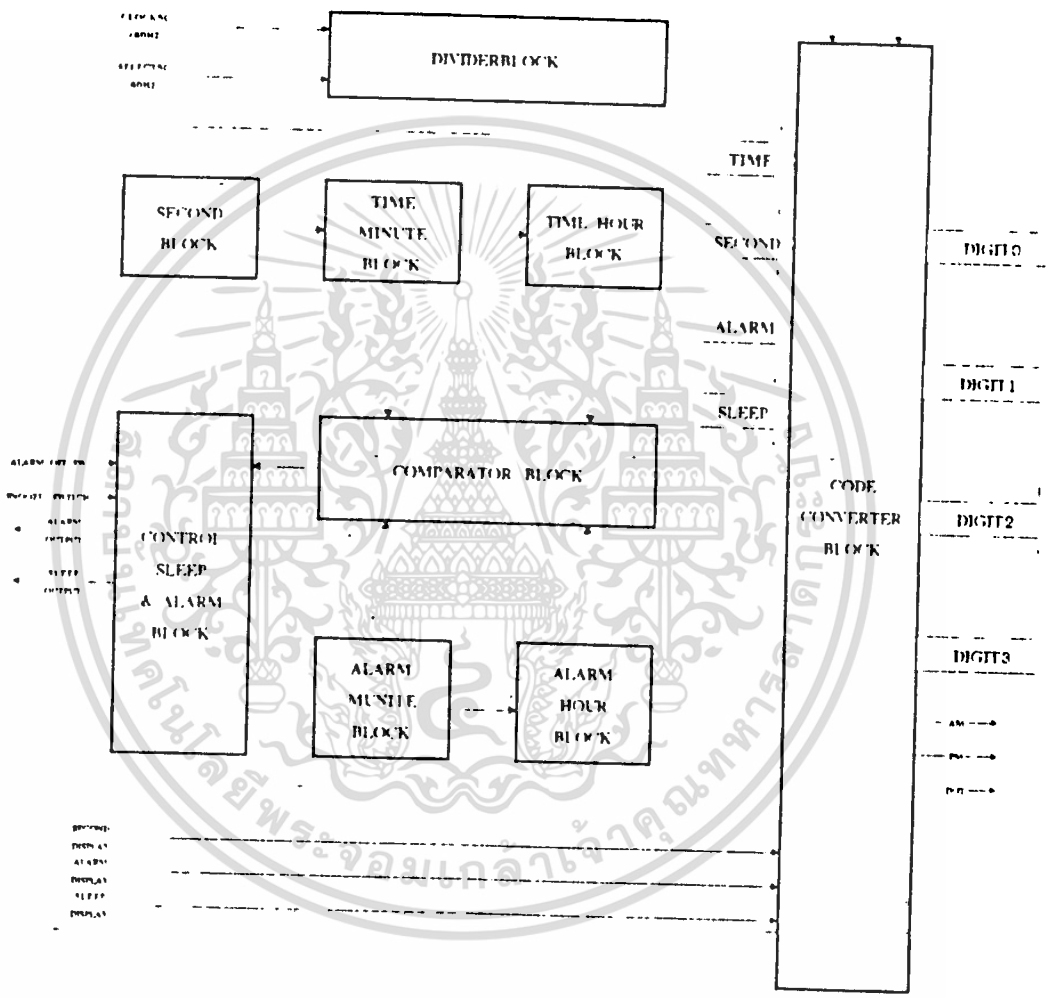
3.TIME MINUTE COUNTER,TIME HOUR COUNTER ทำหน้าที่เป็นวงจรนับเวลาจริงหลักนาทีและชั่วโมง

4.ALARM MINUTE COUNTER,ALARM HOUR COUNTER ทำหน้าที่เป็นวงจรนับสำหรับตั้งเวลาปลุกหลักนาทีและชั่วโมง

5.DIVIDER ทำหน้าที่เป็นวงจรหารเพื่อกำเนิดสัญญาณนาฬิกาปลุกให้กับทุก ๆ ส่วนภายในระบบโดยรับ Input 50 Hz หรือ 60 Hz ตามต้องการ

6.TIME SECOND COUNTER ทำหน้าที่เป็นวงจรนับหลักวินาที

7.ALARM & SLEEP CIRCUIT เป็นวงจรควบคุมการปลุกและการเปิดปิดอุปกรณ์ไฟฟ้า



BLOCK DIAGRAM

ขั้นตอนการพัฒนาโครงงาน

เป้าหมายในการทำงานคือการทำ VHDL Model ของโครงงานชิ้นนี้ เริ่มจากการศึกษาตัว Application Digital Alarm Clock ให้ละเอียด ซึ่งการศึกษาระดับ Block Diagram ก็คิดว่าเพียงพอแล้วที่จะเริ่มสร้าง Model ขึ้นมาโดยใช้ VHDL เมื่อเข้าใจรายละเอียดการทำงานแต่ Block แล้วจึงเริ่มลงเมื่อเขียนโปรแกรม โดยมองว่าการทำงานในภายใน Block เป็น State Machine,Truth Table,PLA table ตามต้องการขึ้นอยู่กับผู้ออกแบบ Component สามารถสร้างได้จริงโดยการ Synthesis เมื่อเริ่มทำการเขียนก็สามารถแยกรายละเอียดภายใน Block ลงไปได้อีกหลาย ๆ Component ซึ่งแต่ละ Component เขียนอธิบายด้วย Behavioral Style Modelling เมื่อ Compile แต่ละส่วนแล้วทดสอบ (Simulation)เพื่อตรวจสอบการทำงานให้ตรงตามที่ออกแบบในแต่ละส่วนไป เมื่อได้ทุกส่วนครบจึงเอามาประกอบกันจนเป็น Model ที่สมบูรณ์ซึ่งทั้งหมดให้ VHDL พัฒนาอยู่ในรูปของ Source Code ส่วนการสร้าง (MAP) เป็น Hardware เราใช้ Field Programmable Gate Array (FPGA) ในที่นี้จะกล่าวถึง Tools ต่าง ๆทั้ง Hardware และ Software ที่ใช้ในการทำงาน ซึ่งมีดังนี้

ฮาร์ดแวร์ (HARDWARE)

1.Microcomputer PC 486DX-66 RAM 16 MB
2.XC3000 Design Demostration Board
3.XC4000 Design Demostration Board
4.Xchecker
5.XPP(XILINX PROM Programmer)
6.Universal EPROM Programmer

ซอฟท์แวร์ (SOFTWARE)

1.Workview PLUS for Windows ของ VIEWLOGIC inc. เป็น Software CAE ทางด้านไฟฟ้าอิเลคทรอนิกส์ ทำงานบน PC Platform ภายใต้ระบบ Windows มี User Inerface แบบ WINDOWS ภายใน Workview PLUS ประกอบไปด้วย Software หลายตัว ตั้งแต่การสร้างออกแบบ,การทดสอบการทำงาน ซึ่งจัดว่าเป็นชุด Software ที่สมบูรณ์มากที่สุดชุดหนึ่งที่ทำงานบน PC Platform ในขณะนี้ วงจรดิจิตอลและอนาล็อกได้ทุกประเภท ภายในชุด Software ที่กล่าวมา มีโปรแกรมต่าง ๆที่นำมาใช้ทำงานดังนี้

- VIEWDRAW เป็น Software ใช้ในการสร้างและออกแบบระบบ (Design Creation) โดยวิธีการสร้างสามารถทำได้โดยวาด Schematic Diagram
- VIEWSIM,VIEWTRACE เป็น Software ใช้ในการจำลองการทำงาน (Design Verification) ดู Timing Waveform ของ Design ที่เราได้ออกแบบ
- VIEWSYNTHESIS,VIEWGEN เป็น Software ใช้สังเคราะห์ Schematic ขึ้นมาจาก VHDL Model

2. XACT Development System ของ XILINX

เป็น Software ที่จัดการเรื่อง Implementation FPGA บน Chip XC3000,XC4000 XILINX Software จะทำหน้าที่แปลงวงจร Schematic ของเราให้อยู่ในรูปของ Bit Stream Configuration ที่สามารถจะไป Configuration ให้ XILINX FPGA ทำงานตามวงจรที่เราต้องการ ภายในชุด XACT development Software ที่ Software หลาย ๆตัวภายในที่ช่วยทำหน้าที่ต่าง ๆ โดยสามารถทำแบบอัตโนมัติหรือเลือกทำแบบ manual ก็ได้ ตัว Software มาพร้อมกับชุดทดลอง demo boud และสาย cable xchuker ช่วย Download Bit Stream จาก PC ผ่านทาง comport ส่งไปยัง XILINX FPGA demestration Board เพื่อทดสอบการทำงานได้จริง หน้าที่การทำงานของ Software ต่าง ๆ ภายใน XACT แยกออกเป็นส่วนต่าง ๆ ตามหน้าที่ดังนี้

ส่วนของการทำ Design entry และ Conversion

XMake

ช่วยแปลง Schematic ให้เป็น LCA Bit Stream โดยอัตโนมัติ

MemGen

สร้าง RAM,ROM ขนาดตามต้องการได้ภายใน XC4000

ส่วนของการทำ Design Implementation

XNFMerge

รวมเอา XNF File MAP File เข้าด้วยกันเป็น XNF ไฟล์เดียว

XNFDRC

เป็นการตรวจสอบ Design Rule เพื่อหัวข้อผิดพลาดในช่วงการออกแบบตอนแรก ๆ

XNFmap

ทำการ MAP Logic ที่อยู่ใน XNF File ไปยัง LCA element (CLB,IOB,TBUFS)

MAP2LCA

แปลง MAP File ไปเป็น LCA File เพื่อเป็น Input APR

APR (Automatic Place & Route)

ทำการ Place & Route Design ที่ทำบน XC2000,XC3000

PPR (Partition,Place & route)

การแยกส่วน Partition และ Plan & Route Design XC4000

MakeBits

สร้าง Configuration Bit Stream สำหรับ LCA

MakePROM

แปลง Configuration Bit Stream (BIT) ให้อยู่ในรูปของ File ที่สามารถ Download ลงสู่ PROM นอกจากนี้ยังรวมเอา Bit File หลายๆ ไฟล์มารวมกันเพื่อใช้ LCA ในลักษณะของ Daisy Chain

ส่วนของการทำ Design verification

Xdelay

เป็น Software ที่วิเคราะห์ static-timing ซึ่งรายงานข้อมูลฟังก์ชันเวลาของ Design ที่เราออกแบบและสามารถใช้เป็นตัววิเคราะห์ประสิทธิภาพของ Design โดยรวม

LCA2XNF

แปลง LCA File ไปเป็น XILINX Netlist format(XNF) สำหรับการเกิด Timing Simulation

BAX

สร้าง XNF File ซึ่งรวมเอา delay information ของ Design ใช้ชื่อ net ของ Original Design

XDE (XACT Design Editor)

XACT Design editor ให้เราสามารถแก้ไขวิเคราะห์ LCA Design ด้วยตนเอง (manually)

ส่วนของ Periphral/Hardware

XC3000 Design Demostration Board

ใช้สำหรับทดสอบการทำงานของ LCA ตระกูล XC2000,XC3000

XC4000 Design Demonstration Board

ใช้สำหรับทดลองการทำงานของ LCA ตระกูล XC4000

Xchecker

เป็น Cable และ Software สำหรับ Download,readblock แก้ไข Configuration data และดูสถานะของ logic state ภายใน XILINX FPGA

XPP (XILINX PROM Programmer)

เป็น Software มากับ DS112 Programmer ใช้สำหรับ serial PROM

วิธีทำโครงงาน

การออกแบบโครงงานนี้ ใช้หลักการออกแบบวงจรอิเลคโทรนิกส์แบบอัตโนมัติ (Electronic Design Automation, EDA) เนื่องจากเป็นแนวทางใหม่ที่ช่วยในการออกแบบวงจร ทำได้ง่ายและรวดเร็วมีประสิทธิภาพมากขึ้น ลดค่าใช้จ่ายและเวลาในการออกแบบโครงงานแต่ละชิ้นลงไปมาก หลังจากที่ได้กำหนดคุณสมบัติ (Specification) ของโครงงานแล้วก็ทำการเขียน VHDL Mode ขึ้นมาซึ่งลักษณะการเขียนดูได้จาก VHDL description for Synthesis เราสามารถใช้ VHDL อธิบายการทำงานของ Digital Circuit ในรูปแบบของ

- Combinatorial logic
  - Truth table
  - PLA table
  - Boolean Equation
- Sequential logic
- Finite state machine

วิธีการที่เลือกใช้ในการสร้าง Electronics Design ในโครงงานนี้เลือกใช้ VHDL Model ในแทนรูปแบบดังกล่าว โดยมีขั้นตอนดังนี้

1. เลือกวิธีการในการสร้าง Model โดยใช้ภาษา VHDL
2. แยกหน้าที่ในระบบออกเป็น Block หรือ Module ตามหน้าที่ หรือเขียนในรูปของ State Machine,Truth Table
3. สร้าง Model โดยใช้ภาษา VHDL ใช้ VHDL analyzer เป็นตัว Compiler
4. ทดสอบและจำลองการทำงาน VHDL Model โดยใช้ VIEWSIM
5. ให้ Synthesis Tools สร้างวงจร Schematic Gate Level จาก VHDL Model โดยใช้ VIEWSYNTHESIS(VHDLDES)
6. สร้างฮาร์ดแวร์ Prototype บน Field Programmable Gate Array (FPGA) ตระกูล XC40000จากบริษัท XILINX ใช้ XACT Development Software

ขั้นตอนหลักในการสร้างมี 4 หัวข้อดังนี้

1.Design Entry ใช้ Software VHDL Analyzer ของ VIEWLOGIC inc.

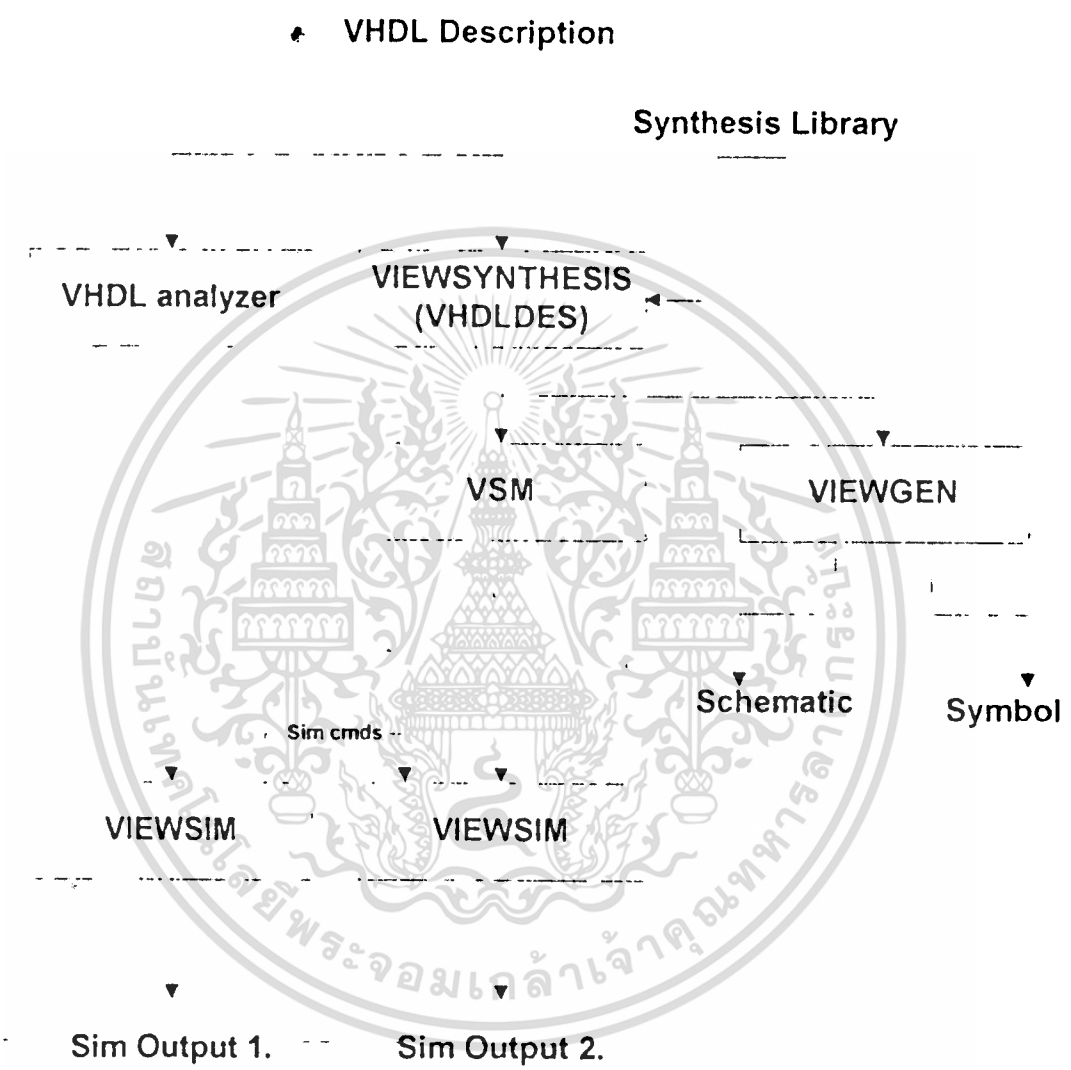
2.Design Simulation ใช้ Software VIEWSIM ของ VIEWLOGIC inc.

3.Design Synthesis ใช้ Software VIEWSYNTHESIS ของ VIEWLOGIC inc.

4.Design Implementation ใช้ Software XACT Development System ของ XILINX

อธิบายขั้นตอนการทำงานแต่ละขั้นตอน

1. DESIGN ENTRY



Flow Chart รูปที่ 1 แสดงการทำ Design Entry โดยใช้ VHDL

Flow Chart รูปที่ 1 แสดงขั้นตอนการทำ Design Entry จะเห็นว่า เริ่มแรกจะทำการเขียน VHDL Code ขึ้นมาก่อน จาก Specification ที่เรากำหนด จากนั้น จึงค่อยทำการตรวจสอบไวยากรณ์ โดยใช้ VHDL analyzer เป็นตัวคอมพายล์ (Compile) โดยลักษณะของการเขียนด้วยภาษา VHDL นี้จะต้องตามหลักของ VHDL Synthesis Guide เพื่อที่ VHDL Model ที่เขียนจะสามารถแปลงออกมาให้เป็นวงจร Schematic diagram ได้

2. DESIGN SIMULATION

VHDL Model ที่ผ่านการคอมพายล์แล้วจะต้องนำมาทำการ Simulation เพื่อตรวจสอบว่าทำงานได้ตามต้องการหรือไม่ซึ่งมีวิธีทำได้ 2 ทางคือ

ทางเลือกที่ 1 หลังจากทำการ Compile VHDL Model แล้วให้ใช้ VIEWSIM ทำการ Simulation สามารถเก็บไว้เป็นไฟล์ได้

ทางเลือกที่ 2 ถ้าต้องการทดสอบให้ลักษณะของ Schematic ให้ใช้ VIEWSYNTHESIS ทำการสังเคราะห์แปลง VHDL Model ให้กลายเป็นวงจร Schematic ก่อนโดยต้องมีการกำหนดด้วยว่าจะใช้ Library ของอุปกรณ์จากที่ใด (ในที่นี้ใช้ XC4000 ของบริษัท XILINX) เมื่อได้วงจร Schematic ของ Design แล้วก็ใช้ VSM ทำการแปลง Schematic ให้เป็น Netlist ซึ่งสามารถทำการ Simulate ได้ด้วย VIEWSIM

เมื่อ VSM ทำการแปลงแล้วจะได้ File (XXX.VSM) แล้งจึงทำการ Simulation โดยอาจจะเอา Command Sequence ที่ได้จากการ Simulation VHDL Model มาใช้ก็ได้เพื่อทำการเปรียบเทียบว่า ผลลัพธ์จะเป็นอย่างไร หลังจากขั้นตอนนี้ถ้าได้ผลลัพธ์ความต้องการแล้วก็เข้าสู่การทำงานขั้นตอนต่อไป

3. DESIGN SYNTHESIS

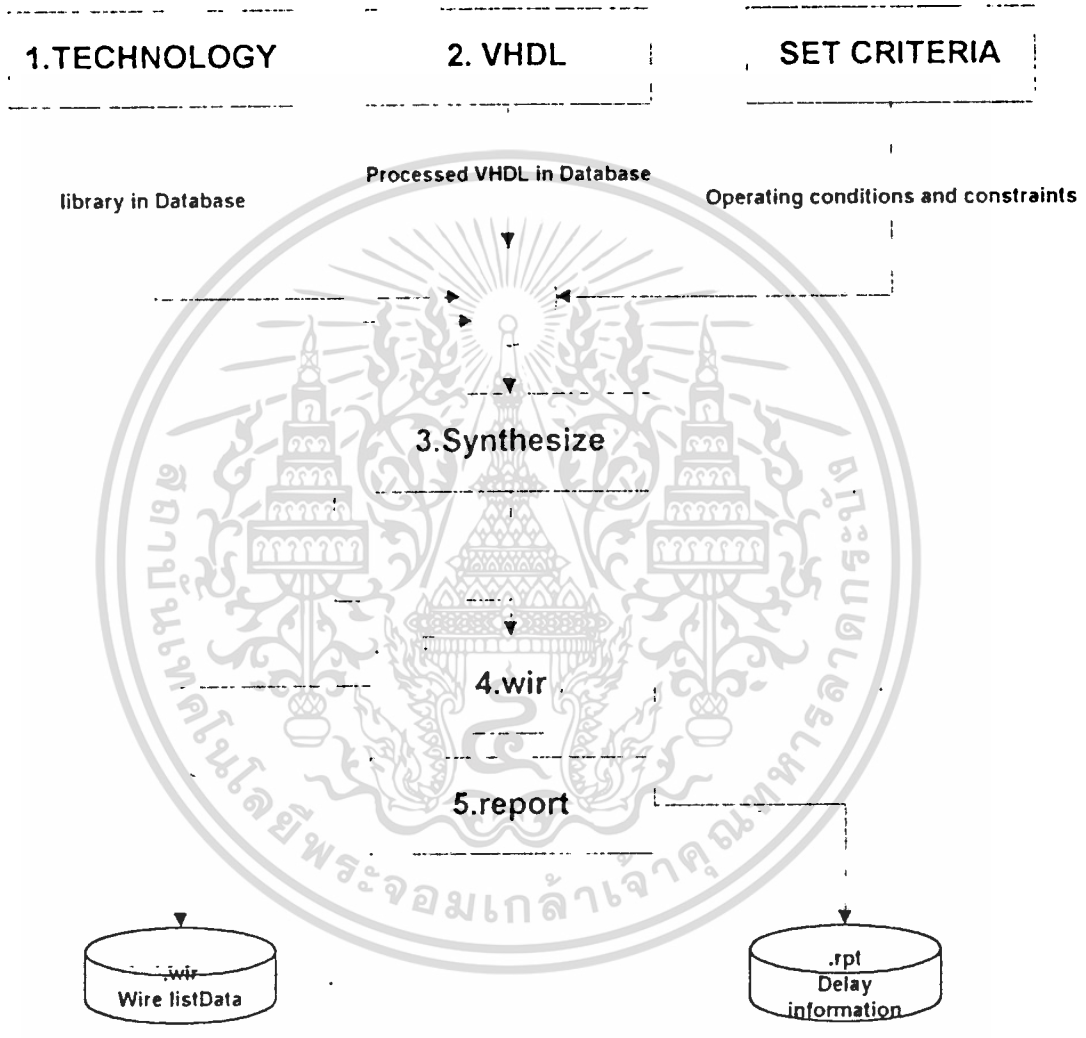
จาก Flow Chart รูปที่ 2 จะเห็นว่าก่อนที่จะทำการ Synthesis ได้นั้นจะต้องมี Input 3 อย่าง คือ

1. Technology ว่าต้องการจะสร้างวงจรด้วย component จากที่ใด ซึ่งในโครงงานนี้เลือกใช้ XC4000 ของ XILINX

2. VHDL code โดยที่ VIEWSYNTHESIS จะต้องอ่าน VHDL ที่ผ่านการ Compile แล้วเข้ามาใน Database ภายใน

3. การกำหนดข้อบังคับในการทำ Synthesis ต่างๆ

เมื่อกำหนด 3 อย่างตามต้องการแล้ว Software ที่จะเริ่มทำการ Synthesis ผลลัพธ์ที่ได้จะอยู่ในรูปของ Netlist จากนั้นให้ใช้คำสั่ง wir เพื่อสร้าง wirelist ของวงจรและใช้คำสั่ง rep เพื่อสร้าง Report รายงานเหตุการณ์ต่างๆ ที่เกิดขึ้น output จากการทำ Synthesis ตรงนี้คือ XXX.1 คือ wirelist และ XXXX.rpt คือ Report File หลังจากนี้สามารถใช้ VIEWGEN ทำการสร้าง Schematic ของวงจรเพื่อจะได้เรียกขึ้นมาดูได้ใน VIEWDRAW



รูปที่ 2 แสดงการทำ Design Synthesis

4. DESIGN IMPLEMENTATION

จากรูปแสดงให้เห็นขบวนการขั้นตอนการสร้าง Design ลงบน FPGA หรือ LCA ของ XILINX ซึ่งจะแบ่งการทำงานออกเป็น 3 ส่วนใหญ่ๆ ได้แก่

1. Design Entry

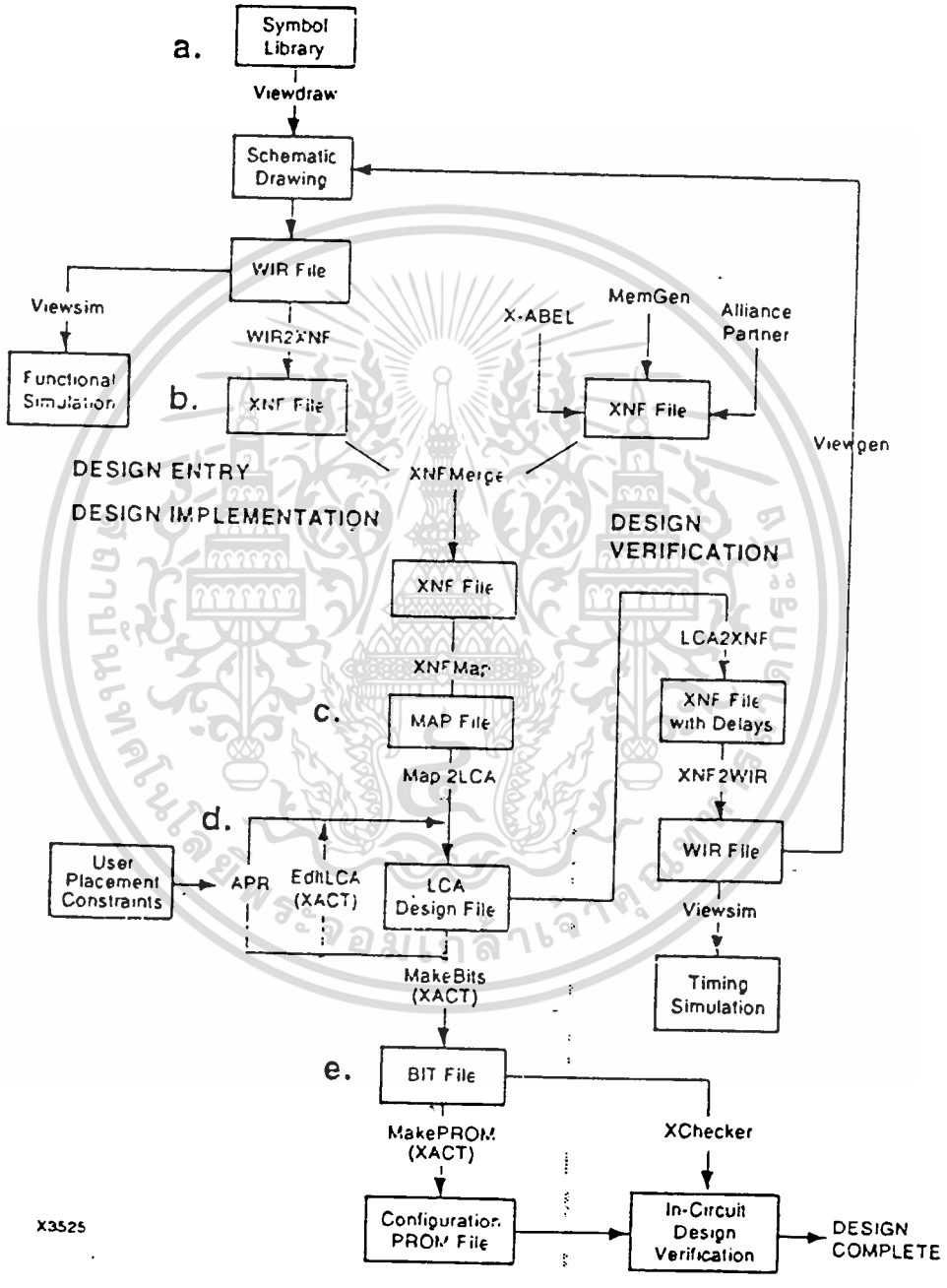
2. Design Implementation

3. Design verification

เริ่มต้นจากจุด a. เป็นการสร้าง Design จาก symbol ที่อยู่ใน Library ซึ่งเราไม่จำเป็นจะต้องทำ เนื่องจากว่าเราได้ Schematic จากการ Synthesis เรียบร้อยแล้วแต่บางครั้งจำเป็นจะต้องใช้ VIEWDRAW ทำการเพิ่มเติมส่วนต่างๆ เล็กๆน้อยๆ เข้าไปเพื่อให้ Schematic สมบูรณ์ยิ่งขึ้น จากนั้นจะได้ Output เป็น WIR File ตรงนี้สามารถทำ Simulation โดยใช้ VIEWSIM ก็ได้หรือไม่ต้องการ Simulation ก็สามารถใช้ WIR2XNF ทำการแปลง Viewlogic File ให้อยู่ใน Format ของ XILINX เพื่อเข้าสู่ XNFMerge ต่อไป

จุด b แต่ก่อนที่จะเข้า XMFMerge ถ้า Design ของเรามีการใช้ Memory ภายใน LCA (XC4000) หรือมี Design อื่นๆ เราสามารถนำมารวมกันได้ ณ. จุดนี้ ผ่านขั้นตอนนี้ไป File ของ Design ทั้งหมดก็จะถูกแปลงให้อยู่ในรูปของ XNF File format เท่านั้น

ที่จุด c. XNF File ของ Design จะถูกส่งเข้ามายัง XNFmap เพื่อทำการแปลงให้อยู่ในรูปของ LCA Design File ตรงนี้จะมีการทำ ADR (Automatic place & route) เพื่อที่จะทำการเชื่อมต่อ CLB,IOB ต่างๆ ภายใน LCA เข้าด้วยกันให้ทำงานเป็นวงจรที่เราต้องการ โดยก่อนทำ APR User สามารถกำหนดข้อบังคับต่างๆ ในการทำงานให้ได้ เสร็จแล้วให้ MakeBits เพื่อที่จะทำการแปลงการเชื่อมต่อภายใน LCA ให้เป็นลักษณะของ Bit File เมื่อได้ Bit File แล้วเราสามารถนำเอาไปพัฒนาให้หลายลักษณะตามต้องการ เช่น นำ bit File ส่งผ่านในรูป slave mode ไปยัง Communication Port เพื่อที่จะเข้าไป Config LCA ที่อยู่บน Demoboard หรือ board ที่สร้างขึ้นมาเพื่อให้ LCA ทำงาน หรืออีกทางเลือกก็นำเอา bit File มาเป็น Input แก่ MakePROM เพื่อที่จะทำการแปลง bit File ให้อยู่ในรูปของ data ที่จะบรรจุลง EPROM สุดท้ายจะได้ Configuration EPROM File จากนั้น LCA ก็จะสามารถทำงานได้จริงในวงจรที่สร้างขึ้นในโครงงานนี้ได้สร้าง Demoboard XC4000 ขึ้นมา 1 board เพื่อทดสอบการใช้งานว่าทำได้จริงขั้นตอนนี้ Design จะเสร็จสมบูรณ์ แต่ถ้าต้องการทดสอบการทำงาน simulation หลังจากที่ config ลงไปแล้วก็สามารถกระทำได้ โดยที่จุด d เมื่อเราได้ LCA Design File เราก็นำเอา LCA มาแปลงให้เป็น XNF โดยใช้ LCA2XNF เราก็จะได้ XNF File เพื่อใช้ VIEWSIM ทำการ Simulate การทำงานเมื่อมี Delay ของ Component ต่างๆ ใน LCA ด้วยเพื่อดูว่าการทำงานยังคงเหมือนเดิมหรือไม่อย่างไร



รูปที่ 3 LCA Design Flow

บทที่ 7
บทสรุปและวิจารณ์

บทสรุป

การออกแบบวงจรรวมปัจจุบันทำได้รวดเร็วและมีประสิทธิภาพมาก เนื่องจากการพัฒนาเครื่องมือ (Tools) ต่าง ๆ ทั้งทางด้าน Hardware และ Software ระบบที่ช่วยออกแบบทางอิเลคทรอนิกส์ (electronic design automation, EDA) ก็ได้พัฒนาขึ้นมาจนทำให้การออกแบบสามารถทำได้อย่างรวดเร็ว

VHDL เป็นภาษาที่ใช้เป็นภาษาเขียนอธิบายการทำงานของวงจรใด ๆ แทนการสร้างวงจรขึ้นมาจริง ๆ ใช้ประโยชน์ในการทำ Simulation เพื่อนำไปสร้างวงจรจริง ๆ ได้ ปัจจุบันจะเห็นว่ามีวงจรรวม ASIC ใหม่ ๆ เป็นผลิตภัณฑ์ออกมามากมาย เนื่องมาจากเทคโนโลยีในการออกแบบวงจรรวมได้พัฒนาขึ้นไปนั่นเอง การศึกษาโครงงานนี้ทำให้สามารถนำความรู้ที่ได้ไปใช้ประโยชน์ได้ในวงการอุตสาหกรรมจริง ๆ และคิดว่าในอนาคตคงจะมีการใช้เครื่องมือที่มีอยู่ออกแบบและพัฒนาสิ่งประดิษฐ์ใหม่ ๆ สู่วงการอิเลคทรอนิกส์ต่อไป

บทวิจารณ์

ระบบ Electronic Design Automation เป็นการออกแบบที่ใช้คอมพิวเตอร์ทั้ง Hardware และ Software ช่วยในการออกแบบวงจรอิเลคทรอนิกส์ต่าง ๆ เป็นเทคโนโลยีแบบใหม่ทันสมัยมาก มีมากมายหลาย Platform ทั้ง PC และ Workstation ซอฟท์แวร์ที่ใช้บนแต่ละ Platform ก็แตกต่างกันไป การเรียนรู้แต่ละรูปแบบใช้เวลานานมากเนื่องจากไม่คุ้นเคย ทำให้การทำงานโครงงานเสียเวลาไปมากกับการศึกษา Tools ต่าง ๆ เหล่านี้บางครั้งต้องเสียเวลากับ การแก้ปัญหาที่เกิดขึ้นทำให้การทำ Application ออกมาไม่ดีเท่าที่ควร ซอฟแวร์ที่มีใช้อยู่ยังไม่มีตัวใดที่สามารถทำงานได้ตลอดขบวนการพัฒนาจะต้องใช้หลาย ๆ ตัวมาช่วยพัฒนา บางครั้งเกิดความเข้ากันไม่ได้ระหว่าง Software ต่าง ๆ เหล่านั้น ๆ ทำให้เกิดปัญหาในการพัฒนาต่อ ฉะนั้นในการออกแบบควรจะศึกษา Tools ที่จะใช้ให้ดี ศึกษาความเป็นไปได้ในการใช้ Software Tools ต่าง ๆ ให้ดีเสียก่อนที่จะเริ่มลงมือทำการออกแบบ

กิตติกรรมประกาศ

ผู้จัดทำขอขอบคุณ อาจารย์ บรรจง ปิยธำรง เป็นอย่างยิ่งที่ท่านได้ให้ความรู้พื้นฐานการออกแบบโดยใช้ CAD/CAE ให้แนวทางในการทำงาน จัดหาเครื่องมือต่าง ๆทั้งทางด้าน Hardware และ Software ตลอดจนให้คำแนะนำและช่วยแก้ปัญหาต่าง ๆที่เกิดขึ้นตลอดการทำโครงงาน

ขอขอบคุณศูนย์อิเลคทรอนิคส์และคอมพิวเตอร์แห่งชาติ(NECTEC)ที่ช่วยเอื้อเฟื้อซอฟแวร์ VIEWLOGIC ที่ใช้ในการทำโครงงาน

ขอขอบคุณเจ้าหน้าที่ศูนย์วิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (CAD/CAM Center) ทุกท่านที่ช่วยความสะดวกในการใช้เครื่อง Workstation ทั้งในเวลาและนอกเวลาราชการ,การให้ยืมคู่มือและเอกสาร,ช่วยแก้ปัญหาการใช้ Hardware และ Software ในศูนย์วิจัย ตลอดจนจัดให้มีการฝึกอบรมการใช้ซอฟแวร์ Mentor Graphics

ขอขอบคุณห้องปฏิบัติการไมโครคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้า มหาวิทยาลัยเกษตรศาสตร์ ที่จัดให้มีการอบรมซอฟแวร์ Workview Plus for Windows ของบริษัท Viewlogic และซอฟแวร์ XACT Development Tools ของ XILINX

ขอขอบคุณบริษัท Antech Communications ที่ช่วยเอื้อเฟื้อให้ยืมใช้ Software และ Library ต่าง ๆ ตลอดจนเป็นธุระในการจัดหา License ของ Software ที่จำเป็นในการทำโครงงาน

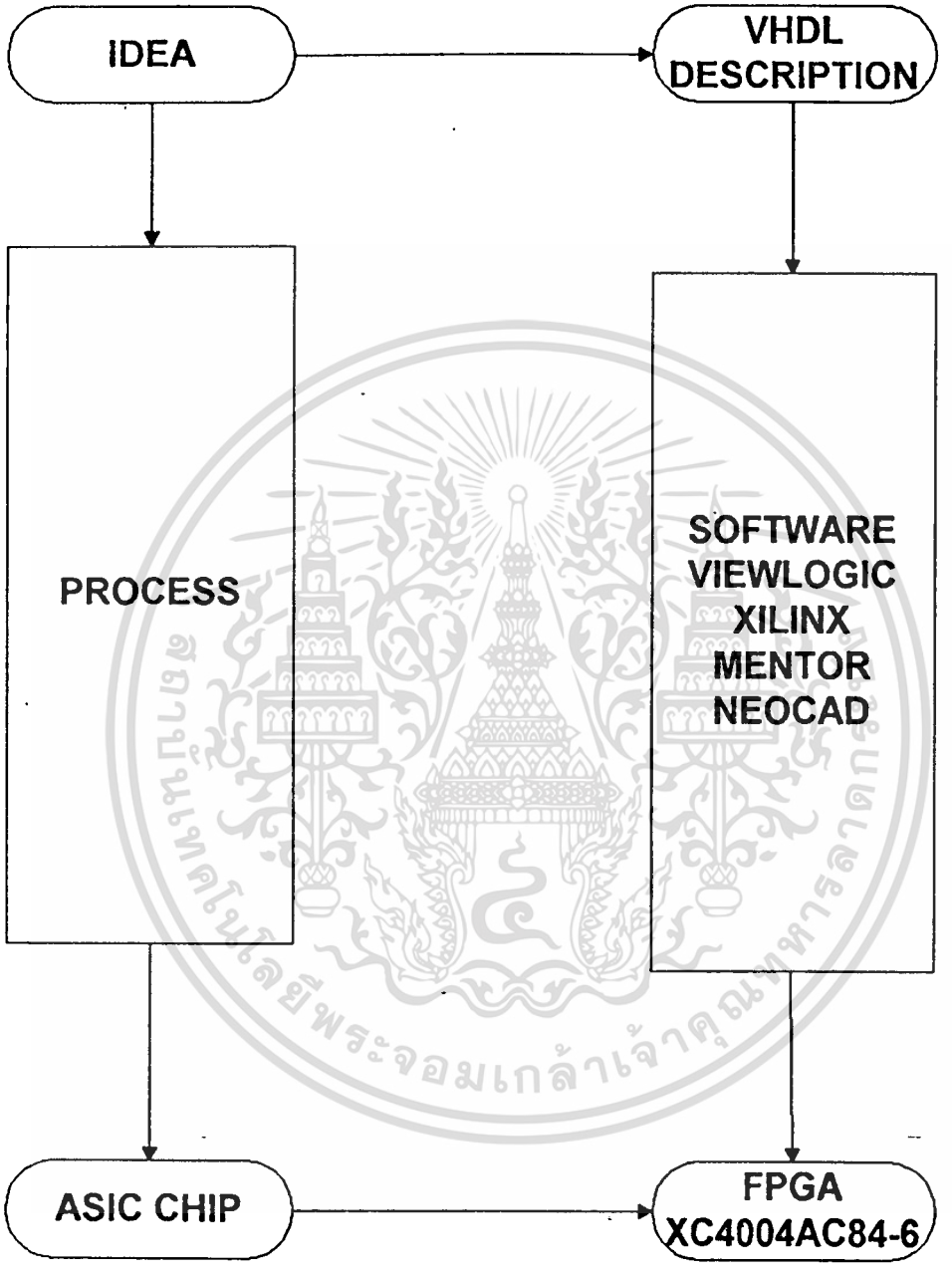
สุดท้ายที่จะลืมเสียไม่ได้คือ พ่อแม่ ที่ได้ให้กำเนิดเลี้ยงดูอบรมให้การศึกษามาเป็นอย่างดี,เพื่อน ๆที่ช่วยเป็นที่ปรึกษาและให้กำลังใจในการทำงานตลอดมา

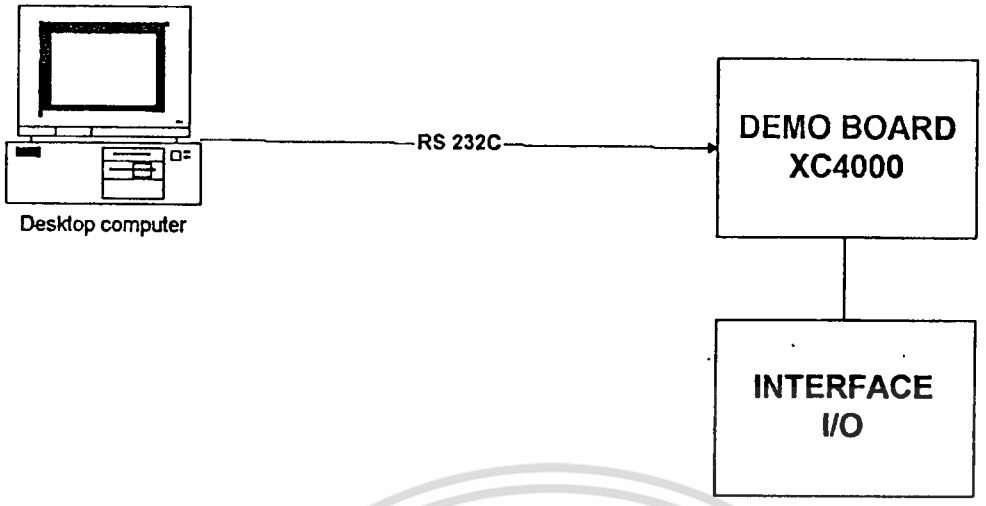
ผู้จัดทำ

บรรณานุกรม

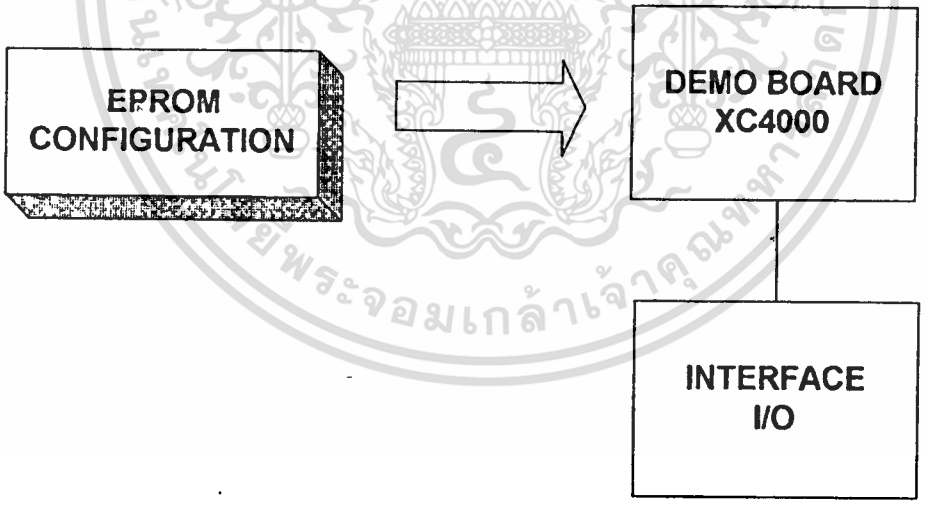
1. Jayaram Bhasker,"VHDL Primer",Printice Hall,First Edition, 253 p,1992.
2. Zainalabedin Navabi,"VHDL analysis and Modelling Digital System",Mcgraw-Hill Internation Editions,375 p.,1993.
3. Mentor Graphics,"An Introduction to Modelling with VHDL.",Mentor Graphics,1992.
4. Mentor Graphics, "Introduction to VHDL.",Mentor Graphics,1992.
5. Mentor Graphics,"Mentor Graphics VHDL reference Manual,"Mentor Graphics,1992.
6. Viewlogic,"Using Workview Plus for Windows.",Viewlogic,1993.
7. Viewlogic,"ViewDraw Reference Manual."Viewlogic,1993.
8. Viewlogic,"Viewtrace User's Guide."Viewlogic,1993.
9. Viewlogic,"Viewsim Reference Manual."Viewlogic,1993.
10. Viewlogic,"VHDL User's Guide.",Viewlogic,1993.
11. Viewlogic,"VHDL Reference Manual.",Viewlogic,1993.
12. XILINX,"XACT Development System Reference Guide",XILINX,1994
13. XILINX,"XACT Development System Library Guide",XILINX,1994
14. XILINX,"XACT Viewlogic Interface User Guide",XILINX,1994
15. XILINX,"The Programmable Logic Data Book",XILINX,1994

ภาคผนวก

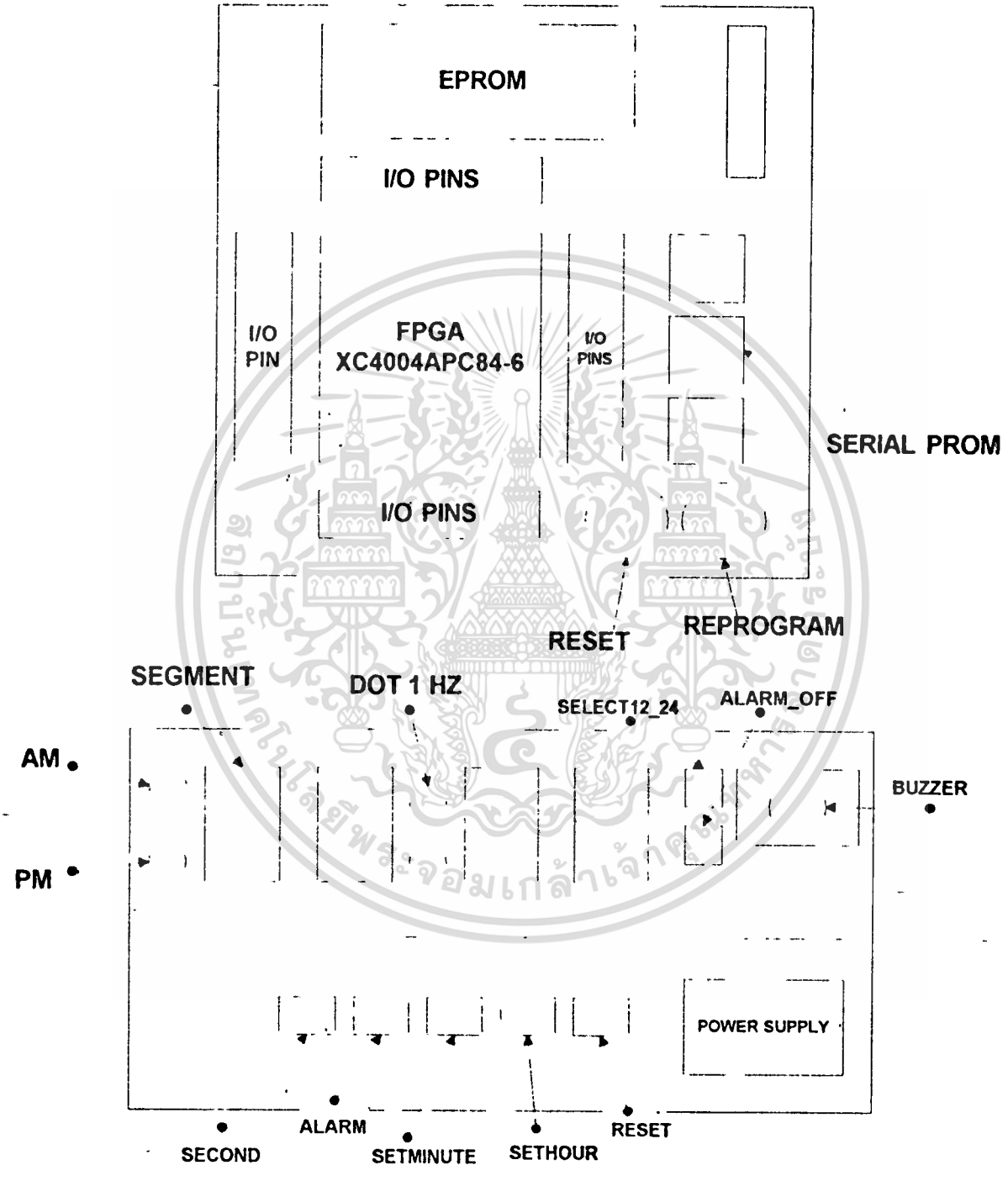




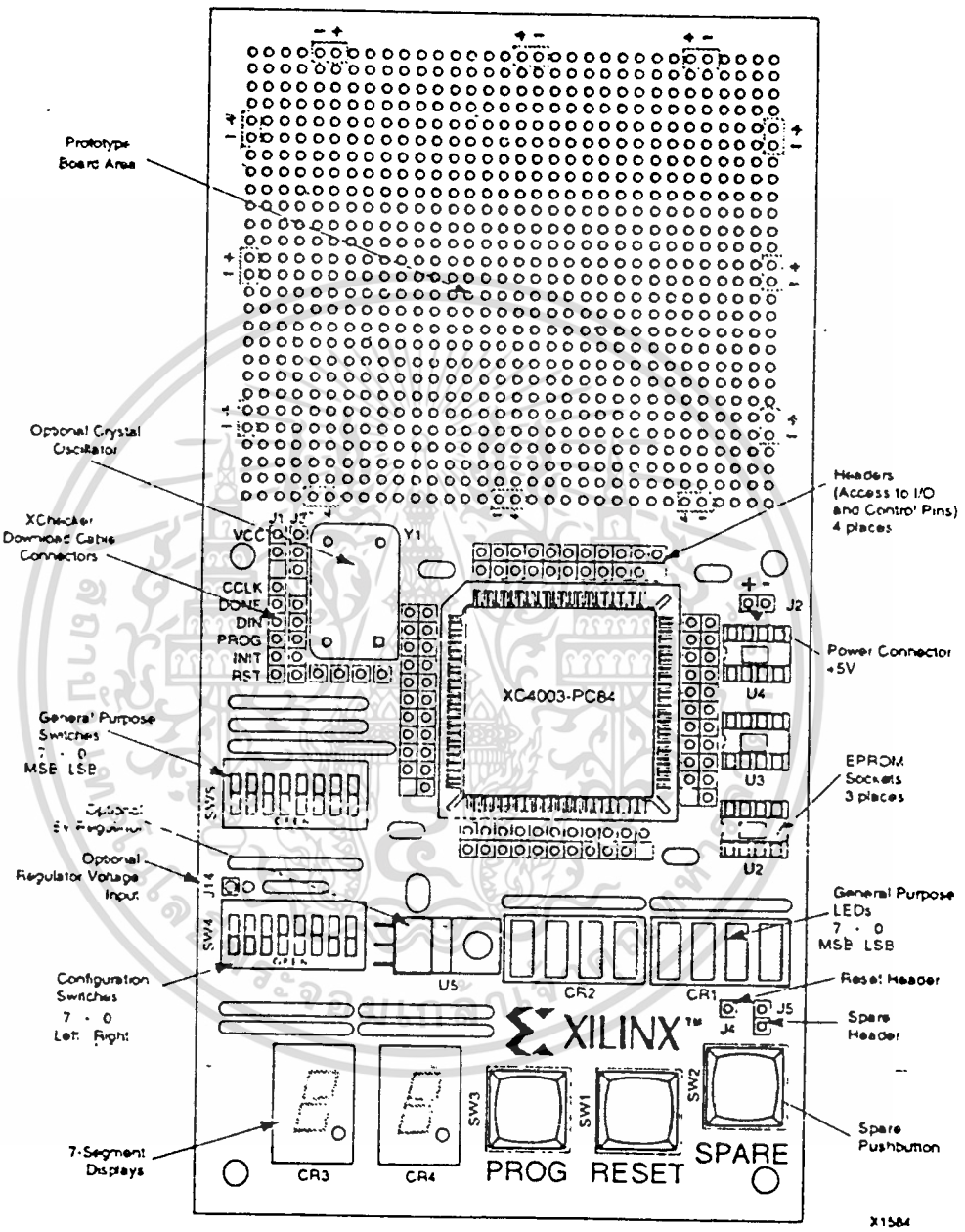
XCHECKER CABLE



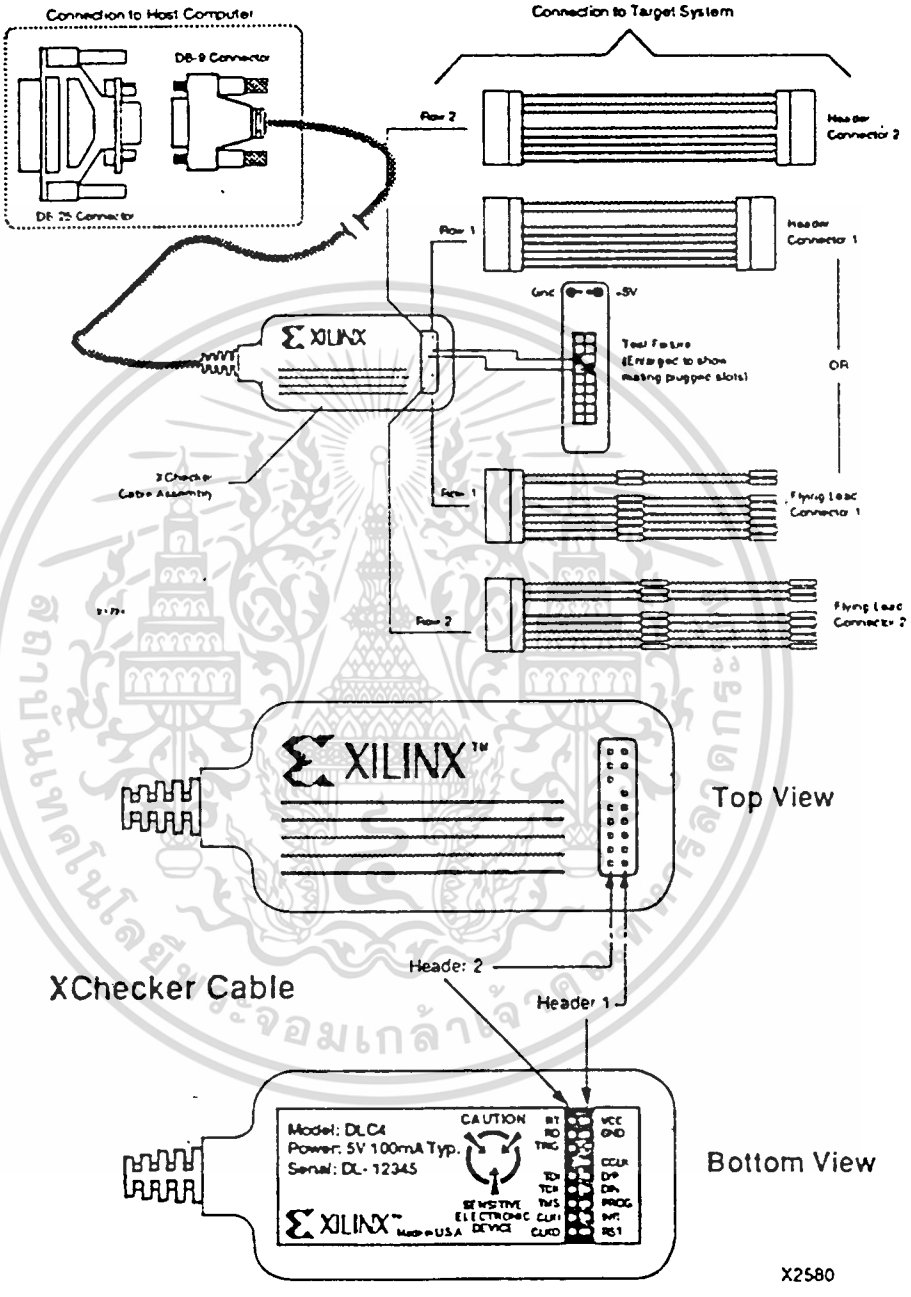
DEMO BOARD



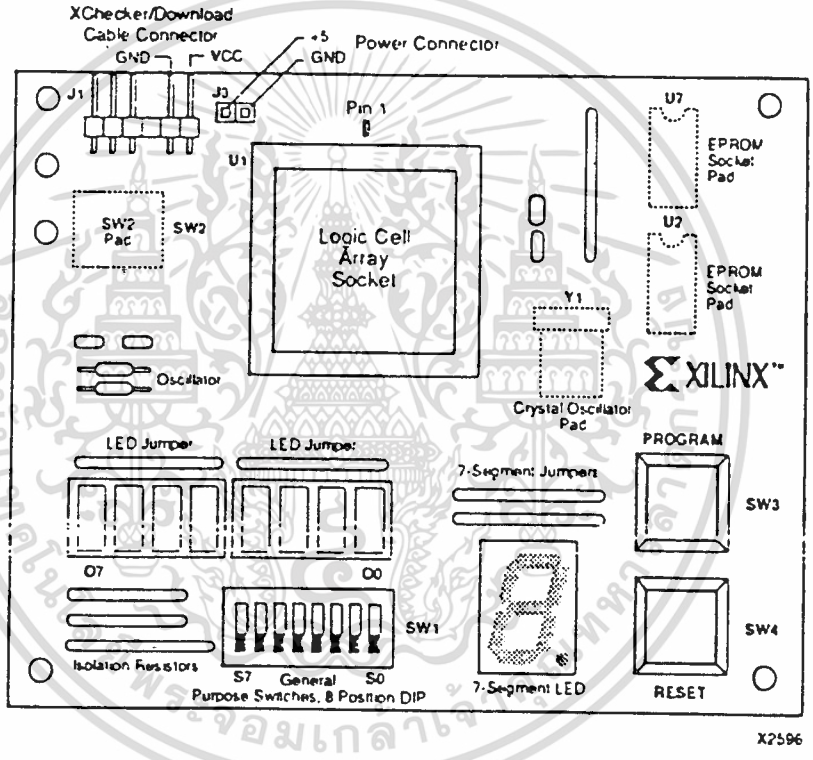
INTERFACE I/O



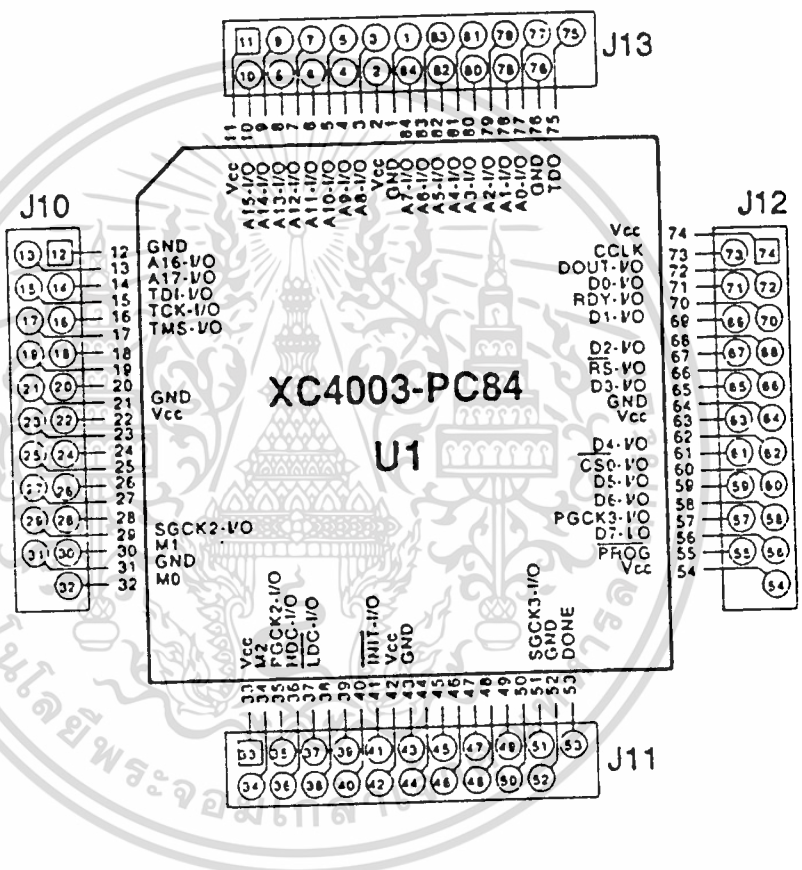
Demonstration Board Layout.



XChecker Hardware and Accessories



Demonstration Board Layout



XC3000
Logic Cell Array Family

Product Specification

Features

- Industry-leading FPGA family with five device types
  - Logic densities from 1,000 to 6,000 gates
  - Up to 144 user-definable I/Os
- Guaranteed 70- to 125-MHz toggle rates, 9 to 5.5 ns logic delays
- Advanced CMOS static memory technology
  - Low quiescent and active power consumption
- XC3000-specific features
  - Ultra-low current option in Power-Down mode
    4-mA output sink and source current
  - Broad range of package options includes plastic and ceramic quad flat packs, plastic leaded chip carriers and pin grid arrays
  - 100% bitstream compatible with the XC3100 family
  - Commercial, industrial, military, "high rel", and MIL-STD-883 Class B grade devices
  - Easy migration to XC3300 series of HardWire mask-programmed devices for high-volume production

Description

XC3000 is the original family of devices in the XC3000 class of Field Programmable Gate Array (FPGA) architectures. The XC3000 family has a proven track record in addressing a wide range of design applications, including general logic replacement and sub-systems integration. For a thorough description of the XC3000 architecture see the preceding pages of this data book.

The XC3000 Family covers a range of nominal device densities from 2,000 to 9,000 gates, practically achievable densities from 1,000 to 6,000 gates. Device speeds, described in terms of maximum guaranteed toggle frequencies, range from 70 to 125 MHz. The performance of a completed design depends upon placement and routing implementation, so, like with any gate array, the final verification of device utilization and performance can only be known after the design has been placed and routed.

Device	CLBs	Array	User I/Os Max	Flip-Flops	Horizontal Longlines	Configuration Data Bits
XC3020	64	8 x 8	64	256	16	14,779
XC3030	100	10 x 10	80	360	20	22,176
XC3042	144	12 x 12	96	480	24	30,784
XC3064	224	16 x 14	120	688	28	46,064
XC3090	320	16 x 20	144	928	40	64,160

Xilinx maintains test specifications for each product as controlled documents. To insure the use of the most recently released device performance parameters, please request a copy of the current test-specification revision.

Absolute Maximum Ratings

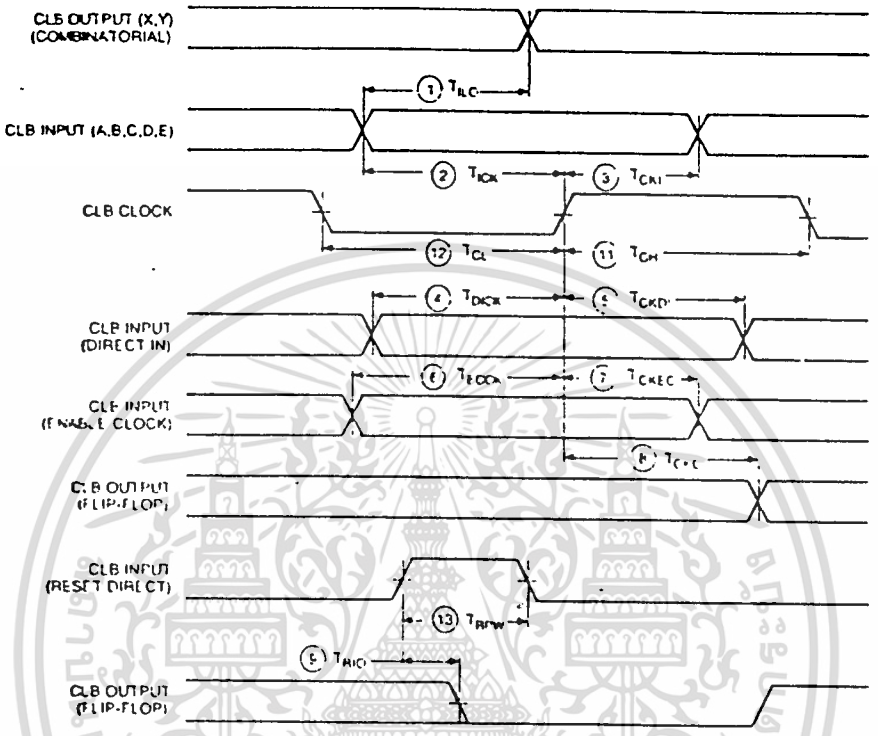
Symbol	Description		Units
V_{CC}	Supply voltage relative to GND	−0.5 to +7.0	V
V_{IN}	Input voltage with respect to GND	−0.5 to V_{CC} +0.5	V
V_{TS}	Voltage applied to 3-state output	−0.5 to VCC +0.5	V
T_{STG}	Storage temperature (ambient)	−65 to +150	°C
T_{SOL}	Maximum soldering temperature (10 s @ 1/16 in.)	+260	°C
T_J	Junction temperature plastic	+125	°C
	Junction temperature ceramic	+150	°C

Note: 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 Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

Operating Conditions

Symbol	Description	Min	Max	Units
V_{CC}	Supply voltage relative to GND Commercial 0°C to +70°C	4.75	5.25	V
	Supply voltage relative to GND Industrial −40°C to +85°C	4.5	5.5	V
V_{IH}	High-level input voltage — TTL configuration	2.0	V_{CC}	V
V_{IL}	Low-level input voltage — TTL configuration	0	0.8	V
V_{IHC}	High-level input voltage — CMOS configuration	70%	100%	V_{CC}
V_{ILC}	Low-level input voltage — CMOS configuration	0	20%	V_{CC}
T_{IN}	Input signal transition time		250	ns

CLB Switching Characteristic Guidelines



Buffer (Internal) Switching Characteristic Guidelines

Description	Speed Grade / Symbol	-70 Max	-100 Max	-125 Max	Units
Global and Alternate Clock Distribution*					
Either: Normal IOB input pad through clock buffer to any CLB or IOB clock input	T_{PID}	8.0	7.5	7.0	ns
Or: Fast (CMOS only) input pad through clock buffer to any CLB or IOB clock input	T_{PIDC}	6.5	6.0	5.7	ns
TBUF driving a Horizontal Longline (L.L.)*					
I to L.L. while T is Low (buffer active)	T_{IO}	5.0	4.7	4.5	ns
T↓ to L.L. active and valid with single pull-up resistor	T_{ON}	11.0	10.0	9.0	ns
T↓ to L.L. active and valid with pair of pull-up resistors	T_{ON}	12.0	11.0	10.0	ns
T↑ to L.L. High with single pull-up resistor	T_{PUS}	24.0	22.0	17.0	ns
T↑ to L.L. High with pair of pull-up resistors	T_{PUF}	17.0	15.0	12.0	ns
BIDI					
Bidirectional buffer delay	T_{BIDI}	2.0	1.8	1.7	ns

* Timing is based on the XC3042, for other devices see XACT timing calculator.

CLB Switching Characteristic Guidelines (continued)

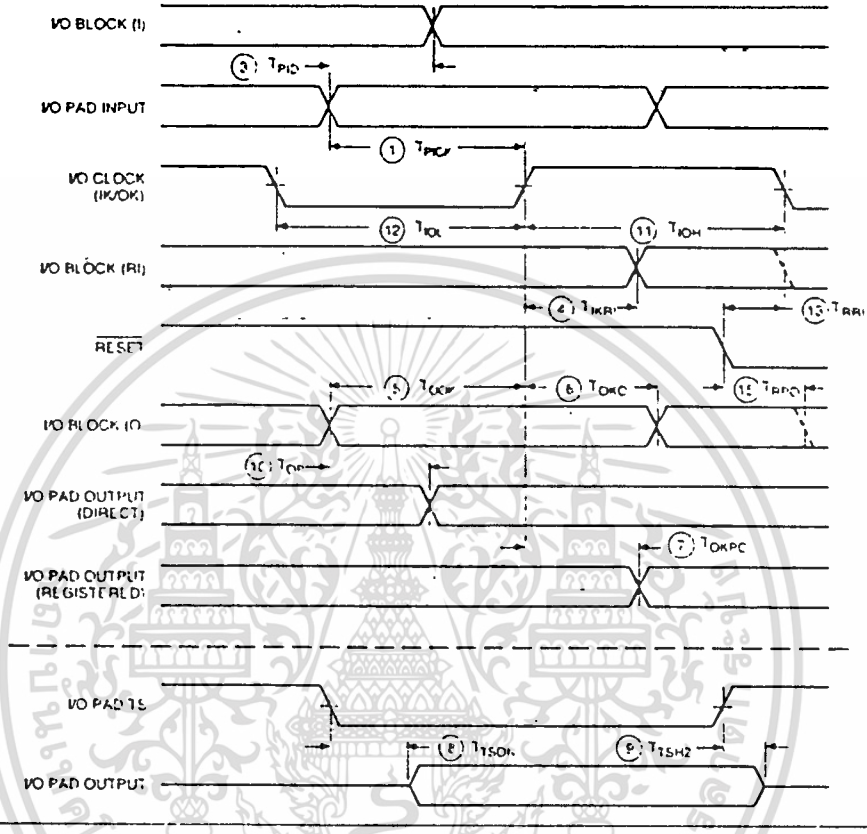
Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

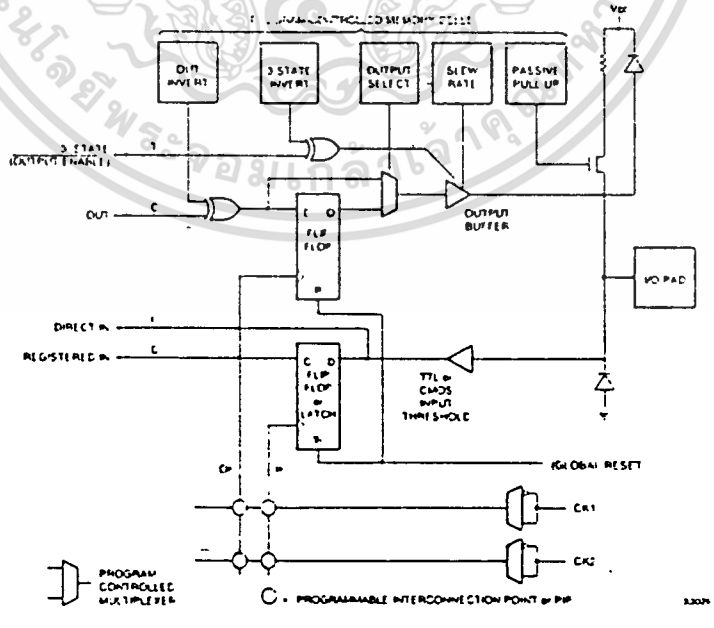
Speed Grade			-70		-100		-125		
Description	Symbol		Min	Max	Min	Max	Min	Max	Units
Combinatorial Delay									
Logic Variables A, B, C D, E to outputs X or Y	1	T_{ILO}		9.0		7.0		5.5	ns
Sequential delay									
Clock k to outputs X or Y	8	T_{CKO}		6.0		5.0		4.5	ns
Clock k to outputs X or Y when Q is returned through function generators F or G to drive X or Y		T_{QLO}		13.0		10.0		8.0	ns
Set-up time before clock K									
Logic Variables A, B, C, D, E	2	T_{ICK}	8.0		7.0		5.5		ns
Data In DI	4	T_{DICK}	5.0		4.0		3.0		ns
Enable Clock EC	6	T_{ECCK}	7.0		5.0		4.5		ns
Reset Direct inactive RD			1.0		1.0		1.0		ns
Hold Time after clock K									
Logic Variables A, B, C, D, E	3	T_{CKI}	0		0		0		ns
Data In DI	5	T_{CKDI}	4.0		2.0		1.5		ns
Enable Clock EC	7	T_{CKEC}	0		0		0		ns
Clock									
Clock High time	11	T_{CH}	5.0		4.0		3.0		ns
Clock Low time	12	T_{CL}	5.0		4.0		3.0		ns
Max flip-flop toggle rate		F_{CLK}	70		100		125		MHz
Reset Direct (RD)									
RD width	13	T_{RPW}	8.0		7.0		6.0		ns
delay from rd to outputs X or Y	9	T_{RIO}		8.0		7.0		6.0	ns
Global Reset (RESET Pad)*									
RESET width (Low)		T_{MRW}	25.0		21.0		20.0		ns
delay from RESET pad to outputs X or Y		T_{MRO}		23.0		19.0		17.0	ns

*Timing is based on the XC3042, for other devices see XACT timing calculator.

Note The CLB K to Q output delay (T_{CKO}, #8) of any CLB, plus the shortest possible interconnect delay, is always longer than the Data In hold time requirement (T_{CKDI}, #5) of any CLB on the same die.

IOB Switching Characteristic Guidelines





IOB 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. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Description	Speed Grade	Symbol	-70 Min	-70 Max	-100 Min	-100 Max	-125 Min	-125 Max	Units
Propagation Delays (Input)									
Pad to Direct In (I)	3	T_{PID}		6		4		3	ns
Pad to Registered In (Q) with latch transparent		T_{PTG}		21		17		16	ns
Clock (IK) to Registered In (Q)	4	T_{IKRI}		5.5		4		3	ns
Set-up Time (Input)									
Pad to Clock (IK) set-up time	1	T_{PICK}	20		17		16		ns
Propagation Delays (Output)									
Clock (OK) to Pad (fast)	7	T_{OKPO}		13		10		9	ns
same (slew rate limited)	7	T_{OKPO}		33		27		24	ns
Output (O) to Pad (fast)	10	T_{OPF}		9		6		5	ns
same (slew-rate limited)	10	T_{OPS}		29		23		20	ns
3-state to Pad begin hi-Z (fast)	9	T_{TSHZ}		8		8		7	ns
same (slew-rate limited)	9	T_{TSHZ}		28		25		24	ns
3-state to Pad active and valid (fast)	8	T_{TSON}		14		12		11	ns
same (slew-rate limited)	8	T_{TSON}		34		29		27	ns
Set-up and Hold Times (Output)									
Output (O) to clock (OK) set-up time	5	T_{OOK}	10		9		8		ns
Output (O) to clock (OK) hold time	6	T_{OKO}	0		0		0		ns
Clock									
Clock High time	11	T_{IOH}	5		4		3		ns
Clock Low time	12	T_{IOL}	5		4		3		ns
Max. flip-flop toggle rate		F_{CLK}	70		100		125		MHz
Global Reset Delays (based on XC3042)									
RESET Pad to Registered In (Q)	13	T_{RRI}		25		24		23	ns
RESET Pad to output pad (fast)	15	T_{RPO}		35		33		29	ns
(slew-rate limited)	15	T_{RPO}		53		45		42	ns

Notes: 1 Timing is measured at pin threshold, with 50 pF external capacitive loads (incl. test fixture). For larger capacitive loads, see XAPP 024. Typical slew rate limited output rise/fall times are approximately four times longer.

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

3 Input pad set-up time is specified with respect to the internal clock (IK). In order to calculate system set-up time, subtract clock delay (pad to IK) from the input pad set-up time value. Input pad holdtime with respect to the internal clock (IK) is negative. This means that pad level changes immediately before the internal clock edge (IK) will not be recognized

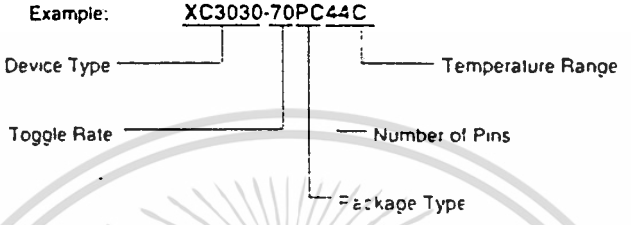
For a detailed description of the device architecture, see pages 2-105 through 2-123.

For a detailed description of the configuration modes and their timing, see pages 2-124 through 2-132.

For detailed lists of package pin-outs, see pages 2-140 through 2-150.

For package physical dimensions and thermal data, see Section 4.

Ordering Information



Component Availability

Pins	44	64	68	84		100				132	144	160	164	175		176	208	223
Type	PLASTIC PLCC	PLASTIC VQFP	PLASTIC PLCC	PLASTIC PLCC	CERAMIC PGA	PLASTIC PQFP	PLASTIC TQFP	PLASTIC VQFP	TQFP METRIC (0.4 mm) CQFP	PLASTIC PGA	CERAMIC PGA	PLASTIC TQFP	PLASTIC PQFP	TQFP METRIC (0.4 mm) CQFP	PLASTIC PGA	CERAMIC PGA	PLASTIC TQFP	PLASTIC PQFP	CERAMIC PGA
Code	PC44	VQ64	PC68	PC84	PG84	PC100	TQ100	VQ100	CB100	PP132	PG132	TQ144	PQ160	CB164	PP175	PG175	TQ176	PQ208	PG223
XC3020 -50					M				M										
XC3020 -70			C	C, I	C, M, B	C, I			C, M, B										
XC3020 -100			C	C, I	C, M, B	C, I			C, M, B										
XC3020 -125			C	C	C	C													
XC3030 -50					M														
XC3030 -70	C, I		C	C	C, M	C, I	C												
XC3030 -100	C, I		C	C, I	C, M	C, I	C												
XC3030 -125	C		C	C	C	C	C												
XC3042 -50					M				M		M								
XC3042 -70				C, I	C, M, B	C, I	C		C, M, B		C, M, B								
XC3042 -100				C, I	C, M, B	C, I	C		C, M, B		C, M, B								
XC3042 -125				C	C	C	C				C								
XC3064 -50											M								
XC3064 -70				C, I							C, M, B		C, I						
XC3064 -100				C							C, M, B		C, I						
XC3064 -125				C							C		C						
XC3090 -50														MP		MP			
XC3090 -70				C, I									C, I	C, M, B	C, I	C, M, B		C, I	
XC3090 -100				C, I									C, I	C, M, B	C, I	C, M, B		C, I	
XC3090 -125				C									C		C		C		

C = Commercial = 0° to +70° C I = Industrial = -40° to +85° C M = Mil Temp = -55° to +125° C B = MIL-STD-883C Class B

Parentheses indicate future product plans

XC4000
Logic Cell Array Family

Product Specifications

Features

- Third Generation Field-Programmable Gate Arrays
  - Abundant flip-flops
  - Flexible function generators
  - On-chip ultra-fast RAM
  - Dedicated high-speed carry-propagation circuit
  - Wide edge decoders (four per edge)
  - Hierarchy of interconnect lines
  - Internal 3-state bus capability
  - Eight global low-skew clock or signal distribution network
- Flexible Array Architecture
  - Programmable logic blocks and I/O blocks
  - Programmable interconnects and wide decoders
- Sub-micron CMOS Process
  - High-speed logic and interconnect
  - Low power consumption
- Systems-Oriented Features
  - IEEE 1149.1-compatible boundary-scan logic support
  - Programmable output slew rate (2 modes)
  - Programmable input pull-up or pull-down resistors
  - 12-mA sink current per output
  - 24-mA sink current per output pair
- Configured by Loading Binary File
  - Unlimited reprogrammability
  - Six programming modes
- XACT Development System runs on '386/'486-type PC, NEC PC, Apollo, Sun-4, and Hewlett-Packard 700 series
  - Interfaces to popular design environments like Viewlogic, Mentor Graphics and OrCAD
  - Fully automatic partitioning, placement and routing
  - Interactive design editor for design optimization
  - 288 macros, 34 hard macros, RAM/ROM compiler

Description

The XC4000 family of Field-Programmable Gate Arrays (FPGAs) provides the benefits of custom CMOS VLSI, while avoiding the initial cost, time delay, and inherent risk of a conventional masked gate array.

The XC4000 family provides 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).

XC4000 devices have generous routing resources to accommodate the most complex interconnect patterns. They are customized by loading configuration data into the internal memory cells. The FPGA can either actively read its configuration data out of external serial or byte-parallel PROM (master modes), or the configuration data can be written into the FPGA (slave and peripheral modes).

The XC4000 family is supported by powerful and sophisticated software, covering every aspect of design, from schematic entry, to simulation, to automatic block placement and routing of interconnects, and finally the creation of the configuration bit stream.

Since 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 the design and development cycle, but they also offer a cost-effective solution for production rates well beyond 1000 systems per month.

For a detailed description of the device features, architecture, configuration methods and pin descriptions, see pages 2-9 through 2-45.

Table 1. The XC4000 Family of Field-Programmable Gate Arrays

Device	XC4003	XC4005	XC4006	XC4008	XC4010/10D	XC4013	XC4025
Appr. Gate Count	3,000	5,000	6,000	8,000	10,000	13,000	25,000
CLB Matrix	10 x 10	14 x 14	16 x 16	18 x 18	20 x 20	24 x 24	32 x 32
Number of CLBs	100	196	256	324	400	576	1,024
Number of Flip-Flops	360	616	768	936	1,120	1,536	2,560
Max Decode Inputs (per side)	30	42	48	54	60	72	96
Max RAM Bits	3,200	6,272	8,192	10,368	12,800*	18,432	32,768
Number of IOBs	80	112	128	144	160	192	256

*XC4010D has no RAM

Absolute Maximum Ratings

Symbol	Description		Units
V_{CC}	Supply voltage relative to GND	−0.5 to 7.0	V
V_{IN}	Input voltage with respect to GND	−0.5 to 7	V
V_{TS}	Voltage applied to 3-state output	−0.5 to 7	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.	+ 150	°C

Note: 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 Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

Operating Conditions

Symbol	Description		Min	Max	Units
V_{CC}	Supply voltage relative to GND	Commercial: 0°C to 70°C	4.75	5.25	V
	Supply voltage relative to GND	Industrial: −40°C to 85°C	4.5	5.5	V
	Supply voltage relative to GND	Military: −55°C to 125°C	4.5	5.5	V
V_{IH}	High-level input voltage (XC4000 has TTL-like input thresholds)		2.0	V_{CC}	V
V_{IL}	Low-level input voltage (XC4000 has TTL-like input thresholds)		0	0.8	V
T_{IN}	Input signal transition time			250	ns

DC Characteristics Over Operating Conditions

Symbol	Description	Min	Max	Units
V_{OH}	High-level output voltage @ I_{OH} = −4.0 mA, V_{CC} min	2.4		V
V_{OL}	Low-level output voltage @ I_{OL} = 12.0 mA, V_{CC} max (Note 1)		0.4	V
I_{CCO}	Quiescent LCA supply current (Note 2)		10	mA
I_{IL}	Leakage current	−10	+10	µA
C_{IN}	Input capacitance (sample tested)		15	pF
I_{PIN}	Pad pull-up (when selected) @ V_{IN} = 0V (sample tested)	0.02	0.25	mA
I_{BLL}	Horizontal Long Line pull-up (when selected) @ logic Low	0.2	2.5	mA

Note: 1. With 50% of the outputs simultaneously sinking 12 mA.
2. With no output current loads, no active input or long line pull-up resistors, all package pins at V_{CC} or GND, and the LCA configured with a MakeBits tie option.

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. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Description	Symbol	Speed Grade / Device	-6 Max	-5 Max	-4 Max	Units
Full length, both pull-ups, inputs from IOB I-pins	T_{WAF}	XC4003	9.0	8.0	5.0	ns
		XC4005	10.0	9.0	6.0	ns
		XC4006	11.0	10.0	7.0	ns
		XC4008	12.0	11.0	8.0	ns
		XC4010	13.0	12.0	9.0	ns
		XC4013	15.0	14.0	11.0	ns
Full length, both pull-ups, inputs from internal logic	T_{WAFL}	XC4003	12.0	11.0	7.0	ns
		XC4005	13.0	12.0	8.0	ns
		XC4006	14.0	13.0	9.0	ns
		XC4008	15.0	14.0	10.0	ns
		XC4010	16.0	15.0	11.0	ns
		XC4013	18.0	17.0	13.0	ns
Half length, one pull-up, inputs from IOB I-pins	T_{WAO}	XC4003	9.0	8.0	6.0	ns
		XC4005	10.0	9.0	7.0	ns
		XC4006	11.0	10.0	8.0	ns
		XC4008	12.0	11.0	9.0	ns
		XC4010	13.0	12.0	10.0	ns
		XC4013	15.0	14.0	12.0	ns
Half length, one pull-up, inputs from internal logic	T_{WAOI}	XC4003	12.0	11.0	8.0	ns
		XC4005	13.0	12.0	9.0	ns
		XC4006	14.0	13.0	10.0	ns
		XC4008	15.0	14.0	11.0	ns
		XC4010	16.0	15.0	12.0	ns
		XC4013	18.0	17.0	14.0	ns

Note: 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 on page 2-52.

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. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

		Speed Grade	-6	-5	-4	
Description	Symbol	Device	Max	Max	Max	Units
Global Signal Distribution						
From pad through primary buffer, to any clock K	T_{PG}	XC4003	7.8	5.8	5.1	ns
		XC4005	8.0	6.0	5.5	ns
		XC4006	8.2	6.2	5.7	ns
		XC4008	8.6	6.6	6.1	ns
		XC4010	9.0	7.0	6.5	ns
		XC4013	10.0	8.0	7.5	ns
From pad through secondary buffer, to any clock K	T_{SG}	XC4003	8.8	6.8	6.3	ns
		XC4005	9.0	7.0	6.7	ns
		XC4006	9.2	7.2	6.9	ns
		XC4008	9.6	7.6	7.3	ns
		XC4010	10.0	8.0	7.7	ns
		XC4013	11.0	9.0	8.7	ns

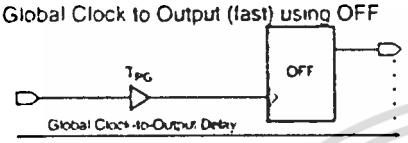
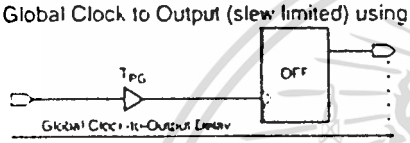
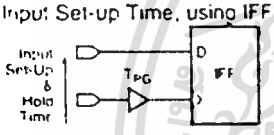
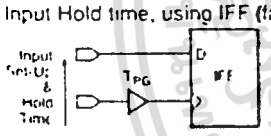
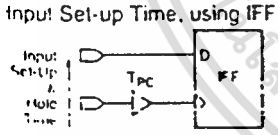
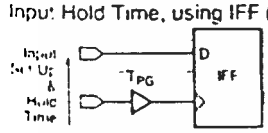
Horizontal Longline Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

		Speed Grade	-6	-5	-4	
Description	Symbol	Device	Max	Max	Max	Units
TBUF driving a Horizontal Longline (L.L.) I going High or Low to L.L. going High or Low, with T is Low, i.e. buffer is constantly active	T_{IO1}	XC4003	8.8	6.2	4.4	ns
		XC4005	10.0	7.0	5.5	ns
		XC4006	10.6	7.5	6.0	ns
		XC4008	11.1	8.0	6.5	ns
		XC4010	11.7	8.5	7.0	ns
		XC4013	13.0	9.5	7.5	ns
I going Low to L.L. going from resistive pull-up High to active Low. (TBUF configured as open drain)	T_{IO2}	XC4003	9.3	6.7	5.0	ns
		XC4005	10.5	7.5	6.0	ns
		XC4006	11.1	8.0	6.5	ns
		XC4008	11.6	8.5	7.0	ns
		XC4010	12.2	9.0	7.5	ns
		XC4013	13.5	10.0	8.0	ns
T going Low to L.L. going from resistive pull-up or floating High to active Low. (TBUF configured as open drain or active buffer with I = Low)	T_{ON}	XC4003	10.7	9.0	7.2	ns
		XC4005	12.0	10.0	8.0	ns
		XC4006	12.6	10.5	8.5	ns
		XC4008	13.2	11.0	9.0	ns
		XC4010	13.8	11.5	9.5	ns
		XC4013	15.1	12.6	11.1	ns
T going High to TBUF going inactive, not driving L.L.	T_{OFF}	All devices	3.0	2.0	1.6	ns
T going High to L.L. going from Low to High, pulled up by a single resistor	T_{PUS}	XC4003	24.0	26.0	14.0	ns
		XC4005	26.0	22.0	16.0	ns
		XC4006	28.0	24.0	16.0	ns
		XC4008	30.0	26.0	20.0	ns
		XC4010	32.0	28.0	22.0	ns
		XC4013	36.0	32.0	26.0	ns
T going High to L.L. going from Low to High, pulled up by two resistors	T_{PUF}	XC4003	11.6	9.0	7.0	ns
		XC4005	12.0	10.0	8.0	ns
		XC4006	13.0	11.0	9.0	ns
		XC4008	14.0	12.0	10.0	ns
		XC4010	15.0	13.0	11.0	ns
		XC4013	17.0	15.0	13.0	ns

Guaranteed Input and Output Parameters (Pin-to-Pin)

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 these two methods, the values listed below should be used, and the derived values must be ignored.

Description	Symbol	Speed Grade / Device	-6	-5	-4	Units
Global Clock to Output (fast) using OFF 	T_{ICKOF} (Max)	XC4003	15.1	12.5	11.6	ns
		XC4005	15.5	13.0	12.0	ns
		XC4006	15.7	13.2	12.2	ns
		XC4008	16.1	13.6	12.6	ns
		XC4010	16.5	14.0	13.0	ns
		XC4013	17.5	15.0	14.0	ns
Global Clock to Output (slew limited) using OFF 	T_{ICKO} (Max)	XC4003	19.9	15.2	14.4	ns
		XC4005	20.5	16.0	15.0	ns
		XC4006	20.7	16.2	15.2	ns
		XC4008	21.1	16.6	15.6	ns
		XC4010	21.5	17.0	16.0	ns
		XC4013	22.5	18.0	17.0	ns
Input Set-up Time, using IFF (fast) 	T_{PSUF} (Min)	XC4003	2.4	2.0	1.6	ns
		XC4005	2.0	1.5	1.2	ns
		XC4006	1.8	1.3	1.0	ns
		XC4008	1.4	0.9	0.6	ns
		XC4010	1.0	0.5	0.2	ns
		XC4013	0.5	0	0	ns
Input Hold time, using IFF (fast) 	T_{PHF} (Min)	XC4003	5.1	4.0	4.0	ns
		XC4005	5.5	4.5	4.5	ns
		XC4006	5.7	4.7	4.7	ns
		XC4008	6.1	5.1	5.1	ns
		XC4010	6.5	5.5	5.5	ns
		XC4013	7.5	6.5	6.5	ns
Input Set-up Time, using IFF (with delay) 	T_{PSU} (Min)	XC4003	21.5	18.5	12.0	ns
		XC4005	21.0	18.0	12.0	ns
		XC4006	20.8	17.8	12.0	ns
		XC4008	20.4	17.4	12.0	ns
		XC4010	20.0	17.0	12.0	ns
		XC4013	19.0	16.0	12.0	ns
Input Hold Time, using IFF (with delay) 	T_{PH} (Min)	XC4003	0	0	0	ns
		XC4005	0	0	0	ns
		XC4006	0	0	0	ns
		XC4008	0	0	0	ns
		XC4010	0	0	0	ns
		XC4013	0	0	0	ns

Timing is measured at pin threshold, with 50 pF external capacitive loads (incl. test fixture). When testing fast outputs, only one output switches. When testing slew-rate limited outputs, half the number of outputs on one side of the device are switching. These parameter values are tested and guaranteed for worst-case conditions of supply voltage and temperature, and also with the most unfavorable clock polarity choice.

T_{PDLI} for -4 Speed Grade

Pad to I1, I2 via transparent latch, with delay	XC4003	17.6 ns
	XC4005	17.9 ns
	XC4006	18.0 ns
	XC4008	18.3 ns
	XC4010	18.6 ns
	XC4013	19.3 ns

T_{PICKD} for -4 Speed Grade

Input set-up time pad to clock (IK) with delay	XC4003	15.6 ns
	XC4005	15.9 ns
	XC4006	16.0 ns
	XC4008	16.3 ns
	XC4010	16.6 ns
	XC4013	17.3 ns

X5282

IOB 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. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Description	Speed Grade / Symbol	-6 Min	-6 Max	-5 Min	-5 Max	-4 Min	-4 Max	Units
Input								
Propagation Delays								
Pad to I1, I2	T_{PID}		4.0		3.0		2.8	ns
Pad to I1, I2, via transparent latch (fast)	T_{PLI}		8.0		7.0		6.0	ns
Pad to I1, I2, via transparent latch (with delay)	T_{PDLI}		26.0		24.0		**	ns
Clock (IK) to I1, I2, (flip-flop)	T_{IKRI}		8.0		7.0		6.0	ns
Clock (IK) to I1, I2 (latch enable, active Low)	T_{IKLI}		8.0		7.0		6.0	ns
Set-up Time (Note 3)								
Pad to Clock (IK), fast	T_{PICK}	7.0		6.0		4.0		ns
Pad to Clock (IK) with delay	T_{PICKD}	25.0		24.0		**		ns
Hold Time (Note 3)								
Pad to Clock (IK), fast	T_{IKPI}	1.0		1.0		1.0		ns
Pad to Clock (IK) with delay	T_{IKPID}	neg		neg		neg		ns
Output								
Propagation Delays								
Clock (OK) to Pad (fast)	T_{OKPOF}		7.5		7.0		6.5	ns
same (slew rate limited)	T_{OKPOS}		11.5		10.0		9.5	ns
Output (O) to Pad (fast)	T_{OPF}		9.0		7.0		5.5	ns
same (slew-rate limited)	T_{OPS}		13.0		10.0		8.5	ns
3-state to Pad begin hi-Z (slew-rate independent)	T_{TSHZ}		9.0		7.0		6.5	ns
3-state to Pad active and valid (fast)	T_{TSONF}		13.0		10.0		9.5	ns
same (slew -rate limited)	T_{TSONS}		17.0		13.0		12.5	ns
Set-up and Hold Times								
Output (O) to clock (OK) set-up time	T_{OOK}	8.0		6.0		5.5		ns
Output (O) to clock (OK) hold time	T_{OKO}	0		0		0		ns
Clock								
Clock High or Low time	T_{CH}/T_{CL}	5.0		4.0		4.0		ns
Global Set/Reset								
Delay from GSR net through Q to I1, I2	T_{RRI}		14.5		13.5		13.5	ns
Delay from GSR net to Pad	T_{RPO}		18.0		17.0		14.0	ns
GSR width*	T_{MRW}	21.0		18.0		18.0		ns

* Timing is based on the XC4005. For other devices see XACT timing calculator.
** See preceding page

Notes 1 Timing is measured at pin threshold, with 50 pF external capacitive loads (incl. test fixture). Slew rate limited output rise/fall times are approximately two times longer than fast output rise/fall times. A maximum total external capacitive load for simultaneous fast mode switching in the same direction is 200 pF per power/ground pin pair. For slew-rate limited outputs this total is two times larger. Exceeding this maximum capacitive load can result in ground bounce of >1.5 V amplitude, <5 ns duration, which might cause problems when the LCA drives clocks and other asynchronous signals.

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

3 Input pad setup times and hold times are specified with respect to the internal clock (IK). To calculate system setup time, subtract clock delay (clock pad to IK) from the specified input pad setup time value, but do not subtract below zero. Negative hold time means that the delay in the input data is adequate for the external system hold time to be zero, provided the input clock uses the Global signal distribution from pad to IK.

CLB 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. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Speed Grade		-6		-5		-4		
Description	Symbol	Min	Max	Min	Max	Min	Max	Units
Combinatorial Delays								
F/G inputs to X/Y outputs	T_{ILO}		6.0		4.5		4.0	ns
F/G inputs via H' to X/Y outputs	T_{IHO}		8.0		7.0		6.0	ns
C inputs via H' to X/Y outputs	T_{HHO}		7.0		5.0		4.5	ns
CLB Fast Carry Logic								
Operand inputs (F1,F2,G1,G4) to Cout	T_{OPCY}		7.0		5.5		5.0	ns
Add/Subtract input (F3) to Cout	T_{ASCY}		8.0		6.0		5.5	ns
Initialization inputs (F1,F3) to Cout	T_{INCY}		6.0		4.0		3.5	ns
C_{IN} through function generators to X/Y outputs	T_{SUM}		8.0		6.0		5.5	ns
C_{IN} to C_{OUT}, bypass function generators	T_{BYP}		2.0		1.5		1.5	ns
Sequential Delays								
Clock K to outputs Q	T_{CKO}		5.0		3.0		3.0	ns
Set-up Time before Clock K								
F/G inputs	T_{ICK}	6.0		4.5		4.5		ns
F/G inputs via H'	T_{IHCK}	8.0		6.0		6.0		ns
C inputs via H1	T_{HHCK}	7.0		5.0		5.0		ns
C inputs via DIN	T_{DICK}	4.0		3.0		3.0		ns
C inputs via EC	T_{ECCK}	7.0		4.0		3.0		ns
C inputs via S/R, going Low (inactive)	T_{RCK}	6.0		4.5		4.0		ns
C_{IN} input via F'/G'	T_{CCK}	8.0		6.0		5.5		ns
C_{IN} input via F'/G' and H'	T_{CHCK}	10.0		7.5		7.3		ns
Hold Time after Clock K								
F/G inputs	T_{CKI}	0		0		0		ns
F/G inputs via H'	T_{CKIH}	0		0		0		ns
C inputs via H1	T_{CKHH}	0		0		0		ns
C inputs via DIN	T_{CKDI}	0		0		0		ns
C inputs via EC	T_{CKEC}	0		0		0		ns
C inputs via S/R, going Low (inactive)	T_{CKR}	0		0		0		ns
Clock								
Clock High time	T_{CH}	5.0		4.5		4.5		ns
Clock Low time	T_{CL}	5.0		4.5		4.5		ns
Set/Reset Direct								
Width (High)	T_{RPW}	5.0		4.0		4.0		ns
Delay from C inputs via S/R, going High to Q	T_{RIO}		9.0		8.0		7.0	ns
Master Set/Reset*								
Width (High or Low)	T_{MRW}	21.0		18.0		16.0		ns
Delay from Global Set/Reset net to Q	T_{MRQ}		33.0		31.0		28.0	ns

* Timing is based on the XC4005. For other devices see XACT timing calculator.

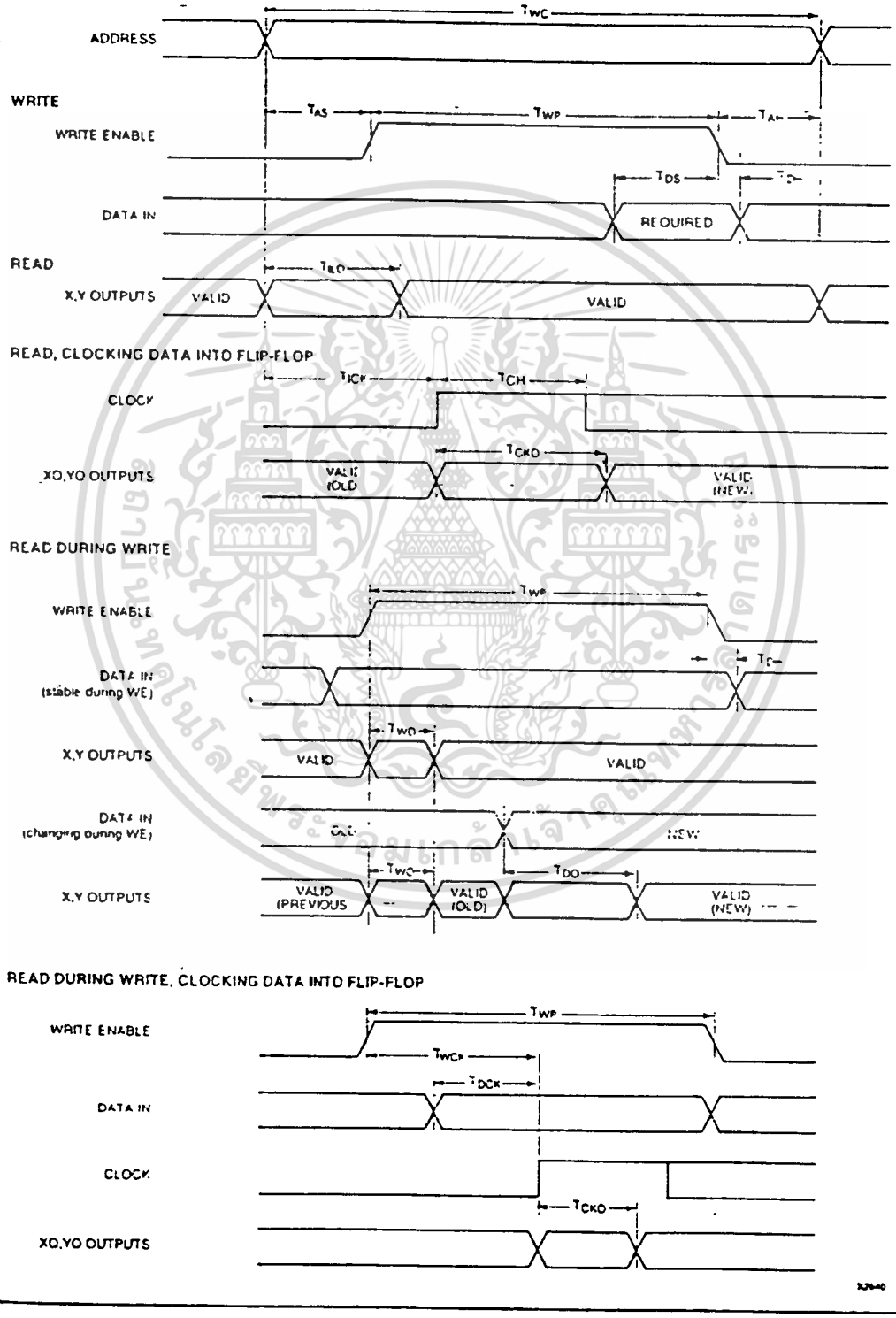
CLB Switching Characteristic Guidelines (continued)

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

CLB RAM Option	Speed Grade		-6		-5		-4		
Description	Symbol		Min	Max	Min	Max	Min	Max	Units
Write Operation									
Address write cycle time	16 x 2	T_{WC}	9.0		8.0		8.0		ns
	32 x 1	T_{WCT}	9.0		8.0		8.0		ns
Write Enable pulse width (High)	16 x 2	T_{WP}	5.0		4.0		4.0		ns
	32 x 1	T_{WPT}	5.0		4.0		4.0		ns
Address set-up time before beginning of WE	16 x 2	T_{AS}	2.0		2.0		2.0		ns
	32 x 1	T_{AST}	2.0		2.0		2.0		ns
Address hold time after end of WE	16 x 2	T_{AH}	2.0		2.0		2.0		ns
	32 x 1	T_{AHT}	2.0		2.0		2.0		ns
DIN set-up time before end of WE	16 x 2	T_{DS}	4.0		4.0		4.0		ns
	32 x 1	T_{DST}	5.0		5.0		5.0		ns
DIN hold time after end of WE	both	T_{DH}	2.0		2.0		2.0		ns
Read Operation									
Address read cycle time	16 x 2	T_{RC}	7.0		5.5		5.0		ns
	32 x 1	T_{RCT}	10.0		7.5		7.0		ns
Data valid after address change (no Write Enable)	16 x 2	T_{ILO}		6.0		4.5		4.0	ns
	32 x 1	T_{IHO}		8.0		7.0		6.0	ns
Read Operation, Clocking Data into Flip-Flop									
Address setup time before clock K	16 x 2	T_{ICK}	6.0		4.5		4.5		ns
	32 x 1	T_{IHCK}	8.0		6.0		6.0		ns
Read During Write									
Data valid after WE going active (DIN stable before WE)	16 x 2	T_{WO}		12.0		10.0		9.0	ns
	32 x 1	T_{WOT}		15.0		12.0		11.0	ns
Data valid after DIN (DIN change during WE)	16 x 2	T_{DO}		11.0		9.0		8.5	ns
	32 x 1	T_{DOT}		14.0		11.0		11.0	ns
Read During Write, Clocking Data into Flip-Flop									
WE setup time before clock K	16 x 2	T_{WCK}	12.0		10.0		9.5		ns
	32 x 1	T_{WCKT}	15.0		12.0		11.5		ns
Data setup time before clock K	16 x 2	T_{DCK}	11.0		9.0		9.0		ns
	32 x 1	T_{DCKT}	14.0		11.0		11.0		ns

Note: Timing for the 16 x 1 RAM option is identical to 16 x 2 RAM timing.

CLB RAM Timing Characteristics



READ DURING WRITE, CLOCKING DATA INTO FLIP-FLOP

X5640

XC4003 Pinouts

Pin Description	PC84	PQ100	PG120	Bound Scan
VCC	2	92	G3	-
I/O (A8)	3	93	G1	32
I/O (A9)	4	94	F1	35
I/O	-	95	E1	38
I/O	-	96	F2	41
I/O (A10)	5	97	F3	44
I/O (A11)	6	98	D1	47
-	-	-	E2*	-
I/O (A12)	7	99	C1	50
I/O (A13)	8	100	D2	53
-	-	-	E3*	-
-	-	-	B1*	-
I/O (A14)	9	1	C2	56
SGCK1 (A15, I/O)	10	2	D3	59
VCC	11	3	C3	-
GND	12	4	C4	-
PGCK1 (A16, I/O)	13	5	B2	62
I/O (A17)	14	6	B3	65
-	-	-	A1*	-
-	-	-	A2*	-
I/O (TDI)	15	7	C5	68
I/O (TCK)	16	8	B4	71
-	-	-	A3*	-
I/O (TMS)	17	9	B5	74
I/O	18	10	A4	77
I/O	-	-	C6	80
I/O	-	11	A5	83
I/O	19	12	B6	86
I/O	20	13	A6	89
GND	21	14	B7	-
VCC	22	15	C7	-
I/O	23	16	A7	92
I/O	24	17	A8	95
I/O	-	18	A9	98
I/O	-	-	B8	101
I/O	25	19	C8	104
I/O	26	20	A10	107
I/O	27	21	B9	110
I/O	-	22	A11	113
-	-	-	B10*	-
I/O	28	23	C9	116
SGCK2 (I/O),	29	24	A12	119
O (M1)	30	25	B11	122
GND	31	26	C10	-
I (M0)	32	27	C11	125[1]
VCC	33	28	D11	-
I (M2)	34	29	B12	126[1]
PGCK2 (I/O)	35	30	C12	127
I/O (HDC)	36	31	A13	130
-	-	-	B13*	-
-	-	-	E11*	-
I/O	-	32	D12	133
I/O (LDC)	37	33	C13	136
I/O	38	34	E12	139
I/O	39	35	D13	142
I/O	-	36	F11	145
I/O	-	37	E13	148
I/O	40	38	F12	151
I/O (INIT)	41	39	F13	154
VCC	42	40	G12	-

Pin Description	PC84	PQ100	PG120	Bound Scan
GND	43	41	G11	-
I/O	44	42	G13	157
I/O	45	43	H13	160
I/O	-	44	J13	163
I/O	-	45	H12	166
I/O	46	46	H11	169
I/O	47	47	K13	172
I/O	48	48	J12	175
I/O	49	49	L13	176
-	-	-	K12*	-
-	-	-	J11*	-
I/O	50	50	M13	181
SGCK3 (I/O)	51	51	L12	184
GND	52	52	K11	-
DONE	53	53	L11	-
VCC	54	54	L10	-
PROG	55	55	M12	-
I/O (D7)	56	56	M11	187
PGCK3 (I/O),	57	57	N13	190
-	-	-	N12*	-
-	-	-	L9*	-
I/O (D6)	58	58	M10	193
I/O	-	59	N11	196
I/O (D5)	59	60	M9	199
I/O (CS0)	60	61	N10	202
I/O	-	62	L8	205
I/O	-	63	N9	208
I/O (D4)	61	64	M8	211
I/O	62	65	N8	214
VCC	63	66	M7	-
GND	64	67	L7	-
I/O (D3)	65	68	N7	217
I/O (RS)	66	69	N6	220
I/O	-	70	N5	223
I/O	-	-	M6	226
I/O (D2)	67	71	L6	229
I/O	68	72	N4	232
I/O (D1)	69	73	M5	235
I/O (RCLK-BUSY/RDY)	70	74	N3	238
-	-	-	M4*	-
-	-	-	L5*	-
I/O (D0, DIN),	71	75	N2	241
SGCK4 (DOUT, I/O)	72	76	M3	244
CCLK	73	77	L4	-
VCC	74	78	L3	-
O (TDO)	75	79	M2	-
GND	76	80	K3	-
I/O (A0, WS)	77	81	L2	2
PGCK4 (A1, I/O)	78	82	N1	5
-	-	-	M1*	-
-	-	-	J3*	-
I/O (CS1, A2)	79	83	K2	8
I/O (A3)	80	84	L1	11
I/O (A4)	81	85	J2	14
I/O (A5)	82	86	K1	17
I/O	-	87	H3	20
I/O	-	88	J1	23
I/O (A6)	83	89	H2	26
I/O (A7)	84	90	H1	29
GND	1	91	G2	-

* Indicates unconnected package pins.

1 Contributes only one bit (J) to the boundary scan register

Boundary Scan Bit 0 = TDO.T

Boundary Scan Bit 1 = TDO.O

Boundary Scan Bit 247 = BSCANT.UPD

```
--------------------------------------------------------------------------------
-- Project          Digital Alarmminute Clock
-- By               Mr.Suracbet Sripolkrang 35103255
-- Model            clock top level of design
--                  Implementation on FPGA XC4004APC84C-6
--------------------------------------------------------------------------------
LIBRARY synth;
USE synth.stdsynth.ALL;
ENTITY Clock is
  PORT (
    SIGNAL  reset                 : IN vlbit;
    SIGNAL  clk_60hz              : IN vlbit;
    SIGNAL  Alarmminute_off_sw    : IN vlbit;
    SIGNAL  dispalarm             : IN vlbit;
    SIGNAL  dispsecond            : IN vlbit;
    SIGNAL  select12_24           : IN vlbit;
    SIGNAL  sethour               : IN vlbit;
    SIGNAL  setminute             : IN vlbit;
    SIGNAL  am,pm                 : OUT vlbit;
    SIGNAL  dot_1hz               : OUT vlbit;
    SIGNAL  Alarmminute_output    : OUT vlbit;
    SIGNAL  seg0,seg1             : OUT vlbit_1d(6 downto 0);
    SIGNAL  seg2,seg3             : OUT vlbit_1d(6 downto 0) );
END Clock;
ENTITY first of Clock is
---------- COMPONENT Decalration ----------
---------- divider block ----------
COMPONENT Divider
  PORT (
    SIGNAL reset, up_enable, clk          : IN      vlbit;
```

```
    SIGNAL cout                          : OUT     vlbit);
  END COMPONENT;
  ---------- second block ----------
  COMPONENT Second
    PORT (
      SIGNAL reset, up_enable, clk       : IN      vlbit;
      SIGNAL cout                        : OUT     vlbit;
      SIGNAL bcdout                      : OUT     vlbit_1d(5 downto 0);
      SIGNAL secondsetenable             : IN      vlbit;
      SIGNAL sethour,setminute           : IN      vlbit;
      SIGNAL resetout                    : OUT     vlbit );
  END COMPONENT;
  ---------- time block ----------
  COMPONENT Time
  PORT (
      SIGNAL reset, up_enable, clk_input : IN      vlbit;
      SIGNAL clk_1sec,clk_60hz           : IN      vlbit;
      SIGNAL sethour,setminute           : IN      vlbit;
      SIGNAL timesetenable               : IN      vlbit;
      SIGNAL timeminute                  : OUT     vlbit_1d(5 downto 0);
      SIGNAL timehour                    : OUT     vlbit_1d(5 downto 0);
      SIGNAL timeam,timepm               : OUT     vlbit;
      SIGNAL secondreset                 : IN      vlbit );
  END COMPONENT;
  ---------- control block ----------
  COMPONENT Control
  PORT (
      SIGNAL comp_input,Alarmminute_off_sw   :IN   vlbit;
      SIGNAL reset                           :IN   vlbit;
      SIGNAL Alarmminute_outputt             :OUT  vlbit);
```

```
END COMPONENT;
---------- comparator ----------
COMPONENT Comparator
  PORT (
    SIGNAL  timeminute                : IN  vlbit_1d(5 downto 0);
    SIGNAL  Alarmminute               : IN  vlbit_1d(5 downto 0);
    SIGNAL  alarmhour,timehour        : IN  vlbit_1d(5 downto 0);
    SIGNAL  timeam,timepm             : IN  vlbit;
    SIGNAL  alarmam,alarmpm           : IN  vlbit;
    SIGNAL  reset,clk                 : IN  vlbit;
    SIGNAL  comp_OUT                  : OUT vlbit );
END COMPONENT;
---------- Alarmminute COMPONENT ----------
COMPONENT Alarmminute
  PORT (
    SIGNAL reset, up_enable                 : IN    vlbit;
    SIGNAL Alarmminute                      : OUT   vlbit_1d(5 downto 0);
    SIGNAL alarmhour                        : OUT   vlbit_1d(5 downto 0);
    SIGNAL sethour,setminute                : IN    vlbit;
    SIGNAL clk_1sec,clk_60hz                : IN    vlbit;
    SIGNAL alarmsetenable                   : IN    vlbit;
    SIGNAL alarmam,alarmpm                  : OUT   vlbit
);
END COMPONENT;
---------- muxtiplexer COMPONENT ----------
COMPONENT Divider1
  PORT (
    SIGNAL  dispalarm                 : IN vlbit;
    SIGNAL  select12_24               : IN vlbit;
    SIGNAL  alarmam,alarmpm           : IN vlbit;
```

```
    SIGNAL  timeam,timepm                   : IN vlbit;
    SIGNAL  timeminute                      : IN vlbit_1d(5 downto 0);
    SIGNAL  Alarmminute                     : IN vlbit_1d(5 downto 0);
    SIGNAL  timehour,alarmhour              : IN vlbit_1d(5 downto 0);
    SIGNAL  timesetenable,alarmsetenable,secondsetenable: OUT vlbit;
    SIGNAL  bcddigit0,bcddigit1             : OUT vlbit_1d(3 downto 0);
    SIGNAL  bcddigit2,bcddigit3             : OUT vlbit_1d(3 downto 0);
    SIGNAL  am,pm                           : OUT vlbit;
    SIGNAL  dispsecond                      : IN vlbit;
    SIGNAL  timesecond                      : IN vlbit_1d(5 downto 0)
  );
END COMPONENT;
---------- segment decoder ----------
COMPONENT Divider4
  PORT (
    SIGNAL bcd                              : IN    vlbit_1d(3 downto 0);
    SIGNAL seg                              : OUT   vlbit_1d(6 downto 0) );
END COMPONENT;
---------- segment decoder blank first digit ----------
COMPONENT Divider5
  PORT (
    SIGNAL bcd          : IN    vlbit_1d(3 downto 0);
    SIGNAL seg          : OUT   vlbit_1d(6 downto 0)
  );
END COMPONENT;
---------- SIGNAL declaration ----------
  SIGNAL    pullup          : vlbit;
  SIGNAL    tmp1hz          : vlbit;
  SIGNAL    tmp1minute      : vlbit;
  SIGNAL    bustimeminute   : vlbit_1d(5 downto 0);
```

```
SIGNAL   bustimehour      : vlbit_1d(5 downto 0);
SIGNAL   bussethour       : vlbit;
SIGNAL   bussetminute     : vlbit;
SIGNAL   bustimeset       : vlbit;
SIGNAL   busalarmset      : vlbit;
SIGNAL   bussecondset     : vlbit;
SIGNAL   bustimeam        : vlbit;
SIGNAL   bustimepm        : vlbit;
SIGNAL   busselect1224    : vlbit;
SIGNAL   busalarmminute   : vlbit_1d(5 downto 0);
SIGNAL   busalarmhour     : vlbit_1d(5 downto 0);
SIGNAL   busalarmam       : vlbit;
SIGNAL   busalarmpm       : vlbit;
SIGNAL   bussecond        : vlbit_1d(5 downto 0);
SIGNAL   busresetout      : vlbit;
SIGNAL   buscomp          : vlbit;
SIGNAL   bus60hz          : vlbit;
SIGNAL   bcdd0            : vlbit_1d(3 downto 0);
SIGNAL   bcdd1            : vlbit_1d(3 downto 0);
SIGNAL   bcdd2            : vlbit_1d(3 downto 0);
SIGNAL   bcdd3            : vlbit_1d(3 downto 0);

BEGIN
    pullup          <= '1';
    dot_1hz         <= tmp1hz;
    bussethour      <= sethour;
    bussetminute    <= setminute;
    bus60hz         <= clk_60hz;
    busselect1224   <= select12_24;
    blocks1:Divider
```

```
PORT map (      reset           => reset,
                up_enable       => pullup,
                clk             => bus60hz,
                cout            => tmp1hz
        );
blocks2:Second
 PORT map (     reset           => reset,
                up_enable       => pullup,
                clk             => tmp1hz,
                cout            => tmp1minute,
                bcdout          =>  bussecond,
                secondsetenable =>  bussecondset,
                sethour  =>  bussethour,
                setminute       => bussetminute,
                resetout =>  busresetout );
blocks3 : Time
PORT map (      reset           => reset,
                up_enable       => pullup,
                clk_input       => tmp1minute,
                clk_1sec        => tmp1hz,
                clk_60hz        => bus60hz,
                sethour         => bussethour,
                setminute       => bussetminute,
                timesetenable   => bustimeset,
                timeminute      => bustimeminute,
                timehour        => bustimehour,
                timeam          => bustimeam,
                timepm          => bustimepm,
                secondreset     => busresetout          );
blocks4:Alarmminute
```

```
PORT map (
                reset           => reset,
                up_enable       => pullup,
                Alarmminute     => busalarmminute,
                alarmhour       => busalarmhour,
                sethour         => bussethour,
                setminute       => bussetminute,
                clk_1sec        => tmp1hz,
                clk_60hz        => bus60hz,
                alarmsetenable  => busalarmset,
                alarmam         => busalarmam,
                alarmpm         => busalarmpm          );
blocks5:Comparator
  PORT map (
                timeminute      => bustimeminute,
                Alarmminute     => busalarmminute,
                alarmhour       => busalarmhour,
                timehour        => bustimehour,
                timeam          => bustimeam,
                timepm          => bustimepm,
                alarmam         => busalarmam,
                alarmpm         => busalarmpm,
                reset           => reset,
                clk             => bus60hz,
                comp_OUT        => buscomp             );
blocks6:Control
        PORT map (
                comp_input              => buscomp,
                Alarmminute_off_sw      => Alarmminute_off_sw,
                reset                   => reset,
```

```
                Alarmminute_output      => Alarmminute_output                );
blocks7:Multiplexer
        PORT map (
                dispalarm       =>      dispalarm,
                select12_24     =>      busselect1224,
                alarmam         =>      busalarmam,
                alarmpm         =>      busalarmpm,
                timeam          =>      bustimeam,
                timepm          =>      bustimepm,
                timeminute      =>      bustimeminute,
                Alarmminute     =>      busalarmminute,
                timehour        =>      bustimehour,
                alarmhour       =>      busalarmhour,
                timesetenable   =>      bustimeset,
                alarmsetenable  =>      busalarmset,
                secondsetenable =>      bussecondset,
                bcddigit0       =>      bcdd0,
                bcddigit1       =>      bcdd1,
                bcddigit2       =>      bcdd2,
                bcddigit3       =>      bcdd3,
                am              =>      am,
                pm              =>      pm,
                dispsecond      =>      dispsecond,
                timesecond      =>      bussecond
                );
blocks8:Segment1
PORT map (      bcd     =>      bcdd0,
                seg     =>      seg0);
blocks9:Segment1
PORT map (      bcd     => bcdd1,
```

```
                    seg     => seg1);
    blocks10:Segment1
    PORT map (      bcd     =>      bcdd2,
                    seg     =>      seg2);
    blocks11:Segment2
    PORT map (      bcd => bcdd3,
                    seg => seg3);
END first;
```

```
--------------------------------------------------------------------------------
-- Project          Digital Alarmminute Clock
-- By               Mr.Surachet Sripolkrang 35103255
-- Model        clock top level of design
--               Implementation on FPGA XC4004APC84C-6
--------------------------------------------------------------------------------
LIBRARY synth;
USE synth.stdsynth.ALL;

ENTITY Second is
  PORT (
    SIGNAL reset, up_enable, clk        : IN        vlbit;
    SIGNAL cout                         : OUT       vlbit;
    SIGNAL bcdout                       : OUT       vlbit_1d(5 downto 0);
    SIGNAL secondsetenable              : IN        vlbit;
    SIGNAL sethour,setminute            : IN        vlbit;
    SIGNAL resetout                     : OUT       vlbit );
END Second;

ARCHITECTURE first of Second is

  constant trans_tbl : vlbit_2d(0 to 59, 11 downto 0 ) := (
--
--          IN   OUT  .
----------------+--------------------
      B"000000_000001",
      B"000001_000010",
      B"000010_000011",
      B"000011_000100",
      B"000100_000101",
```

B"000101_000110",

B"000110_000111",

B"000111_001000",

B"001000_001001",

B"001001_001010",

B"001010_001011",

B"001011_001100",

B"001100_001101",

B"001101_001110",

B"001110_001111",

B"001111_010000",

B"010000_010001",

B"010001_010010",

B"010010_010011",

B"010011_010100",

B"010100_010101",

B"010101_010110",

B"010110_010111",

B"010111_011000",

B"011000_011001",

B"011001_011010",

B"011010_011011",

B"011011_011100",

B"011100_011101",

B"011101_011110",

B"011110_011111",

B"011111_100000",

B"100000_100001",

B"100001_100010",

B"100010_100011",

```
        B"100011_100100",
        B"100100_100101",
        B"100101_100110",
        B"100110_100111",
        B"100111_101000",
        B"101000_101001",
        B"101001_101010",
        B"101010_101011",
        B"101011_101100",
        B"101100_101101",
        B"101101_101110",
        B"101110_101111",
        B"101111_110000",
        B"110000_110001",
        B"110001_110010",
        B"110010_110011",
        B"110011_110100",
        B"110100_110101",
        B"110101_110110",
        B"110110_110111",
        B"110111_111000",
        B"111000_111001",
        B"111001_111010",
        B"111010_111011",
        B"111011_000000" );
SIGNAL incz,muxz,bcd          : vlbit_1d(5 downto 0);
SIGNAL gate_reset     : vlbit;
SIGNAL gate_clk       : vlbit;
SIGNAL to_reset,to_clk  : vlbit;
```

```
BEGIN
   to_reset <= reset or gate_reset;
   to_clk <= clk and gate_clk;
   bcdout <= bcd;
   PROCESS(secondsetenable,sethour,setminute)
   BEGIN
     IF  (secondsetenable = '1') THEN
         IF (sethour = '0') THEN
               IF (setminute = '0') THEN
                    resetout <= '0';
                    gate_reset <= '0';
                    gate_clk <= '1';
               ELSIF (setminute = '1') THEN
                    -- hold on
                    resetout <= '0';
                    gate_reset <= '0';
                    gate_clk <= '0';
               END IF;
         ELSIF (sethour = '1') THEN
               IF (setminute = '0') THEN
                    -- reset second
                    resetout <= '0';
                    gate_reset <= '1';
                    gate_clk <= '1';
               ELSIF (setminute = '1') THEN
                    resetout <= '1';
                    gate_reset <= '0';
                    gate_clk <= '1';
               END IF;
         END IF;
```

```
    ELSIF (secondsetenable = '0') THEN
                resetout <= '0';
                gate_reset <= '0';
                gate_clk <= '1';
    END IF;
  END PROCESS;
  pla_table (bcd, incz, trans_tbl);
  muxz <= incz WHEN ('1' = up_enable) ELSE
        bcd ;
  PROCESS
  BEGIN
  WAIT UNTIL prising(to_clk) or (to_reset = '1');
   IF (to_reset = '1') THEN
          cout <= '0';
   ELSE
          IF  (incz = B"000000") THEN
              cout <= '1';
          ELSE
              cout <= '0';
          END IF;
   END IF;
   END PROCESS;

  dffc_v (muxz,to_reset, to_clk, bcd);

END first;
```

```
--------------------------------------------------------------------------------
-- Project          Digital Alarmminute Clock
-- By               Mr.Surachet Sripolkrang 35103255
-- Model        clock top level of design
--               Implementation on FPGA XC4004APC84C-6
--------------------------------------------------------------------------------

LIBRARY synth;
USE synth.stdsynth.ALL;

ENTITY Time is
 PORT (
   SIGNAL reset, up_enable, clk_input     : IN vlbit;
   SIGNAL clk_1sec,clk_60hz               : IN vlbit;
   SIGNAL sethour,setminute               : IN vlbit;
   SIGNAL timesetenable                   : IN vlbit;
   SIGNAL timeminute                      : OUT vlbit_1d(5 downto 0);
   SIGNAL timehour                        : OUT vlbit_1d(5 downto 0);
   SIGNAL timeam,timepm                   : OUT vlbit;
   SIGNAL secondreset                     : IN vlbit
 );
END Time;
ARCHITECTURE first of Time is
 COMPONENT Timeminute
 PORT (
   SIGNAL reset, up_enable, clk           : IN    vlbit;
   SIGNAL cout                            : OUT   vlbit;
   SIGNAL bcdout                          : OUT   vlbit_1d(5 downto 0)
 );
 END COMPONENT;
```

```
COMPONENT Timehour
 PORT (
  SIGNAL reset, up_enable, clk    : IN     vlbit;
  SIGNAL bcdout                   : OUT    vlbit_1d(5 downto 0);
  SIGNAL am,pm                    : OUT    vlbit );
 END COMPONENT;
 SIGNAL   carrytohour             : vlbit;
 SIGNAL   busmin                  : vlbit_1d(5 downto 0);
 SIGNAL   bushr                   : vlbit_1d(5 downto 0);
 SIGNAL   gate_clk                : vlbit;
 SIGNAL   to_reset                : vlbit;

BEGIN
  to_reset <= reset or secondreset;
  PROCESS(sethour,setminute,timesetenable,clk_1sec,clk_input,clk_60hz)
  BEGIN
   IF (timesetenable = '1') THEN
    IF (sethour =  '0')  THEN
        IF (setminute = '0') THEN
          gate_clk <= clk_input;
        ELSIF (setminute = '1') THEN
          gate_clk <= clk_1sec;
        END IF;
    ELSIF (sethour = '1') THEN
       IF (setminute = '0') THEN
          gate_clk <= clk_60hz;
        ELSIF (setminute = '1') THEN
          gate_clk <= clk_60hz;
        END IF;
```

```
    END IF;
  ELSIF (timesetenable = '0') THEN
     gate_clk <= clk_input;
  END IF;
END PROCESS;

zz:Timeminute
  PORT map (reset => to_reset,
          up_enable => up_enable,
          clk => gate_clk,
          cout => carrytohour,
          bcdout => busmin);
xx:Timehour
   PORT map (reset => to_reset,
          up_enable => up_enable,
          clk => carrytohour,
          bcdout => bushr,
          am => timeam,
          pm => timepm);
timeminute <= busmin;
timehour <= bushr;
END first;
```

```
-------------------------------------------------------
-- Project        Digital Alarm Clock
-- By             Mr.Surachet Sripolkrang 35103255
-- Model       clock top level of design
--             Implementation on FPGA XC4004APC84C-6
-------------------------------------------------------
LIBRARY synth;
USE synth.stdsynth.ALL;
ENTITY Timeminute is
  PORT (
    SIGNAL reset, up_enable, clk        : IN      vlbit;
    SIGNAL cout                         : OUT     vlbit;
    SIGNAL bcdout                       : OUT     vlbit_1d(5 downto 0)  );
END Timeminute;
ARCHITECTURE first of Timeminute is
   constant trans_tbl : vlbit_2d(0 to 59, 11 downto 0 ) := (
--
--          IN   OUT
-------------+-------------
       B"000000_000001",
       B"000001_000010",
       B"000010_000011",
       B"000011_000100",
       B"000100_000101",
       B"000101_000110",
       B"000110_000111",
       B"000111_001000",
       B"001000_001001",
       B"001001_001010",
       B"001010_001011",
```

B"001011_001100",
B"001100_001101",
B"001101_001110",
B"001110_001111",
B"001111_010000",
B"010000_010001",
B"010001_010010",
B"010010_010011",
B"010011_010100",
B"010100_010101",
B"010101_010110",
B"010110_010111",
B"010111_011000",
B"011000_011001",
B"011001_011010",
B"011010_011011",
B"011011_011100",
B"011100_011101",
B"011101_011110",
B"011110_011111",
B"011111_100000",
B"100000_100001",
B"100001_100010",
B"100010_100011",
B"100011_100100",
B"100100_100101",
B"100101_100110",
B"100110_100111",
B"100111_101000",
B"101000_101001",

```
        B"101001_101010",
        B"101010_101011",
        B"101011_101100",
        B"101100_101101",
        B"101101_101110",
        B"101110_101111",
        B"101111_110000",
        B"110000_110001",
        B"110001_110010",
        B"110010_110011",
        B"110011_110100",
        B"110100_110101",
        B"110101_110110",
        B"110110_110111",
        B"110111_111000",
        B"111000_111001",
        B"111001_111010",
        B"111010_111011",
        B"111011_000000"
);
SIGNAL          incz,muxz,bcd   : vlbit_1d(5 downto 0);
BEGIN
    bcdout <= bcd;
    pla_table (bcd, incz, trans_tbl);
    muxz <= incz WHEN ('1' = up_enable) ELSE
            bcd ;
    PROCESS
    BEGIN
    WAIT UNTIL prising(clk) or reset = '1';
    IF reset = '1' THEN
```

```
        cout <= '0';
    ELSE
        IF (incz = B"000000") THEN
            cout <= '1';
        ELSE
            cout <= '0';
        END IF;
    END IF;
    END PROCESS;
  dffc_v (muxz, reset, clk, bcd);
END first;
```

```
------------------------------------------------------------
-- Project        Digital Alarm Clock
-- By             Mr.Surachet Sripolkrang 35103255
-- Model       clock top level of design
--             Implementation on FPGA XC4004APC84C-6
------------------------------------------------------------
LIBRARY synth;
USE synth.stdsynth.ALL;
ENTITY Timehour is
  PORT (
    SIGNAL reset, up_enable, clk          : IN        vlbit;
    SIGNAL bcdout                         : OUT       vlbit_1d(5 downto 0);
    SIGNAL am,pm                          : OUT       vlbit   );
END Timehour;
ARCHITECTURE first of Timehour is
    constant trans_tbl : vlbit_2d(0 to 23, 11 downto 0 ) := (
--
--          IN   OUT
-----------------+-----------
       B"000000_000001",
       B"000001_000010",
       B"000010_000011",
       B"000011_000100",
       B"000100_000101",
       B"000101_000110",
       B"000110_000111",
       B"000111_001000",
       B"001000_001001",
       B"001001_001010",
       B"001010_001011",
```

```
    B"001011_001100",
    B"001100_001101",
    B"001101_001110",
    B"001110_001111",
    B"001111_010000",
    B"010000_010001",
    B"010001_010010",
    B"010010_010011",
    B"010011_010100",
    B"010100_010101",
    B"010101_010110",
    B"010110_010111",
    B"010111_000000"          );
SIGNAL incz, muxz,bcd    : vlbit_1d (5 downto 0);
BEGIN
bcdout <= bcd;
pla_table (bcd, incz, trans_tbl);
muxz <= incz WHEN ('1' = up_enable) ELSE
        bcd;
 PROCESS
 BEGIN
    WAIT UNTIL prising(clk) or (reset = '1') ;
    IF (reset = '1') THEN
          am <= '1';
          pm <= '0';
    ELSE
      IF (incz > B"001100") or (incz = B"001100")  THEN
         am   <= '0';
         pm   <= '1';
      ELSIF (incz < B"001100") or (incz = B"000000") THEN
```

```
            am <= '1';
            pm <= '0';
          END IF;
        END IF;
      END PROCESS;
      dffc_v (muxz, reset, clk, bcd);
END first;
```

```
------------------------------------------------------------
-- Project        Digital Alarm Clock
-- By             Mr.Surachet Sripolkrang 35103255
-- Model          clock top level of design
--                Implementation on FPGA XC4004APC84C-6
------------------------------------------------------------
LIBRARY synth;
USE synth.stdsynth.ALL;

ENTITY Controlalarm is
PORT (
   SIGNAL comp_input,Alarmminute_off_sw      : IN vlbit;
   SIGNAL reset                              : IN vlbit;
   SIGNAL Alarmminute_output                 : OUT vlbit);
END Controlalarm;
ARCHITECTURE first of Controlalarm is
BEGIN
    Alarmminute_output <= (not reset) and (comp_input) and (not alarmminute_off_sw);
END first;
```

```
-------------------------------------------------------------------------------
-- Project      Digital Alarmminute Clock
-- By           Mr.Surachet Sripolkrang 35103255
-- Model        comparator module
-------------------------------------------------------------------------------
LIBRARY synth;
USE synth.stdsynth.ALL;
ENTITY Comparator is
  PORT (
    SIGNAL  timeminute          : IN  vlbit_1d(5 downto 0);
    SIGNAL  Alarmminute         : IN  vlbit_1d(5 downto 0);
    SIGNAL  alarmhour,timehour  : IN  vlbit_1d(5 downto 0);
    SIGNAL  timeam,timepm       : IN vlbit;
    SIGNAL  alarmam,alarmpm     : IN vlbit;
    SIGNAL  reset,clk           : IN  vlbit;
    SIGNAL  comp_OUT            : OUT vlbit );
END Comparator;

ARCHITECTURE first of Comparator is
BEGIN
        compare: PROCESS
            BEGIN
            WAIT  UNTIL prising (clk) or reset = '1';
            IF    (reset = '1') THEN
                comp_OUT <= '0';
            ELSE
                IF (Alarmminute = timeminute) and (alarmhour = timehour)
                   and (alarmam = timeam) and (alarmpm = timepm) THEN
                    comp_OUT <= '1';
                ELSE
```

```
            comp_OUT <= '0';
        END IF;
    END IF;
END PROCESS compare;
END first;
```

```
--------------------------------------------------------------------------------
-- Project      Digital Alarmminute Clock
-- By           Mr.Surachet Sripolkrang 35103255
-- Model        Alarmminute counter module
--------------------------------------------------------------------------------

LIBRARY synth;
USE synth.stdsynth.ALL;

ENTITY Alarmminute is
 PORT (
   SIGNAL reset, up_enable       : IN vlbit;
   SIGNAL Alarmminute            : OUT vlbit_1d(5 downto 0);
   SIGNAL alarmhour              : OUT vlbit_1d(5 downto 0);
   SIGNAL sethour,setminute      : IN vlbit;
   SIGNAL clk_1sec,clk_60hz      : IN vlbit;
   SIGNAL alarmsetenable         : IN vlbit;
   SIGNAL alarmam,alarmpm        : OUT vlbit
 );
END Alarmminute;
ARCHITECTURE first of Alarmminute is
 COMPONENT Alarmhour
 PORT (
   SIGNAL reset, up_enable, clk  : IN      vlbit;
   SIGNAL cout                   : OUT     vlbit;
   SIGNAL bcdout                 : OUT     vlbit_1d(5 downto 0) );
 END COMPONENT;

 COMPONENT Ahourcount
 PORT (
```

```
  SIGNAL reset, up_enable, clk  : IN    vlbit;
  SIGNAL bcdout                 : OUT   vlbit_1d(5 downto 0);
  SIGNAL am,pm                  : OUT   vlbit );
 END COMPONENT;

 SIGNAL  carrytohour : vlbit;
 SIGNAL  gate_clk    : vlbit;
 SIGNAL  gate_reset  : vlbit;
 SIGNAL  to_reset    : vlbit;
BEGIN
  to_reset <= reset or gate_reset;
  PROCESS(sethour,setminute,alarmsetenable,clk_1sec,clk_60hz)
  BEGIN
   IF (alarmsetenable = '1')  THEN
    IF (sethour =  '0')  THEN
       IF (setminute = '0') THEN
         gate_clk <= '1';
         gate_reset <= '0';
       ELSIF (setminute = '1') THEN
         gate_clk <= clk_1sec;
         gate_reset <= '0';
       END IF;
     ELSIF (sethour = '1') THEN
       IF (setminute = '0') THEN
         gate_clk <= clk_60hz;
         gate_reset <= '0';
       ELSIF (setminute = '1') THEN
         gate_clk <= '1';
         gate_reset <= '1';  -- reset to BEGIN state
       END IF;
```

```
    END IF;
  ELSIF (alarmsetenable = '0') THEN

     gate_clk <= '1';
     gate_reset <= '0';
  END IF;
END PROCESS;

xx:Alarmhour
  PORT map (reset       => to_reset,
        up_enable           => up_enable,
        clk           => gate_clk,
        cout          => carrytohour,
        bcdout              => Alarmminute);

zz:Ahourcount
   PORT map (reset      => to_reset,
        up_enable           => up_enable,
        clk           => carrytohour,
        bcdout              => alarmhour,
        am            => alarmam,
        pm            => alarmpm
   );
END first;
```

```
------------------------------------------------------
-- Project        Digital Alarm Clock
-- By             Mr.Surachet Sripolkrang 35103255
-- Model          clock top level of design
--                Implementation on FPGA XC4004APC84C-6
------------------------------------------------------

LIBRARY synth;
USE synth.stdsynth.ALL;
ENTITY Alarmhour is
  PORT (
    SIGNAL reset, up_enable, clk      : IN     vlbit;
    SIGNAL cout                       : OUT    vlbit;
    SIGNAL bcdout                     : OUT    vlbit_1d(5 downto 0)  );
END Alarmhour;
ARCHITECTURE first of Alarmhour is
   constant trans_tbl : vlbit_2d(0 to 59, 11 downto 0 ) := (
--
--          IN   OUT
-------------+-------------
       B"000000_000001",
       B"000001_000010",
       B"000010_000011",
       B"000011_000100",
       B"000100_000101",
       B"000101_000110",
       B"000110_000111",
       B"000111_001000",
       B"001000_001001",
       B"001001_001010",
```

B"001010_001011",

B"001011_001100",

B"001100_001101",

B"001101_001110",

B"001110_001111",

B"001111_010000",

B"010000_010001",

B"010001_010010",

B"010010_010011",

B"010011_010100",

B"010100_010101",

B"010101_010110",

B"010110_010111",

B"010111_011000",

B"011000_011001",

B"011001_011010",

B"011010_011011",

B"011011_011100",

B"011100_011101",

B"011101_011110",

B"011110_011111",

B"011111_100000",

B"100000_100001",

B"100001_100010",

B"100010_100011",

B"100011_100100",

B"100100_100101",

B"100101_100110",

B"100110_100111",

B"100111_101000",

```
        B"101000_101001",
        B"101001_101010",
        B"101010_101011",
        B"101011_101100",
        B"101100_101101",
        B"101101_101110",
        B"101110_101111",
        B"101111_110000",
        B"110000_110001",
        B"110001_110010",
        B"110010_110011",
        B"110011_110100",
        B"110100_110101",
        B"110101_110110",
        B"110110_110111",
        B"110111_111000",
        B"111000_111001",
        B"111001_111010",
        B"111010_111011",
        B"111011_000000" );

SIGNAL incz,muxz,bcd    : vlbit_1d(5 downto 0);
BEGIN
    bcdout <= bcd;
    pla_table (bcd, incz, trans_tbl);
    muxz <= incz WHEN ('1' = up_enable) ELSE
          bcd ;
    PROCESS
    BEGIN
    WAIT UNTIL prising(clk) or (reset = '1');
```

```
    IF (reset = '1') THEN
            cout <= '0';
    ELSE
      IF  (incz = B"000000") THEN
            cout <= '1';
      ELSE
            cout <= '0';
      END IF;
    END IF;
    END PROCESS;
   dffc_v (muxz, reset, clk, bcd);
END first;
```

****

Gate Usage Summary

****

Cell	Count	Area/Cell	Cell	Count	Area/Cell
MX4000:FDR	43	0.00	MX4000:FDS	2	0.00
X4000:AND2	115	0.25	X4000:AND3	7	0.50
X4000:AND4	5	0.75	X4000:INV	105	0.00
X4000:NAND2	105	0.25	X4000:NAND3	29	0.50
X4000:NAND4	4	0.75	X4000:NOR3	24	0.50
X4000:NOR4	6	0.75	X4000:OR2	189	0.25
X4000:OR3	24	0.50	X4000:OR4	7	0.75
X4000:XOR2	15	0.25			

Total Cells : 680 Total Area : 164.50

Partition, Place and Route Summary

Input XNF Design Statistics (Note 1)

Number of Logic Symbols: 635

Number of Flip Flops: 45

Number of 3-State Buffers: 0

Number of IO Pads: 40

Number of Hard Macros: 0

Number of Nets: 730

Number of Pins: 2228

Equivalent "Gate Array" Gates: 1468 (Note 2)

- From Logic: 1468
- From Random Access Memories: 0
- From Read Only Memories: 0

Partitioned Design Utilization Using Part 4004APC84-6 (Note 3)

Occupied CLBs (Note 4):	Used=143 Max=144 Util=99%
Packed CLBs (Note 4):	Used=126 Max=144 Util=87%
Package Pins (Note 5):	Used=40 Max=96 Util=41%
FG Function Generators:	Used=253 Max=288 Util=87%
H Function Generators:	Used=56 Max=144 Util=38%
Flip Flops (Note 5):	Used=43 Max=480 Util=8%
Memory Write Controls:	Used=0 Max=144 Util=0%
3-State Buffers:	Used=0 Max=384 Util=0%
3-State Buffer Output Lines:	Used=0 Max=48 Util=0%
Address Decoders:	Used=0 Max=144 Util=0%
Address Decoder Output Lines:	Used=0 Max=32 Util=0%

Routing Summary

Number of unrouted connections:	0
Number of pips used:	2337
Number of local lines used:	741
Number of double lines used:	478
Number of long lines used:	178
Number of global lines used:	3
Number of decoder lines used:	0