การออกแบบระบบเชิงตัวเลขด้วยวีแอชดีแอล
DIGITAL SYSTEM DESIGN WITH VHDL.

โดย

นาย สมหวัง ปิยภาณีรัตน์ 35103253

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิศวกรรมศาสตร์
สาขาวิชาวิศวกรรมคอมพิวเตอร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบระบบเชิงตัวเลขด้วยวีแอชดีแอล

ผู้จัดทำ

1. นาย สมหวัง ปียภาณีรัตน์ 35103253

.....อาจารย์ที่ปรึกษา

(อาจารย์ บรรจง ปียธำรง)

การออกแบบวงจรรวมดิจิตอลโดยใช้วีเอชดีแอล

นาย สมหวัง ปิยภาณีรัตน์ 35103253
อาจารย์ บรรจง ปิยธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2537

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการออกแบบระบบเชิงตัวเลขด้วยภาษาวีเฮชดีแอล(VHDL) คำว่า VHDL ย่อมาจาก VHSIC Hardware Description Languge เป็นภาษาคอมพิวเตอร์ระดับสูงใช้ในการเขียนบรรยายฟังก์ชันการทำงานของระบบเชิงตัวเลข โดยที่ตัวภาษาออกแบบมาเพื่อให้บรรยายได้หลายๆลักษณะตามความเหมาะสมได้แก่ การอธิบายพฤติกรรมของระบบ(Behavioral), การไหลของข้อมูลในระบบ(Dataflow) จนถึงอธิบายถึงการเชื่อมต่อกันระหว่าง Component ต่างๆ จนเป็นโครงสร้างของระบบ(Structural) ด้วยประโยชน์ของภาษานี้รวมกับความสามารถของระบบที่ช่วยออกแบบทางอิเลคทรอนิกส์แบบอัตโนมัติ (Electronic Design Automation,EDA)ทำให้การออกแบบระบบเชิงตัวเลขเป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ โครงงานนี้ได้ทดลองทำระบบจำลองการทำงานของเครื่องบันทึกเทป(Tape Recorder Simulater)ขึ้นมา 1 ชิ้น โดยกำหนดคุณสมบัติ(Specification)ขึ้นมา เริ่มต้นด้วยการสร้าง Source Code เพื่ออธิบายฟังก์ชันการทำงานขึ้นมาด้วยภาษา VHDL จากนั้นทำการคอมพายล์ (Compile), ดีบัก (Debug) และจำลองการทำงาน (Simulate) จนได้ฟังก์ชันการทำงานตามต้องการ เสร็จแล้วใช้ความสามารถของซอฟแวร์สังเคราะห์ (Synthesis) ทำการแปลงโมเดล (Model) ที่อยู่ในรูปของ Source Code ภาษา VHDL ให้เป็นผังวงจร (Schematic Diagram) หลังจากนั้นก็นำผังวงจร (Schematic Diagram) ไปสร้างฮาร์ดแวร์เป็นต้นแบบใช้งานจริงบน Field Programmable Gate Array (FPGA)

DIGITAL DESIGN USING VHDL

Mr.Somwang Piyapaneerut

Mr.Bunjong Piyatamrong Advisor

1994

Abstract

This thesis presents the new trend for designing the digital system with 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 tape recorder simulater. First, We created the VHDL model of tape recorder simulator, Compiled and debugged. Until we got a complete VHDL source code for the tape recorder simulator. 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 tape recorder simulator on FPGA.

คำนำ

รายงานฉบับนี้แสดงรายละเอียดขั้นตอนการออกแบบวงจรรวมดิจิตอลโดยใช้ภาษาอธิบายฮาร์ดแวร์ (Hardware Description Language) รวมกับการใช้คอมพิวเตอร์ช่วยในการออกแบบ(CAD/ CAE TOOLS) เพื่อนำเสนอการออกแบบวงจรในแนวใหม่ซึ่งสะดวกรวดเร็วและมีประสิทธิภาพ โดยเครื่องมือและทรัพยากรที่มีอยู่ซึ่งอาจารย์ที่ปรึกษาได้เล็งเห็นประโยชน์อันจะเกิดแก่อุตสาหกรรมของไทยในอนาคต จึงชี้แนวทางและในคำแนะนำตลอดจนส่งเข้ารับการฝึกอบรมในหลายๆ คอร์ส เกี่ยวกับเนื้อหาในการออกแบบ เพื่อให้ผู้เขียนได้มีความรู้เพียงพอที่จะพัฒนาระบบขึ้นมาเองได้ โดยโครงงานนี้ได้ทำการออกแบบระบบจำลองการทำงานของเครื่องบันทึกเทปขึ้นมา ตลอดภาคการศึกษาผู้เขียนได้ศึกษาเนื้อหามากมายเกี่ยวกับตัวภาษา VHDL , เครื่องมือในการพัฒนา (VIEWlogic) และสภาพแวดล้อมในการพัฒนาแบบใหม่ซึ่งไม่คุ้นเคย ซึ่งย่อมต้องมีความผิดพลาดและไม่สมบูรณ์อยู่ในตัวรายงานซึ่งผู้เขียนยินดีรับคำติชมจากผู้อ่านทุกประการ เพื่อที่จะได้ปรับปรุงแก้ไขให้ดีขึ้นต่อไป ในที่นี้ขอขอบคุณอาจารย์บรรจง ปิยธำรง อาจารย์ที่ปรึกษา ซึ่งได้ชี้แนวทางในการทำงานและจัดหาเครื่องมือให้ได้ใช้จนพัฒนางานขึ้นมาได้ หวังว่ารายงานประกอบโครงงานนี้คงมีประโยชน์ต่อท่านผู้ที่สนใจเพื่อเป็นแนวทางในการพัฒนาสิ่งใหม่ๆต่อไป

นาย สมหวัง ปิยภาณีรัตน์ 35103253
ห้อง 3P
ผู้จัดทำ

สารบัญ

บทที่ 1
กล่าวนำ

ปริญญานิพนธ์ฉบับนี้นำเสนอการออกแบบระบบเชิงตัวเลขด้วยภาษา VHDL โดยใช้คอมพิวเตอร์ช่วยในการออกแบบ(Computer Aided Engineering, CAE Tools) เพื่อนำเสนอการออกแบบในแนวใหม่ซึ่งสะดวกรวดเร็วและมีประสิทธิภาพ โดยใช้เครื่องมือและทรัพยากรที่มีอยู่ ภาษา VHDL เป็นภาษาบรรยายการทำงานของฮาร์ดแวร์ เพื่อใช้อธิบายการทำงานของระบบเชิงตัวเลขตัวภาษานั้นประกอบไปด้วยรายละเอียด และส่วนประกอบต่างๆ ซึ่งใช้อธิบายพฤติกรรม (Behavioral) หรือโครงสร้าง(Structural) ของระบบ โดยพิจารณาถึงฐานเวลา(Timing) ของระบบเป็นหลักเนื่องจากว่าระบบที่เราจะออกแบบโดย VHDL นั้น ผู้ออกแบบไม่จำเป็นต้องรู้ถึงวงจรภายในว่ามีอะไรต่อกันอย่างไรบ้าง เพียงแต่กำหนด Input, Output, และฟังก์ชั่นการทำงานของระบบ เขียนเป็นโปรแกรมแสดงการไหลของข้อมูล(Dataflow), การทำงานของระบบ(Behavioral) หรือโครงสร้างของระบบ(Structural)ขึ้นมา จากนั้นก็ทำการ Compile และ Simulate Model ที่ออกแบบมา เพื่อทำการวิเคราะห์ดูฟังก์ชั่นฐานเวลาของระบบว่าตรงตามต้องการหรือไม่ ถ้ามีการแก้ไขอย่างไรก็ทำการแก้ไข Source Code ใหม่ แล้ว Compile และ Simulate ซ้ำๆ จนกว่าจะได้ Model ที่มีฐานเวลาของระบบตามต้องการ ซึ่งจะเห็นว่ามีความง่าย และสะดวกรวดเร็วกว่าเมื่อก่อนมาก เพราะไม่ต้องเสียเวลากับการสร้างวงจรทางฮาร์ดแวร์จริงๆขึ้นมาทดสอบ ซึ่งทำให้เสียเวลาและค่าใช้จ่ายสูง Model ที่ออกแบบโดยใช้ VHDL สามารถทำการสังเคราะห์(Synthesis) เพื่อให้ได้ผังวงจรและนำเอาไปสร้างเป็นวงจรจริงๆได้ โดยอาจนำเอาไปสร้างเป็น ASIC หรือนำไป Burn ลง EPLD หรือ FPGA เพื่อนำไปใช้งานจริงๆต่อไป ประโยชน์ของภาษา VHDL ใช้กันมากในการออกแบบ Chip ASIC (Application Specific Integrated Circuit) หรือแม้กระทั่งการออกแบบไมโครโปรเซอร์ในปัจจุบันก็ใช้ภาษาVHDL ในการออกแบบ จะเห็นได้ว่าภาษา VHDL มีประโยชน์อย่างยิ่งในการออกแบบระบบเชิงตัวเลขในปัจจุบันและมี Software หลายตัวซึ่งสนับสนุนภาษา VHDL ตั้งแต่การCompile, Simulate, Synthesisได้แก่ Mentor Graphics ,VIEWlogic เป็นต้น Software CAE ที่ใช้ในการพัฒนาโครงงานนี้ขึ้นมาคือ VIEWlogic ตั้งแต่การเขียน Source Code ของภาษา VHDL, การ compile, การทดสอบและจำลองฟังก์ชันการทำงาน เมื่อได้ Source Code ที่สมบูรณ์ของ Model แล้ว ก็ทำการแปลงSource Codeให้อยู่ในรูปของผังวงจร เพื่อที่จะไปสร้างบน FPGA ซึ่งผู้จัดทำได้รับการแนะนำและความช่วยเหลือเป็นอย่างดีจากอาจารย์ที่ปรึกษาซึ่งท่านได้ชี้แนวทางการทำงานตลอดจนส่งเข้ารับการฝึกอบรมเกี่ยวกับเนื้อหาในการออกแบบ เพื่อให้ผู้จัดทำ

ได้มีความรู้เพียงพอที่จะพัฒนาระบบขึ้นมาเองได้ โครงงานนี้ได้ทดลองพัฒนาระบบจำลองการทำงานของเครื่องบันทึกเทป ตลอดภาคการศึกษาผู้จัดทำได้ศึกษาเนื้อหามากมายเกี่ยวกับตัวภาษา VHDL , ซอฟแวร์ CAE Tools ที่ใช้ในการพัฒนาได้แก่ VIEWlogic และ XACT Development ซึ่งล้วนเป็นสภาพแวดล้อมในการพัฒนาแบบใหม่ ผู้จัดทำยอมรับว่าต้องมีความผิดพลาดและไม่สมบูรณ์อยู่ในตัวรายงานและโครงงานซึ่งผู้เขียนเองยินดีรับคำติชมจากท่านผู้อ่านทุกประการเพื่อที่จะได้ปรับปรุงแก้ไขให้ดีขึ้นต่อไป ในที่นี้ขอขอบคุณท่านอาจารย์บรรจง ปิยธำรง อาจารย์ที่ปรึกษา ซึ่งได้ชี้แนวทางในการทำงานและจัดหาเครื่องมือต่างๆให้ได้ใช้จนพัฒนางานขึ้นมาได้ หวังว่ารายงานเล่มนี้จะเป็นประโยชน์ต่อท่านผู้ที่สนใจเพื่อเป็นแนวทางประดิษฐ์สิ่งใหม่ๆขึ้นมาต่อไป

นาย สมหวัง ปิยภาณีรัตน์

ผู้จัดทำ

บทที่ 2
ขั้นตอนการออกแบบ

2.1 ขั้นตอนที่ใช้ในการออกแบบ

ขั้นตอนที่ใช้ในการออกแบบส่วนใหญ่มีดังนี้

2.1.1 ความคิดริเริ่มและวิเคราะห์ระบบที่จะออกแบบ (Design Idea)

2.1.2 การออกแบบตัวบรรยายพฤติกรรม (Behavioral Design)

2.1.3 การออกแบบทางวงจรตรรก และการสังเคราะห์ (Logic Design and Synthesis)

2.1.4 การออกแบบทางกายภาพ และการทำระบบต้นแบบ (Physical Design and Prototyping)

2.1.5 การจำลอง และการทดสอบ (Simulation and Testing)

2.1.1 ความคิดริเริ่มและวิเคราะห์ระบบที่จะออกแบบ

ในขั้นนี้เราจะต้องทำการกำหนดสิ่งต่างๆ ที่จะนำไปใช้ในขั้นต่อไปดังนี้

2.1.1.1 กำหนดคุณสมบัติของระบบเชิงตัวเลข เช่น ระบบนี้ให้ผลลัพธ์อะไรบ้าง เป็นต้น

2.1.1.2 เขียนแผนผังการติดต่อของระบบย่อย(System Block Diagram)ที่ต้องใช้

2.1.1.3 กำหนดสัญญาณที่ใช้ในการติดต่อ เช่น ชื่อ, ลักษณะที่ใช้ติดต่อ, จำนวน เป็นต้น

2.1.1.4 เขียนอธิบายการทำงานอย่างคร่าวๆ

2.1.2 การออกแบบตัวบรรยายพฤติกรรม

จากขั้นตอนที่ผ่านมาเรานำเอาข้อกำหนดต่างๆที่ได้มาทำการออกแบบในขั้นนี้โดย

2.1.2.1 เขียนประกาศเอ็นติตี้(Entity Declaration) จากการกำหนดสัญญาณในข้อ 2.1.1.3

2.1.2.2 เขียนผังการทำงาน(Flow Chart) โดยละเอียดในแต่ละส่วนของการทำงาน

2.1.2.3 เขียนฐานเวลา(Timing Diagram) จากผังการทำงาน โดยดูความสัมพันธ์ระหว่างสัญญาณต่างๆ ว่าสัญญาณใดเมื่อเปลี่ยนแปลงจะมีผลทำให้สัญญาณอื่นๆมีการเปลี่ยนแปลง และเปลี่ยนแปลงไปอย่างไร

2.1.2.4 เขียนผังสถานะ(State Diagram) จากฐานเวลา

2.1.2.5 เขียนตารางการโอนถ่ายสถานะ(Transition Table) จากผังสถานะ

2.1.2.6 ทำการกำหนดสถานะ ซึ่งประกอบด้วย

2.1.2.6.1 รหัสเลขฐานสอง

2.1.2.6.2 การเปลี่ยนแปลงของสัญญาณที่เกิดขึ้น ณ. สถานะนั้นๆ

2.1.2.7 เขียนประกาศสัญญาณที่ถูกใช้ภายในสถาปัตยกรรมของระบบ (Architecture Declaration)

2.1.2.8 เขียน Source Code

2.1.3 การออกแบบทางวงจรตรรก และการสังเคราะห์

ในขั้นตอนนี้ เป็นการใช้เครื่องมือทางซอฟท์แวร์ในการออกแบบ อันได้แก่ ViewSynthesis ของ VIEWlogic

2.1.3.1 ViewSynthesis Inputs

มี input อยู่ 2 แบบ คือ

2.1.3.1.1 Source Code จากข้อ 2.1.2.8 ซึ่งมีนามสกุลเป็น .vhd

2.1.3.1.2 ส่วนประกอบที่ถูกนิยามในรูปแบบฟังก์ชั่นจาก source technology library [library.sml]

2.1.3.2 ViewSynthesis Outputs

ViewSynthesis จะสังเคราะห์ inputs และผลิต wirelist ที่จะถูกใช้ในการจำลองตัวระบบที่ทำการออกแบบ และสร้างผังวงจร(schematic)ของระบบที่ทำการออกแบบ

โดย Viewgen จะใช้ wirelist เป็นสารสนเทศ(information)เพื่อที่จะกำเนิดแฟ้มผังวงจร ViewSynthesis ยังจะให้กำเนิดแฟ้มบันทึก(log file), รายงาน(แฟ้มที่มีนามสกุลเป็น .rpt) ภายในรายงานประด้วย

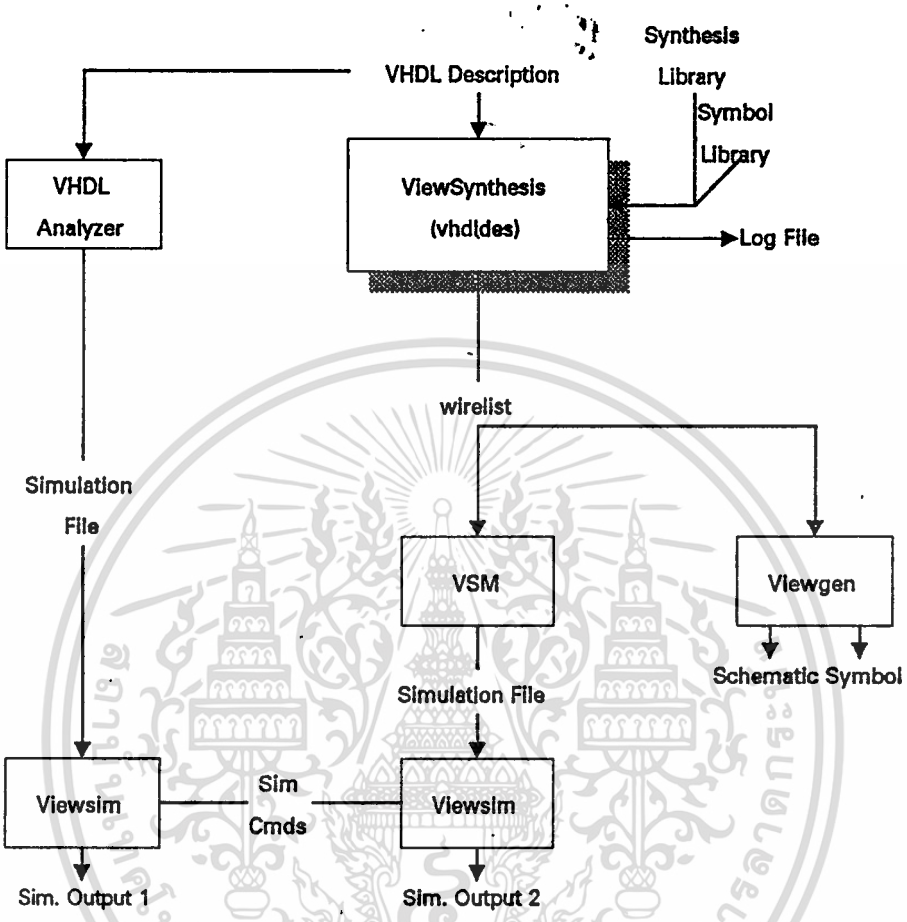
2.1.3.3.1 เวลาที่ใช้

2.1.3.2.2 จำนวนเกท(gate) ที่ถูกใช้

2.1.3.2.3 สถิติของ wirelist

2.1.3.2.4 Output drive

2.1.3.2.5 สารสนเทศของ Input loading



รูปที่ 2-1 Design Flow

2.1.3.3 ตัววิเคราะห์ VHDL (VHDL Analyzer)

ในส่วนที่ 1 ของรูปที่ 2-1 ตัววิเคราะห์ VHDL จะรับSource Codeจากข้อ 2.1.2.8 ซึ่งมีนามสกุลเป็น .vhd และใช้เป็นสารสนเทศในการสร้างแฟ้มทั้งสามดังตารางที่ 2-1

แฟ้ม	คำอธิบาย
แฟ้มที่มีนามสกุลเป็น .lis	ข้อความภายในแสดงโปรแกรมภาษา VHDL และโน๊ตความผิดพลาดที่เกิดขึ้นทาง syntax
แฟ้มที่มีนามสกุลเป็น .vli	เป็นแฟ้มภาษา VHDL ที่แปลงรูปแบบเพื่อใช้ในการผลิตแฟ้มที่จะใช้จำลองการทำงาน

แฟ้ม	คำอธิบาย
แฟ้มที่มีนามสกุลเป็น .vsm	เป็นแฟ้มที่อยู่ในรูปแบบที่ Viewsim ต้องการใช้ในการจำลองการทำงาน

ตารางที่ 2-1 VHDL Analyzer Created Files

2.1.3.4 Viewsim Outputs

ครั้งแรกที่เราจะจำลองการทำงาน Viewsimจะผลิตแฟ้ม viewsim.log ให้เราทำการเปลี่ยนชื่อเป็น vhdl.out แล้วใช้ Viewsim อีกครั้งจำลองการทำงานของเกทต่างๆที่ถูกผลิตโดย VHDLDes และ จะเป็นการผลิตแฟ้ม viewsim.log อีกให้เปลี่ยนชื่อแฟ้มเป็น wire.out

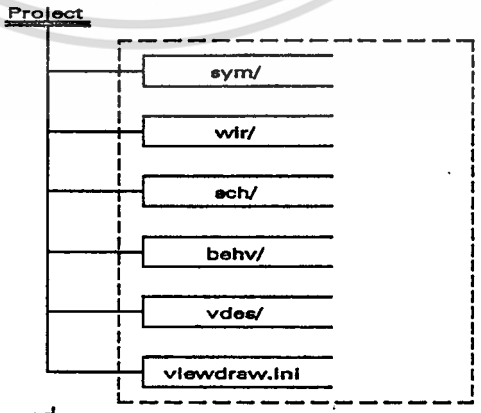
แฟ้ม	คำอธิบาย
vhdl.out	ข้อความภายในเป็นผลลัพธ์ของการจำลองบนพื้นฐานของสารสนเทศจาก ตัววิเคราะห์ VHDL ซึ่งแสดงอยู่ในส่วนที่ 1 ของรูปที่ 2-1
wire.out	ข้อความภายในเป็นผลลัพธ์ของการจำลองบนพื้นฐานของสารสนเทศจาก ViewSynthesis ซึ่งแสดงอยู่ในส่วนที่ 2 ของรูปที่ 2-1

ตารางที่ 2-2 Viewsim Outputs

2.1.3.5 แฟ้ม ViewSynthesis

2.1.3.5.1 โครงสร้างแฟ้ม(File Structure)

โครงสร้างแฟ้มของ ViewSynthesis ดังแสดงในรูปที่ 2-2



รูปที่ 2-2 Directory Contents

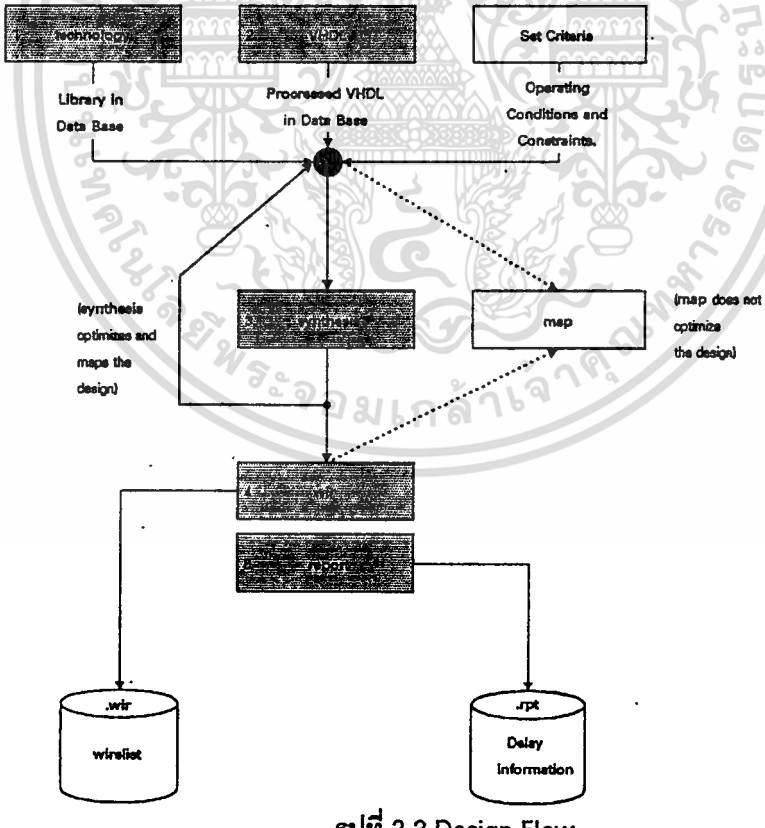
2.1.3.5.2 Directory ที่บรรจุอยู่(Directory Contents)

Directory	Contents...
sym	บรรจุ Symbolic ต่างๆของ model ที่ประกอบขึ้น
wir	บรรจุ wirelist ที่เชื่อมต่อ element ต่างๆในวงจรด้วย input และ output ที่ออกแบบไว้ และยังบรรจุ wirelist ของอุปกรณ์ทุกตัว
sch	บรรจุผังวงจรของระบบที่ออกแบบ และของแต่ละอุปกรณ์
behv	บรรจุสารสนเทศของตัวบรรยาย VHDL. ที่ถูกเก็บในแฟ้มนามสกุล .vhd
vdes	บรรจุสารสนเทศของ wirelist ทั่วๆไปที่ถูกใช้โดย ViewSynthesis เพื่อใช้สังเคราะห์ระบบที่ทำการออกแบบ

ตารางที่ 2-3 Directory Contents

2.1.3.6 ขั้นตอนการใช้งาน ViewSynthesis มีดังนี้

2.1.3.6.1 การไหลผ่านของการออกแบบเบื้องต้น (Basic Design Flow) รูปที่ 2-3 แสดงการไหลผ่านของการออกแบบ เพื่อที่จะนำเข้าสู่ ViewSynthesis โดยส่วนที่แรเงาทึบ จะแสดงขั้นตอนที่จำเป็นต้องทำ



รูปที่ 2-3 Design Flow

2.1.3.6.2 คำสั่ง technology

ใช้คำสั่ง technology เพื่อที่จะอ่าน technology library เข้าสู่ ViewSynthesis ตัวอย่างเช่น อ่านแฟ้ม lsi10k ที่เป็นแฟ้มของ technology library โดยใช้คำสั่งดังนี้

VHDLDes>technology lsi10k

ถ้าเราไม่ใส่ชื่อแฟ้ม ViewSynthesis จะแสดงแฟ้มของ technology library ที่มีอยู่ออกมา โดย ViewSynthesis จะหาจาก directory ปัจจุบัน และ WDIR path สำหรับทุกๆแฟ้มที่มีนามสกุล .lib

เมื่อ ViewSynthesis ทำการอ่านแฟ้มจบ จะแสดงข่าวสารออกมาว่า "Library '*library name*' is in Database"

2.1.3.6.3 คำสั่ง vhdl

ใช้คำสั่ง vhdl เพื่อที่จะอ่านตัวภาษา VHDL.ที่ทำการออกแบบไว้เข้าสู่ ViewSynthesis ตัวอย่างเช่น เราจะนำแฟ้ม seven เข้ามา ให้ใช้คำสั่งดังนี้

VHDLDes>vhdl seven

ถ้าเราไม่ใส่ชื่อแฟ้ม ViewSynthesis จะแสดงแฟ้มของภาษา VHDL. ที่มีอยู่ออกมา โดย ViewSynthesis จะหาจาก directory ปัจจุบัน และ ./ behv/ path สำหรับทุกๆแฟ้มที่มีนามสกุล .vhd.

ViewSynthesis จะอ่านตัวที่ทำการออกแบบตามลำดับ เป็นขั้นต่อขั้น (step-by-step) และจุดที่สิ้นสุดในแต่ละส่วน และยังจะแสดงจำที่ผิดพลาด และคำเตือน(ถ้ามี) บนหน้าจอ

2.1.3.6.4 คำสั่ง synthesize

คำสั่ง synthesize จะทำการ optimize และ map ตัวที่ออกแบบ ขั้นตอนการทำ optimize เป็นการรวมเอาตัวที่ออกแบบไว้กับ technology library

และ operating parameters ที่ตัวออกแบบใช้กับส่วนประกอบต่างๆ ให้ใช้คำสั่งดังนี้

VHDLDes>synthesize

2.1.3.6.5 คำสั่ง wire

wirelist คือสิ่งที่ถูกสร้างจากผลลัพธ์ของการสังเคราะห์ ให้ใช้คำสั่งดังนี้

VHDLDes>wire

แฟ้มของ wirelist จะมีนามสกุลเป็น .1 อยู่ภายใน directory ./wire/

2.1.3.6.6 คำสั่ง report

รายงานที่ถูกแสดงออกทางจอ เมื่อทำการสังเคราะห์จบสมบูรณ์แล้ว เราสามารถเก็บรายงานเหล่านี้ลงแฟ้ม โดยใช้คำสั่ง

VHDLDes>report

2.1.4 การออกแบบทางกายภาพ และการทำระบบต้นแบบ

ในการออกแบบทางกายภาพนั้นจะเป็นการใช้ tool ที่มีชื่อว่า ViewDraw ของ VIEWlogic กับ XACT ของ XILINX ส่วนการทำระบบต้นแบบจะเป็นการประกอบอุปกรณ์ต่างๆเข้าเป็นระบบที่จะนำไปทดสอบหรือใช้งานจริง

2.1.4.1 Viewgen

จาก ViewSynthesis เราจะใช้ Viewgen ทำการสร้างผังวงจรเพื่อใช้ในการทำการจำลอง และ port mapping

2.1.4.2 Port Mapping

port mapping จะเป็นการนำ I/O port มาจับคู่เข้ากับขาของ FPGA หรือเรียกอีกอย่างว่า IOB Configuration.

2.1.4.2.1 IOB Pad Options

Pad	Description
IPAD	Input pad
OPAD	Output pad
BPAD	Bidirectional pad
UPAD	Unbounded pad

ตารางที่ 2-4

2.1.4.2.2 IOB Buffer Options

Buffer	Description
IBUF	Input buffer
OBUF	Output buffer
OBUFT	Output buffer with 3-state control

ตารางที่ 2-5

2.1.4.2.3 IOB Latch Options

Latch	Description
INFF	Input flip-flop
INLAT	Input latch
OUTFF	Output flip-flop
OUTFFT	3-state output flip-flop

ตารางที่ 2-6

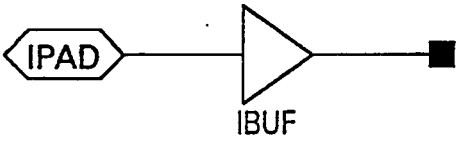
2.1.4.2.4 ขั้นตอนการทำ port mapping

ในการทำ port mapping นั้นเราจะใช้ ViewDraw ในการทำ โดยมีขั้นตอนดังนี้

2.1.4.2.4.1 เราจะต้องหาขาที่ไม่ได้ถูกใช้งานของ FPGA ก่อน โดยดูจาก Data Sheet ของ FPGA ในรุ่นที่จะใช้ ซึ่งขาที่อนุญาตให้ใช้งานทั่วไปได้ จะเขียนว่า "I/O"

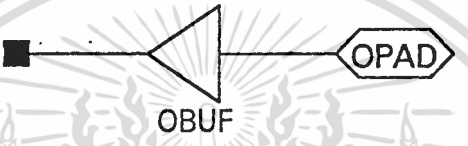
2.1.4.2.4.2 เรียก ViewDraw ขึ้นมา แล้วเลือกผังวงจรที่จะทำ port mapping

2.1.4.2.4.3 Input Port Mapping เราจะทำการ map ได้ดังรูปที่ 2-4



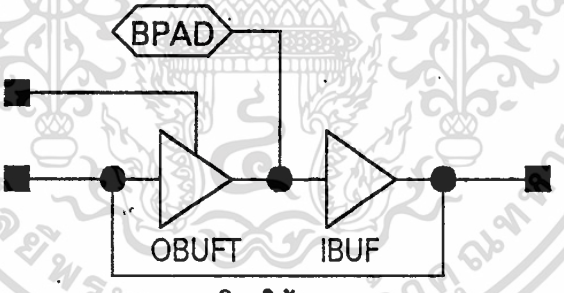
รูปที่ 2-4 Input Port Mapping โดยใช้ input buffer

2.1.4.2.4.4 Output Port Mapping เราจะทำการ map ได้ดังรูปที่ 2-5



รูปที่ 2-5 Output Port Mapping โดยใช้ output buffer

2.1.4.2.4.4 Output Port Mapping เราจะทำการ map ได้ดังรูปที่ 2-6



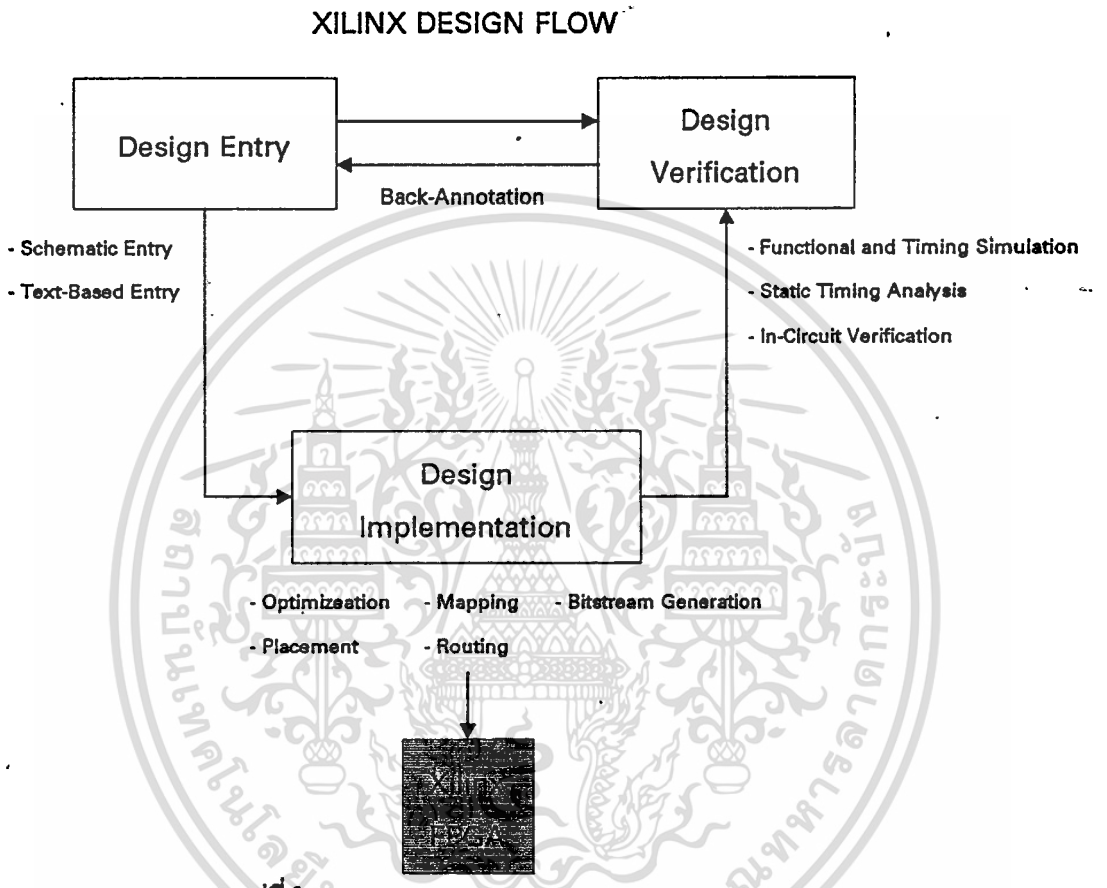
รูปที่ 2-6 Bidirection Port Mapping โดยใช้ output 3-state buffer และ input buffer

2.1.4.2.4.5 ที่ pad เราจะต้องใส่หมายเลขขา(pin number)ของ FPGA โดยการ add attribute แล้วใส่ค่า LOC=P*pin_No*

2.1.4.3 การแปลงรูปจากแฟ้มของ ViewDraw สู่แฟ้ม bit stream ของ FPGA ในการแปลงรูปนั้นเราจะใช้ tool ที่มีชื่อว่า XACT ของ XILINX ทำการแปลงรูปเพื่อจะนำไปโปรแกรมตัว FPGA

2.1.4.3.1 การไหลผ่านในการออกแบบของ XILINX

รูปที่ 2-7 แสดงการไหลผ่านในการออกแบบของ XILINX ซึ่งประกอบไปด้วย 3 ส่วนใหญ่ๆ คือ Design Entry, Design Implementation, และ Design Verification



รูปที่ 2-7 The XACT Development System Design Flow

2.1.4.3.2 Design Entry และ Convertion

เป็นการนำตัวระบบที่ทำการออกแบบเข้าสู่ XACT และทำการแปลงให้อยู่ในรูปแบบแฟ้มชนิด netlist แฟ้มที่สามารถนำเข้ามาได้มี แฟ้มที่เป็นผังวงจร และแฟ้มที่เป็นประเภท text ได้แก่ แฟ้มชนิด Boolean หรือ state expression กับแฟ้มภาษา VHDL โปรแกรมที่ถูกใช้งานในส่วนนี้ได้แก่

2.1.4.3.2.1 XMake - ทำการแปลงแฟ้มชนิดผังวงจรเข้าสู่ LCA และแฟ้มชนิด BIT

2.1.4.3.2.2 MemGen - ทำการสร้าง RAMs หรือ ROMs ของทุกๆขนาดที่ใช้สำหรับชิป XC4000

2.1.4.3.2.3 XNFCvt - ทำการแปลง XNF netlist จากเวอร์ชั่น 4 ไปเป็นเวอร์ชั่น 2 หรือ 1 .

2.1.4.3.2.4 XNFUpd - โปรแกรมนี้จะผลิตแฟ้มเวอร์ชั่น 4 ของ XNF จากเวอร์ชั่น 1 หรือ เวอร์ชั่น 2 XNFUpd จะ upgrade จากแฟ้ม XNF ของชิป XC3000 เป็นแฟ้ม XNF ของชิป XC4000

2.1.4.3.3 Design Implementation

2.1.4.3.3.1 XNFMerge - ทำการรวมแฟ้มชนิด XNF กับชนิด MAP ให้เป็นแฟ้มชนิด XNF เพียงแฟ้มเดียว

2.1.4.3.3.2 XNFDRC - เป็นการทำการตรวจสอบอย่างง่ายๆในขบวนการคำนวณ

2.1.4.3.3.3 XNFMap - ทำการ map วงจร logic ที่ถูกกำหนดโดยแฟ้มชนิด XNF เข้าสู่ LCA elements (CLBs, IOB, and TBUFs)

2.1.4.3.3.4 Map2LCA - ทำการเปลี่ยนชนิดแฟ้มของ MAP เป็นแฟ้มชนิด LCA เพื่อเป็น input ให้ APR

2.1.4.3.3.5 APR (Automatic Place and Route) - ทำการ Places and Routes ของชิป XC2000 และ XC3000

2.1.4.3.3.6 PPR (Partition, Place and Route) - Partitions, Places, and Routes ของชิป XC4000

2.1.4.3.3.7 MakeBits - จะสร้าง configuration bitstream ให้กับ LCA

2.1.4.3.3.8 MakePROM - จะทำการแปลงแฟ้มของ configuration bitstream(BIT) ไปเป็นแฟ้มที่จะนำไป download ลงสู่ PROM และยังนำไปใช้ในการทำ daisy chain ของ LCA device

2.1.4.3.4 Design Verification

เป็นการทำการจำลอง, วิเคราะห์หา static-timing, และ ทำ in-circuit verification โดยใช้โปรแกรมต่อไปนี้

2.1.4.3.4.1 XDelay - จะทำการวิเคราะห์หา static-timing แล้วรายงานผลถึงรายละเอียดต่างๆของฐานเวลาที่เกี่ยวข้องกับการออกแบบ และยังสามารถใช้สำหรับการวิเคราะห์หา performance อีกด้วย

2.1.4.3.4.2 LCA2XNF - ทำการแปลงแฟ้มของ LCA ไปเป็นแฟ้มชนิด XNF(Xilinx Netlist Format) เพื่อใช้ในการจำลองหาฐานเวลา

2.1.4.3.4.3 BAX - จะสร้างแฟ้มชนิด XNF ที่มีสารสนเทศของค่าที่หน่วงเวลา

2.1.4.3.4.4 XDE (XACT Design Editor) - ใช้ในการทำการออกแบบ LCA ด้วยตัวผู้ออกแบบเอง

2.1.5 การจำลอง และการทดสอบ

ในการจำลองการทำงานและการทดสอบนั้น เราจะใช้ tool ที่มีชื่อว่า ViewSim ของ VIEWlogic ใช้ในการจำลองหาค่าของฐานเวลา เพื่อที่จะนำไปเปรียบเทียบกับค่าของฐานเวลาที่ได้ออกแบบไว้ การทำการจำลองเราสามารถจะทำการจำลองได้ในทุกขั้นตอนของการออกแบบตั้งแต่ การออกแบบตัวบรรยายภาษา VHDL จนถึงการทำเครื่องต้นแบบ

2.1.5.1 ViewSim Input

แฟ้มข้อมูลที่เป็น input ให้กับ ViewSim คือ

2.1.5.1.1 แฟ้มข้อมูลที่มีนามสกุลเป็น .vsm ซึ่งได้มาจากโปรแกรม VSM ของ VIEWlogic ภายในแฟ้มนี้จะประกอบไปด้วย wirelist ที่ ViewSim ต้องใช้

2.1.5.1.2 แฟ้มข้อมูลที่มีนามสกุล .vsm และ .vli ได้มาจาก VHDL Analyzer

2.1.5.1.3 แฟ้มข้อมูลที่มีนามสกุล .cmd ซึ่งเป็นคำสั่งที่ผู้ออกแบบหรือผู้ใช้ทั่วไปใช้เป็นคำสั่งให้ ViewSim ทำงาน

2.1.5.2 ViewSim Output

สิ่งที่ ViewSim สร้างเป็น outputs คือ

2.1.5.2.1 แฟ้มข้อมูลที่มีนามสกุล .wfm เป็น waveform data stream สำหรับใช้ใน ViewTrace

2.1.5.2.2 ค่า back-annotation ที่จะส่งให้ ViewDraw

2.1.5.2.3 ข้อมูลที่แสดงออกทางหน้าจอ

2.1.5.2.4 ข้อมูลที่แสดงออกทางเครื่องพิมพ์

2.1.5.2.5 แฟ้มที่จดบันทึกการจำลองการทำงาน

2.1.5.3 การจำลองการทำงานทาง hardware

ในการจำลองการทำงานทาง hardware นี้ จะเป็นการต่อวงจรระหว่าง FPGA board กับ application board แล้วใช้โปรแกรม XChecker ในการทำ FPGA configuration จากนั้นให้จำลองการทำงาน โดยการให้ inputs ที่เป็นไปได้ทั้งหมด แล้วดู outputs เป็นไปตามต้องการหรือไม่

2.2 สรุป

ในการออกแบบระบบเชิงตัวเลขนั้น โดยหลักใหญ่ๆแล้ว มีดังนี้

2.2.1 ความคิดริเริ่มและวิเคราะห์ระบบที่จะออกแบบ (Design Idea)

2.2.2 การออกแบบตัวบรรยายพฤติกรรม (Behavioral Design)

2.2.3 การออกแบบทางวงจรตรรก และการสังเคราะห์ (Logic Design and Synthesis)

2.2.4 การออกแบบทางกายภาพ และการทำระบบต้นแบบ (Physical Design and Prototyping)

2.2.5 การจำลอง และการทดสอบ (Simulation and Testing)

ซึ่งในแต่ละขั้นตอนนั้น ก็จะเป็นการใช้ tools ในการออกแบบทำให้สะดวก และหาข้อผิดพลาดได้ง่าย รวมทั้งยังสามารถทำการดัดแปลงแก้ไข (Modify) ในขั้นตอนใดๆก็ได้

บทที่ 3
ภาษา VHDL

บทนี้จะแนะนำประวัติความเป็นมาอย่างย่อๆ ของภาษา VHDL(VHSIC Hardware Description Langauge)ว่ามีความเป็นมาอย่างไร

3.1 ภาษา VHDL

VHDL ย่อมาจากคำว่า VHSIC Hardware Description Langauge(VHSIC ย่อมาจาก Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง(High Level Langauge) ซึ่งใช้อธิบายการทำงานของระบบเชิงตัวเลข สามารถใช้อธิบายฟังก์ชันการทำงานได้หลายๆระดับ ตั้งแต่ระดับผังงานจนถึงระดับวงจรเกท ความซับซ้อนของระบบสามารถจะเขียนได้ตั้งแต่ระดับวงจรเกทประกอบกันจนเป็นระบบที่สมบูรณ์ รูปแบบของภาษา VHDL นั้น จะประกอบไปด้วย 2 ส่วนใหญ่ๆได้แก่ ส่วนของ Sequencial Langauge และ Concurrent Langauge การโปรแกรมด้วยภาษา VHDL สามารถจะเขียนได้ทั้ง 2 รูปแบบรวมกัน เพราะในการทำงานของระบบใดๆย่อมจะมีการทำงานในแบบ Sequential และ Concurrent อยู่รวมกัน นอกจากนี้ตัวภาษา VHDL ยังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยๆ เข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้ ตัวภาษา VHDL นอกจากจะกำหนดรูปแบบไวยากรณ์ (Syntax)ของตัวภาษาแล้ว ยังมีการตรวจสอบความหมายของตัวภาษาว่าจะทำการจำลองการทำงานได้หรือไม่ เพราะว่าโปรแกรมที่เขียนโดย VHDL ต้องผ่านการจำลองการทำงานเพื่อตรวจสอบดูการทำงาน ฉะนั้นในการ Compile จะมีการตรวจสอบทั้ง Syntax และ Simulation Semantics อย่างไรก็ตามตัวภาษานั้นถึงจะมีความซับซ้อนมากมายในรูปแบบและกฎเกณฑ์ของภาษาแต่การเรียนรู้เพียงบางส่วนของตัวภาษาก็สามารถนำมาใช้งานได้โดยไม่จำเป็นจะต้องศึกษารายละเอียดทั้งหมด เนื่องจาก ตัวภาษา VHDL ออกแบบมาให้ใช้ออกแบบได้ตั้งแต่วงจรที่มีขนาดเล็กจนถึงวงจรที่มีขนาดใหญ่และซับซ้อน

3.2 ประวัติความเป็นมาของภาษา VHDL

ความต้องการภาษานี้เริ่มจากโครงการ VHSIC ของ Department OF Defence(DOD)ของสหรัฐอเมริกาเนื่องจากมีบริษัทที่สร้าง VHSIC Chip หลายบริษัทได้ร่วมโครงการที่จะพัฒนาในขณะนั้นหลายๆ บริษัทใช้ภาษา VHDL ซึ่งแตกต่างกัน ในการที่จะอธิบายการทำงาน Chip ของตนด้วยเหตุนี้ทำให้เกิดความแตกต่าง แต่ละบริษัทไม่สามารถแลกเปลี่ยนเทคโนโลยี่ให้กันและกันได้

ทำให้ DOD เกิดปัญหาในการที่จะพัฒนาและซ่อมบำรุงในภายหลัง จึงเกิดความต้องการภาษา VHDL ซึ่งเป็นมาตราฐานในการที่จะอธิบายถึงตัวที่ทำการออกแบบนั้นๆ 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, มหาวิทยาลัยและ DOD ก็มีการปรับปรุงและจดมาตราฐานใหม่ IEEE ในปี 1987 อีกครั้งซึ่งเป็นที่รู้จักกันในชื่อของ IEEE STD 1076-1987 หลังจากนั้นยายน1988 บริษัทใดๆ ที่ทำการพัฒนา ASIC CHIP ใน DOD ของอเมริกาต้องส่งตัว VHDL Model พร้อมกับ TEST BENCH ตามมาตราฐานที่ได้กำหนดเอาไว้

3.3 ความสามารถของภาษา VHDL (CAPABILITY)

หัวข้อดังต่อไปนี้คือความสามารถหลักๆของตัวภาษา VHDL

3.3.1 ตัวภาษา VHDL สามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิต CHIP กับผู้ออกแบบ(CAD Tools)

3.3.2 ใช้เป็นการสื่อกลางในการแลกเปลี่ยนสื่อสารระหว่าง CAE และ CAD Tools เช่น Source Codeของภาษา VHDL สามารถใช้Compile โดยใช้Compiler&Simulator ได้หลายตัวแตกต่างกัน

3.3.3 ภาษาวีเอชดีแอล สนับสนุนการออกแบบ Top-Down design และ Bottom-Up design หรือผสมกันทั้ง 2 แบบ

3.3.4 ตัวภาษาวีเอชดีแอลเป็น Generic คือไม่อิงเทคโนโลยี่อันใดอันหนึ่ง(แต่สามารถอิงเทคโนโลยี่ใดก็ได้และในขณะเดียวกัน ก็สามารถสนับสนุนหลาย ๆ เทคโนโลยี่)

3.3.5 สนับสนุนการออกแบบทั้งระบบ Synchronuos และ Asynchronous

3.3.6 สนับสนุนการออกแบบระบบเชิงตัวเลขในหลายๆเทคนิค เช่น Finite State Machine, Algorithmic หรือ Boolean Equation

3.3.7 ตัวภาษา VHDL สามารถอ่านและทำความเข้าใจได้โดยมนุษย์(Human Readable)

3.3.8 ภาษา VHDL เป็นมาตราฐานรับรองโดย IEEE และ ANSI ทำให้ Model ที่ออกแบบโดย VHDL สามารถเคลื่อนย้าย(Portable) ไปยังระบบใดๆ ก็ได้และสามารถนำกลับมาใช้ใหม่ได้ (Reuse)

3.3.9 ภาษา VHDL สนับสนุนรูปแบบการเขียนถึง 3 รูปแบบ ได้แก่ Behavioral Stye, Structural Stye, Dataflow Stye หรือสามารถเขียนรวมกันทั้ง 3 รูปแบบ(Mixed Style)

3.3.10 สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของ Component, Function. Procedure และ Package

3.3.11 ไม่จำเป็นต้องศึกษา Software Simulator เพราะ Simulation Model สามารถเขียนได้โดยใช้ภาษา VHDL เช่นกัน

3.3.12 สามารถเขียน Model ได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของ Model (ขึ้นอยู่กับ Software Tools)

3.3.13 สามารถอธิบาย Parameter ที่เกี่ยวกับฟังก์ชั่นทางด้านเวลา เช่น Propagation Delay, Min-Max Delay, Setup, Holding Time, Spike Detection สามารถอธิบายได้ภายในตัวภาษา

3.3.14 GENERICS ช่วยให้เราสามารถสร้าง Parameter ของตัวที่ทำการออกแบบ

3.3.15 Model ที่สร้างด้วยภาษา VHDL นั้น ไม่เพียงแต่จะอธิบายฟังก์ชั่นการทำงานเท่านั้น แต่ยังสามารถอธิบายถึงรายละเอียดของตัว Model เช่น Total Area และ Speed ของ Model

3.3.16 ภาษา VHDL เป็นมาตรฐานใช้โดยบริษัทและผู้ออกแบบหลายๆแห่ง ฉะนั้นจึงเป็นการง่ายที่จะทำความเข้าใจ ถึงแม้ว่าจะมาจากแหล่งต่างๆ

3.3.17 Model ที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะว่าตัวแปลภาษาได้ตรวจสอบไวยากรณ์ทางด้าน Simulation Semantic ไว้ด้วย

3.3.18 การอธิบาย Model ด้วย Behavioral Style สามารถสังเคราะห์ไปเป็นระดับวงจรเกทได้ถ้าทำตามกฎของ Synthesis Guideline

3.3.19 มีความสามารถที่ให้เราออกแบบข้อมูลชนิดใหม่ๆได้ทำให้ VHDL Model เป็นการออกแบบในระดับสูงที่ไม่ต้องคำนึงถึงว่าจะสร้างตัว Model นั้นขึ้นมาได้อย่างไร

3.4 หลักการสร้างโมเดลโดยภาษา VHDL(General VHDL Modelling Principles)

3.4.1 VHDLเป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้(Human Readable) ซึ่งช่วยในการสร้างและออกแบบวงจรของระบบเชิงตัวเลข (Digital Hardware System, Circuit Board) และอุปกรณ์ต่างๆ อาจใช้อธิบายระบบทั้งระบบหรือ อธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของแผนผังอุปกรณ์(Component Block)จากนั้นก็ทำการจำ ลองการทำงาน โดยที่ตัวที่ทำการออกแบบนั้นยังไม่ได้สร้างขึ้นจริงหรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังจากจำลองการทำงานจนได้ตามที่ต้องการจึงนำไป ทำการสังเคราะห์เพื่อให้ได้วงจรเกทต่อไป

ประโยชน์จริงๆของการใช้ VHDL เป็น Tool แทนการสร้างระบบต้นแบบขึ้นมาจริงแบบเมื่อก่อน ก็คือเราสามารถอธิบาย Product Idea, Product Proposal, Product Speciification เป็นลักษณะในรูปของแฟ้ม Text จากนั้นก็นำไป Compile เพื่อดูผังเวลาการทำงาน จากนั้นก็ทำการ Refine แก้ไขจนกว่าจะได้ Specification ตามต้องการ เมื่อผลิตภัณฑ์ได้ผลตามที่ต้องการแล้วจึงนำไปเข้าสู่การสังเคราะห์เพื่อให้ได้ผังวงจรในระดับวงจรเกท แล้วนำไปสร้างเป็น ระบบต้นแบบจริงต่อไป ซึ่งระบบต้นแบบที่สร้างนั้นทำงานได้จริงเพราะได้ทำการจำลองการทำงานมาเรียบร้อยแล้ว เป็นการลดเวลาและค่าใช้จ่ายในการสร้างระบบต้นแบบได้มาก

เนื่องจากว่าภาษา VHDL เป็นภาษาที่มีประสิทธิภาพสูง เราจึงใช้ VHDL อธิบายฮาร์ดแวร์เพราะว่ามีข้อดี 2 ประการ คือ

3.4.1.1 เข้าใจได้ง่าย (Easy To Understand)

3.4.1.2 สามารถแก้ไขได้ง่าย (Modificable)

การเข้าใจได้ง่ายมีประโยชน์ต่อใครก็ได้ซึ่งมีความจำเป็นที่จะต้องอ่าน Code ที่ได้ออกแบบมาล้วโดยไม่จำเป็นต้องให้ผู้ออกแบบมาอธิบายให้ฟัง ตัวภาษา VHDL อธิบายการทำงานภายในตัวอยู่แล้ว ส่วนอีกประการหนึ่งก็คือความต้องการในการเปลี่ยนแปลง Hardware ที่ได้ออกแบบแล้วก็คือว่า หลังจากทดสอบแล้วพบข้อที่ผิดพลาดซึ่งต้องแก้ไข หรือว่าระหว่างพัฒนามีการเปลี่ยนแปลงความต้องการของระบบหรือต้องการเพิ่มการทำงานบางส่วน

กรณีอื่นๆที่ VHDL มีประโยชน์ก็คือ ตัวภาษานั้นสนับสนุนหลักการต่างๆ ให้เขียน แก้ ไข และบำรุงรักษาระบบเชิงตัวเลขที่มีความซับซ้อนเป็นไปได้อย่างรวดเร็ว และมีประสิทธิ ภาพ ซึ่งหลักการตัวที่กล่าวมีดังนี้

•Top Down Design

•Modularity

•Abstraction

•Information Hiding

•Uniformity

ซึ่งจะอธิบายประโยชน์ของหลักการต่างๆในหัวข้อต่อไปและแสดงให้เห็นว่า VHDL นั้นช่วยในการออกแบบระบบเชิงตัวเลขขนาดใหญ่และซับซ้อนนั้น ให้อ่านเข้าใจได้ง่ายและแก้ไขได้ง่ายอย่างไร

3.4.2 Top Down Design

ในการพัฒนาระบบเชิงตัวเลขขนาดใหญ่ที่มีความซับซ้อน วิศวกร หรือผู้ออกแบบมักจะมองระบบที่จะออกแบบให้อยู่ในรูปของแผนผังงานเสียก่อน ก่อนที่จะย่อยระบบที่จะออกแบบให้ลึกถึงรายละเอียดต่อไป ซึ่ง VHDL นั้นอนุญาติให้

3.4.2.1 อธิบายการทำงานของแต่ละส่วน

3.4.2.2 วิเคราะห์การทำงาน

3.4.2.3 จัดการแก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ ก่อนที่จะทำการออกแบบให้ละเอียดลึกลงไปในขั้นตอนต่อไป การแก้ไขในขั้นตอนนี้จะทำให้ลดค่าใช้จ่ายกว่าการไปแก้ไขในช่วงของการพัฒนาในระดับสร้าง Silicon CHIP

3.4.3 Modularity

Modularity คือหลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ออกเป็นส่วนย่อยเล็กๆลงไปซึ่งปกติการทำงานของฮาร์ดแวร์ใหญ่ต้องประกอบด้วยฮาร์ดแวร์ส่วนย่อยๆลงไป ซึ่งวงจรระดับเกททั้งหมดจะอยู่ในรูปๆเดียว(Flatten Design)หลังจากนั้น จะตัดออกส่วน ย่อยๆเล็กลงมา เมื่อเราออกแบบโดยใช้ VHDL หน้าที่การทำงานของแต่ละส่วนสามารถอธิบายได้โดยได้ Module ของระหัส(คล้าย Function หรือ Procedure) ซึ่งแสดงการทำงานของส่วนย่อยนั้นอย่างชัดเจน ซึ่งการแยกระบบที่จะออกแบบใหญ่ๆ ออกเป็นส่วนย่อยๆนี้ทำให้ง่ายต่อการจัดการและง่ายต่อการทำความเข้าใจ

ตัวภาษา VHDL ประกอบขึ้นมาด้วย Language Building Block ซึ่งประกอบไปด้วย 75 Reserved Word และมากกว่า 200 Combination words

3.4.4 Abstraction

คำนิยามของระบบที่จะออกแบบ จะอธิบายการทำงานของตัวระบบมากกว่าที่จะอธิบายถึงว่าจะพัฒนาตัวระบบนั้นอย่างไร หลักการนี้จะมีความสัมพันธ์อย่างใกล้ชิดกับหลักการ Modularity

อีกวิธีการหนึ่งซึ่งแสดงถึงการอธิบายการทำงานของงระบบโ โดยใช้ VHDL ในหลายๆระดับของการนิยาม ROM(READ ONLY MEMORY)อธิบายโดยใช้ภาษาระดับสูงแสดงถึง Address ต่างๆซึ่งเก็บ DATA ไว้ใน Address นั้นๆที่ระดับนี้ไม่ต้องสนใจถึง Address Line, Data Line หรือ Control Line เราสามารถเพ่งจุดสนใจไปที่ขนาดของ DATA โดยไม่ต้องคิดถึงสัญ ญาณควบคุมต่างๆมากมายภายใน เพราะว่าส่วนนั้นจะถูกจัดการเองในระดับต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ในการ จัดการสัญญาณภายในทุกเส้นภายในการที่จะอ่านข้อมูลหรือโปรแกรมข้อมูลใน ROM ถ้าต้องการเปลี่ยนค่าข้อมูลภายใน ROM เราควรขึ้นมาแก้ไขในระดับที่สูงขึ้นมา จะทำให้ง่ายกว่าในการที่จะควบคุมสัญญาณภายใน ซึ่งเราจะเห็นว่าในแต่ละระดับมีความเหมาะสม แตกต่างกันไป และตรงจุดนี้เองทำให้ระบบที่เราออกแบบง่ายต่อการแก้ไขโดยการใช้ประโยชน์ของ Abstraction

3.4.5 Information Hiding

เมื่อเราทำการเขียนระหัสต้นแบบของภาษา VHDL ขึ้นมาเพื่ออธิบายการทำงานของฮาร์ดแวร์ตัวหนึ่ง บางครั้งเราอาจต้องการที่จะซ่อนรายละเอียดการพัฒนา Module นั้นๆ โดย ไม่ต้องการให้ส่วน Module อื่นๆรู้การทำงานภายใน Information Hiding มีประโยชน์ก็คือทำให้ระบบที่ออกแบบนั้น สามารถจัดการได้ง่ายและสามารถอ่านและทำความเข้าใจได้ง่ายกกว่า หลักการนี้จะใช้สนับสนุนหลักการ Abstraction คือจะสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่าระบบนั้นจะถูกสร้างขึ้นมาอย่างไรมีวงจรอย่างไรบ้างเป็นต้น การซ่อนรายละเอียดภายใน Module ทำให้ความสนใจของผู้ออกแบบสนใจไปในส่วนที่สำคัญมาก กว่า ในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ เช่น คำอธิบายของ Nand Gate นั้นจะถูกปิดบังเอาไว้จากคนที่เขียนอธิบาย Flip-Flop คนที่เขียนอธิบายการทำงานของ Flip-Flop ไม่ต้องสนใจเลยว่า Nand Gate จะทำงานอย่างไรจะต่อกันภายในอย่างไร โดย Nand Gate สามารถเขียนขึ้นมาแล้วคอมไพล์เก็บไว้ใน LIBRARY ผู้ที่ออกแบบ Flip-Flop ระดับสูงขึ้นมาเพียงแต่ต้องรู้ว่าจะเชื่อมต่อ Input/Output ของ Nand Gate มาใช้งานได้อย่างไร และประโยชน์อีกอย่างของ Information Hiding ก็คือป้องกัน

ข้อมูลภายใน ในกรณีที่แจกจ่าย VHDL Model ไปยังที่อื่น ๆ เช่น ส่งไปให้อีกบริษัทใช้พัฒนาร่วมกับ VHDL อื่นๆ โดยเป็นการแจกจ่าย อาจ ส่งไปแค่ VHDL ที่คอมไพล์แล้วไม่ต้องส่งตัวระหัสต้นแบบไป ทำให้เราป้องกันทรัพย์สินทางปัญญาได้ในอีกระดับหนึ่ง

3.4.6 Uniformility

Uniformility เป็นหลักการอีกอย่างที่ช่วยในการอธิบายระบบ ด้วยภาษา VHDL นั้น อ่านได้ง่ายขึ้น Uniformility หมายถึงการสร้าง Module ของระหัสในลักษณะคล้ายกัน โดยใช้ตัวภาษา VHDL BUIDING Block ทำให้เกิดการเขียนระหัสต้นแบบที่ดีอย่างเช่น มีการใช้ย่อ หน้า มีการใช้ Comment อธิบาย เป็นต้น ทำให้การพัฒนา Module อ่านและทำการเข้าใจง่าย

3.5 ระดับของการอธิบายระบบ(LEVEL OF ABSTRACTION)

การออกแบบโดยใช้ภาษา VHDL Description ก็มีหลายระดับแต่ละระดับก็เหมาะสมและมีข้อเสียต่างกันไปซึ่งรูปแบบของภาษา VHDL มี 3 ระดับดังนี้

3.5.1 Behavioral Description

3.5.2 Dataflow Description

3.5.3 Structural Description

3.5.1 Behavioral Description

เป็นระดับที่เป็นนามธรรมที่สุด โดยภาษา VHDL นั้น มีรูปแบบนี้เป็นการอธิบายการทำงานของระบบคล้ายๆกับโปรแกรมคอมพิวเตอร์ คือมี Procedural Form และ Function Form ซึ่งไม่ต้องกล่าวถึงรายละเอียดและการสร้างตัวระบบเลย การใช้ภาษา VHDL รูปแบบนี้เหมาะสำหรับอธิบายระบบที่มีความซับซ้อนมากๆ โดยนักออกแบบไม่ต้องทราบเลยว่าระบบจะประกอบด้วยอุปกรณ์อะไรบ้าง ทำให้ VHDL รูปแบบนี้เหมาะกับผู้ที่ไม่ใช่วิศวกรและผู้ใช้งานทั่วๆไปและเป็นการอธิบายตัวระบบที่ดีไปในตัว(Good Documentation)

3.5.2 Dataflow Description

เป็นระดับที่ต่ำกว่า Behavioral คือแสดงการไหลของข้อมูลภายในหน่วยย่อยๆของระบบ โดยตัวภาษาจะอธิบายการสื่อสารข้อมูลระหว่างหน่วยย่อยๆนั้น ผู้ออกแบบจะต้องรู้การทำงานของหน่วยย่อยๆ แต่ละหน่วยและรูปแบบของข้อมูลที่จะส่งไปมา การอธิบาย

ในระดับ นี้ไม่เหมาะกับผู้ใช้ทั่วๆไป แต่จะเหมาะกับระดับช่างเทคนิคหรือวิศวกรเท่านั้น การเขียนอธิบายแบบนี้จะเสียเวลามากกว่าแบบ Behavioral ก็จริงแต่ข้อดีก็คือสามารถที่จะสังเคราะห์ออกมาได้ดีกว่า เพราะแสดงรายละเอียดของระบบออกมาในระดับผังงาน ซึ่งมากกว่าแบบ Behavioral ซึ่งไม่แสดงอะไรเลย แต่อย่างไรก็ตามเราจะใช้รูปแบบ Dataflow ในการอธิบายการทำงานของระบบโดยการไหลข้อมูลระหว่าง Register และ Bus

3.5.3 Structural Description

เป็นการอธิบายฮาร์ดแวร์ในระดับต่ำสุด โดยอธิบายถึงการเชื่อมต่ออุปกรณ์แต่ละตัวเพื่อที่จะให้ได้เป็นระบบขึ้นมา ระดับนี้เป็นระดับที่สังเคราะห์ได้ง่ายที่สุดเพราะแสดงรายละเอียดของระบบได้มากที่สุด อย่างไรก็ตามภาษา VHDL ในระดับนี้อธิบายฮาร์ดแวร์ได้ขัด เจนที่สุด บางครั้งเรียกว่า Hardware Description at Gate Level ระดับนี้เข้าใจได้ยากที่สุด แต่เหมาะสำหรับการจำลองการทำงานที่ต้องการผังเวลาที่ละเอียดที่สุด

3.6 สรุป

3.6.1 ภาษา VHDL คือตัวภาษาที่ออกแบบมาเฉพาะเพื่อใช้อธิบายการทำงานของฮาร์ดแวร์ให้อยู่ในรูปที่สามารถอ่านทำความเข้าใจได้ (Human Readable) สามารถอธิบายได้ถึงการจัดระบบและการทำงานของวงจรระดับเกท, วงจรระดับ Board และอุปกรณ์ต่างๆ

3.6.2 เหตุผลที่ทำให้ภาษา VHDL ใช้ในการออกแบบและจำลองการทำงานของผลิตภัณฑ์ ตัวหนึ่งซึ่งยังไม่ได้สร้างจริงๆ เพื่อดูการทำงานก่อนลงมือสร้าง หรืออาจใช้เป็นตัวแทนแนวคิดในผลิตภัณฑ์นั้นๆ มีดังนี้

3.6.2.1 ภาษา VHDL อนุญาตให้เราออกแบบ, จำลองการทำงาน และทดสอบระบบโดยใช้รูปแบบของภาษาระดับสูงจนถึงระดับวงจรเกท

3.6.2.2 ภาษา VHDL ถ้าเราเขียนตามรูปแบบของ VHDL Synthesis Guide จะทำให้เราสามารถใช้รหัสภาษา VHDL นั้นไปทำการสร้างวงจรได้โดยใช้ VHDL Synthesis Tools

3.6.2.3 เพราะว่าภาษา VHDL เป็นภาษาที่กำหนดเป็นมาตรฐาน IEEE 1076-1987 วิศวกร หรือผู้ออกแบบทั่วๆไปสามารถใช้ภาษานี้ในการพัฒนาได้เหมือนกันลดปัญหาความเข้ากันไม่ได้ลงไป

3.6.3 VHDL มีคุณสมบัติที่ดีที่ทำให้เราสามารถเขียนและแก้ไขระบบเชิงตัวเลขที่มีขนาดใหญ่และซับซ้อนได้อย่างสะดวกรวดเร็วและมีประสิทธิภาพ ดังนี้คือ

3.6.3.1 Top Down Design วิธีการนี้ ทำให้เราสามารถอธิบายการทำงานของระบบได้ในลักษณะของ Block ใหญ่ๆ จากนั้นทำการวิเคราะห์จำลองการทำงานและแก้ไขให้ได้คุณสมบัติตามที่เราต้องการ ณ. ระดับ Block ก่อนที่จะลงลึกในระดับต่ำต่อไป

3.6.3.2 Modularity วิธีการที่แยกส่วน (หรือการประกอบส่วนย่อยๆ ขึ้นมา) ระบบที่เราออกแบบออกเป็นส่วนย่อยๆเล็กๆออกมา

3.6.3.3 Abstraction เป็นรายละเอียดใน Module ซึ่งอธิบายการทำงานของ Module มากกว่าที่จะอธิบายถึงการพัฒนาและการสร้าง Module นั้น

3.6.3.4 Information Hiding การพัฒนา Module ใหม่จาก Module อื่นๆที่สร้างขึ้นมาแล้ว

3.6.3.5 Uniformility การสร้าง Module โดยใช้ตัวภาษา VHDL Building Blocks

บทที่ 4

Field Programmable Gate Array(FPGA)

4.1 รู้จักกับ Programmable Gate Array

ความก้าวหน้าของอุตสาหกรรมอิเลคทรอนิกส์ในปัจจุบันทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆมากมายซึ่งทำให้เกิดการลดราคาค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาด ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครเปรเซสเซอร์และหน่วยความจำ ทุกๆครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างวงจรรวม VLSI และ IC standard มากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวน Logic Function ที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตให้ขนาดมากๆ 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 FPGAสามารถออกแบบและทดสอบภายในเวลาเพียง 2-3 วันเท่านั้นตรงข้ามกับการออกแบบโดยใช้ Custom gate array ซึ่งใช้เวลาหลายอาทิตย์ การเปลี่ยนแปลงแก้ไขแบบก็เช่นเดียวกัน จากผลประโยชน์ของ FPGA ซึ่งที่ได้กล่าวมา ทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ลดความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์,ลดค่า NRE (Nonrecurring engineering cost) ลงไปด้วย

4.2 Field Programmable Gate Array (FPGA)

คล้ายๆกับ Gate array ทั่วไป FPGA ไม่ต้องมีค่าใช้จ่ายคงที่และไม่มีค่า Fabrication เพราะว่า FPGA ทุกค่ามีโครงสร้างที่เหมือนกันใช้จ่ายในการผลิตมีลักษณะ คล้ายๆกับการผลิต IC standard ที่มีปริมาณมากๆโดยทั่วไป

4.3 Programmable Logic Device (PLD)

มีลักษณะ PLDs โดยส่วนใหญ่จะใช้แทนที่ SSt/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

4.4 Standard Cell & Custom IC

การผลิตแบบนี้ ต้องการหน้ากาก Mark สำหรับทุก Layer ในการผลิตทำให้ต้องมีค่าใช้จ่ายพิเศษ และมีความล่าช้าในการพัฒนา แต่ผลดีคือให้ต้นทุนต่อหน่วยต่ำที่สุดเหมาะสำหรับการผลิตให้จำนวนมากๆ standard cell ให้ประโยชน์ในการ high level building blocks

4.5 Gate Array

เราใช้ gate array สร้างวงจรลอจิกโดยการเชื่อมต่อทรานซิสเตอร์และ gate ต่างๆเข้าด้วยกันจนได้ฟังก์ชั่นการทำงานที่ตามที่ต้องการซึ่งเป็นขั้นตอนสุดท้ายของกระบวนการผลิต gate array มีความหนาแน่นของ gate ประมาณ 100,000 gate หรือมากกว่านั้น ใช้ประโยชน์ประมาณ 80-90% สำหรับอุปกรณ์เล็กๆ และ 40-60% สำหรับอุปกรณ์ใหญ่ ลักษณะไม่เหมือน IC standard products, gate-array มีค่าใช้จ่าย fixed cost และ product cost การใช้งาน gate array จะประหยัดที่สุดเมื่อจำนวนการผลิตสูงมากจนทำให้ค่าใช้จ่ายคงที่หายไป

4.6 Programmable Gate Array Architecture

Logic Cell Array (LCA) เป็นสิ่งประดิษฐ์ของบริษัท XILINX มีสถาปัตยกรรมภายในคล้ายๆ 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 ส่วนประกอบหน่วยความจำและการโปรแกรมการเชื่อมต่อต่างๆ ความเร็วของ System clock rate ถูกกำหนดด้วย toggle flip-flop สำหรับการประยุกต์ใช้โดยทั่วไปจะอยู่ที่ประมาณ 1/3 ถึง 1/2 ของ maximum toggle gate

4.6.1 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

4.6.2 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

4.6.3 Interconnect

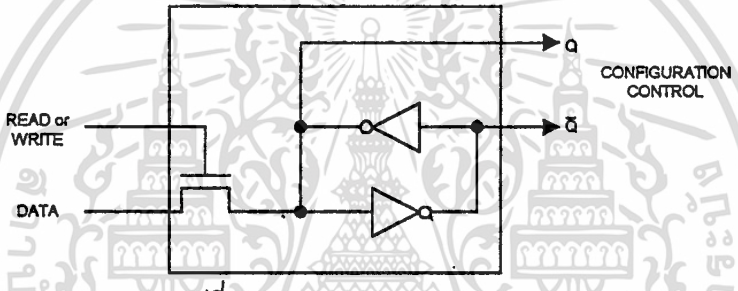
ความยืดหยุ่นของการใช้ LCA มาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรม resource ต่างๆที่อยู่ภายในเข้าด้วยกันการที่จะควบคุมการเชื่อมตัวระหว่างจุดสองจุดภายใน chip เหมือนกับ gate array ทั่วๆไป การเชื่อมต่อภายใน LCA ประกอบด้วย network 2 ทิศทางคือทาง vertical และ horizontal หรือ(row และ column)ซึ่งวางอยู่ระหว่าง CLB programmable switch จะทำการเชื่อมต่อ input และ output ของ IOBs และ CLBs ที่จุดต่อร่วมระหว่าง row กับ column สามารถ switch signal จาก path ไปยังส่วนอื่นๆ เส้น long line จะลากผ่านตลอด chip เพื่อแก้ปัญหาเรื่อง critical signal ด้วย Minimum delay หรือ skew ที่เกิดขึ้น

4.7 สถาปัตยกรรมของ LCA (Logic Cell Array Architecture)

I/O block ที่โปรแกรมการทำงานได้ เรียงล้อมรอบตัว CLB อยู่รอบนอกมีไว้เพื่อเชื่อมต่อกับ Package pin ชุดของ CLB ทำหน้าที่เป็นวงจรใดก็ได้ตามต้องการแล้วแต่ผู้ใช้จะโปรแกรมการ

เชื่อมต่อภายในระหว่าง resource ที่มีอยู่การส่งผ่านลอจิก ระหว่าง block ต่างๆ เปรียบเสมือนสัญญาณต่างๆที่ส่งผ่านระหว่างอุปกรณ์ในรูปแบบ 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



รูปที่ 4-1 Static Configuration Memory Cell

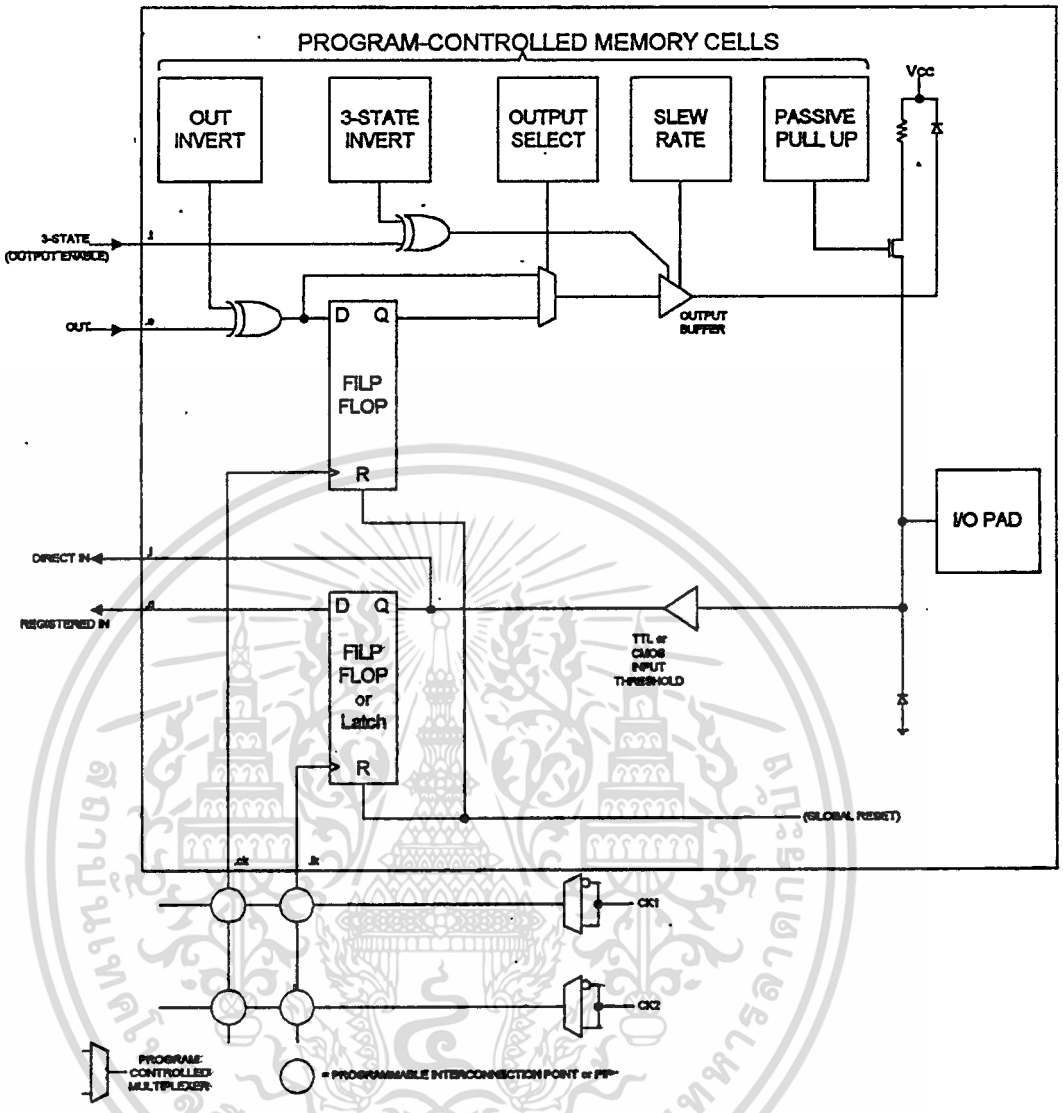
4.7.1 Configuration Memory

static memory cell ภายใน LCA ถูกออกแบบเป็นอย่างดีมีความเชื่อถือได้สูงและป้องกันสัญญาณรบกวนได้ดี ความแน่นอนของ Memory ภายใน LCA ซึ่งได้รับการออกแบบเป็นอย่างดีนี้ ทำงานได้แม้ภายใต้สภาวะที่เลวร้าย ถ้าเทียบกับการโปรแกรมในแบบอื่นๆ static memory ให้ความเชื่อถือได้ดี,มีความหนาแน่นสูง,มีประสิทธิภาพสูงและสามารถทดสอบได้ จาก รูปที่ 4-1 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 Q̄ ใช้ระดับ GND และ VCC เพื่อทำให้เกิดความแน่นอนและควบคุมได้โดยตรง comparative load และ sense amplify ทำให้การทำงานมีความเที่ยงตรงสูงเนื่องจากโครงสร้างของ configuration memory cell เป็นอย่างที่กล่าวมาทำให้ค่า LCA ไม่มีการได้รับผลกระทบใดๆ เกิดขึ้นที่เกิดจากการเปลี่ยน

แปลงของ power-supply อย่างรุนแรง ,การแพร่กระจายของรังสี alpha ที่มีความเข้มสูง ในการทดสอบความแน่นอนในการทำงานไม่พบข้อผิดพลาดใดๆ 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 เป็นไปได้ในหลายๆลักษณะเช่น Synchronous, Serial หรือ Daisy chain ได้

4.7.2 I/O Block

IOB แต่ละตัวดัง รูปที่ 4-2 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



รูปที่ 4-2 Input/Output Block

4.7.2.1 IOB output buffer มีลักษณะเป็น CMOS compatible 4 ma source or sink ขับสัญญาณ high out CMOS หรือ TTL compatible ขา IOB pin [f] สามารถควบคุมการทำงานของ output ได้ configuration program bit สามารถกำหนดการทำงานของ IOB ให้เป็นไปได้ในหลายๆลักษณะ เช่น optimal output register,logical signal inversion และ 3 state;slew rate control ของ output

4.7.2.2 Program controlled memory cell ดัง รูปที่ 4-2 ควบคุมลักษณะต่างๆดังต่อไปนี้

4.7.2.2.1 logic inversion of output ถูกควบคุมโดย 1 configuration bit แต่ละ IOB

4.7.2.2.2 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

4.7.2.2.3 direct or register output เลือกได้แต่ละ IOB register ใช้สัญญาณขอบหน้า trig

4.7.2.2.4 เพิ่มค่า output transition speed

4.7.2.2.5 high impedance pull-up resistor ใช้สำหรับป้องกันการ floating เมื่อไม่ได้ใช้ขานั้นทำงาน

4.7.2.3 สรุปการทำงาน I/O

4.7.2.3.1 inputs

4.7.2.3.2 direct

4.7.2.3.3 flip-flop latch

4.7.2.3.4 CMOS/TTL threshold

4.7.2.3.5 pull-up resistor/open circuit

4.7.2.3.6 outputs

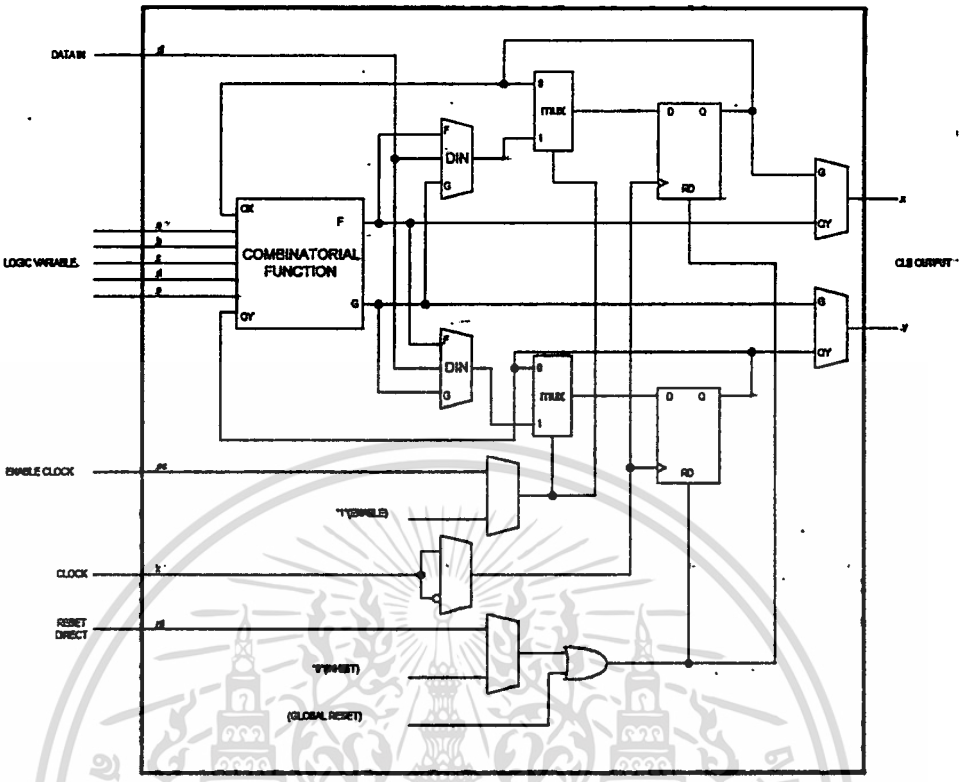
4.7.2.3.7 direct/resistor

4.7.2.3.8 inverted/not

4.7.2.3.9 3 state/on/off

4.7.2.3.10 full speed/slew limit

4.7.2.3.11 3 state/output enable



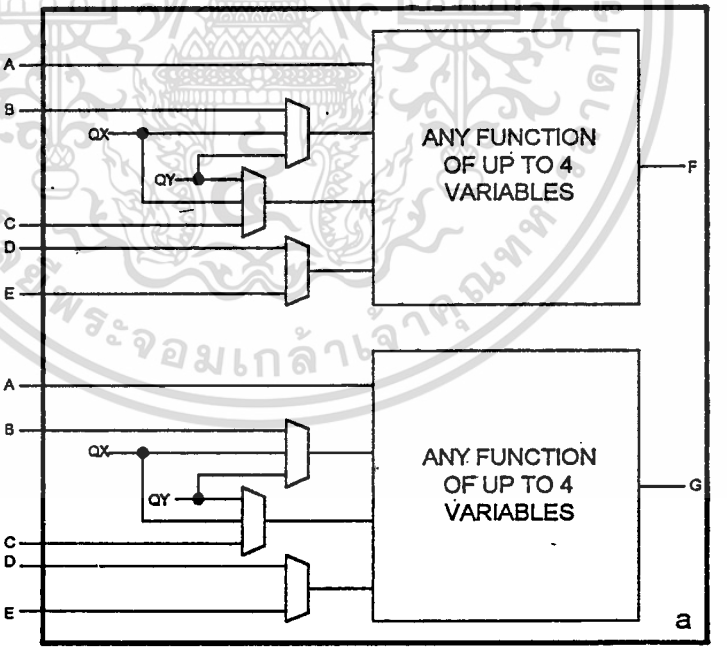
รูปที่ 4-3 Configurable Logic Block

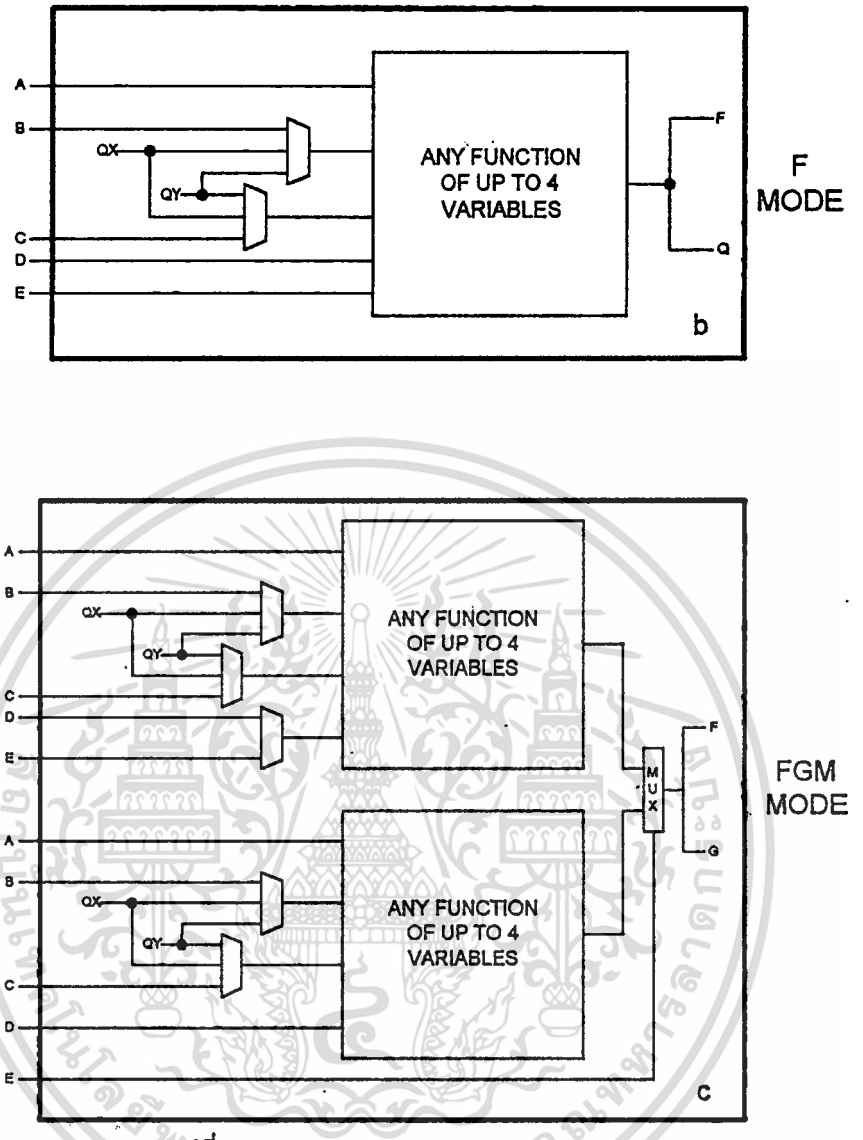
4.7.3 Configuration Lgic Block (CLB)

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 ในรูปแบบต่างๆ CLB แต่ละตัวมีส่วนของ combinatorial logic , flip-flop 2 ตัว และส่วนควบคุมภายใน (internal control section) ดังรูปที่ 3-3 และรูปที่ 4-4 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

Data input ของ filp-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 CLBแต่ละตัวมีขาใช้ขา asynchronous ร่วมกันซึ่งเมื่อ enabled และได้รับ logic high จะมีความสำคัญเหนือกว่า clock inputs flip-flop ทุกตัว ใน CLB ถูก resetโดย active low

input ขา RESET หรือระหว่างการทำ configuration นอกจากนี้ filp-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 4-4a หรือ signal function ดังรูป 4-4d หรือ function พิเศษ ที่ใช้ 7 variable ดังรูป 3-4c บางส่วนของ partial function ที่มี 6 ตัวแปร หรือ 7 ตัวแปร สามารถสร้างได้โดยใช้ขา input variable (e) ในการที่จะเลือกระหว่าง 2 function ที่มี 4 variable สำหรับ 2 function ที่มี 4 variable





รูปที่ 4-4 Combinatorial Function Mode

รูปที่ 3-4a combinatorial logic option FG จะสร้างฟังก์ชัน 2 ฟังก์ชั่นที่มีตัวแปร 4 ตัวแต่ละฟังก์ชัน ตัวแปร A ต้องใช้ร่วมกันระหว่าง 2 function ตัวแปรตัวที่ 2 หรือ 3 สามารถเป็นตัวเลือกอะไรก็ได้ b,c,qx หรือ qy ตัวแปรตัวที่ 4 จะเป็น D หรือ E อะไรก็ได้ แล้วแต่จะเลือก

รูปที่ 4-4b combinatorial logic option จะสร้างฟังก์ชั่นอะไรก็ได้ที่มี 5 ตัวแปร a,d,e และ เลือกมาอีก 2 ตัวจาก b,c,qx,qy

รูปที่ 4-4c 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

4.7.4 การเชื่อมต่อภายในที่สามารถโปรแกรมได้ (Programmable Interconnect)

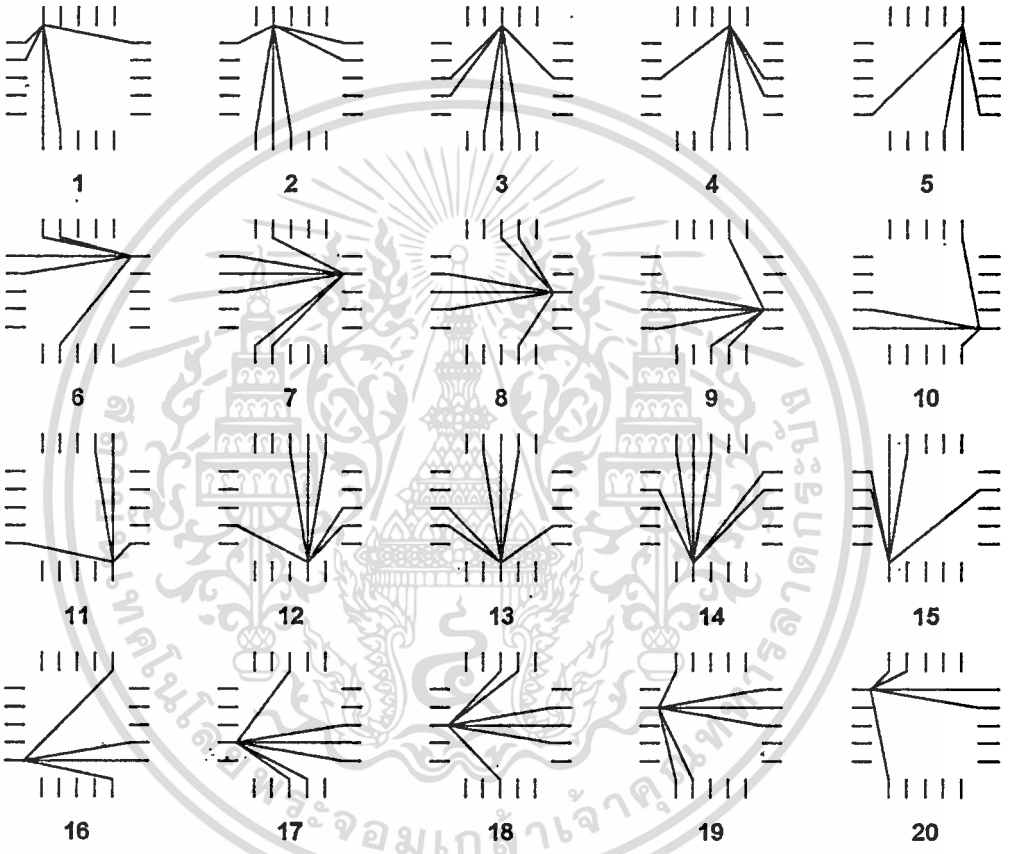
Resource ภายใน FPGA ที่ช่วยอำนวยความสะดวกในการเชื่อมต่อ logic cell array ต่างๆเข้าด้วยกันได้แก่ทางเดิน (Path) ที่ช่วยในการเชื่อมโยง Input และ Output ของ I/O Block เข้ากับ Logic Block ให้กลายเป็น Logic Network การเชื่อมต่อภายในระหว่าง Block ประกอบไปด้วย Metal Segment จำนวน 2 Layer ออกแบบพิเศษโดยใช้ Pass 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 และการ Routing ในการที่จะ Access logic block input variable, control input และ block output resource metal 3 ชนิด อำนวยความสะดวกในการที่จะเชื่อมต่อ Networkเข้าไว้ในหลายรูปแบบตามความต้องการดังนี้

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

4.7.4.1 General purpose Interconnect

การเชื่อมต่อแบบ General Purpose Interconnect 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 ที่ไม่ได้ถูกโปรแกรมจะอยู่ในสภาพที่ไม่เป็นตัวนำใดๆ ทั้งสิ้น การเชื่อมต่อภายใน Switching Matrix สามารถทำได้ทั้ง

แบบ Automatic หรือโดย Manualโดยใช้ Edinet ในการที่จะเลือกคู่เชื่อมต่อตามต้องการ รูปที่ 4-5 แสดงรูปแบบการเชื่อมต่อต่างๆภายใน Switching Matrix Buffer แบบพิเศษเพื่อออกแบบขึ้นมาใช้ใน General interconnect zone ช่วยให้เกิดการ Isolation และ Restoration เพื่อที่จะปรับปรุงคุณภาพของสัญญาณที่วิ่งผ่าน Net ที่มีขนาดยาวๆ



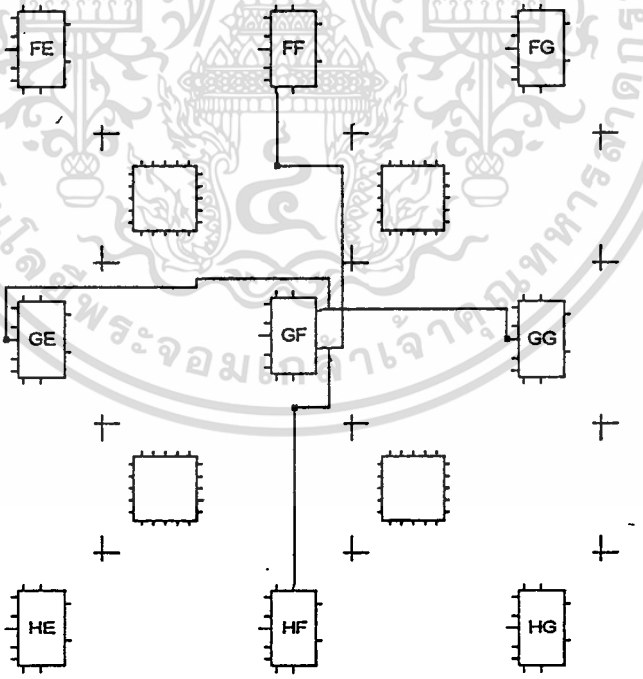
รูปที่ 4-5 Switch Matrix Interconnection Option.

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 โดย

อัตโนมัติ การคำนวณการหน่วงเวลาในแต่ละ Path ที่เลือกถูกคำนวณอย่างอัตโนมัติโดย Development software

4.7.4.2 Direct Interconnect

การเชื่อมต่อโดยตรง (Direct Interconnect) ดังแสดงในรูปที่ 4-6 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)



รูปที่ 4-6 CLB .X and .Y Outputs

4.7.4.3 Long line

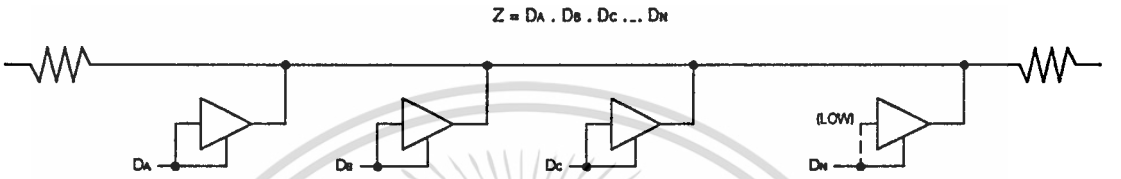
long line จะเป็นเส้นทางเดินของสัญญาณที่ไม่ผ่าน switch matrices และมีจุดประสงค์หลักที่ช่วยให้สัญญาณที่จะต้องเดินทางผ่านระยะทางไกลๆ หรือช่วยให้ Signal มี Minimum skew rate ระหว่างปลายทางหลายๆ จุด Long line ลากในแนวนอนและแนวตั้งในส่วนสูงและความกว้างของ Interconnect area ในแต่ละ Interconnect column มี 3 เส้นในแนวตั้งและในแต่ละ 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 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

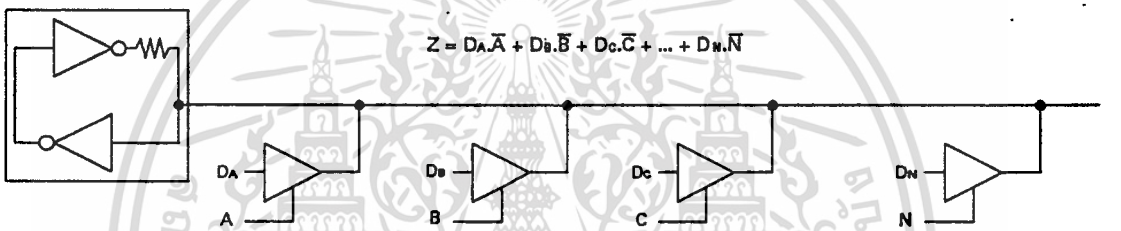
4.7.4.4 Internal Busses

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 ดังรูปที่ 4-7a 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 4-7b 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



รูปที่ 4-7a 3-State Buffers Implement a Wired-AND Function

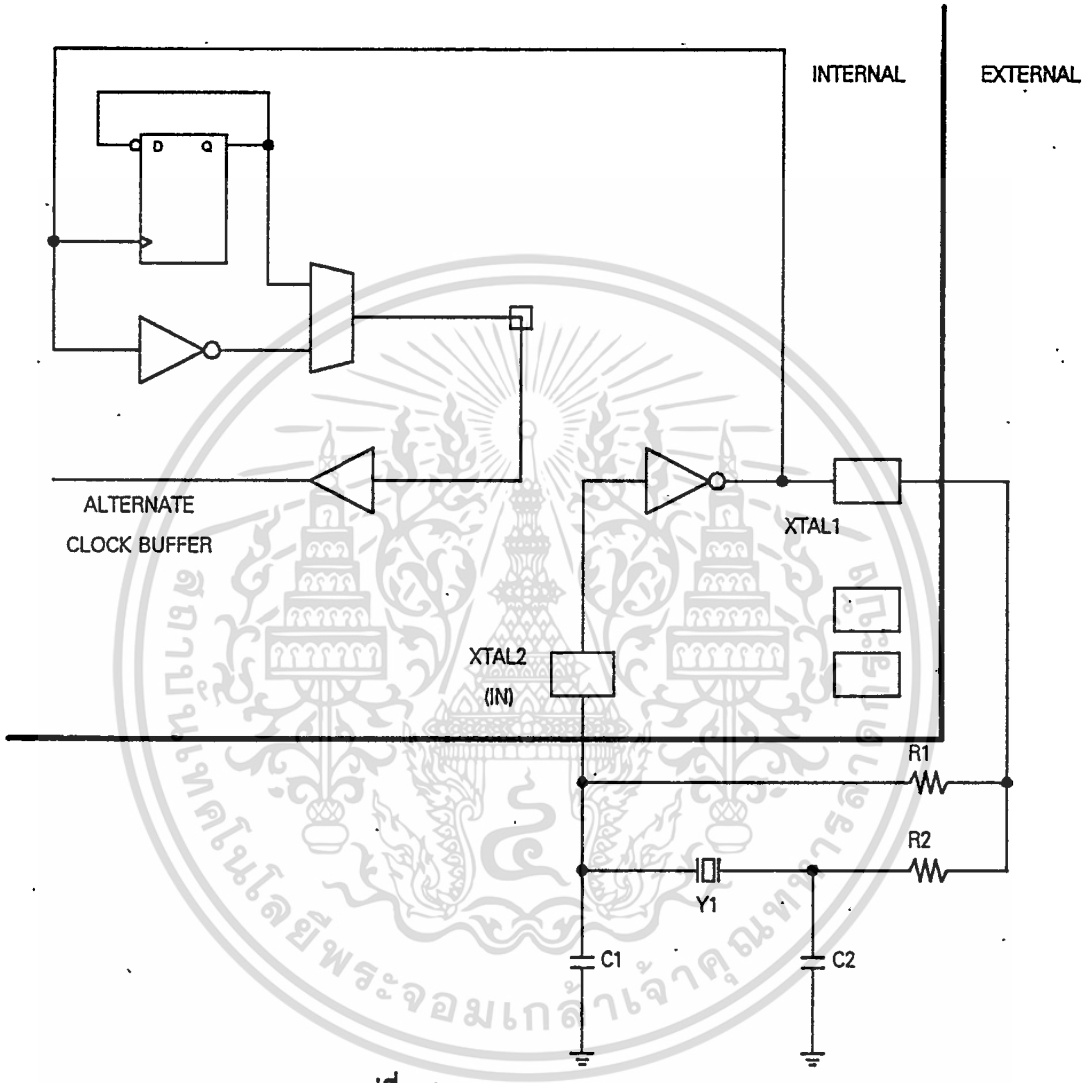


รูปที่ 4-7b 3-State Buffers Implement a Multiplexer

4.7.5 คริสตอลออสซิลเลเตอร์ (Crystal Oscillator)

รูปที่ 4-8 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 ดังรูปที่ 4-8 ร่วมกับอุปกรณ์ Oscillator จากภายนอกวงจร Oscillator จะต้อง active ก่อนที่ขบวนการทำ configuration จะสมบูรณ์เพื่อที่จะทำให้ Oscillator นั้นอยู่ในสภาวะคงที่ การเชื่อมต่อภายในจะต่อให้จริงๆ ก็ต่อเมื่อขบวนการ Configuration เสร็จสิ้น ในรูปที่ 4-8 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 กลับมาเกินสามารถแก้ไขให้มีค่าถูก

ต้องได้โดยใช้อัตราส่วน C2/C1 Amplifier ถูกออกแบบมาเพื่อให้ใช้กับความถี่ตั้งแต่ 1 MHz จนถึง 1/2 ของความถี่ CLB Toggle frequency เมื่อไม่มีการใช้งาน Oscillator invertor IOBs ที่ใช้สามารถถูกใช้งาน User I/O ได้ตามปกติ



รูปที่ 4-8 Crystal Oscillator Inverter

4.7.6การโปรแกรม (Programming)

4.7.6.7 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 จะถูกรีเซ็ตให้ทำงาน เพื่อรอให้ power supply คงที่ก่อน วงจร initialization state time-out (ประมาณ

11-33 Ms) ถูกกำหนดโดย 14 bit counter ภายใน LCA เอง จากตารางที่ 4-1 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	Byte wide addr = 0000 up
0	1	0	-	reserved	-
0	1	1	Active	Master	Byte wide addr = FFFF down
1	0	0	-	reserved	-
1	0	1	Passive	Peripheral	Byte wide
1	1	0	-	reserved	-
1	1	1	Passive	Slave	Bit serial

ตารางที่ 4-1

4.7.6.8 Configuration data

configuration data เป็นตัวกำหนดการเชื่อมต่อ resource ภายใน LCA เพื่อทำให้ LCA ทำงานเป็นหน้าที่ตามที่ต้องการ ซึ่ง data สามารถ load จากภาย

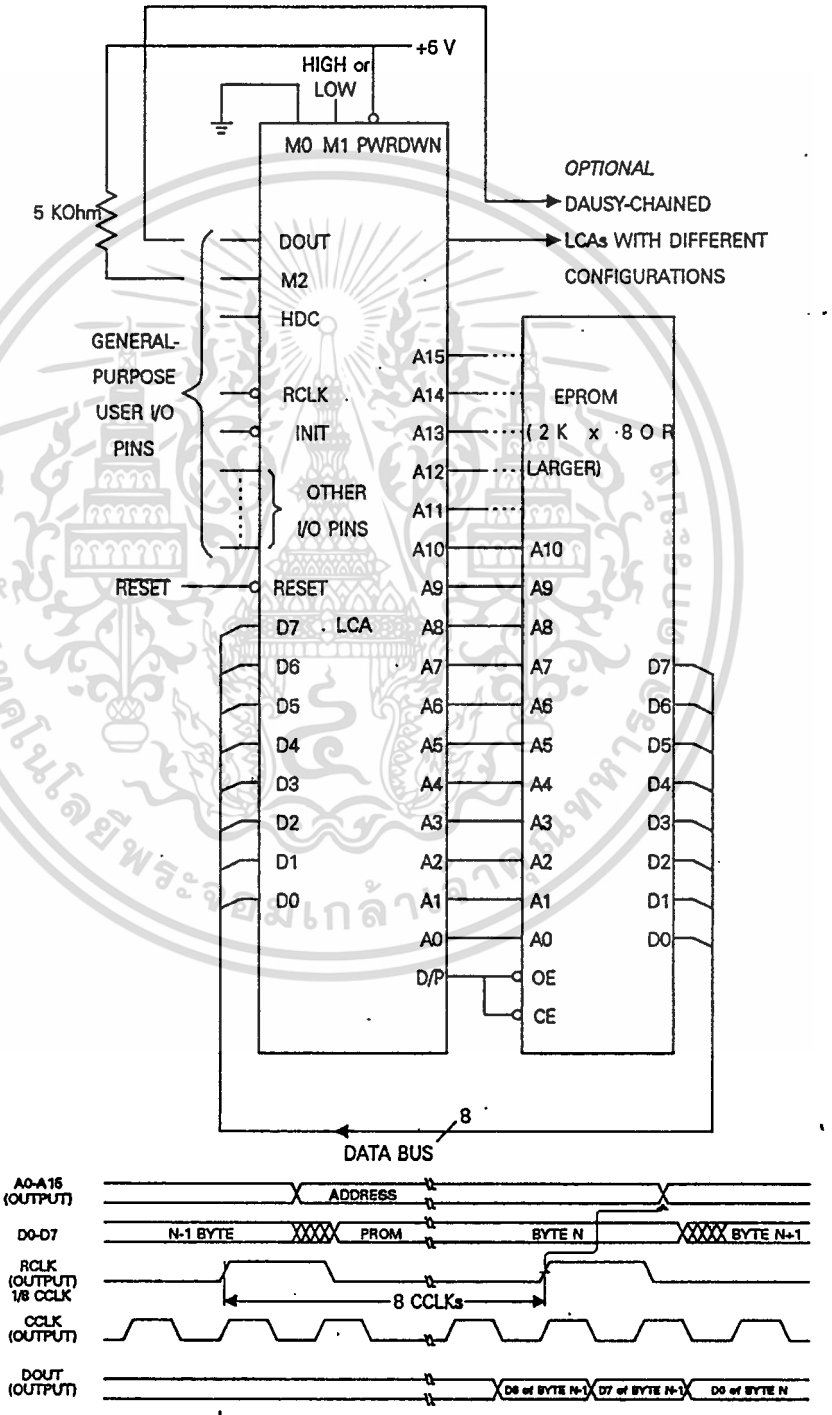
นอกเมื่อจ่ายไฟให้ LCA หรือ ทางการสั่งงานโดยสัญญาณคำสั่ง(Command)มีรูปแบบในการโปรแกรมหลายรูปแบบให้เลือกตามความเหมาะสม รูปแบบแต่ละอย่างเลือกได้โดยขา select mode pin โดยขาถูกอ่านเข้ามาก่อนที่จะเริ่ม configuration time รูปแบบของ data อ่านจะอยู่ในรูป serial หรือ byte parale ขึ้นอยู่กับ configuration mode

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 ที่ต้องการ ได้ถูกเขียนลงไปแล้ว 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

4.7.6.9 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 รูปที่ 4-9 แสดงตัวอย่างการต่อ Parallel Mode LCA เริ่มต้นที่ address 0000 และ

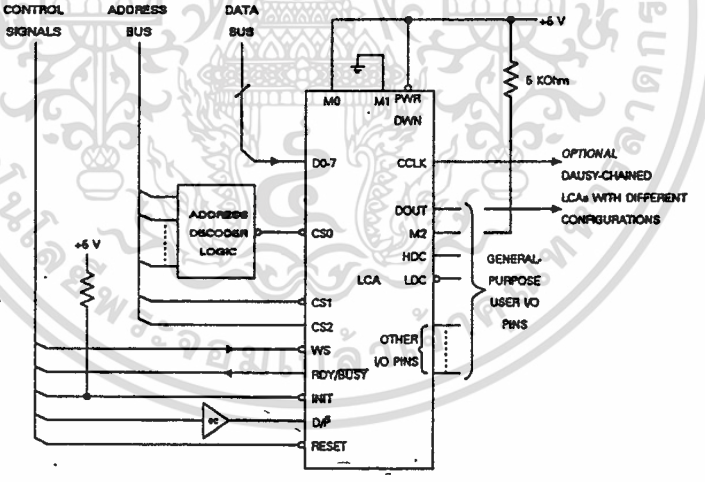
เพิ่มขึ้นเรื่อยๆ สำหรับ Master low ส่วน Master High จะเริ่มที่ FFFF และจะลดค่าลงมาเรื่อยๆ 2 mode ที่สร้างขึ้นมาเพื่อให้ใช้งานได้กับ Microprocessors ที่เริ่มต้นทำงานในลักษณะต่างๆ กัน สำหรับทั้ง Master high และ Master Low ข้อมูล (data byte) จะถูกอ่านแบบขนานทุกๆ Read Clock (RCLK) และส่งเข้าไปภายในแบบ Serial โดย Configuration Clock

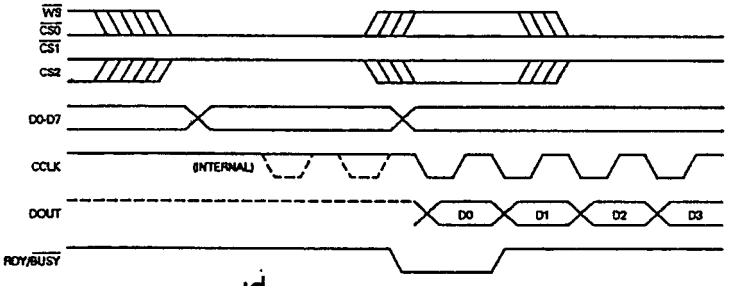


รูปที่ 4-9 Master Parallel Mode

4.7.6.10 Peripheral Mode

Peripheral Mode เป็นการส่งผ่านข้อมูลโดยผ่านการ Load แบบ byte wide ลักษณะคล้ายๆ Microprocessors interface ดังรูปที่ 4-10 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 SByte เสร็จพร้อมที่จะ Load byte ใหม่ต่อไปคล้ายกับ Master Mode Peripheral Mode อาจใช้เป็นแบบ Lead device หรือ Daisy-chain device





รูปที่ 4-10 Peripheral Mode

4.7.6.11 Slave Mode

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

4.7.6.12 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 ครบ หลังจากนั้น 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

4.7.6.13 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

4.7.6.13.1 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 เพิ่มขึ้นมาช่วย สำหรับ 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

4.7.6.13.2 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 Q ของ IOB แต่ละตัว ข้อมูลเหล่านี้จะถูกรวมเข้ากับ Configuration bit ในตำแหน่งที่ไม่ได้ใช้ข้อมูลเหล่านี้เป็นประโยชน์ต่อ XACT development system ในการที่จะตรวจสอบสภาพ การทำงานของ Logic ภายในตัว LCA

4.7.6.14 Reprogram

LCA configuration Memory สามารถเขียนเข้าไปใหม่อีกครั้งได้ ขณะที่กำลังทำงานอยู่ในระบบ ในการเริ่มต่นที่จะทำการ Reprogram ทำได้โดยให้สัญญาณขอบหน้า (low to high) เข้าไปที่ขา DONE/PROG ในการที่จะลด Noise ที่จะเข้ามา สัญญานขอบหน้าที่เข้ามาจะถูกหารโดย Internal timing generation ก่อนประมาณ 2 cycle เพื่อให้แน่ใจ เมื่อการ Reprogram เริ่มต้นขึ้นขา 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

4.7.6.15 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

4.7.6.16 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

4.7.6.17 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

4.7.7 ประสิทธิภาพ (Performance)

4.7.7.1 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

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 ในลักษณะต่างๆ

4.7.7.2 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

4.7.7.3 Interconnect Performance

ประสิทธิภาพของการเชื่อมต่อขึ้นอยู่กับการ routing resource ที่มีอยู่ภายในในการที่จะสร้างทางเดินของสัญญาณภายใน LCA ดังที่กล่าวมาแล้วข้างต้น ในเรื่องของProgrammable 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 แบบ bidirection และจำนวนloading บนทางเดินของสัญญาณทั้งหมดภายใน path นั้น ในการคำนวณ worst-case ใน general interconnect โดย XACT development software นั้นรายงาน element ทั้งหมด

4.7.8 อธิบายหน้าที่การทำงานแต่ละขา Pin description

4.7.8.1 กลุ่มขาที่ใช้งานแบบถาวรใช้เป็น I/O ของ user ไม่ได้

- VCC

มีจำนวน 2 ถึง 8 ขา ขึ้นอยู่กับขนาดของตัวถังต่อเข้ากับไฟ +15v 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 จะ

เริ่มต้นทำงานเมื่อขา 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 ต่างๆจะ active หลังจากที่ LCA ได้ถูกโปรแกรมไปครั้งหนึ่ง ถ้ามีการเปลี่ยนระดับสัญญาณจาก high to low ทำให้ LCA เริ่มกระบวนการ Reconfiguration ตัวเองขึ้นมาใหม่

- MO/RTRIG

ขณะอยู่ใน Mode 0 ขา input ขานี้ ร่วมกับขา M1,M2 ถูกอ่านค่าเข้าไปก่อนเพื่อกำหนด Mode ที่จะเริ่มในการทำกระบวนการ configuration ถ้ามี

การเปลี่ยนแปลงระดับของสัญญาณจาก 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

4.7.8.2 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 เสร็จสิ้น ขานี้จะเป็น User I/O pin

- DOUT

ระหว่างกระบวนการ configuration ขานี้ใช้สำหรับเป็น output สำหรับ serial data ส่งไปยัง pin serial data input ของ slave ตัวต่อไปในลักษณะของ 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 จนกว่าจะจบการ configuration

4.8 Logic cell array family

4.8.1 XC3000

4.8.1.1 คุณลักษณะ (Features)

4.8.1.1.1 ประสิทธิภาพสูงใช้งานได้ที่ความถี่ 70-,100- 125 Mhz ขึ้นไป

4.8.1.1.2 เป็น Genration ที่ 2 ของ FPGA

- I/O function
- Digital logic function
- Interconnection

4.8.1.1.3 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

4.8.1.1.4 เป็นผลิตภัณฑ์มาตรฐาน

- ใช้พลังงานต่ำ ,เป็น CMOS ,ใช้เทคโนโลยี Static Memory
- ประสิทธิภาพเทียบเท่ากับการใช้ TTL SSI/MIS
- ผ่านการทดสอบจากโรงงานแล้ว 100%
- สามารถเลือก Configuration Mode ได้
- มี Development software ช่วยในการพัฒนา (XACT development software)สมบูรณ์แบบ
- สามารถ capture ผังวงจรได้
- มี Automatic Place/Route

- ทำการจำลองวงจร และหาฐานเวลาได้
- มีตัวที่ใช้แก้ไขในการออกแบบได้
- มี library และ user macros
- ทำการคำนวณหาค่าของฐานเวลาได้
- มี XACTOR In-Circuit Verifier
- มีมาตราฐานของแฟ้มข้อมูลของตัว PROM

4.8.1.2 รายละเอียด (description)

LCA ตระกูล XC3000 Logic cell array (LCA) family ประกอบไปด้วยวงจรลอจิกที่มีความหนาแน่นและประสิทธิภาพสูง ความยืดหยุ่น,และการโปรแกรมได้ user array architecture เกิดขึ้นมาจาก configuration program ที่อยู่ภายใน และส่วนประกอบที่สามารถปรับเปลี่ยนเพื่อให้ได้วงจรตามต้องการได้ 3 แบบ คือ IOB, array of CLB, เชื่อมต่อภายในตามความต้องการของผู้ใช้ logic function และ การเชื่อมต่อภายในถูกกำหนดด้วย โปรแกรมที่อยู่ในหน่วยความจำ ภายในตัว LCA โปรแกรมนี้ Load เข้าสู่ Memory ได้ในหลายๆ รูปแบบตามความเหมาะสมที่ใช้งานโปรแกรมถูกบรรจุอยู่ใน 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

ตารางที่ 4-2 เปรียบเทียบขนาดของ XC3000 LCA family

4.8.2 XC4000

4.8.2.1 คุณลักษณะ

4.8.2.1.1 เป็น Generation ที่ 3 ของ FPGA

- มี Flip-Flop เป็นจำนวนมาก
- ในการผลิตฟังก์ชั่นของการทำงานมีความยืดหยุ่นสูง
- มี ultra-fast RAM ในตัว
- ใช้กับงานที่ต้องการความเร็วสูง
- มี wide edge decoder
- Interconnect line เป็นแบบ Hierachy
- สามารถใช้ 3-State Bus ภายในได้
- มีการกระจายกำลังงานของสัญญาณต่ำ

4.8.2.1.2 มีสถาปัตยกรรมภายในที่ยืดหยุ่น

- มี logic blocks และ I/O blocks ที่โปรแกรมได้
- มี interconnect และ wide decoder ที่โปรแกรมได้

4.8.2.1.3 ทำกระบวนการ sub-micron ชนิด CMOS ได้

- มี logic และ interconnect ที่มีความเร็วสูง
- ใช้กำลังงานต่ำ

4.8.2.1.4 คุณลักษณะทาง System-Oriented

- รองรับมาตราฐาน IEEE 1149.1 ในการทำ boundary-scan logic
- สามารถโปรแกรมค่า output slew rate ได้
- สามารถโปรแกรมให้ input มี pull-up หรือ pull-downResister ได้
- ให้กระแส output ได้ตั้งแต่ 12-24 mA (แล้วแต่รุ่น)

4.8.2.1.5 ทำการ config โดยการโหลดเอาแฟ้มข้อมูล Binary

- ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
- มี mode ในการโปรแกรมให้เลือก 6 modes

4.8.2.1.6 มีโปรแกรม XACT Development System ที่ทำงานบน PC '386/'486, NEC PC, Apollo, Sun-4, และ Hewlett-Packard 700 series

- สามารถติดต่อกับโปรแกรมอื่นๆได้ เช่น VIEWlogic, Mentor Graphics และ OrCAD
- มี automatic Place and Routing ที่ครบ
- มี Interactive Design Editor ที่ใช้สำหรับการทำ optimization
- มี 288 macros, 34 hard macros, RAM/ROM compiler

4.8.2.2 รายละเอียด

LCA ตระกูล XC4000 ประกอบไปด้วยวงจร logic ที่มีความหนาแน่นสูง, มีความหยืดหยุ่น, การโปรแกรมสถาปัตยกรรมของ CLBs ที่ต่อกันภายในมีประสิทธิภาพมากในวิธีการแบบ Hierachy, และการต่อ IOBs ที่อยู่รอบๆก็ใช้วิธีการ perimeter

บทที่ 5
การประยุกต์ใช้งาน

ในบทนี้จะนำเสนอการออกระบบเชิงตัวเลข โดยใช้วิธีการออกแบบเป็นรูปเดียว(Flatten Design)

5.1 แนวความคิดในการออกแบบ

5.1.1 ชื่อระบบเชิงตัวเลข

ระบบจำลองการทำงานของเครื่องบันทึกเทป

(Tape Recorder Simmulater.)

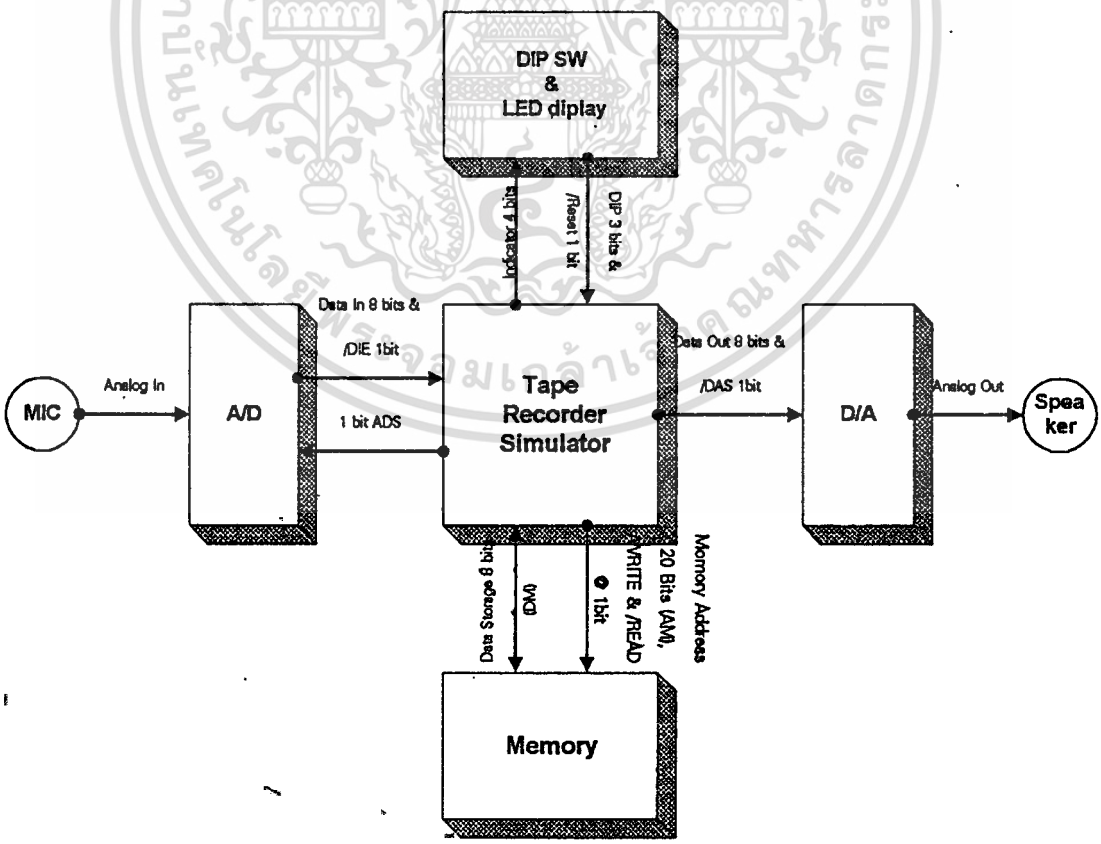
5.1.2 คุณสมบัติ

5.1.2.1 เล่นและเก็บเสียง โดยผ่าน memory แทนเทปแม่เหล็ก

5.1.2.2 เมื่อเก็บเสียงจนเต็มแล้ว จะต้องมีการแสดงผล

5.1.2.3 มีการกรอกลับมาเริ่มต้นใหม่

5.1.3 แผนผังการทำงาน



รูปที่ 5-1

5.1.4 การทำงานเบื้องต้น

5.1.4.1 การเล่นกลับ (PLAY(back) MODE)

5.1.4.1.1 ถ้าสวิทช์ /PLAY = 0 ให้รอสัญญาณนาฬิกาเป็นขอบขาขึ้น ไปทำข้อ 5.1.4.1.2 ต่อ ถ้าไม่ใช่ให้รอจนกว่าจะเป็นจริง

5.1.4.1.2 นำข้อมูลที่ถูกเก็บอยู่ใน memory ออกไปให้ D/A

5.1.4.1.3 เพิ่ม AM ขึ้น 1

5.1.4.1.4 กลับไปทำข้อ 5.4.1.1 ใหม่

5.1.4.2 การบันทึกเสียง (RECORD MODE)

5.1.4.2.1 ถ้าสวิทช์ /RECORD = 0 ให้รอสัญญาณนาฬิกาเป็นขอบขาขึ้น ไปทำข้อ 5.1.4.2.2 ต่อ ถ้าไม่ใช่ให้รอจนกว่าจะเป็นจริง

5.1.4.2.2 นำข้อมูลจาก A/D ไปเก็บใน memory

5.1.4.2.3 เพิ่ม AM ขึ้น 1

5.1.4.2.4 กลับไปทำข้อ 5.4.2.1 ใหม่

5.1.4.3 การกรอกลับ (REWIND MODE)

5.1.4.3.1 ถ้าสวิทช์ /REWIND = 0 ให้รอสัญญาณนาฬิกาเป็นขอบขาขึ้น ไปทำข้อ 5.1.4.3.2 ต่อ ถ้าไม่ใช่ให้รอจนกว่าจะเป็นจริง

5.1.4.3.2 ให้ AM = 0

5.1.4.3.3 กลับไปทำข้อ 1 ใหม่

5.1.5 ตารางการทำงานของสวิทช์

/PLAY	/RECORD	/REWIND	MODE
OFF	OFF	OFF	No Operate
OFF	OFF	ON	REWIND
OFF	ON	OFF	RECORD
OFF	ON	ON	REWIND
ON	OFF	OFF	PLAY
ON	OFF	ON	PLAY
ON	ON	OFF	PLAY
ON	ON	ON	PLAY

ตารางที่ 5-1

5.2 การออกแบบตัวบรรยายภาษา VHDL

เราได้ Block Diagram ของระบบ และการทำงานเบื้องต้น ต่อไปเราจะทำการสร้าง source code โดยมีขั้นตอนดังนี้

5.2.1Entity Design

เราจะดูจาก Block Diagram ซึ่งจะได้ port ดังนี้

5.2.1.1 Input port ประกอบด้วย

- SW (/RESET, /PLAY, /RECORD, /REWIND)
- CLK (clock)
- /DIE (Data In Enable)
- ADS (Analog to Digital Select)
- DI_7 - DI_0 (Data In 8 bits)

5.2.1.2 Output port ประกอบด้วย

- /DAS (Digital to Analog Select)
- /READ (Memory Read)
- /WRITE (Memory Write)
- AM_{19} - AM_0 (Address Memory 20 bits)
- LED (Play mode indecater, Record mode indecater, Rewind mode indicater, Memory full indicater)

5.2.1.3 In/Out port

- DM_7 - DM_0 (Data Memory 8 bits)

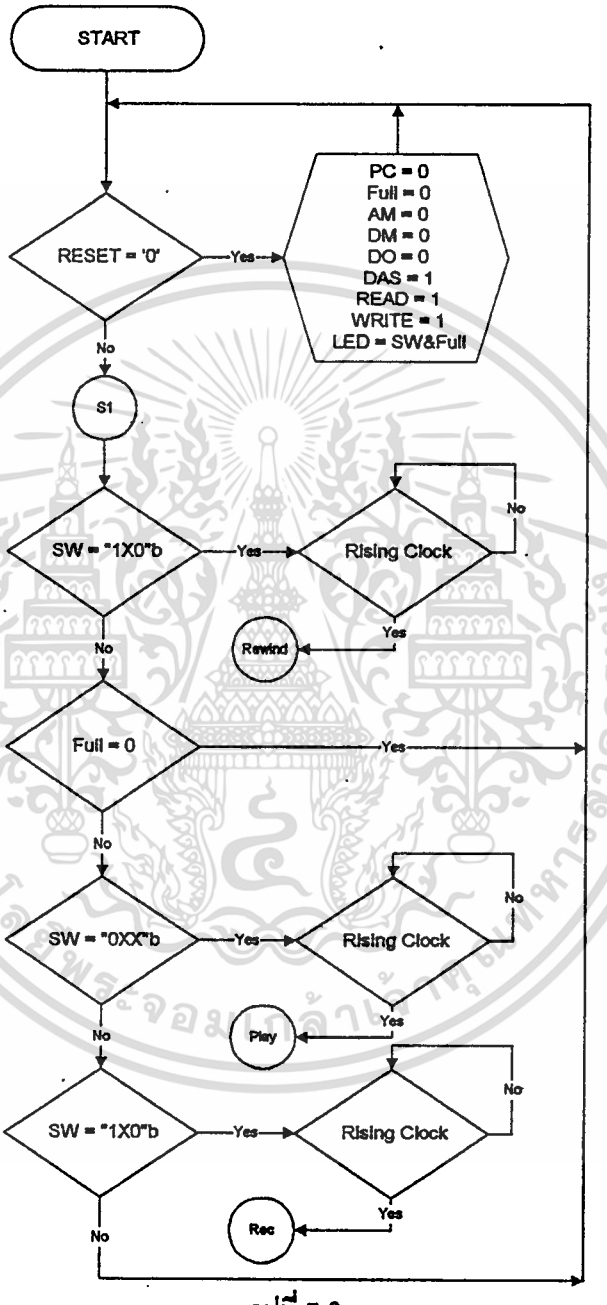
5.2.1.4 entity declaration

```
entity TapeSim is
port ( signal clk, reset, DIE : in vlbit;
       signal SW : in vlbit_vector(2 downto 0);
       signal DI : in vlbit_vector(7 downto 0);
       signal DM : inout vlbit_vector(7 downto 0);
       signal DO : out vlbit_vector(7 downto 0);
       signal AM : out vlbit_vector(19 downto 0);
       signal LED : out vlbit_vector(3 downto 0);
       signal WRITE, READ, DAS, ADS : out vlbit);
end TapeSim;
```

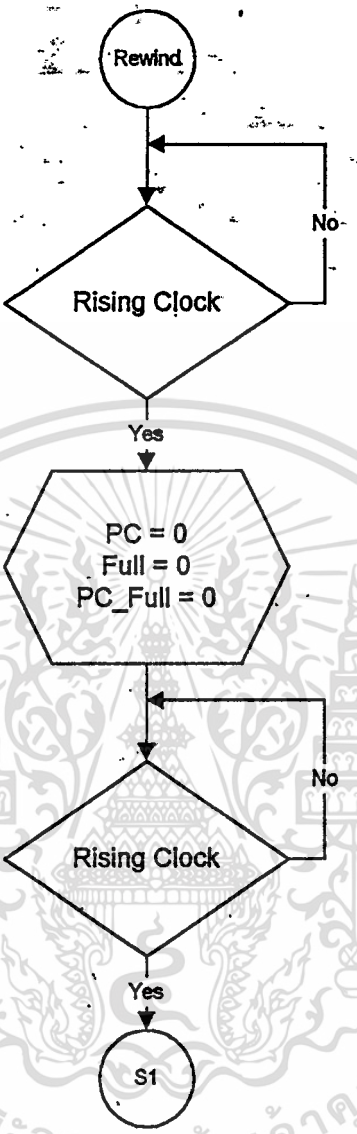
5.2.2 Architecture Design

จากการทำงานเบื้องต้นเรานำมาใช้ในการออกแบบ โดยมีขั้นตอนดังนี้

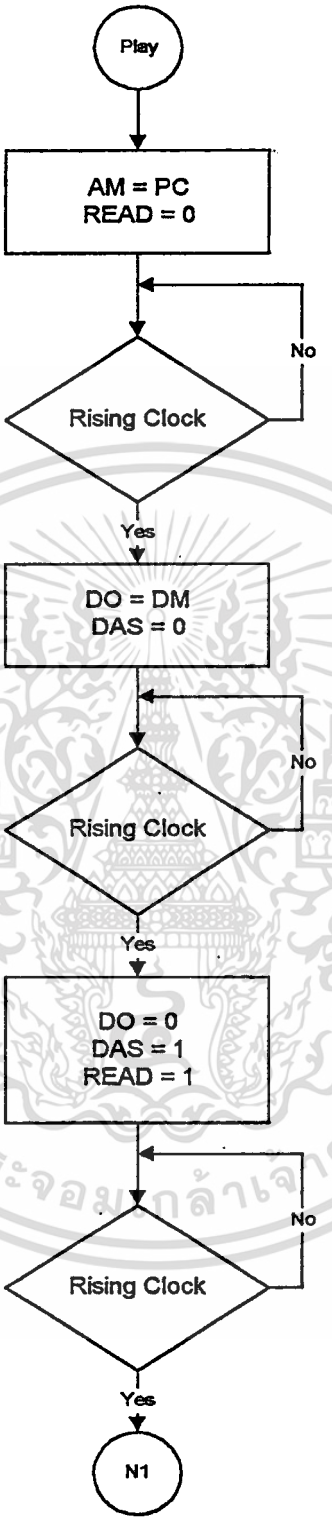
5.2.2.1 ทำการเขียน Flow chart



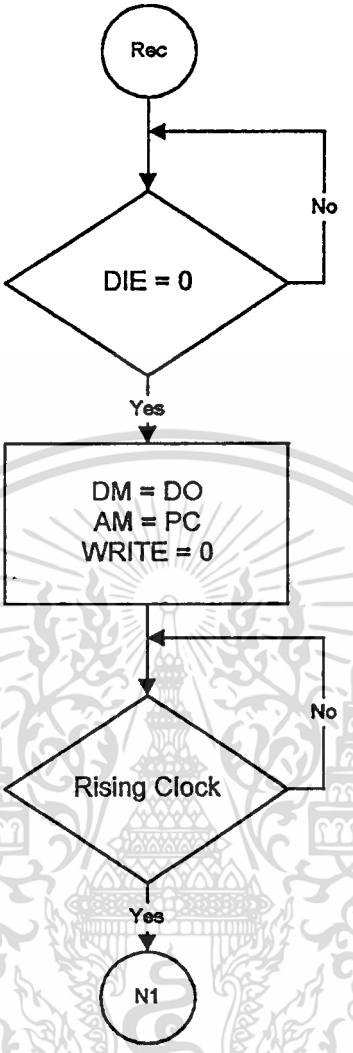
รูปที่ 5-2



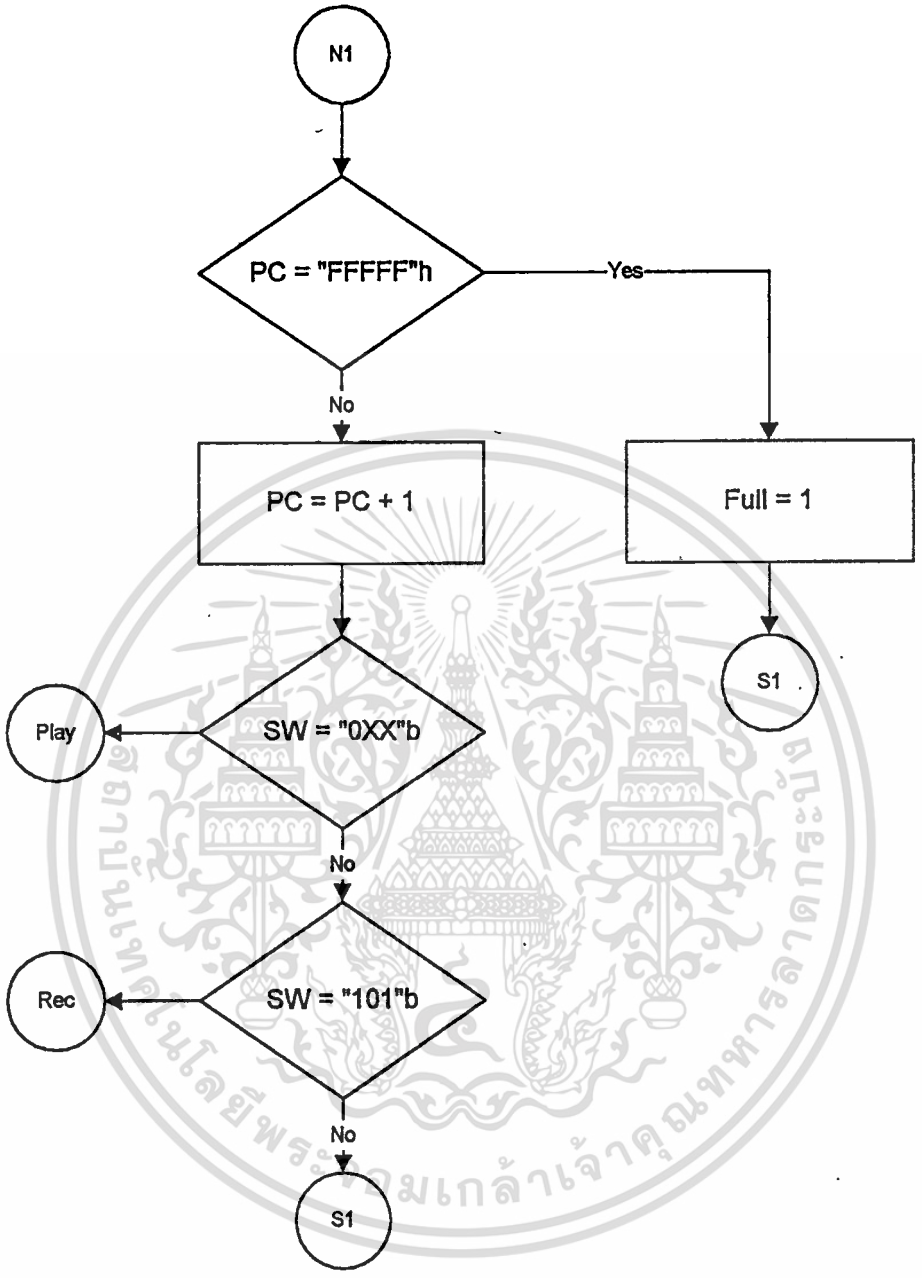
รูปที่ 5-3



รูปที่ 5-4

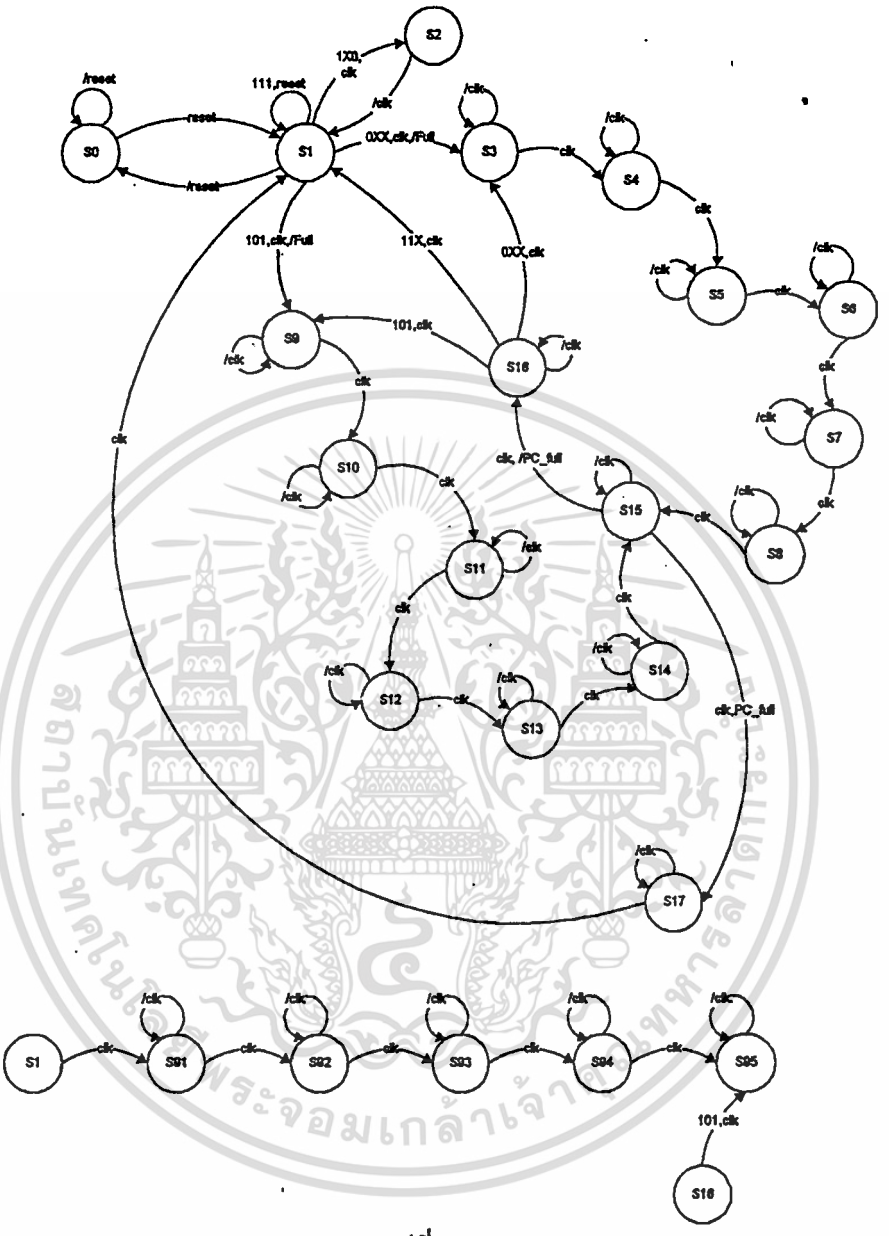


รูปที่ 5-5



รูปที่ 5-6

5.2.2.2 จาก Flow Chart มาเขียนเป็น State Diagram ได้



รูปที่ 5-7

– S0 – Reset Mode

– PC_full = 0

– PC = 0

– Full = 0

– READ = 1

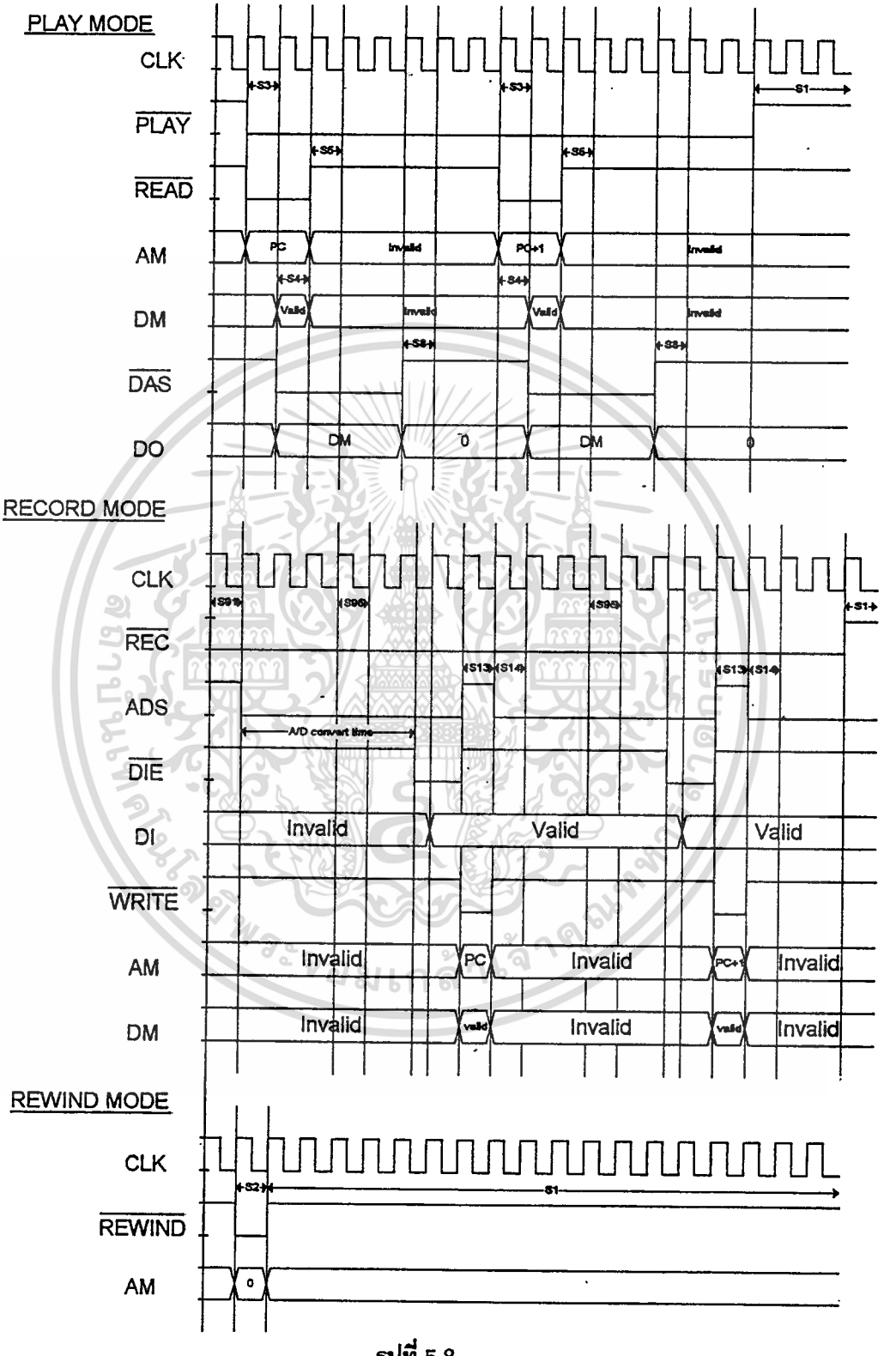
– WRITE = 1

– DAS = 1

- ADS = 0
- AM = 0
- DO = 0

- S1 - No Operate Mode
- S2 - Rewind Mode
  - PC = 0
  - Full = 0
  - PC_full = 0
- S3 - Play Mode
  - READ = 0
  - AM = PC
- S4 - DAS = 0
  - DM = DO
- S5 - READ = 1
- S8 - DAS = 1
  - DO = 0
- S9 - Record Mode
- S91 - ADS = 1
- S92 - ADS = 0
- S13 - DM = DI
  - AM = PC
  - WRITE = 0
  - ADS = 1
- S14 - WRITE = 1
  - ADS = 0
- S15 - IF PC = "FFFFF"h THEN PC_full = 1
  ELSE PC_full = 0
- S16 - PC = PC+1
- S17 - Full = 1

5.2.2.3 เขียน Timming Diagram ได้ดังรูป



รูปที่ 5-8

5.2.2.4 กำหนด state ให้เป็นค่า binary

จาก state diagram จะเห็นว่าจำนวน state ทั้งหมดมี 27 states จะต้องใช้ binary อย่างน้อย 5 bits เพราะฉะนั้นจะได้ค่า binary ของ state ต่างๆ ดังนี้

S0 = "00000"b

S1 = "00001"b

S2 = "00010"b

S3 = "00011"b

S4 = "00100"b

S5 = "00101"b

S6 = "00110"b

S7 = "00111"b

S8 = "01000"b

S91 = "01001"b

S92 = "10010"b

S93 = "10011"b

S94 = "10100"b

S95 = "10101"b

S10 = "01010"b

S11 = "01011"b

S12 = "01100"b

S13 = "01101"b

S14 = "01110"b

S15 = "01111"b

S16 = "10000"b

S17 = "10001"b

5.2.2.5 เขียนเป็นตารางได้ดังนี้

SW	Full	PC_full	STATE	NEXT STATE
111	X	X	S1	S1
1X0	X	X	S1	S2
XXX	X	X	S2	S1
0XX	0	X	S1	S3
XXX	X	X	S3	S4
XXX	X	X	S4	S5
XXX	X	X	S5	S6
XXX	X	X	S6	S7
XXX	X	X	S7	S8
XXX	X	X	S8	S15
101	0	X	S1	S91
XXX	X	X	S91	S92
XXX	X	X	S92	S93
XXX	X	X	S93	S94
XXX	X	X	S94	S95
XXX	X	X	S95	S10
XXX	X	X	S10	S11
XXX	X	X	S11	S12
XXX	X	X	S12	S13
XXX	X	X	S13	S14
XXX	X	X	S14	S15
XXX	X	0	S15	S16
XXX	X	1	S15	S17
0XX	X	X	S16	S3
11X	X	X	S16	S1
101	X	X	S16	S95
XXX	X	X	S17	S1

ตารางที่ 5-1

5.2.2.6 Architecture declaration

```
-------------------------------------
-- Architectural behavioral description
architecture main_control of TapeSim is

  constant  one : vlbit_1d (18 downto 0) :=
B"000_0000_0000_0000_0001";
  constant PC_MAX : vlbit_1d (19 downto 0) := B"1111
_1111_1111_1111_1111";

-------------------------------------
--      Finite State Machine Aligment
  constant S1  : vlbit_1d(4 downto 0) := B"00001";
  constant S2  : vlbit_1d(4 downto 0) := B"00010";
  constant S3  : vlbit_1d(4 downto 0) := B"00011";
  constant S4  : vlbit_1d(4 downto 0) := B"00100";
  constant S5  : vlbit_1d(4 downto 0) := B"00101";
  constant S6  : vlbit_1d(4 downto 0) := B"00110";
  constant S7  : vlbit_1d(4 downto 0) := B"00111";
  constant S8  : vlbit_1d(4 downto 0) := B"01000";
  constant S91 : vlbit_1d(4 downto 0) := B"01001";
  constant S92 : vlbit_1d(4 downto 0) := B"10010";
  constant S93 : vlbit_1d(4 downto 0) := B"10011";
  constant S94 : vlbit_1d(4 downto 0) := B"10100";
  constant S95 : vlbit_1d(4 downto 0) := B"10101";
  constant S10 : vlbit_1d(4 downto 0) := B"01010";
  constant S11 : vlbit_1d(4 downto 0) := B"01011";
  constant S12 : vlbit_1d(4 downto 0) := B"01100";
  constant S13 : vlbit_1d(4 downto 0) := B"01101";
  constant S14 : vlbit_1d(4 downto 0) := B"01110";
  constant S15 : vlbit_1d(4 downto 0) := B"01111";
  constant S16 : vlbit_1d(4 downto 0) := B"10000";
```

```
  constant S17 : vlbit_1d(4 downto 0) := B"10001";

  constant  table  :  vlbit_2d (0 to 27,14 downto 0) := (
-- SW,Full,PC_full I State I Next_state
-------------------------------
-----        No Operate        -----
-------------------------------
 B"111XX_00001_00001",  -- S1 => S1
 B"XXX1X_00001_00001",
-------------------------------
-----        Rewind State        -----
-------------------------------
 B"1X0XX_00001_00010",  -- S1 => S2
 B"XXXXX_00010_00001",  -- S2 => S1
-------------------------------
-----        Play State        -----
-------------------------------
 B"0XX0X_00001_00011",  -- S1 => S3
 B"XXXXX_00011_00100",  -- S3 => S4
 B"XXXXX_00100_00101",  -- S4 => S5
 B"XXXXX_00101_00110",  -- S5 => S6
 B"XXXXX_00110_00111",  -- S6 => S7
 B"XXXXX_00111_01000",  -- S7 => S8
 B"XXXXX_01000_01111",  -- S8 => S15 -- to N1
-------------------------------
-----        Record State        -----
-------------------------------
 B"1010X_00001_01001",  -- S1 => S91
 B"XXXXX_01001_10010",  -- S91=> S92
 B"XXXXX_10010_10011",  -- S92=> S93
 B"XXXXX_10011_10100",  -- S93=> S94
 B"XXXXX_10100_10101",  -- S94=> S95
```

```
B"XXXXX_10101_01010",  -- S95=> S10
B"XXXXX_01010_01011",  --S10 => S11
B"XXXXX_01011_01100",  --S11 => S12
B"XXXXX_01100_01101",  --S12 => S13
B"XXXXX_01101_01110",  --S13 => S14
B"XXXXX_01110_01111",  --S14 => S15 -- to N1
-------------------------------
----          N1          -----
-------------------------------
B"XXXX0_01111_10000",  --S15 => S16
B"XXXX1_01111_10001",  --S15 => S17

B"0XXXX_10000_00011",  --S16 => S3
B"11XXX_10000_00001",  --S16 => S1
B"101XX_10000_10101",  --S16 => S95

B"XXXXX_10001_00001"); --S17 => S1
-------------------------------

signal htbl : vlbit_1d (9 downto 0);
signal state, next_s : vlbit_1d (4 downto 0);
signal PC : vlbit_1d (19 downto 0);
signal Full, PC_full : vlbit;
signal Din : vlbit_1d (7 downto 0);
signal Disp : vlbit_1d (2 downto 0);

begin

 pla_table (htbl, next_s, table);
 htbl <= SW&Full&PC_full&state;
 LED <= Disp&Full;
```

```
main: process
 begin
  wait until (prising (clk) or (reset = '0'));
  if reset = '0' then
     state <= S1; -- start state at S1
     PC_full <= '0';
     PC <= B"0000_0000_0000_0000_0000";
     Full <= '0';
     READ <= '1';
     WRITE <= '1';
     DAS <= '1';
     ADS <= '0';
     DO <= B"0000_0000";
     DIn <= B"0000_0000";
     Disp <= B"000";
   else
     state <= next_s;
     if    state = S1   then Disp <= B"000";
     elsif  state = S2   then PC <= B"0000_0000_0000_0000_0000";
                         Full <= '0';
                         PC_Full <= '0';
                         Disp(0) <= '1';
     elsif  state = S3   then AM <= PC;
                         READ <= '0';
                         Disp(2) <= '1';
     elsif  state = S4   then DO <= DMI;
                         DAS <= '0';
     elsif  state = S5   then READ <= '1';
     elsif  state = S8   then DO <= B"0000_0000";
                         DAS <= '1';
     elsif  state = S91  then Disp(1) <= '1';
                         ADS <= '1';
```

```
elsif  state = S92  then ADS <= '0';
elsif  state = S13  then AM <= PC;
                         DMO <= DIn;
                         WRITE <= '0';
                         ADS <= '1';
elsif  state = S14  then WRITE <= '1';
                         ADS <= '0';
elsif  state = S15  then if PC = PC_MAX then PC_full <= '1';
                         else PC_full <= '0';
                         end if;
elsif  state = S16  then if PC(19) = '1' then
                          PC <= addum(PC(18 downto 0), one);
                          PC(19) <= '1';
                         else
                          PC <= addum(PC(18 downto 0), one);
                         end if;
    elsif  state = S17  then Full <= '1';
    end if;
    if DIE = '0' then DIn <= DI;
    end if;
  end if;
 end process;
end main_control;
```

5.4 การออกแบบทางวงจรดรรก และการสังเคราะห์

การออกแบบทางวงจรดรรก และการสังเคราะห์ เราจะนำเอาSource Codeภาษา VHDL มาเป็นข้อมูลให้กับ ViewSynthesis จากขั้นตอนที่ผ่าน โดยทำดังนี้

5.4.1 เรียกโปรแกรม ViewSynthsis

```
C:SOMWANG> VHDLDES.
VHDLDes - V2.21; Workview 4.1.3 062292, 3000 Series
c Copyright 1985,1992 by Viewlogic Systems, Inc.
1:  VHDLDes> _
```

5.4.2 เลือก technology library

โดยพิจารณาจาก จำนวนขาที่ใช้, จำนวน CLBs ที่ต้องใช้ และความเร็วในการตอบสนองที่ระบบต้องการ (ดูจาก Data Sheet ของ FPGA) จากระบบที่ทำการออกแบบนี้ต้องการใช้ขา I/O ทั้งหมด 58 ขา จำนวน CLBs ที่ใช้ประมาณ 2,200 ชุด ความเร็วสัญญาณนาฬิกา ประมาณ 64 KHz เราจะเลือกใช้ชิปเบอร์ XC4003 -6 เพราะฉะนั้นจะใช้ technology library ของ X4000

```
1: VDHLDes> technology X4000
 -- reading library file C:>WORKVIEW\STANDARD\X4000.SML
Processing Viewlogic X4000 Library, Version - 2/22/93

Library 'X4000' is in database.
```

5.4.3 ทำการตรวจสอบระหัสต้นแบบภาษา VHDL

```
2: VHDLDes> vhdl tapesim.vhd

>> Compiling <tapesim.vhd> ...

-- analyzing ENTITY tapesim at line 28 ...

   0 errors
   0 warnings

-- analyzing ARCHITECTURE main_control of ENTITY tapesim at line 39 ...

   0 errors
   0 warnings

0 errors
0 warnings

>>> VHDL analysis done.

Design is in database.
```

5.4.4 ทำการสังเคราะห์วงจรตรรก

3: VHDLDes> synthesize

----- Date : Sat Apr 29 17:03:17 1995
----- Module Name : TAPESIM
----- Library Name : X4000
----- Output Directory : .
----- Operating Voltage: 5.000000
----- Operating Temp : 25
----- Process : TYPICAL
----- Logic Type : MIXED
----- area/speed Factor: 1.000000
----- Package : DEFAULT

****

Output Drives

****

Output	Drive		Output	Drive	
	rise	fall		rise	fall
DO7	0.0000	0.0000	DO6	0.0000	0.0000
DO5	0.0000	0.0000	DO4	0.0000	0.0000
DO3	0.0000	0.0000	DO2	0.0000	0.0000
DO1	0.0000	0.0000	DO0	0.0000	0.0000
DMO7	0.0000	0.0000	DMO6	0.0000	0.0000
DMO5	0.0000	0.0000	DMO4	0.0000	0.0000
DMO3	0.0000	0.0000	DMO2	0.0000	0.0000
DMO1	0.0000	0.0000	DMO0	0.0000	0.0000
AM19	0.0000	0.0000	AM18	0.0000	0.0000

AM17	0.0000	0.0000	AM16	0.0000	0.0000
AM15	0.0000	0.0000	AM14	0.0000	0.0000
AM13	0.0000	0.0000	AM12	0.0000	0.0000
AM11	0.0000	0.0000	AM10	0.0000	0.0000
AM9	0.0000	0.0000	AM8	0.0000	0.0000
AM7	0.0000	0.0000	AM6	0.0000	0.0000
AM5	0.0000	0.0000	AM4	0.0000	0.0000
AM3	0.0000	0.0000	AM2	0.0000	0.0000
AM1	0.0000	0.0000	AM0	0.0000	0.0000
LED3	0.0000	0.0000	LED2	0.0000	0.0000
LED1	0.0000	0.0000	LED0	0.0000	0.0000
WRITE	0.0000	0.0000	READ	0.0000	0.0000
DAS	0.0000	0.0000	ADS	0.0000	0.0000

****

Input Loadings

****

Input	Load	Input	Load
CLK	78.000	RESET	1.000
DIE	9.000	SW2	3.000
SW1	1.000	SW0	1.000
DI7	1.000	DI6	1.000
DI5	1.000	DI4	1.000
DI3	1.000	DI2	1.000
DI1	1.000	DI0	1.000
DMI7	1.000	DMI6	1.000
DMI5	1.000	DMI4	1.000
DMI3	1.000	DMI2	1.000

DMI1 1.000 DMI0 1.000

****

Timing Report

****

Output	Delay		Required		Load
	rise	fall	rise	fall	
AM0MEM	7.00	7.00	<none>	<none>	1.0
AM1MEM	7.00	7.00	<none>	<none>	1.0
AM2MEM	7.00	7.00	<none>	<none>	1.0
AM3MEM	7.00	7.00	<none>	<none>	1.0
AM4MEM	7.00	7.00	<none>	<none>	1.0
AM5MEM	7.00	7.00	<none>	<none>	1.0
AM6MEM	7.00	7.00	<none>	<none>	1.0
AM7MEM	7.00	7.00	<none>	<none>	1.0
AM8MEM	7.00	7.00	<none>	<none>	1.0
AM9MEM	7.00	7.00	<none>	<none>	1.0
AM10MEM	7.00	7.00	<none>	<none>	1.0
AM11MEM	7.00	7.00	<none>	<none>	1.0
AM12MEM	7.00	7.00	<none>	<none>	1.0
AM13MEM	7.00	7.00	<none>	<none>	1.0
AM14MEM	7.00	7.00	<none>	<none>	1.0
AM15MEM	7.00	7.00	<none>	<none>	1.0
AM16MEM	7.00	7.00	<none>	<none>	1.0
AM17MEM	7.00	7.00	<none>	<none>	1.0
AM18MEM	7.00	7.00	<none>	<none>	1.0

AM19MEM	7.00	7.00	<none>	<none>	1.0
READMEM	6.00	6.00	<none>	<none>	1.0
READSET	1.00	1.00	<none>	<none>	50.0
DO0MEM	6.00	6.00	<none>	<none>	1.0
DO0CLR	1.00	1.00	<none>	<none>	50.0
DO1MEM	6.00	6.00	<none>	<none>	1.0
DO1CLR	1.00	1.00	<none>	<none>	50.0
DO2MEM	6.00	6.00	<none>	<none>	1.0
DO2CLR	1.00	1.00	<none>	<none>	50.0
DO3MEM	6.00	6.00	<none>	<none>	1.0
DO3CLR	1.00	1.00	<none>	<none>	50.0
DO4MEM	6.00	6.00	<none>	<none>	1.0
DO4CLR	1.00	1.00	<none>	<none>	50.0
DO5MEM	6.00	6.00	<none>	<none>	1.0
DO5CLR	1.00	1.00	<none>	<none>	50.0
DO6MEM	6.00	6.00	<none>	<none>	1.0
DO6CLR	1.00	1.00	<none>	<none>	50.0
DO7MEM	6.00	6.00	<none>	<none>	1.0
DO7CLR	1.00	1.00	<none>	<none>	50.0
ADSMEM	5.00	5.00	<none>	<none>	1.0
ADSCLR	1.00	1.00	<none>	<none>	50.0
DASMEM	5.00	5.00	<none>	<none>	1.0
DASSET	1.00	1.00	<none>	<none>	50.0
DMO0MEM	6.00	6.00	<none>	<none>	1.0
DMO1MEM	6.00	6.00	<none>	<none>	1.0
DMO2MEM	6.00	6.00	<none>	<none>	1.0
DMO3MEM	6.00	6.00	<none>	<none>	1.0
DMO4MEM	6.00	6.00	<none>	<none>	1.0
DMO5MEM	6.00	6.00	<none>	<none>	1.0
DMO6MEM	6.00	6.00	<none>	<none>	1.0
DMO7MEM	6.00	6.00	<none>	<none>	1.0
L1_PC_E0MEM	6.00	6.00	<none>	<none>	1.0

L1_PC_E0CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E1MEM	6.00	6.00	<none>	<none>	1.0
L1_PC_E1CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E2MEM	7.00	7.00	<none>	<none>	1.0
L1_PC_E2CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E3MEM	8.00	8.00	<none>	<none>	1.0
L1_PC_E3CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E4MEM	9.00	9.00	<none>	<none>	1.0
L1_PC_E4CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E5MEM	10.00	10.00	<none>	<none>	1.0
L1_PC_E5CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E6MEM	11.00	11.00	<none>	<none>	1.0
L1_PC_E6CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E7MEM	12.00	12.00	<none>	<none>	1.0
L1_PC_E7CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E8MEM	13.00	13.00	<none>	<none>	1.0
L1_PC_E8CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E9MEM	14.00	14.00	<none>	<none>	1.0
L1_PC_E9CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E10MEM	15.00	15.00	<none>	<none>	1.0
L1_PC_E10CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E11MEM	16.00	16.00	<none>	<none>	1.0
L1_PC_E11CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E12MEM	17.00	17.00	<none>	<none>	1.0
L1_PC_E12CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E13MEM	18.00	18.00	<none>	<none>	1.0
L1_PC_E13CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E14MEM	19.00	19.00	<none>	<none>	1.0
L1_PC_E14CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E15MEM	20.00	20.00	<none>	<none>	1.0
L1_PC_E15CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E16MEM	21.00	21.00	<none>	<none>	1.0

L1_PC_E16CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E17MEM	22.00	22.00	<none>	<none>	1.0
L1_PC_E17CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E18MEM	23.00	23.00	<none>	<none>	1.0
L1_PC_E18CLR	1.00	1.00	<none>	<none>	50.0
L1_PC_E19MEM	20.00	21.00	<none>	<none>	1.0
L1_PC_E19CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E0MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E0CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E1MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E1CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E2MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E2CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E3MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E3CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E4MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E4CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E5MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E5CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E6MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E6CLR	1.00	1.00	<none>	<none>	50.0
L1_DIN_E7MEM	3.00	3.00	<none>	<none>	1.0
L1_DIN_E7CLR	1.00	1.00	<none>	<none>	50.0
L1_DISP_E0MEM	5.00	5.00	<none>	<none>	1.0
L1_DISP_E0SET	1.00	1.00	<none>	<none>	50.0
L1_DISP_E1MEM	5.00	5.00	<none>	<none>	1.0
L1_DISP_E1CLR	1.00	1.00	<none>	<none>	50.0
L1_DISP_E2MEM	5.00	5.00	<none>	<none>	1.0
L1_DISP_E2CLR	1.00	1.00	<none>	<none>	50.0
L1_FULLMEM	5.00	5.00	<none>	<none>	1.0
L1_FULLCLR	1.00	1.00	<none>	<none>	50.0
WRITEMEM	6.00	6.00	<none>	<none>	1.0

WRITESET	1.00	1.00	<none>	<none>	50.0
L1_PC_FULLMEM	20.00	21.00	<none>	<none>	1.0
L1_PC_FULLCLR	1.00	1.00	<none>	<none>	50.0
L1_STATE_E0MEM	6.00	6.00	<none>	<none>	1.0
L1_STATE_E0SET	1.00	1.00	<none>	<none>	50.0
L1_STATE_E1MEM	8.00	8.00	<none>	<none>	1.0
L1_STATE_E1CLR	1.00	1.00	<none>	<none>	50.0
L1_STATE_E2MEM	7.00	7.00	<none>	<none>	1.0
L1_STATE_E2CLR	1.00	1.00	<none>	<none>	50.0
L1_STATE_E3MEM	7.00	7.00	<none>	<none>	1.0
L1_STATE_E3CLR	1.00	1.00	<none>	<none>	50.0
L1_STATE_E4MEM	6.00	5.00	<none>	<none>	1.0
L1_STATE_E4CLR	1.00	1.00	<none>	<none>	50.0
DO7	1.00	2.00	<none>	<none>	1.0
DO6	1.00	2.00	<none>	<none>	1.0
DO5	1.00	2.00	<none>	<none>	1.0
DO4	1.00	2.00	<none>	<none>	1.0
DO3	1.00	2.00	<none>	<none>	1.0
DO2	1.00	2.00	<none>	<none>	1.0
DO1	1.00	2.00	<none>	<none>	1.0
DO0	1.00	2.00	<none>	<none>	1.0
DMO7	1.00	1.00	<none>	<none>	1.0
DMO6	1.00	1.00	<none>	<none>	1.0
DMO5	1.00	1.00	<none>	<none>	1.0
DMO4	1.00	1.00	<none>	<none>	1.0
DMO3	1.00	1.00	<none>	<none>	1.0
DMO2	1.00	1.00	<none>	<none>	1.0
DMO1	1.00	1.00	<none>	<none>	1.0
DMO0	1.00	1.00	<none>	<none>	1.0
AM19	1.00	1.00	<none>	<none>	1.0
AM18	1.00	1.00	<none>	<none>	1.0
AM17	1.00	1.00	<none>	<none>	1.0

AM16	1.00	1.00	<none>	<none>	1.0
AM15	1.00	1.00	<none>	<none>	1.0
AM14	1.00	1.00	<none>	<none>	1.0
AM13	1.00	1.00	<none>	<none>	1.0
AM12	1.00	1.00	<none>	<none>	1.0
AM11	1.00	1.00	<none>	<none>	1.0
AM10	1.00	1.00	<none>	<none>	1.0
AM9	1.00	1.00	<none>	<none>	1.0
AM8	1.00	1.00	<none>	<none>	1.0
AM7	1.00	1.00	<none>	<none>	1.0
AM6	1.00	1.00	<none>	<none>	1.0
AM5	1.00	1.00	<none>	<none>	1.0
AM4	1.00	1.00	<none>	<none>	1.0
AM3	1.00	1.00	<none>	<none>	1.0
AM2	1.00	1.00	<none>	<none>	1.0
AM1	1.00	1.00	<none>	<none>	1.0
AM0	1.00	1.00	<none>	<none>	1.0
LED3	1.00	2.00	<none>	<none>	1.0
LED2	1.00	2.00	<none>	<none>	1.0
LED1	2.00	1.00	<none>	<none>	1.0
LED0	1.00	2.00	<none>	<none>	3.0
WRITE	2.00	1.00	<none>	<none>	1.0
READ	2.00	1.00	<none>	<none>	1.0
DAS	2.00	1.00	<none>	<none>	1.0
ADS	1.00	2.00	<none>	<none>	1.0

Total Delta	0.00	0.00
Max. Delay	23.00	23.00
Min. Delay	1.00	1.00

****

Gate Usage Summary

****

Cell	Count	Area/Cell	Cell	Count	Area/Cell
MX4000:FD	28	0.00	MX4000:FDR	45	0.00
MX4000:FDS	5	0.00	X4000:AND2	72	0.25
X4000:INV	57	0.00	X4000:NAND2	122	0.25
X4000:NAND4	2	0.75	X4000:NOR3	7	0.50
X4000:NOR4	2	0.75	X4000:OR2	130	0.25
X4000:OR3	9	0.50	X4000:OR4	5	0.75
X4000:XOR2	2	0.25			

Total Cells : 486 Total Area : 96.25

****

Netlist Statistics

****

Maximum level of gates = 21 Total number of nets = 508

Maximum pins per net = 79 Average pins per net = 3.05

Maximum load per net = 78.00 Average load per net = 1.97

5.4.5 สร้างแฟ้ม wirelist

4: VHDLDes> wire

Netlist (WIR) file was generated for 'TAPESIM'.

5.4.6 สร้างแฟ้มรายงานของการสังเคราะห์

5: VHDLDes> report

Report (RPT) file was generated for 'TAPESIM'.

5.4.7 สร้างผังวงจร

6: VHDLDes> viewgen

7: VHDLDes> quit

5.5 การออกแบบทางกายภาพ และการทำเครื่องต้นแบบ

การออกแบบทางกายภาพนั้น เราจะใช้ Viewdraw มาทำ port mapping แล้วส่งต่อให้ XACT ทำการแปลงให้เป็นแฟ้ม bit stream เพื่อนำลงสู่ EEPROM จากนั้นก็จะทำเครื่องต้นแบบ

5.5.1 การทำ port mapping

จาก data sheet ของชิป XC4003 เราจะได้ I/O pins ทั้งหมดที่สามารถใช้ได้อยู่ 61 pins เราสามารถ mapping ได้ดังตารางที่ 5-2

Input Pin.		Output Pin.		InOut Pin	
signal	Pin No.	signal	Pin No.	signal	Pin No.
RESET	P26	AM0	P77	DM0	P65
CLK	P27	AM1	P78	DM1	P66
DIE	P19	AM2	P79	DM2	P67
SW0	P23	AM3	P80	DM3	P68
SW1	P24	AM4	P81	DM4	P69
SW2	P25	AM5	P82	DM5	P70
DI0	P44	AM6	P83	DM6	P71
DI1	P45	AM7	P84	DM7	P72
DI2	P46	AM8	P3		
DI3	P47	AM9	P4		
DI4	P48	AM10	P5		

Input Pin.		Output Pin.		InOut Pin	
signal	Pin No.	signal	Pin No.	signal	Pin
DI5	P49	AM11	P6		
DI6	P50	AM12	P7		
DI7	P51	AM13	P8		
		AM14	P9		
		AM15	P10		
		AM16	P13		
		AM17	P14		
		AM18	P15		
		AM19	P16		
		DO0	P35		
		DO1	P36		
		DO2	P37		
		DO3	P38		
		DO4	P57		
		DO5	P58		
		DO6	P59		
		DO7	P60		
		LED0	P39		
		LED1	P40		
		LED2	P61		
		LED3	P62		
		WRITE	P17		
		READ	P18		
		DAS	P20		

ตารางที่ 5-2

ในการทำ port mapping นั้น เราจะลดความยุ่งยากลง โดยการนำเอาหลักการของ Top-Down Design เข้ามาช่วย เราจะสร้างส่วนของ top-level ขึ้นมาครอบ มีลักษณะคล้ายกับ socket IC ซึ่งเขียนเป็นSource Codeภาษา VHDL ได้ดังนี้

```
1: ------------------------------
2: -- File    : socket.vhd
3: -- Designer : Mr.Somwang Piyapaneerut
4: -- Last update : 24/04/95
5: ------------------------------
6:
7: Library synth;
8: Use synth.stdsynth.all;
9:
10: -- Entity Declaration --- Declare Ports
11: ------------------------------
12: entity socket is
13:   port ( signal clk, reset, DIE : in vlbit;
14:        signal SW : in vlbit_vector(2 downto 0);
15:        signal DI : in vlbit_vector(7 downto 0);
16:        signal DM : inout vlbit_vector(7 downto 0);
17:        signal DO : out vlbit_vector(7 downto 0);
18:        signal AM : out vlbit_vector(19 downto 0);
19:        signal LED : out vlbit_vector(3 downto 0);
20:        signal WRITE, READ, DAS, ADS : out vlbit);
21: end socket;
22: ------------------------------
23: -- Architecture behavioral describtion
24: architecture struct of socket is
25:
26: --      Component Decalration      --
27:
28: component TapeSim
29:   port ( signal clk, reset, DIE : in vlbit;
```

```
30:         signal SW : in vlbit_vector(2 downto 0);
31:         signal DI, DMI : in vlbit_vector(7 downto 0);
32:         signal DO, DMO : out vlbit_vector(7 downto 0);
33:         signal AM : out vlbit_vector(19 downto 0);
34:         signal LED : out vlbit_vector(3 downto 0);
35:         signal WRITE, READ, DAS, ADS : out vlbit);
36:  end component;
37:
38: signal DMIn, DMOut : vlbit_vector (7 downto 0);
39: signal DMS : vlbit;
40:
41: begin
42:
43: DMS <= not WRITE;
44: DMIn <= not DM;
45: DM <= B"ZZZZ_ZZZZ" when (DMS = '1')
46:      else DMOut;
47:
48: U1:Tapesim
49: port map( clk => clk,
50:           reset => reset,
51:           DIE => DIE,
52:           SW => SW,
53:           DI => DI,
54:           DMI => DMIn,
55:           DMO => DMOut,
56:           DO => DO,
57:           AM => AM,
58:           LED => LED,
59:           WRITE => WRITE,
60:           READ => READ,
61:           DAS => DAS,
```

```
62:          ADS => ADS);
63: end struct;
```

ขาที่เป็น I/O ในส่วนที่เป็น flatten นั้น เราจะต้องแปลงให้อยู่ในรูป In หรือ Out เท่านั้น จากบรรทัดที่ 43-46 จะเป็น wire ขาที่เราทำการแปลงให้เป็น I/O อีกครั้ง แล้วทำการแปลงรหัสต้นแบบภาษา VHDL ไปเป็นผังวงจร (เหมือนการทำออกแบบในส่วนวงจรตรรก) ต่อไปในการแปลงผังวงจรไปเป็น bit stream จะกระทำกับแฟ้ม socket จากรูปที่ 2-6 OBFT จะต้องมีสัญญาณที่มาควบคุม OBUFT ให้เป็น 3-state จากขาสัญญาณที่เราออกแบบนั้น ขาสัญญาณ DM จะมีความสัมพันธ์กับสัญญาณ WRITE เราจะนำเอาสัญญาณนี้มาควบคุม เมื่อเรียก Viewdraw แล้วเราจะเปลี่ยน component ต่างๆส่วนของ CLBs ให้เป็น component ในส่วนของ IOBs (ดูคู่มือประกอบ) ดังนี้

5.5.1.1 BUFT.1 เปลี่ยนเป็น \X4000\OBUFT.1

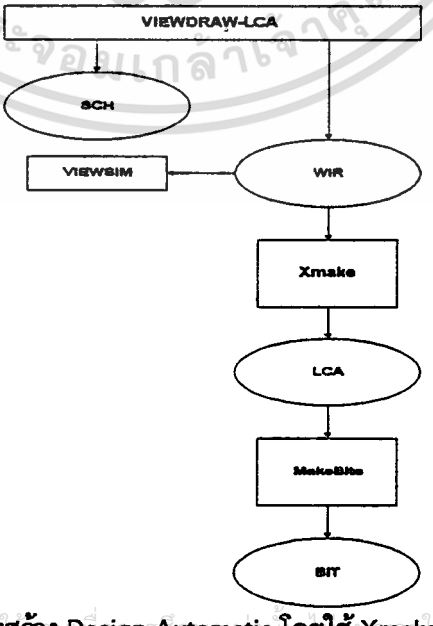
5.5.1.2 INV.1 เปลี่ยนเป็น \X4000\BUF.1

5.5.1.3 BUF.1 เปลี่ยนเป็น \X4000\IBUF.1

จากนั้นทำการลบ port ที่เป็น component ใน \SOMWANG แล้ว add ตัว component ตามข้อ 2.1.4.2.4.3 - 2.1.4.2.4.4

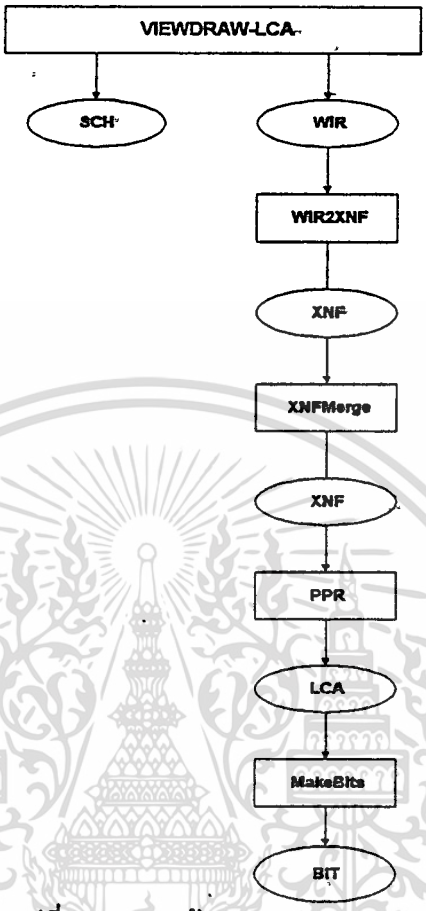
5.5.2 การแปลงแฟ้มผังวงจรเป็นแฟ้ม bit stream

เรียก XDM (Xilinx Design Manager) จากนั้นใช้ Xmake ในการแปลงแฟ้มผังวงจรเป็นแฟ้ม bit stream รูปที่ 5-9 เป็นขั้นตอนการทำ Xmake



รูปที่ 5-9 การสร้าง Design Automatic โดยใช้ Xmake ของ XILINX

หรือจะทำแบบ manual ก็ได้ โดยมีขั้นตอนดังรูปที่ 5-10



รูปที่ 5-10 การสร้าง Manaul Design

จากนั้นให้ใช้ MakePROM สร้างแฟ้มที่จะนำไปโปรแกรมลงสู่ EEPROM เพื่อสร้างเครื่องต้นแบบต่อไป

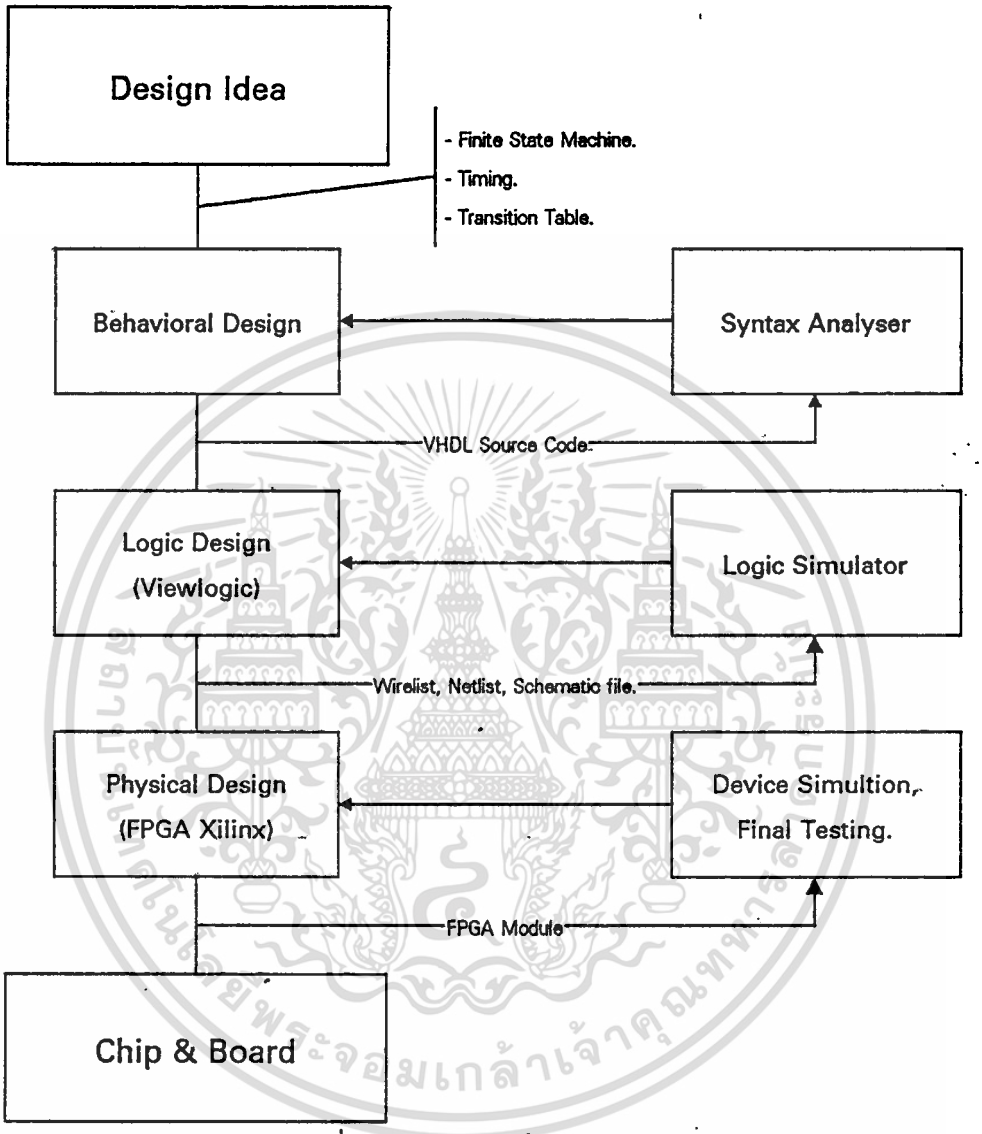
5.5.3 การสร้างเครื่องต้นแบบ

การสร้างเครื่องต้นแบบนั้น เราจะสร้างแผงวงจรจากผังวงจรแล้วประกอบเข้ากัน โดยผู้จัดทำโครงงานนี้ได้สร้างเป็น 3 แผงวงจร คือ แผงวงจร XC4003, แผงวงจร I/O และแผงวงจร Memory

5.6 การจำลอง และการทดสอบ

ในรูปที่ 5-11 แสดงการจำลองและทดสอบในแต่ละส่วนของขั้นตอนการออกแบบ โดยขั้นตอนการออกแบบตัวบรรยายทางภาษา VHDL นั้น จะใช้ตัว VHDL Analyzer ทำการสร้างแฟ้มนามสกุล '.VSM' เป็นข้อมูลให้กับ Viewsim จากนั้นสร้างแฟ้ม "TAPESIM.CMD" เป็น command

file เพื่อใช้ในการจำลองหาค่าของฐานเวลา นำไปเปรียบเทียบกับฐานเวลาที่ได้ออกแบบไว้ ทำลักษณะนี้ ในทุกขั้นตอนเพื่อหาข้อผิดพลาด



รูปที่ 5-11 การจำลองในส่วนต่างๆ

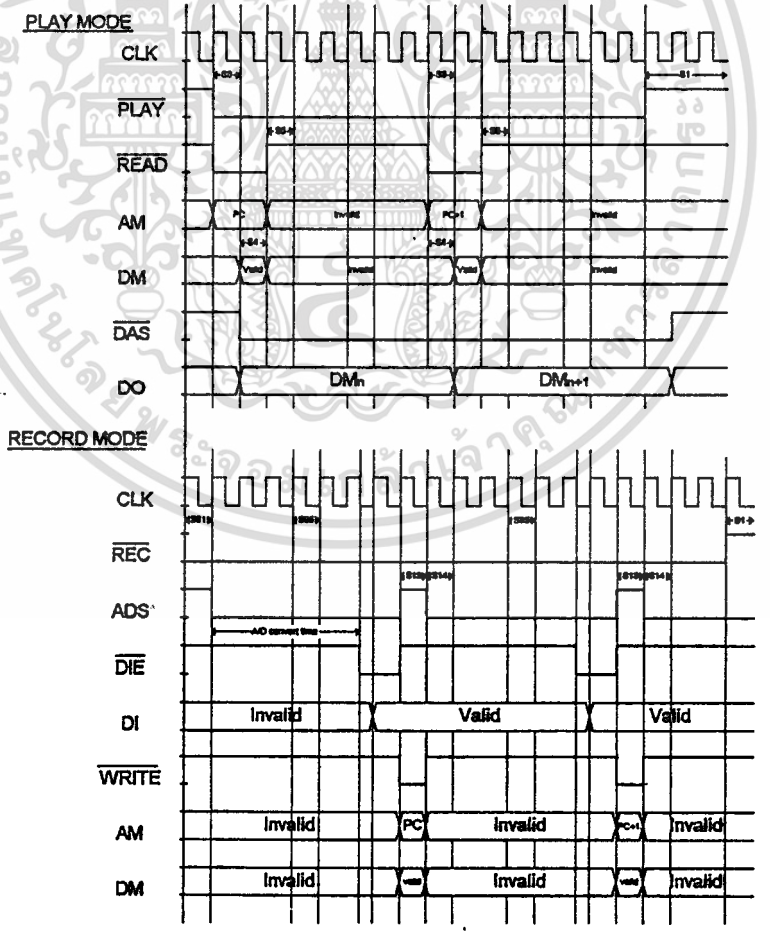
บทที่ 6
การแก้ไขดัดแปลง

6.1 การแก้ไขดัดแปลง

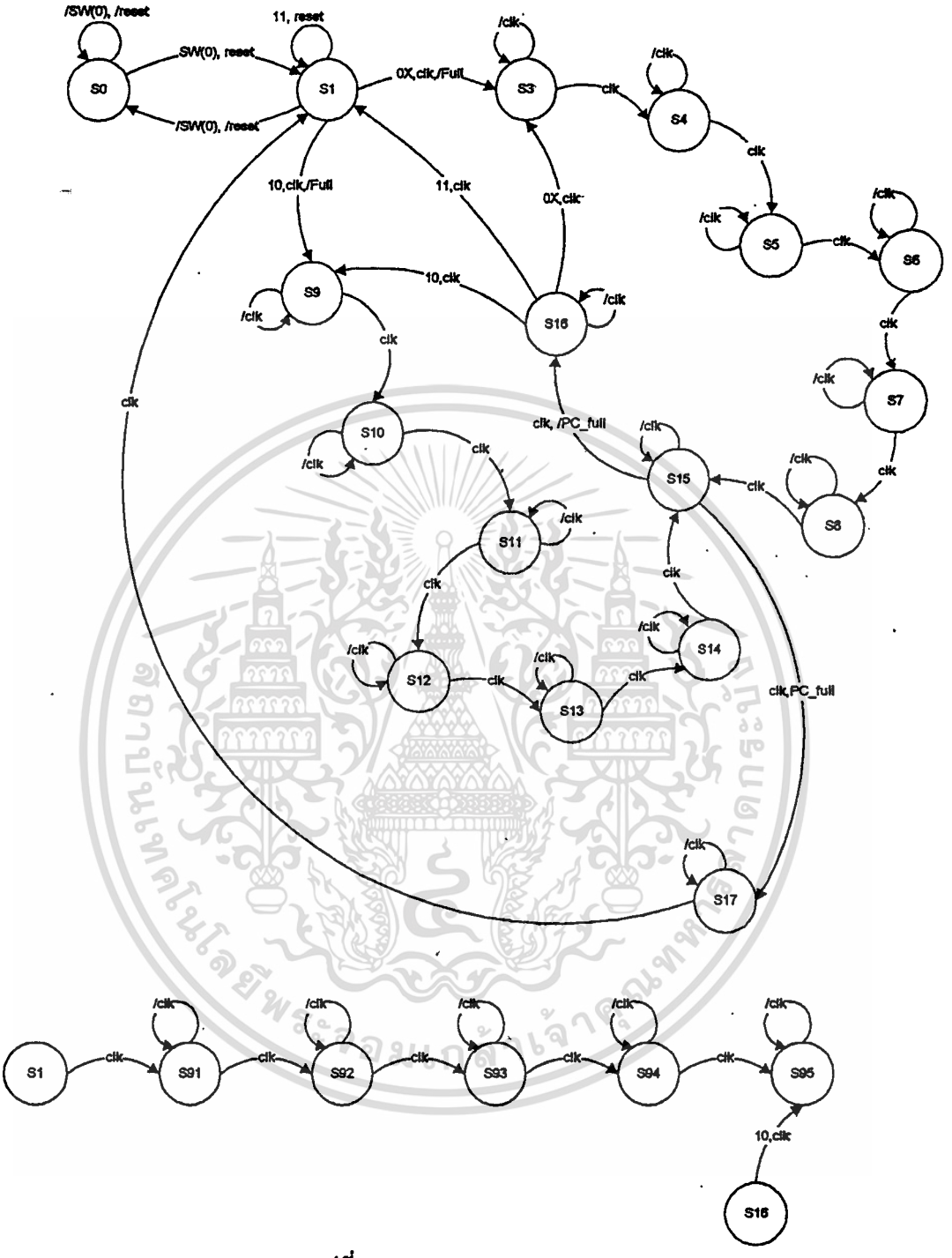
การแก้ไขดัดแปลงนั้น เราสามารถที่จะกระทำในส่วนใดๆของขั้นตอนการออกแบบก็ได้ ในบทนี้ผู้จัดทำจะนำเสนอการดัดแปลงตั้งแต่ขั้นตอนการออกแบบตัวบรรยายภาษา VHDL

6.2 การแก้ไขดัดแปลงการออกแบบตัวบรรยายภาษา VHDL

จากการออกแบบตัวบรรยายภาษา VHDL ของบทที่ 5 นั้น เราจะพบว่า ส่วนการทำงานของ PLAY Mode นั้นทำงานผิด ค่าของฐานเวลาที่ถูกต้อง เป็นตามรูปที่ 6-1 และส่วนการทำงานของ REWIND Mode นั้นทำงานคล้ายกับการ reset เราจะยุบ FSM state ที่ 2 ทิ้ง เพราะฉะนั้น FSM ที่ถูกต้องเป็นตามรูปที่ 6-2



รูปที่ 6-1 Timing Diagram



รูปที่ 6-2 Finite State Machine

Source Code ภาษา VHDL ที่ทำการแก้ไขแล้วมีดังนี้

```
-------------------------------
-- File     : tapesim.vhd
-- Designer : Mr.Somwang Piyapaneerut
-- Last update : 25/04/95
-------------------------------
-------------------------------
--    clk   = clock
--   /reset = system reset.
--   /DIE   = Data In Acknowledge.
--    SW    =   2    1    0
--            /PLAY  /REC  reset
--    DI    = Data In 8 bits (MSB7-LSB0)
--    DM    = Memory Data Bus 8 bits
--    DO    = Data Out 8 bits
--    AM    = Memory Address Bus 20 bits
--    LED   =   3    2    1    0
--            PLAY  REC ready  Full
--   /WRITE = Memory Write
--   /READ  = Memory Read
--   /DAS   = D/A Select or Output Ack.
--    ADS   = A/D Select or Start Converting
-------------------------------
Library synth;
Use synth.stdsynth.all;

-- Entity Declaration — Declare Ports
-------------------------------
entity TapeSim is
  port ( signal clk, reset, DIE : in vlbit;
         signal SW : in vlbit_vector(2 downto 0);
         signal DI, DMI : in vlbit_vector(7 downto 0);
```

```
        signal DO, DMO : out vlbit_vector(7 downto 0);
        signal AM : out vlbit_vector(19 downto 0);
        signal LED : out vlbit_vector(3 downto 0);
        signal WRITE, READ, DAS, ADS : out vlbit);
end TapeSim;
-------------------------------
-- Architectural behavioral description
architecture main_control of TapeSim is

   constant  one : vlbit_1d (18 downto 0) := B"000_0000_0000_0000_0001";
   constant PC_MAX : vlbit_1d (19 downto 0) := B"1111_1111_1111_1111_1111";

-------------------------------
--      Finite State Machine Aligment
   constant S1  : vlbit_1d(4 downto 0) := B"00001";
   constant S2  : vlbit_1d(4 downto 0) := B"00010";
   constant S3  : vlbit_1d(4 downto 0) := B"00011";
   constant S4  : vlbit_1d(4 downto 0) := B"00100";
   constant S5  : vlbit_1d(4 downto 0) := B"00101";
   constant S6  : vlbit_1d(4 downto 0) := B"00110";
   constant S7  : vlbit_1d(4 downto 0) := B"00111";
   constant S8  : vlbit_1d(4 downto 0) := B"01000";
   constant S91 : vlbit_1d(4 downto 0) := B"01001";
   constant S92 : vlbit_1d(4 downto 0) := B"10010";
   constant S93 : vlbit_1d(4 downto 0) := B"10011";
   constant S94 : vlbit_1d(4 downto 0) := B"10100";
   constant S95 : vlbit_1d(4 downto 0) := B"10101";
   constant S10 : vlbit_1d(4 downto 0) := B"01010";
   constant S11 : vlbit_1d(4 downto 0) := B"01011";
   constant S12 : vlbit_1d(4 downto 0) := B"01100";
   constant S13 : vlbit_1d(4 downto 0) := B"01101";
   constant S14 : vlbit_1d(4 downto 0) := B"01110";
```

```
    constant S15 : vlbit_1d(4 downto 0) := B"01111";
    constant S16 : vlbit_1d(4 downto 0) := B"10000";
    constant S17 : vlbit_1d(4 downto 0) := B"10001";

    constant  table  :  vlbit_2d (0 to 25,13 downto 0) := (
-- SW,Full,PC_full I State I Next_state
-------------------------------
-----        No Operate       -----
-------------------------------
 B"11XX_00001_00001",  -- S1 => S1
 B"XX1X_00001_00001",
-------------------------------
-----        Play State       -----
-------------------------------
 B"0X0X_00001_00011",  -- S1 => S3
 B"XXXX_00011_00100",  -- S3 => S4
 B"XXXX_00100_00101",  -- S4 => S5
 B"XXXX_00101_00110",  -- S5 => S6
 B"XXXX_00110_00111",  -- S6 => S7
 B"XXXX_00111_01000",  --S7 => S8
 B"XXXX_01000_01111",  -- S8 => S15 -- to N1
-------------------------------
-----        Record State       -----
-------------------------------
 B"100X_00001_01001",  -- S1 => S91
 B"XXXX_01001_10010",  -- S91=> S92
 B"XXXX_10010_10011",  -- S92=> S93
 B"XXXX_10011_10100",  -- S93=> S94
 B"XXXX_10100_10101",  -- S94=> S95
 B"XXXX_10101_01010",  -- S95=> S10
 B"XXXX_01010_01011",  --S10 => S11
 B"XXXX_01011_01100",  --S11 => S12
```

```
 B"XXXX_01100_01101",  --S12 => S13
 B"XXXX_01101_01110",  --S13 => S14
 B"XXXX_01110_01111",  --S14 => S15 -- to N1
-------------------------------
-----          N1          -----
-------------------------------
 B"XXX0_01111_10000",  --S15 => S16
 B"XXX1_01111_10001",  --S15 => S17

 B"0XXX_10000_00011",  --S16 => S3
 B"11XX_10000_00001",  --S16 => S1
 B"10XX_10000_10101",  --S16 => S95

 B"XXXX_10001_00001"); --S17 => S1
-------------------------------

 signal htbl : vlbit_1d (8 downto 0);
 signal state, next_s : vlbit_1d (4 downto 0);
 signal PC : vlbit_1d (19 downto 0);
 signal Full, PC_full, sreset : vlbit;
 signal Din : vlbit_1d (7 downto 0);
 signal Disp : vlbit_1d (2 downto 0);

begin

  pla_table (htbl, next_s, table);
  htbl <= SW(2 downto 1)&Full&PC_full&state;
  LED <= Disp&Full;
  sreset <= reset AND SW(0);

main: process
 begin
```

```
wait until (prising (clk) or (sreset = '0'));
if sreset = '0' then
   state <= S1; -- start state at S1
   PC_full <= '0';
   PC <= B"0000_0000_0000_0000_0000";
   Full <= '0';
   READ <= '1';
   WRITE <= '1';
   DAS <= '1';
   ADS <= '0';
   DO <= B"0000_0000";
   DIn <= B"0000_0000";
   Disp <= B"000";
else
   state <= next_s;
   if    state = S1   then Disp <= B"001";
   elsif  state = S3   then AM <= PC;
                       READ <= '0';
                       Disp(2) <= '1';
                       Disp(0) <= '0';
   elsif  state = S4   then DO <= DMI;
                       DAS <= '0';
   elsif  state = S5   then READ <= '1';
   elsif  state = S91  then Disp(1) <= '1';
                       Disp(0) <= '0';
                       ADS <= '1';
   elsif  state = S92  then ADS <= '0';
   elsif  state = S13  then AM <= PC;
                       DMO <= DIn;
                       WRITE <= '0';
                       ADS <= '1';
   elsif  state = S14  then WRITE <= '1';
```

```
                    ADS <= '0';
  elsif  state = S15  then if PC = PC_MAX then PC_full <= '1';
                    else PC_full <= '0';
                    end if;
  elsif  state = S16  then if PC(19) = '1' then
                     PC <= addum(PC(18 downto 0), one);
                     PC(19) <= '1';
                    else
                     PC <= addum(PC(18 downto 0), one);
                    end if;
  elsif  state = S17  then Full <= '1';
  end if;
  if DIE = '0' then DIn <= DI;
  end if;
 end if;
end process;
end main_control;
```

จากนั้นทำตามขั้นตอนต่อไปซ้ำแบบเดิม (ตั้งแต่ข้อที่ 5.3 - 5.6) ยกเว้นการทำ port mapping เพราะเราทำ socket ไว้แล้ว

บทที่ 7
สรุป และวิจารณ์

7.1 สรุป และวิจารณ์

จากการศึกษา และทดลองทำการออกระบบเชิงตัวเลขด้วย VHDL นั้นพบว่า ขั้นตอนในการออกแบบมีขั้นตอนใหญ่ๆ ดังนี้

7.1.1 แนวความคิดในการออกแบบ

7.1.2 การออกแบบตัวบรรยายภาษา VHDL

7.1.3 การออกแบบทางวงจรตรรก และการสังเคราะห์

7.1.4 การออกแบบทางกายภาพ และการสร้างเครื่องต้นแบบ

7.1.5 การจำลอง และการทดสอบ

ในแต่ละขั้นตอนจะให้ผลลัพธ์ที่จะเป็นข้อมูลให้กับขั้นตอนต่อไป เพราะฉะนั้นการออกแบบไม่สามารถกระทำข้ามขั้นตอนได้ ในขั้นตอนการออกแบบตัวบรรยายภาษา VHDL นั้น เป็นขั้นตอนที่มีความยืดหยุ่นสูงพอสมควร มีรูปแบบต่างๆกันในการออกแบบ ซึ่งสามารถแบ่งออกเป็นรูปแบบได้ ดังนี้

7.1.2.1 รูปแบบ Top-Down Design รูปแบบนี้มีลักษณะคล้ายกับการออกแบบ Software โดยการแบ่งตัวระบบออกเป็น Model ซึ่งจะทำการออกแบบระบบทั้งหมดก่อน โดยการมอง Model หนึ่งๆ เป็น backblock หรือ component ที่ไม่ทราบการทำงานภายใน แต่ทราบว่ามี I/O เป็นอย่างไร จากนั้นจึงทำการออกแบบในส่วนของ Model

7.1.2.2 รูปแบบ Bottom-Up Design รูปแบบนี้จะกลับกันกับแบบ Top-Down Design เพราะจะออกแบบที่ละ Model ก่อน จากนั้นนำ Model ทั้งหมดมาประกอบเข้าด้วยกัน ผลลัพธ์จากการออกแบบจะเป็นรูปแบบ Flatten Design

7.1.2.3 รูปแบบ Flatten Design รูปแบบนี้จะไม่มีการแบ่งแยกการทำงานออกเป็น Model จะมองระบบทั้งหมดแล้วทำการออกแบบ รูปแบบนี้เหมาะสำหรับการออกแบบระบบที่ไม่มีการทำงานซ้ำซ้อนกัน และเป็นระบบที่มีความซับซ้อนมานัก

7.1.2.4 ข้อดี และข้อเสีย ในการออกแบบในแต่ละรูปแบบ

7.1.2.4.1 Top-Down Design

- ข้อดี

- สามารถแก้ไขได้ง่าย

- ในกรณีที่ระบบมีการทำงานซ้ำซ้อนกันมากๆ การออกแบบในรูปแบบนี้จะมีความสะดวกรวดเร็ว

- สามารถแบ่งการออกแบบให้ผู้ออกแบบหลายคนช่วยกันทำ

- ไม่ต้องทำ port mapping ทุกครั้งที่มีการแก้ไข ทำเพียงครั้งเดียวเท่านั้น

- ข้อเสีย

- การ optimize ไม่ดีเท่าที่ควร

- เมื่อทำเสร็จแล้วมีจำนวนวงจรเกทมาก

7.1.2.4.2 Bottom-Up

- ข้อดี

- เป็นแบบที่แก้ข้อเสียแบบ Top-Down

- เหมือนแบบ Top-Down

- ข้อเสีย

- ในระบบที่มีขนาดเล็กจะเสียเวลามาก

7.1.2.4.3 Flatten Design

- ข้อดี

- ใช้เวลาในการออกแบบน้อย

- การ optimize ทำได้ดีกว่าแบบ Top-Down

- ข้อเสีย

- ไม่เหมาะที่จะนำไปใช้ในการออกแบบระบบที่มีขนาดใหญ่ และมีความซับซ้อนสูง

- การทำ port mapping นั้นจะต้องทำทุกครั้งที่มีการแก้ไขระบบ

ภาคผนวก

ขั้นตอนการใช้งาน

การใช้งานเครื่องจำลองการทำงานเครื่องบันทึกเทปนี้ แบ่งออกเป็น 3 mode คือ

1. Play mode

กดปุ่ม PLAY เป็นการเข้าสู่ Play mode และ LEDสีเหลืองเหนือปุ่ม PLAY จะติด modeนี้จะนำเอาเสียงพูดที่ถูกบันทึกไว้มาเล่นกลับ (Play Back)

2. Record mode

กดปุ่ม REC เป็นการเข้าสู่ Record mode และ LEDสีเหลืองเหนือปุ่ม REC จะติด modeนี้จะเก็บเอาเสียงพูดบันทึกลงสู่ memory

3. Rewind mode

กดปุ่ม RW เป็นการเข้าสู่ Rewind mode และ LEDสีเขียวเหนือปุ่ม RW จะแสดงความพร้อมในการใช้งาน modeนี้จะเป็นการ setup ระบบ ให้มาเริ่มต้นที่จะทำการบันทึกเสียงพูดหรือเล่นกลับเสียงพูด

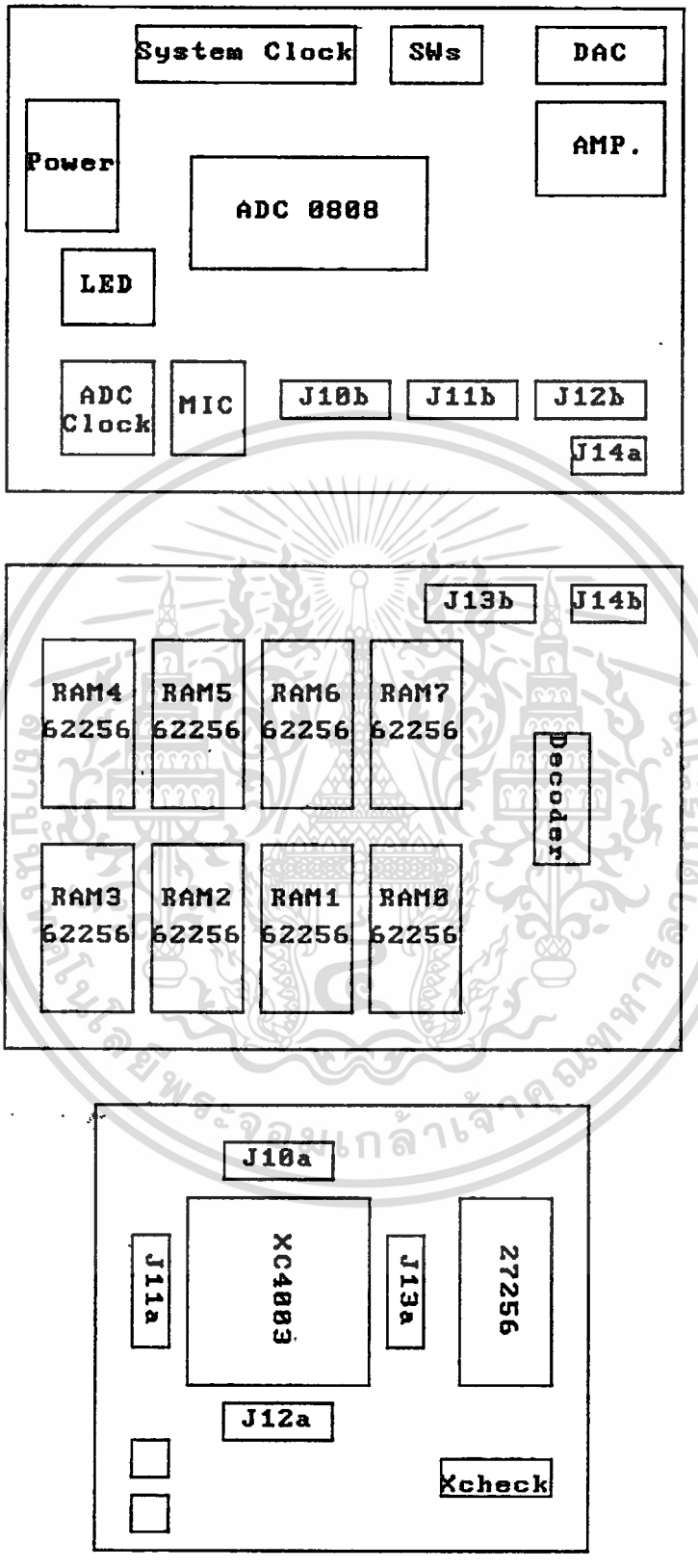
เริ่มต้นการใช้งาน

1. เมื่อ Power On ให้เรากดปุ่ม RW เพื่อทำการ setup ระบบ
2. ทำการล้าง memory ด้วยการกดปุ่ม REC ตามช่วงเวลาที่ขึ้นอยู่กับขนาดของ memory ที่ติดตั้งไว้ คำนวนได้ดังนี้

$$\text{เวลาทั้งหมด} = (1.25*10^{-5})* \text{ขนาดของ RAM}$$

ถ้าติดตั้ง RAM ไว้ 1 MB. ให้สังเกตุจาก LED สีแดง

3. กดปุ่ม RW อีกครั้ง
4. เริ่มทำการบันทึกเสียงได้



National Semiconductor

ADC0808, ADC0809 8-Bit μP Compatible A/D Converters With 8-Channel Multiplexer

General Description

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals.

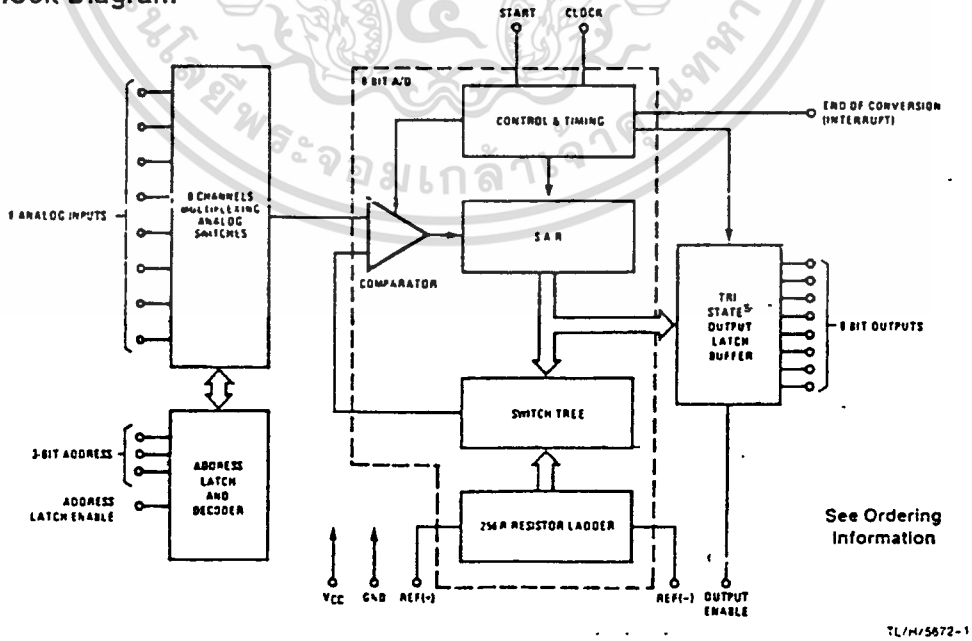
The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE® outputs.

The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications. For 16-channel multiplexer with common output (sample/hold port) see ADC0816 data sheet. (See AN-247 for more information.)

Features

- Resolution—8-bits
- Total unadjusted error— ± ½ LSB and ± 1 LSB
- No missing codes
- Conversion time—100 μS
- Single supply—5 V_{DC}
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- 8-channel multiplexer with latched control logic
- Easy interface to all microprocessors, or operates "stand alone"
- Outputs meet T^2L voltage level specifications
- 0V to 5V analog input voltage range with single 5V supply
- No zero or full-scale adjust required
- Standard hermetic or molded 28-pin DIP package
- Temperature range −40°C to +85°C or −55°C to +125°C
- Low power consumption—15 mW
- Latched TRI-STATE output

Block Diagram



Absolute Maximum Ratings (Notes 1 & 2)

Supply Voltage (V_{CC}) (Note 3) 6.5V
Voltage at Any Pin −0.3V to (V_{CC} + 0.3V)
Except Control Inputs
Voltage at Control Inputs −0.3V to +15V
(START, OE, CLOCK, ALE, ADD A, ADD B, ADD C)
Storage Temperature Range −65°C to +150°C
Package Dissipation at T_A = 25°C 875 mW
Lead Temperature (Soldering, 10 seconds) 300°C

Operating Conditions (Notes 1 & 2)

Temperature Range (Note 1) $T_{MIN} \le T_A \le T_{MAX}$
ADC0808CJ $-55°C \le T_A \le +125°C$
ADC0808CCJ, ADC0808CCN,
ADC0809CCN $-40°C \le T_A \le +85°C$
Range of V_{CC} (Note 1) 4.5 V_{DC} to 6.0 V_{DC}

Electrical Characteristics

Converter Specifications: V_{CC} = 5 V_{DC} = V_{REF+}, $V_{REF(-)}$ = GND, $T_{MIN} \le T_A \le T_{MAX}$ and f_{CLK} = 640 kHz unless otherwise stated.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
	ADC0808 Total Unadjusted Error (Note 5)	25°C T_{MIN} to T_{MAX}			±½ ±¾	LSB LSB
	ADC0809 Total Unadjusted Error (Note 5)	0°C to 70°C T_{MIN} to T_{MAX}			±1 ±1¼	LSB LSB
	Input Resistance	From Ref(+) to Ref(−)	1.0	2.5		kΩ
	Analog Input Voltage Range	(Note 4) V(+) or V(−)	GND−0.10		V_{CC}+0.10	V_{DC}
$V_{REF(+)}$	Voltage, Top of Ladder	Measured at Ref(+)		V_{CC}	V_{CC}+0.1	V
$\frac{V_{REF(+)}+V_{REF(-)}}{2}$	Voltage, Center of Ladder		V_{CC}/2-0.1	V_{CC}/2	V_{CC}/2+0.1	V
$V_{REF(-)}$	Voltage, Bottom of Ladder	Measured at Ref(−)	−0.1	0		V
I_{IN}	Comparator Input Current	f_c = 640 kHz, (Note 6)	−2	±0.5	2	μA

Electrical Characteristics

Digital Levels and DC Specifications: ADC0808CJ 4.5V ≤ V_{CC} ≤ 5.5V, −55°C ≤ T_A ≤ +125°C unless otherwise noted ADC0808CCJ, ADC0808CCN, and ADC0809CCN 4.75 ≤ V_{CC} ≤ 5.25V, −40°C ≤ T_A ≤ +85°C unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
ANALOG MULTIPLEXER						
$I_{OFF(+)}$	OFF Channel Leakage Current	V_{CC} = 5V, V_{IN} = 5V, T_A = 25°C T_{MIN} to T_{MAX}		10	200 1.0	nA μA
$I_{OFF(-)}$	OFF Channel Leakage Current	V_{CC} = 5V, V_{IN} = 0, T_A = 25°C T_{MIN} to T_{MAX}	−200 −1.0	−10		nA μA
CONTROL INPUTS						
$V_{IN(1)}$	Logical "1" Input Voltage		V_{CC}−1.5			V
$V_{IN(0)}$	Logical "0" Input Voltage				1.5	V
$I_{IN(1)}$	Logical "1" Input Current (The Control Inputs)	V_{IN} = 15V			1.0	μA
$I_{IN(0)}$	Logical "0" Input Current (The Control Inputs)	V_{IN} = 0	−1.0			μA
I_{CC}	Supply Current	f_{CLK} = 640 kHz		0.3	3.0	mA

Electrical Characteristics (Continued)

Digital Levels and DC Specifications: ADC0808CJ $4.5V \le V_{CC} \le 5.5V$, $-55°C \le T_A \le +125°C$ unless otherwise noted
ADC0808CCJ, ADC0808CCN, and ADC0809CCN $4.75 \le V_{CC} \le 5.25V$, $-40°C \le T_A \le +85°C$ unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
DATA OUTPUTS AND EOC (INTERRUPT)						
$V_{OUT(1)}$	Logical "1" Output Voltage	$I_O = -360\ \mu A$	$V_{CC} - 0.4$			V
$V_{OUT(0)}$	Logical "0" Output Voltage	$I_O = 1.6$ mA			0.45	V
$V_{OUT(0)}$	Logical "0" Output Voltage EOC	$I_O = 1.2$ mA			0.45	V
I_{OUT}	TRI-STATE Output Current	$V_O = 5V$ $V_O = 0$	 −3		3	μA μA

Electrical Characteristics

Timing Specifications $V_{CC} = V_{REF(+)} = 5V$, $V_{REF(-)} = GND$, $t_r = t_f = 20$ ns and $T_A = 25°C$ unless otherwise noted.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{WS}	Minimum Start Pulse Width	*(Figure 5)*		100	200	ns
t_{WALE}	Minimum ALE Pulse Width	*(Figure 5)*		100	200	ns
t_s	Minimum Address Set-Up Time	*(Figure 5)*		25	50	ns
t_H	Minimum Address Hold Time	*(Figure 5)*		25	50	ns
t_D	Analog MUX Delay Time From ALE	$R_S = 0\Omega$ *(Figure 5)*		1	2.5	μS
t_{H1}, t_{H0}	OE Control to Q Logic State	$C_L = 50$ pF, $R_L = 10$k *(Figure 8)*		125	250	ns
t_{1H}, t_{0H}	OE Control to Hi-Z	$C_L = 10$ pF, $R_L = 10$k *(Figure 8)*		125	250	ns
t_C	Conversion Time	$f_C = 640$ kHz, *(Figure 5)* (Note 7)	90	100	116	μS
f_C	Clock Frequency		10	640	1280	kHz
t_{EOC}	EOC Delay Time	*(Figure 5)*	0		8 + 2 μS	Clock Periods
C_{IN}	Input Capacitance	At Control Inputs		10	15	pF
C_{OUT}	TRI-STATE Output Capacitance	At TRI-STATE Outputs, (Note 12)		10	15	pF

Note 1: Absolute maximum ratings are those values beyond which the life of the device may be impaired.

Note 2: All voltages are measured with respect to GND, unless otherwise specified.

Note 3: A zener diode exists, internally, from V_{CC} to GND and has a typical breakdown voltage of 7 V_{DC}.

Note 4: Two on-chip diodes are tied to each analog input which will forward conduct for analog input voltages one diode drop below ground or one diode drop greater than the V_{CC} supply. The spec allows 100 mV forward bias of either diode. This means that as long as the analog V_{IN} does not exceed the supply voltage by more than 100 mV, the output code will be correct. To achieve an absolute $0V_{DC}$ to $5V_{DC}$ input voltage range will therefore require a minimum supply voltage of 4.900 V_{DC} over temperature variations, initial tolerance and loading.

Note 5: Total unadjusted error includes offset, full-scale, linearity, and multiplexer errors. See *Figure 3*. None of these A/Ds requires a zero or full-scale adjust. However, if an all zero code is desired for an analog input other than 0.0V, or if a narrow full-scale span exists (for example: 0.5V to 4.5V full-scale) the reference voltages can be adjusted to achieve this. See *Figure 13*.

Note 6: Comparator input current is a bias current into or out of the chopper stabilized comparator. The bias current varies directly with clock frequency and has little temperature dependence *(Figure 6)*. See paragraph 4.0.

Note 7: The outputs of the data register are updated one clock cycle before the rising edge of EOC.

Functional Description

Multiplexer. The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. Table I shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

TABLE I

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

CONVERTER CHARACTERISTICS

The Converter

The heart of this single chip data acquisition system is its 8-bit analog-to-digital converter. The converter is designed to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach *(Figure 1)* was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256R network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in *Figure 1* are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached + ½ LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n-iterations are required for an n-bit converter. *Figure 2* shows a typical example of a 3-bit converter. In the ADC0808, ADC0809, the approximation technique is extended to 8 bits using the 256R network.

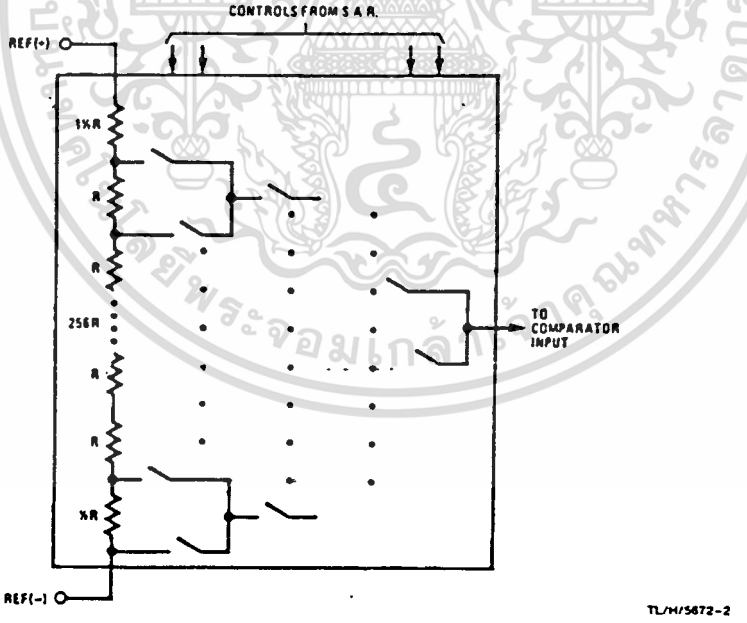


FIGURE 1. Resistor Ladder and Switch Tree

Functional Description (Continued)

The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion.

The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilized comparator provides the most effective method of satisfying all the converter requirements.

The chopper-stabilized comparator converts the DC input signal into an AC signal. This signal is then fed through a high gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier since the drift is a DC component which is not passed by the AC amplifier. This makes the entire A/D converter extremely insensitive to temperature, long term drift and input offset errors.

Figure 4 shows a typical error curve for the ADC0808 as measured using the procedures outlined in AN-179.

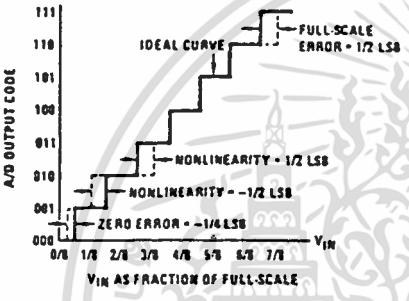


FIGURE 2. 3-Bit A/D Transfer Curve

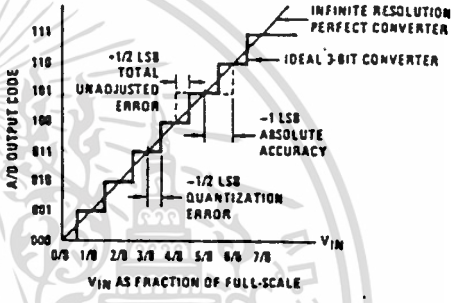


FIGURE 3. 3-Bit A/D Absolute Accuracy Curve

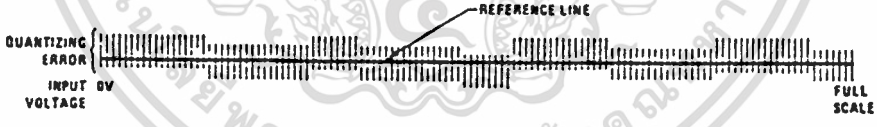
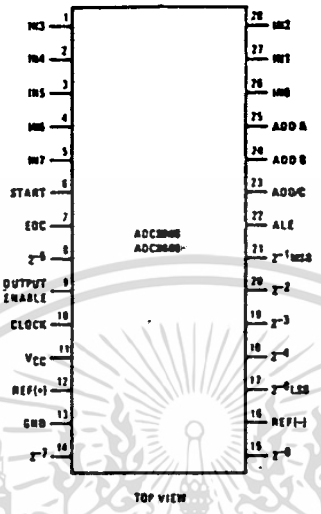


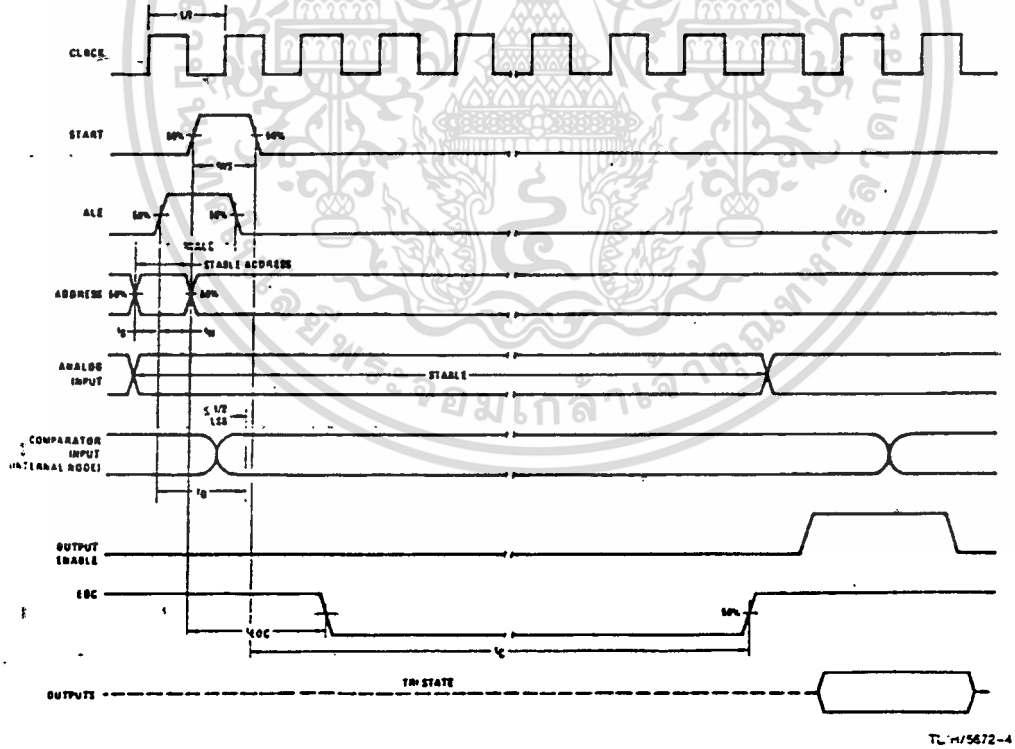
FIGURE 4. Typical Error Curve

Connection Diagram

Dual-In-Line Package



Timing Diagram



TL/H/5672–4

FIGURE 5

Typical Performance Characteristics

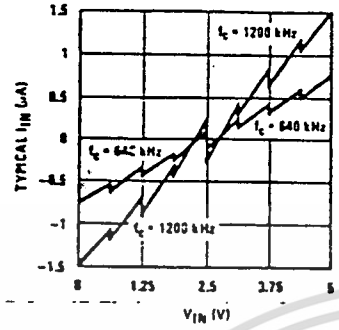
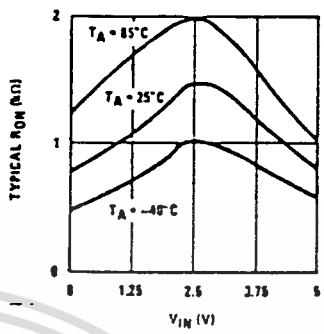


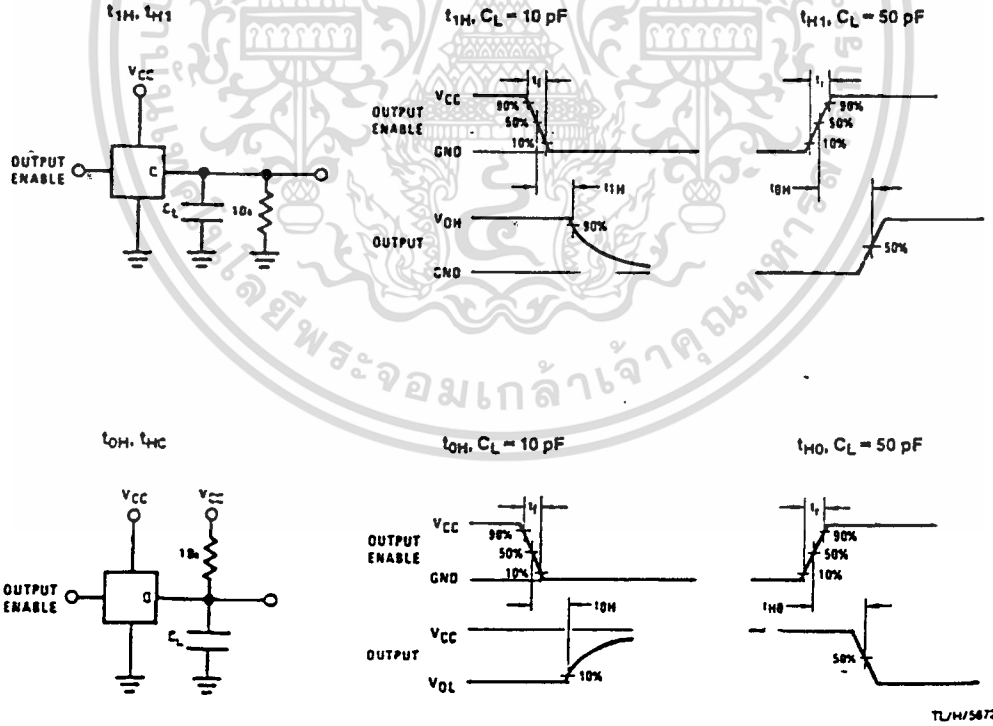
FIGURE 6. Comparator I_{IN} vs V_{IN} ($V_{CC} = V_{REF} = 5V$)



TL/H/5672-5

FIGURE 7. Multiplexer R_{ON} vs V_{IN} ($V_{CC} = V_{REF} = 5V$)

TRI-STATE Test Circuits and Timing Diagrams



TL/H/5672-6

FIGURE 8

Applications Information

OPERATION

1.0 RATIOMETRIC CONVERSION

The ADC0808, ADC0809 is designed as a complete Data Acquisition System (DAS) for ratiometric conversion systems. In ratiometric systems, the physical variable being measured is expressed as a percentage of full-scale which is not necessarily related to an absolute standard. The voltage input to the ADC0808 is expressed by the equation

$$\frac{V_{IN}}{V_{fs}-V_Z}=\frac{D_X}{D_{MAX}-D_{MIN}} \qquad (1)$$

V_{IN} = Input voltage into the ADC0808

V_{fs} = Full-scale voltage

V_Z = Zero voltage

D_X = Data point being measured

D_{MAX} = Maximum data limit

D_{MIN} = Minimum data limit

A good example of a ratiometric transducer is a potentiometer used as a position sensor. The position of the wiper is directly proportional to the output voltage which is a ratio of the full-scale voltage across it. Since the data is represented as a proportion of full-scale, reference requirements are greatly reduced, eliminating a large source of error and cost for many applications. A major advantage of the ADC0808, ADC0809 is that the input voltage range is equal to the supply range so the transducers can be connected directly across the supply and their outputs connected directly into the multiplexer inputs, *(Figure 9)*.

Ratiometric transducers such as potentiometers, strain gauges, thermistor bridges, pressure transducers, etc., are suitable for measuring proportional relationships; however, many types of measurements must be referred to an absolute standard such as voltage or current. This means a system reference must be used which relates the full-scale voltage to the standard volt. For example, if $V_{CC} = V_{REF} = 5.12V$, then the full-scale range is divided into 256 standard steps. The smallest standard step is 1 LSB which is then 20 mV.

2.0 RESISTOR LADDER LIMITATIONS

The voltages from the resistor ladder are compared to the selected into 8 times in a conversion. These voltages are coupled to the comparator via an analog switch tree which is referenced to the supply. The voltages at the top, center and bottom of the ladder must be controlled to maintain proper operation.

The top of the ladder, Ref(+), should not be more positive than the supply, and the bottom of the ladder, Ref(−), should not be more negative than ground. The center of the ladder voltage must also be near the center of the supply because the analog switch tree changes from N-channel switches to P-channel switches. These limitations are automatically satisfied in ratiometric systems and can be easily met in ground referenced systems.

Figure 10 shows a ground referenced system with a separate supply and reference. In this system, the supply must be trimmed to match the reference voltage. For instance, if a 5.12V is used, the supply should be adjusted to the same voltage within 0.1V.

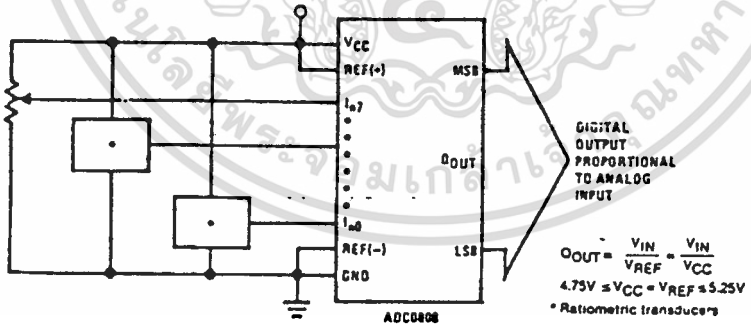


FIGURE 9. Ratiometric Conversion System

Applications Information (Continued)

The ADC0808 needs less than a milliamp of supply current so developing the supply from the reference is readily accomplished. In *Figure 11* a ground referenced system is shown which generates the supply from the reference. The buffer shown can be an op amp of sufficient drive to supply the milliamp of supply current and the desired bus drive, or if a capacitive bus is driven by the outputs a large capacitor will supply the transient supply current as seen in *Figure 12*. The LM301 is overcompensated to insure stability when loaded by the 10 μF output capacitor.

The top and bottom ladder voltages cannot exceed V_{CC} and ground, respectively, but they can be symmetrically less than V_{CC} and greater than ground. The center of the ladder voltage should always be near the center of the supply. The sensitivity of the converter can be increased, (i.e., size of the LSB steps decreased) by using a symmetrical reference system. In *Figure 13*, a 2.5V reference is symmetrically centered about $V_{CC}/2$ since the same current flows in identical resistors. This system with a 2.5V reference allows the LSB bit to be half the size of a 5V reference system.

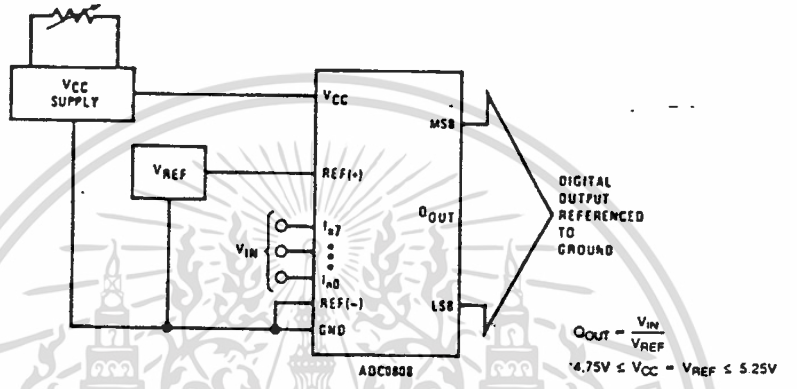


FIGURE 10. Ground Referenced Conversion System Using Trimmed Supply

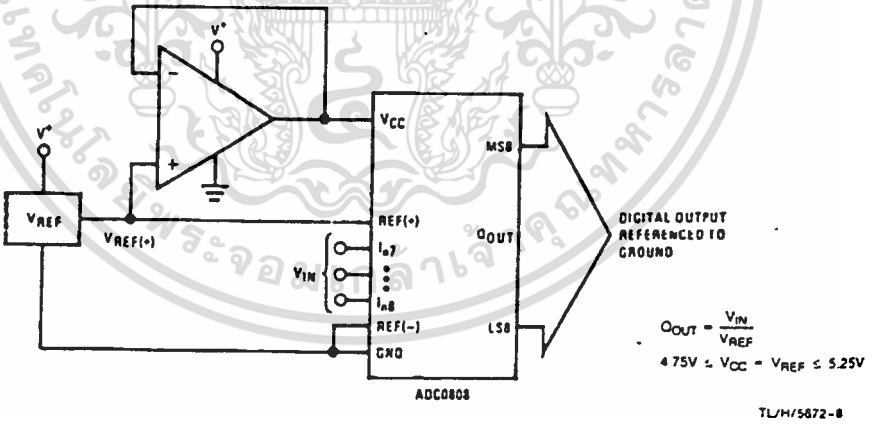


FIGURE 11: Ground Referenced Conversion System with Reference Generating V_{CC} Supply

Applications Information (Continued)

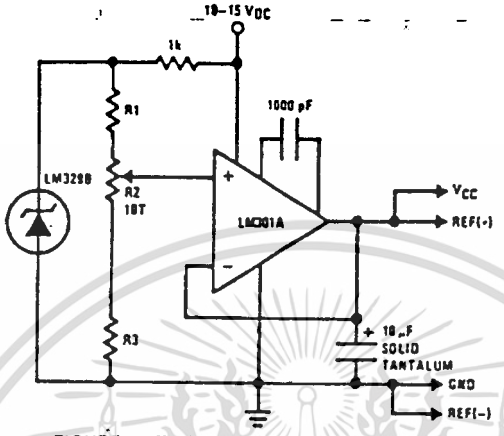
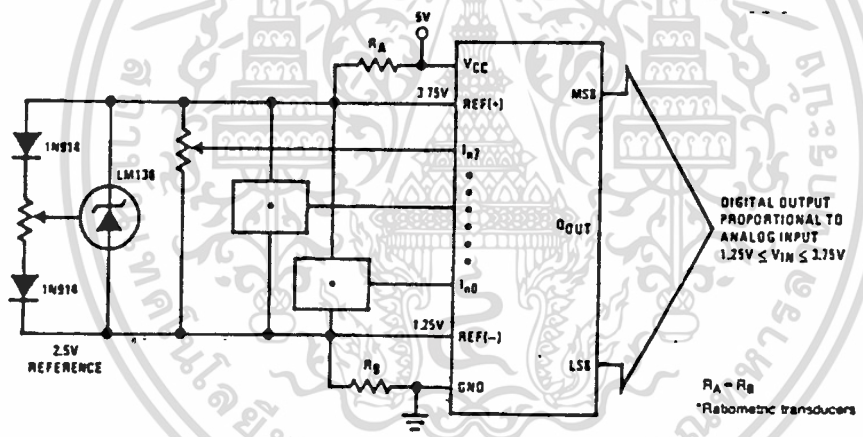


FIGURE 12. Typical Reference and Supply Circuit



TL/H/5672–8

FIGURE 13. Symmetrically Centered Reference

3.0 CONVERTER EQUATIONS

The transition between adjacent codes N and N + 1 is given by:

$$V_{IN} = \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} + \frac{1}{512} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (2)$$

The center of an output code N is given by:

$$V_{IN} = \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (3)$$

The output code N for an arbitrary input are the integers within the range:

$$N = \frac{V_{IN} - V_{REF(-)}}{V_{REF(+)} - V_{REF(-)}} \times 256 \pm \text{Absolute Accuracy} \quad (4)$$

where: V_{IN} = Voltage at comparator input

$V_{REF(+)}$ = Voltage at Ref(+)

$V_{REF(-)}$ = Voltage at Ref(−)

V_{TUE} = Total unadjusted error voltage (typically $V_{REF(+)} \div 512$)

4.0 ANALOG COMPARATOR INPUTS

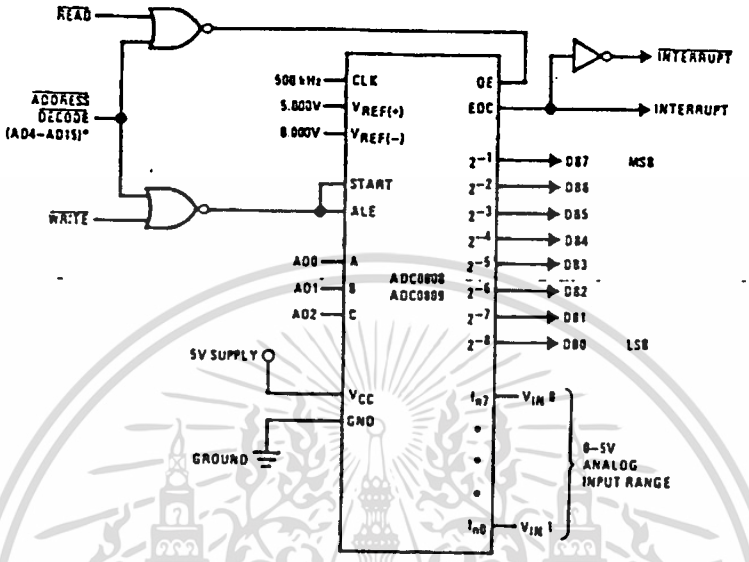
The dynamic comparator input current is caused by the periodic switching of on-chip stray capacitances. These are connected alternately to the output of the resistor ladder/switch tree network and to the comparator input as part of the operation of the chopper stabilized comparator.

The average value of the comparator input current varies directly with clock frequency and with V_{IN} as shown in *Figure 6*.

If no filter capacitors are used at the analog inputs and the signal source impedances are low, the comparator input current should not introduce converter errors, as the transient created by the capacitance discharge will die out before the comparator output is strobed.

If input filter capacitors are desired for noise reduction and signal conditioning they will tend to average out the dynamic comparator input current. It will then take on the characteristics of a DC bias current whose effect can be predicted conventionally.

Typical Application



TL/H/5672–16

*Address latches needed for 8085 and SC/MP interfacing the ADC0808 to a microprocessor

MICROPROCESSOR INTERFACE TABLE

PROCESSOR	READ	WRITE	INTERRUPT (COMMENT)
8080	MEMR	MEMW	INTR (Thru RST Circuit)
8085	RD	WR	INTR (Thru RST Circuit)
Z-80	RD	WR	INT (Thru RST Circuit, Mode 0)
SC/MP	NRDS	NWDS	SA (Thru Sense A)
6800	VMA•φ2•R/W	VMA•φ•R/W	IRQA or IRQB (Thru PIA)

Ordering Information

TEMPERATURE RANGE		−40°C to +85°C		−55°C to +125°C
Error	± ½ Bit Unadjusted	ADC0808CCN	ADC0808CCJ	ADC0808CJ
	± 1 Bit Unadjusted	ADC0809CCN		
Package Outline		N28A Molded DIP	J28A Hermetic DIP	J28A Hermetic DIP

กิตติกรรมประกาศ

ผู้จัดทำขอขอบคุณ อาจารย์ บรรจง ปิยธำรง เป็นอย่างยิ่งที่ท่านได้ให้ความรู้พื้นฐานการออกแบบโดยใช้ CAD/CAE ให้แนวทางในการทำงาน จัดหาเครื่องมือต่างๆทั้งทางด้าน Hardware และ Software ตลอดจนให้คำแนะนำและช่วยแก้ปัญหาต่างๆที่เกิดขึ้นตลอดการทำโครงงาน

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 ต่างๆ ตลอดจนเป็นธุระในการจัดหา License ของ Software ที่จำเป็นในการทำโครงงาน

สุดท้ายที่จะลืมเสียไม่ได้คือ พ่อแม่ ที่ได้ให้กำเนิดเลี้ยงดูอบรมให้การศึกษามาเป็นอย่างดี,เพื่อนๆที่ช่วยเป็นที่ปรึกษาและให้กำลังใจในการทำงานตลอดมา

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 Editons,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