ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

.....

ปริญญานิพนธ์ การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs
IMPLEMENTATION CENTRAL PROCESSING UNIT USING VHDL AND FPGAs DEVICE

นักศึกษา

1. นางสาวเลขา ธรรมนิเวศสกุล รหัสประจำตัว 40031329
2. นายศราวุฒิ เมธาธนบดี รหัสประจำตัว 40031330
3. นายเสริมศักดิ์ ศรีกระจ่าง รหัสประจำตัว 40031339

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ผู้ควบคุมปริญญานิพนธ์

1. อาจารย์กิติพงศ์ มะโน
2. อาจารย์อำพล ทองระอา

คณะกรรมการสอบปริญญานิพนธ์	ลายมือชื่อ
1.อาจารย์วรวิทย์ สมหา	
2.อาจารย์อำพล ทองระอา	
3.อาจารย์พงษ์เกียรติ เชษฐพิทักษ์สกุล	
4. อาจารย์สุระชัย พิมพ์สาลี	
5.อาจารย์อมรชัย ชัยชนะ	

วันเดือนปีที่สอบ วันที่ 22 พฤศจิกายน 2541 เวลา 19.30 น. ถึง 20.20 น.

สถานที่สอบ ห้อง ค.310 คณะครุศาสตร์อุตสาหกรรม

ภาควิชารับรองแล้ว

ลงนาม.....

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 12 เดือน ก.ค. พ.ศ. 2542

ปริญญานิพนธ์

การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

IMPLEMENTATION CENTRAL PROCESSING UNIT USING VHDL AND FPGAs DEVICE

นางสาวเลขา ธรรมนิเวศสกุล

นายศราวุฒิ เมธาธนบดี

นายเสริมศักดิ์ ศรีกระจ่าง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์

เรื่อง การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs
IMPLEMENTATION CENTRAL PROCESSING UNIT USING VHDL AND FPGAs DEVICE

ผู้จัดทำ

1. นางสาวเลขา ธรรมนิเวศสกุล
2. นายศราวุฒิ เมธาธนบดี
3. นายเสริมศักดิ์ ศรีกระจ่าง

อาจารย์ที่ปรึกษา

ลงนาม.....

(อาจารย์กิติพงศ์ มะโน)

ลงนาม.....

(อาจารย์อำพล ทองระอา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

ลงนาม.....

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

ปริญญานิพนธ์

เรื่อง การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs
IMPLEMENTATION CENTRAL PROCESSING UNIT USING VHDL AND FPGAs DEVICE

วัตถุประสงค์

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

ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถใช้โปรแกรมภาษา VHDL และอุปกรณ์ FPGAs
2. ได้ความรู้เกี่ยวกับโปรแกรมการสร้างวงจร และเลียนแบบการทำงานของหน่วยประมวลผลกลางที่ใช้ในการโปรแกรมอุปกรณ์ FPGAs
3. ได้รับความรู้ในการออกแบบวงจรเชิงเลขเฉพาะงาน
4. สร้างวงจรเชิงเลขเฉพาะงานโดยใช้อุปกรณ์ FPGAs ได้
5. สามารถนำวงจรเชิงเลขเฉพาะงานมาใช้ทดสอบและใช้งานได้

การสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

นางสาวเลขา ธรรมนิเวศสกุล
นายศราวุฒิ เมธาธนบดี
นายเสริมศักดิ์ ศรีกระจ่าง

อาจารย์ที่ปรึกษา

อาจารย์กิตติพงศ์ มะโน
อาจารย์อำพล ทองระอา

ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอการสร้างหน่วยประมวลผลกลางโดยใช้ภาษา VHDL และอุปกรณ์ FPGAs ซึ่งสร้างในลักษณะบนลงล่าง และได้ออกแบบระบบคอมพิวเตอร์พื้นฐานโดยใช้ CPU ที่สร้างขึ้นเป็นหน่วยประมวลผลกลาง เพื่อใช้เป็นบอร์ดสำหรับทดสอบการทำงานของ CPU และเพื่อเรียนรู้การประยุกต์ใช้งาน และทำการดาวน์โหลดลงอุปกรณ์ FPGAs เพื่อนำไปประกอบลงบอร์ดใช้งาน โดยหน่วยประมวลผลกลางนี้ใช้สัญญาณนาฬิกาขนาด 2 เมกะเฮิรตซ์ ประกอบด้วยบัสข้อมูลขนาด 4 บิต บัสตำแหน่งขนาด 16 บิต และสามารถกระทำคำสั่งได้ 16 คำสั่ง บอร์ด CPU ED1 นี้สามารถใช้ทดลองกับโปรแกรมภาษา VHDL , การโปรแกรม PAL และการทำงานที่เกี่ยวข้องกับไมโครโปรเซสเซอร์ ผลการสร้างและการทดลองได้ผลถูกต้องตามที่ได้ออกแบบไว้

Implementation Contral Processing Unit Using VHDL and FPGAs Device

MISS LAEKHA THAMNIVESSAKUL
MR. SARAWUT METHATANABADEE
MR. SERMSAK SRIKRAJANG

ADVISORS

MR. KITIPONG MANO
MR. AMPHON THONGRA-AR

1998

ABSTRACT

In this thesis , it shows the central processing unit (CPU). Designed using the principle of top-down design in VHDL language and implement by FPGAs device. The specification of CPU is 4 bit data bus , 16 bit address bus , 16 instruction and 2 MHz clock system. This project designed a single board processing that consist of input unit , output unit , memory unit and constructed CPU. In CPU ED1 board , user can learning about VHDL , FPGAs PAL and microprocessor system according to the lab document

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์นี้สามารถสำเร็จลุล่วงไปได้ด้วยดี จากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน นอกจากนี้ยังได้รับความกรุณาจากอาจารย์กิติพงค์ มะโน อาจารย์ที่ปรึกษาประจำโครงงาน ตลอดจนอาจารย์ทุกท่านในภาควิชาที่ให้ความอนุเคราะห์ในความสะดวกทั้งด้านความรู้ สถานที่ เครื่องมือ และอุปกรณ์ต่าง ๆ และขอขอบคุณบุพการีที่ให้การสนับสนุนในการศึกษา และเพื่อน ๆ ตลอดจนผู้เกี่ยวข้องทุกท่านที่ได้ให้ความช่วยเหลือ แนะนำ จนทำให้โครงงานสำเร็จลุล่วงไปด้วยดี

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

อุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ เป็นอุปกรณ์ที่ได้รับการพัฒนามาเมื่อประมาณ 20 ปีก่อน ในขณะนั้นการพัฒนาอุปกรณ์อิเล็กทรอนิกส์เป็นไปอย่างล่าช้า เพราะเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์ได้รับการพัฒนาไปอย่างรวดเร็วดังที่จะเห็นได้ว่ามีวงจรดิจิตอลอิเล็กทรอนิกส์หลายวงจร ที่แต่เดิมถูกสร้างขึ้นมาจากชิ้นส่วนอุปกรณ์อิเล็กทรอนิกส์จำนวนหลายชิ้น ถูกนำประกอบกันอยู่บนแผงวงจรไฟฟ้า (Printed Circuit Board หรือ PCB) ที่มีขนาดใหญ่ แต่ในปัจจุบันสามารถที่จะใช้เทคโนโลยีการออกแบบและผลิตวงจรรวมขนาดใหญ่มาก (Very Large Scale Integration หรือ VLSI) รวมอุปกรณ์ต่างๆ เหล่านั้นให้อยู่บนชิ้นอุปกรณ์สารกึ่งตัวนำ ที่มีขนาดประมาณ 1-2 ตร.ซม. ได้ ซึ่งเป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้น ตลอดจนความน่าเชื่อถือ และความคงทนต่อสภาพแวดล้อมสูง การที่จะเปลี่ยนอุปกรณ์ใหม่เป็นสิ่งที่ต้องใช้งบประมาณมาก และก็จะประสบกับปัญหาเช่นเดิมคือ อุปกรณ์ใหม่ได้รับการพัฒนามานานแล้วเช่นกัน เพราะในขณะนั้นขั้นตอนของการออกแบบ การผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ต้องใช้วิศวกร และเวลาสำหรับดำเนินการมาก ฉะนั้นวิธีการที่จะช่วยพัฒนาวงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำไปผลิตได้เร็วขึ้น และโครงการดังกล่าวมีชื่อว่า Very High Speed Integrated Circuits หรือ VHSIC ใช้สำหรับบรรยายพฤติกรรมของวงจรหรือ Hardware ของระบบ เป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร (Simulation Language) ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง (High Level Language) เรียกว่า Hardware Description Language หรือ HDL ดังนั้นภาษามาตรฐานนี้จึงมีชื่อว่า VHSIC-HDL หรือ VHDL

ซึ่งทางคณะผู้จัดทำได้เลือกเอาภาษา VHDL (Very High Speed Integrated Circuit Hardware Descriprion Language) ทำการออกแบบเชิงเลขและทำการสังเคราะห์จนได้วงจรระดับเกทแล้ว จึงโปรแกรมลงบนอุปกรณ์ประเภทที่สามารถโปรแกรมได้ (Programmable Logic Device) และได้เลือกใช้อุปกรณ์ FPGAs (Field Programmable Gate Array)

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาการเขียนโปรแกรมภาษา VHDL และอุปกรณ์ FPGAs
2. เพื่อศึกษาโปรแกรมสำหรับออกแบบสร้าง และเลียนแบบการทำงานของหน่วยประมวลผลกลางที่ใช้ในการโปรแกรมและอุปกรณ์ FPGAs
3. เพื่อออกแบบหน่วยประมวลผลโดยใช้ภาษา VHDL
4. เพื่อสร้างหน่วยประมวลผลโดยใช้อุปกรณ์ FPGAs
5. เพื่อนำหน่วยประมวลผลกลางมาทดสอบ
6. เพื่อนำหน่วยประมวลผลกลางมาประยุกต์ใช้งาน

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถใช้โปรแกรมภาษา VHDL และอุปกรณ์ FPGAs
2. ได้ความรู้เกี่ยวกับโปรแกรมการสร้างวงจร และเลียนแบบการทำงานของหน่วยประมวลผลกลางที่ใช้ในการโปรแกรมอุปกรณ์ FPGAs
3. ได้รับความรู้ในการออกแบบวงจรเชิงเลขเฉพาะงาน
4. สร้างวงจรเชิงเลขเฉพาะงานโดยใช้อุปกรณ์ FPGAs ได้
5. สามารถนำวงจรเชิงเลขเฉพาะงานมาทดสอบและใช้งานได้

1.4 เนื้อหาโดยสังเขป

การนำภาษา VHDL มาใช้กำหนดและบรรยายพฤติกรรมฟังก์ชันการทำงานของฮาร์ดแวร์ในระบบดิจิตอลนั้น คือความอ่อนตัวของภาษาที่สามารถจำลองการทำงานจากหลักการของรูปแบบ (Simulate Conceptual Designs) แต่ในขณะเดียวกันก็สามารถจำลองการทำงานของฮาร์ดแวร์ ที่ให้รายละเอียดเกี่ยวกับเวลาอย่างถูกต้อง (Timing Based Simulation) และจากโครงสร้างของภาษายังสามารถจำลองการทำงานในรูปของลำดับชั้น (Hierarchy of Simulation Levels) ความสามารถดังกล่าวนี้จึงช่วยให้ผู้ออกแบบ สามารถที่จะเขียนรูปแบบบรรยาย จากระดับบนสุดของวงจรที่อยู่ในรูปสังเขป (High Level of Abstraction) ลงสู่รายละเอียดในระดับล่าง ของวงจรได้เช่น Gate Level เป็นต้น ภาษา VHDL เป็นภาษาที่สนับสนุนการเขียนรูปแบบในทุกๆ ลักษณะและวิธีการ จากนั้นทำการแปลภาษาและจำลองการทำงานเพื่อตรวจสอบผลการทำงาน หากมีข้อผิดพลาดก็สามารถแก้ไขได้ หลังจากได้ผลการทำงานตามต้องการแล้วต้องนำไปสังเคราะห์เพื่อให้เป็นวงจรแสดงการเชื่อมต่อซึ่งประกอบไปด้วยเกท (Gate) ฟลิป-ฟลอป (Flip-Flop) เสร็จแล้วจึงไปโปรแกรมลงบนอุปกรณ์ FPGAs ซึ่ง FPGAs จะประกอบไปด้วยเส้นทางการเชื่อมต่อระหว่างบล็อก ซึ่งเรา

สามารถโปรแกรมได้ และจะมีเกทให้เลือกใช้ได้ 600 ถึง 25,000 เกท ทำให้เลือกใช้งานได้อย่างกว้างขวางและยังสามารถโปรแกรมใหม่ได้

จากคุณสมบัติและความก้าวหน้าทางเทคโนโลยีของโปรแกรมภาษา VHDL และ อุปกรณ์ FPGAs ทางคณะผู้จัดทำเห็นว่าเป็นเรื่องน่าสนใจที่จะศึกษา จึงได้ทำการออกแบบโครงงานขึ้นมา โดยมีเป้าหมายคือ ใช้โปรแกรมภาษา VHDL เขียนโปรแกรมจำลองการทำงานของหน่วยประมวลผลกลางขนาด 4 บิท และใช้โปรแกรมภาษา VHDL สังเคราะห์เป็นระดับเกท และสร้างเป็นหน่วยประมวลผลขนาด 4 บิท โดยใช้อุปกรณ์ FPGAs ซึ่งเนื้อหาในปริญญานิพนธ์ฉบับนี้ประกอบไปด้วยรายละเอียดต่าง ๆ ที่สำคัญดังนี้

บทที่ 2 ทฤษฎีและหลักการ กล่าวถึงการแนะนำภาษา VHDL อันได้แก่ ประวัติความเป็นมาของภาษา VHDL , Top-Down Design , Terminology และ Conventions , Entity Design Unit , Architecture Design Unit , Packige Design Unit , Configuration Design Unit , Libraries , Design Unit Names , File Organization , Visibility , ชุดคำสั่ง Library , ชุดคำสั่ง Use

บทที่ 3 การออกแบบและการสร้างหน่วยประมวลผลกลางขนาด 4 บิต กล่าวถึงโครงสร้างและการออกแบบ,วิธีการออกแบบ,ลำดับขั้นตอนการออกแบบหน่วยประมวลผลกลางขนาด 4 บิต และการออกแบบวงจรอินเตอร์เฟส

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึงการทดลอง Simulate ในขั้นตอนที่เป็นบล็อก เป็นอุปกรณ์ FPGAs และหน่วยประมวลผลกลางขนาด 4 บิต

บทที่ 5 สรุปอภิปรายและข้อเสนอแนะ กล่าวถึงข้อสรุป , ปัญหา , แนวทางการแก้ไขและแนวทางการพัฒนาเพื่อนำไปประยุกต์ใช้งาน

ภาคผนวก ก โปรแกรมภาษา VHDL ของหน่วยประมวลผลกลางขนาด 4 บิต

ภาคผนวก ข ขั้นตอนการจำลองการทำงานโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก (Viewlogic) และขั้นตอนการติดตั้งโปรแกรม Workview Office และการใช้โปรแกรม XACT Step FOUNDATION

บทที่ 2

ทฤษฎีและหลักการ

2.1 กล่าวนำ

ปัจจุบันการออกแบบระบบเชิงเลขนั้นมีความยุ่งยากและซับซ้อนมากยิ่งขึ้นทำให้คอมพิวเตอร์มีบทบาทที่สำคัญและนำมาเพื่อช่วยในการออกแบบวงจรระบบเชิงเลขเราเรียกว่า CAD (Computer Aided Design) วิธีการออกแบบใหม่ ๆ ได้ถูกพัฒนาขึ้นจากซอฟแวร์ใหม่ ๆ เพื่ออำนวยความสะดวกให้กับนักออกแบบ ดังนั้นโปรแกรมภาษา VHDL เป็นเครื่องมืออย่างหนึ่งที่มีการพัฒนาขึ้นมา และช่วยในกระบวนการของการออกแบบวงจรเชิงเลขให้สะดวกมากยิ่งขึ้น

2.2 ประวัติความเป็นมาของภาษา VHDL

ภาษา VHDL เริ่มประมาณปี ค.ศ. 1981 โดยที่กระทรวงกลาโหมสหรัฐอเมริกา หรือ Department of Defense (DoD) เห็นว่าอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร เป็นอุปกรณ์ที่ได้รับการพัฒนามาเมื่อประมาณ 20 ปีก่อน ในขณะนั้นการพัฒนาอุปกรณ์อิเล็กทรอนิกส์เป็นไปอย่างล่าช้า เพราะเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์ ได้รับการพัฒนาไปอย่างรวดเร็ว ดังที่จะเห็นได้ว่ามีวงจรดิจิตอลอิเล็กทรอนิกส์หลายวงจร ที่แต่เดิมถูกสร้างขึ้นมาจากชิ้นส่วนอุปกรณ์อิเล็กทรอนิกส์จำนวนหลายชิ้น ประกอบกันอยู่บนแผงวงจรไฟฟ้า ที่มีขนาดใหญ่ แต่ในปัจจุบัน สามารถที่จะใช้เทคโนโลยีการออกแบบและผลิตวงจรรวมขนาดใหญ่มาก รวมอุปกรณ์ต่าง ๆ เหล่านั้นให้อยู่บนชิ้นอุปกรณ์สารกึ่งตัวนำ ที่มีขนาดประมาณ 1-2 ตร.ซม. ได้ ซึ่งเป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้น ตลอดจนความน่าเชื่อถือและความคงทนต่อสภาพแวดล้อมสูง ขณะเดียวกันนั้นในวงการทหารได้มีการนำระบบคอมพิวเตอร์และอิเล็กทรอนิกส์ มาใช้ในระบบอาวุธอย่างแพร่หลาย ดังนั้นอุปกรณ์ที่มีใช้อยู่จึงไม่เหมาะสมกับเทคโนโลยีด้านอาวุธของประเทศคู่แข่งขัน การที่จะเปลี่ยนอุปกรณ์ใหม่เป็นสิ่งที่ต้องใช้งบประมาณมาก และก็จะประสบกับปัญหาเช่นเดิมคือ อุปกรณ์ใหม่ได้รับการพัฒนามานานแล้วเช่นกัน เพราะในขณะนั้นขั้นตอนของการออกแบบ การผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ต้องใช้วิศวกร และเวลาสำหรับดำเนินการมาก ฉะนั้นทาง DoD จึงตั้งโครงการขึ้นมาเพื่อศึกษา วิธีการที่จะช่วยพัฒนาวงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำไปผลิตได้เร็วขึ้น และโครงการ

ดังกล่าวมีชื่อว่า Very High Speed Integrated Circuits หรือ VHSIC ในระยะแรกนั้นโครงการเป็นความลับของประเทศและอยู่ในความดูแลควบคุมของ United States International Traffic and Arms Regulations (ITAR) ในปี ค.ศ. 1983 ตามคำแนะนำของคณะทำงาน ("Woods Hole" workshop) ทาง DoD ได้ออกความต้องการมาตรฐานของภาษาที่ใช้สำหรับบรรยายพฤติกรรมของวงจร หรือฮาร์ดแวร์ของระบบสำหรับโครงการ VHSIC ซึ่งมีสาระสำคัญพอสรุปได้ดังนี้

1. ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถจะเข้าใจได้ทั้งคนและเครื่องโดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก

2. สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้ (Project Documentation)

3. ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร (Simulation Language)

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง (High Level Language) เช่นเดียวกับภาษา PASCAL , FORTRAN และ ADA ซึ่งในทางวิศวกรรมการออกแบบฮาร์ดแวร์ เรียกว่า Hardware Description Language หรือ HDL ดังนั้นภาษามาตรฐานนี้จึงมีชื่อว่า VHSIC-HDL หรือ VHDL นั้นเอง

โครงการ DoD ได้มอบหมายให้บริษัท IBM , Texas Instruments และ Intermetrics เป็นผู้ศึกษาและพัฒนา การดำเนินการได้กระทำไปอย่างต่อเนื่อง และได้ผลเป็นที่น่าพอใจ จนกระทั่งปี ค.ศ. 1985 ทาง ITAR ได้ยกเลิกข้อจำกัดในการถ่ายทอดเทคโนโลยีทางทหาร ออกจากโครงการนี้ ดังนั้น VHDL จึงเริ่มเป็นที่รู้จักกันโดยทั่วไป จนกระทั่งทาง IEEE จึงได้รับภาษานี้เข้ามาศึกษา และประมาณปี ค.ศ. 1987 ได้ยอมรับกำหนดมาตรฐานของภาษา โดยให้ชื่อว่า IEEE 1076-1987 และมีชื่อเรียกว่า VHDL มาตรฐานนี้ก็ได้รับการปรับปรุงจนปัจจุบัน ได้ชื่อว่า IEEE 1076-1993 หรือ VHDL 1993

2.3 Top-Down Design

การนำภาษา VHDL มาใช้กำหนดและบรรยายพฤติกรรมฟังก์ชันการทำงานของฮาร์ดแวร์ในระบบดิจิตอลนั้น คือความอ่อนตัวของภาษาที่สามารถจำลองการทำงานจากหลักการของรูปแบบ (Simulate Conceptual Designs) แต่ในขณะเดียวกันก็สามารถจำลองการทำงานของฮาร์ดแวร์ ที่ให้รายละเอียดเกี่ยวกับเวลาอย่างถูกต้อง (Timing Based Simulation) และจากโครงสร้างของภาษายังสามารถจำลองการทำงานในรูปของลำดับชั้น (Hierarchy of Simulation Levels) ความสามารถดังกล่าวนี้จึงช่วยให้การออกแบบสามารถที่จะเขียนรูปแบบบรรยาย จากระดับบนสุดของวงจรที่อยู่ในรูปสังเขป (High Level of Abstraction) ลงสู่รายละเอียดในระดับล่างของวงจรได้เช่น Gate Level เป็นต้น ในเวลานั้นวงการอุตสาหกรรมไมโครอิเล็กทรอนิกส์ ตลอดจนสถาบันวิจัยและศึกษา กำลัง

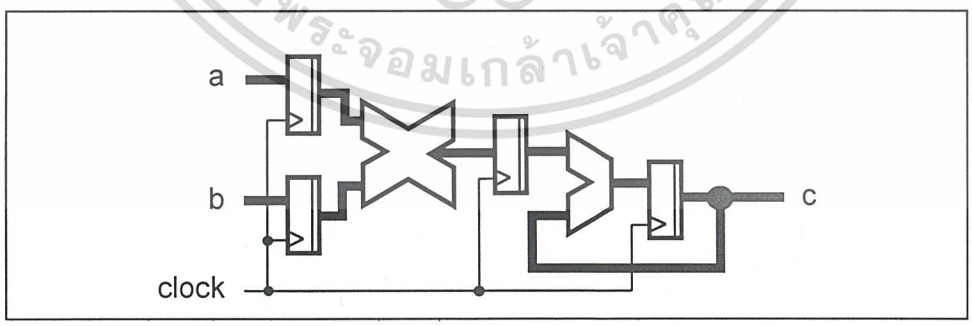
พัฒนาภาษาที่ใช้สำหรับการสังเคราะห์วงจรแบบอัตโนมัติ เพื่อลดเวลาในการพัฒนาวงจรลง ภาษา VHDL จึงถูกนำเข้าพิจารณาในโครงการนี้ด้วย โดยเพิ่มขีดความสามารถของภาษาขึ้น นอกเหนือจากเป็นภาษาที่ใช้สำหรับสังเคราะห์วงจร (Synthesis Language)

ภาษา VHDL สนับสนุนการเขียนรูปแบบในทุก ๆ ลักษณะและวิธีการ ดังเช่นตัวอย่างที่แสดงในรูปที่ 2.1 คือรูปแบบของลำดับขั้นตอนของการคูณและหาผลรวม (Multiply Accumulate algorithm) ของตัวแปรสองตัว **a** และ **b** ส่วนผลลัพธ์คือ **c** ในลักษณะของเลขฐานสอง รูปแบบนี้แสดงในระดับที่สังเขปที่สุดที่จะแก้ปัญหา เพื่อหาผลลัพธ์ โดยไม่คำนึงถึงโครงสร้างของวงจร เช่น อุปกรณ์วงจรรวม Arithmetic and Logic Unit (ALU)

```
FOR i IN 1 TO 1024 LOOP
        result := result + a(i) * b(i);
END LOOP;
c <= result;
```

รูปที่ 2.1 ลำดับขั้นตอนของการคูณและหาผลรวม

อีกด้านหนึ่งของมุมมองในปัญหาเดียวกันนี้ภาษาVHDL ก็สามารถใช้บรรยายลำดับขั้นตอนของการคูณและหาผลรวม (Multiply Accumulate Algorithm) โดยแสดงได้ในรายละเอียดของการสร้างวงจรดังกล่าวจริงๆ ตามที่เห็นได้จากรูปที่ 2.2

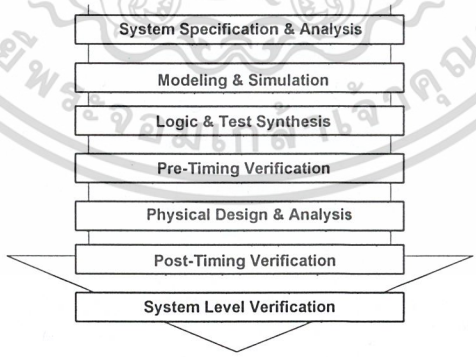


รูปที่ 2.2 โครงสร้างของ Multiply-Accumulate Unit

ฉะนั้น จากความสามารถที่จะเขียนรูปแบบได้ในลักษณะต่าง ๆ นี้เอง จึงเปิดโอกาสให้วิศวกรผู้ออกแบบได้พัฒนาและจำลองการทำงานของรูปแบบได้เร็ว ตั้งแต่ในระยะเริ่มต้นของแนวความคิด

เกี่ยวกับฟังก์ชันการทำงานของวงจรอย่างสังเขปโดยที่ยังไม่ต้องไปคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนั้น VHDL ยังเป็นภาษาที่สนับสนุนลักษณะต่าง ๆ ของระบบดิจิตอล (Digital System) ที่มีความซับซ้อนได้ทั้งหมด

การเริ่มต้นด้วยวิธีการเขียนรูปแบบจากแนวความคิดอย่างสังเขป พร้อมทั้งการจำลองการทำงานของรูปแบบที่เขียนขึ้น เพื่อตรวจสอบความถูกต้อง ประกอบกับการกลั่นกรองเพื่อเพิ่มเติมรายละเอียดลงสู่ระบบดิจิตอลที่สมบูรณ์ในรูปของวงจรไฟฟ้าที่ละขั้นนั้นเป็นขบวนการของ Top-Down Design การที่เริ่มต้นด้วยการเขียนรูปแบบ ในระดับบน (Top-Level) ของแนวความคิดอย่างสังเขปนั้น วิศวกรออกแบบสามารถที่จะพัฒนาสภาพแวดล้อมต่าง ๆ เพื่อการตรวจสอบการทำงานของวงจร (Test Environment) ได้ตั้งแต่ในระยะแรก ๆ ของการออกแบบ เพื่อใช้สำหรับตรวจสอบความถูกต้องของแนวความคิดกับสิ่งที่ต้องการจริงหรือ Specification ของงาน ดังนั้นจึงเป็นไปได้ยากที่ในระดับล่างลงมา โดยที่ได้เพิ่มรายละเอียดของรูปแบบให้มากขึ้นตามลำดับ จะเกิดข้อผิดพลาด หรือเบี่ยงเบนไปจากจุดประสงค์เดิม เพราะในแต่ละขั้นตอนย่อยจะมีการสร้างสภาพแวดล้อมเพื่อการตรวจสอบขึ้นใหม่ โดยอ้างอิงจากระดับที่อยู่สูงกว่าขึ้นไปเสมอ จากรูปที่ 2.3 แสดงให้เห็นขั้นตอนของการออกแบบในลักษณะของ Top-Down Design ทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย ก็เนื่องมาจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้ในหลาย ๆ เทคโนโลยี เช่น Programmable Logic Devices อันได้แก่ PLA , FPGA หรือ CPLD เป็นต้น นอกจากนั้นยังมี Semi-Custom IC (Gate Array, Standard Cell) และ Full Custom IC



รูปที่ 2.3 ขั้นตอนของ Top-Down Design

2.3.1 ขบวนการออกแบบโดยใช้วิธี Top-Down Design

ขั้นตอนการออกแบบ Top-Down Design มีรายละเอียดดังนี้

1. System Specification and Analysis ขั้นตอนของการสร้างข้อกำหนดของความต้องการ (Specification) และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

2. Modeling and Simulation การเขียนรูปแบบของระบบที่ต้องการจะออกแบบ โดยภาษา VHDL หรือ HDL อื่น ๆ จากแนวความคิดอย่างสังเขปที่ได้ สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงานเพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3. Logic and Test Synthesis หลังจากที่ได้หลักการขั้นต้นพร้อมกับแนวความคิดที่ผ่านการตรวจสอบแล้ว หลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมาเป็นลำดับขั้นตอนที่เหมือนกัน คือ Modeling and Simulation จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจร หรือสังเคราะห์ (Synthesis) ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้นและระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้น ให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้น หรือไม่ก็อยู่ในรูปของ Netlist ที่สามารถนำไปผลิตลงบนอุปกรณ์อื่นได้ นอกจากนั้นการผลิตบางเทคโนโลยี อาทิเช่น Gate Array หรือ Standard Cell และ Full Custom IC อาจจะมีความจำเป็นที่ต้องสร้างโครงสร้างของวงจรใหม่หลังจากที่สังเคราะห์ครั้งแรกแล้ว เพื่อความสะดวกต่อการตรวจสอบการทำงานหลังจากที่ผลิตเป็นวงจรต้นแบบแล้ว หรือที่เรียกว่า "Design For Test" (DFT) พร้อมทั้งข้อมูลในการตรวจสอบจะถูกกำหนดในขั้นตอนนี้

4. Pre-Timing Verification หลังจากการสังเคราะห์วงจรให้อยู่ในรูป Gate-Level หรือ Netlist แล้วข้อมูลที่ได้จากผู้ผลิตอุปกรณ์วงจรนั้น นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงานในเรื่องของความถูกต้องของฟังก์ชัน (Functional Simulation) แล้ว ยังมีข้อมูลที่เกี่ยวกับเวลาด้วย ซึ่งเป็นความจริงที่ว่า อุปกรณ์ทางอิเล็กทรอนิกส์ทุกชิ้นจะมี Propagation Delay เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับ nanosecond (10^{-9} second) แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกต ของฟังก์ชันต่าง ๆ จำนวน 10,000 เกตขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจจะทำให้การทำงาน ของวงจรรวมทั้งหมดผิดไป หรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาที่สูง

5. Physical Design and Analysis คือขั้นตอนของการผลิตเป็นวงจรจริง (Technology and Device Mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้าที่ประกอบด้วยอุปกรณ์หลาย ๆ ชิ้น หรืออยู่ในรูปของวงจรรวมเฉพาะงาน (ASIC)

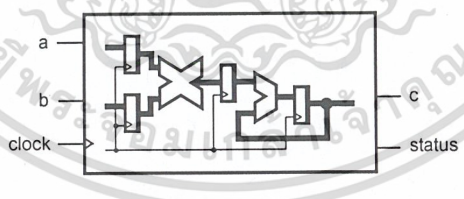
6. Post-Timing Verification หลังจากที่ได้วงจรจริงมาเรียบร้อยแล้ว ยังต้องมีความจำเป็นที่ต้องตรวจสอบการทำงานที่คำนึงถึงเวลาด้วย เพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่น ๆ ให้เป็นระบบดิจิตอล เพราะในขั้นตอนนี้วงจรที่ออกแบบ จะประกอบด้วย Input และ Output Pad ซึ่งเป็นจุดต่อสำหรับรับและส่งสัญญาณกับภายนอก

7. System Level Verification หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่น ๆ ให้เป็นระบบดิจิตอลแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่น ๆ อีกครั้งเป็นการควบคุมคุณภาพของผลิตภัณฑ์

ในตัวอย่างของ multiply accumulate algorithm แสดงให้เห็นขบวนการออกแบบในลักษณะของ Top-Down Design ได้โดยการใช้ Multiply Accumulate Function จุดประสงค์แรกก็คือ การเขียนรูปแบบของฟังก์ชันในระดับบนสุดด้วยภาษา VHDL และจำลองการทำงานของรูปแบบที่ได้เพื่อตรวจสอบความถูกต้องซึ่งฟังก์ชันนี้อาจจะเป็น Discrete Fourier Transform (DFT) ก็ได้ หลังจากที่ได้ฟังก์ชันที่ต้องการแล้วและมีชื่อว่า "multiply_accum" สามารถนำไปใช้ได้ในรูปของ Function Call คือ

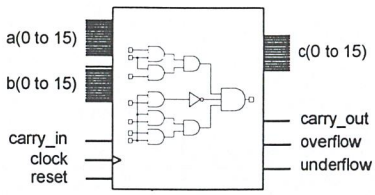
```
C <= multiply_accum(a, b);
```

ถ้าฟังก์ชันเป็นที่พอใจแล้วในขั้นต่อไปจะเป็นการเริ่มของการแปลงแนวความคิดไปสู่การสร้างวงจรจริง ขบวนการดังกล่าวเกิดขึ้นโดยการเพิ่มรายละเอียดให้กับแนวความคิดเดิม รูปที่ 2.4 แสดงให้เห็นวิธีการบรรยายแบบ Dataflow (Dataflow Description) ของ Multiply Accumulate Function



รูปที่ 2.4 Dataflow Description ของ Multiply-Accumulate Function

จาก Dataflow Format ที่ได้นี้ ขบวนการต่อไปคือการสร้างวงจรให้อยู่ในรูปของอุปกรณ์พื้นฐาน หรือ Gate-Level Implementation ซึ่งวิธีนี้อาจใช้วิธีการสังเคราะห์อัตโนมัติ ผลลัพธ์ที่ได้จากการสังเคราะห์จะเป็นการแสดงภาพวงจรด้วยเกต ของฟังก์ชันต่าง ๆ หรือในรูปของ Netlist และจะเป็นการกำหนดเทคโนโลยีจำเพาะที่จะนำไปผลิตในภายหลังในรูปที่ 2.5 แสดงให้เห็นรูปร่างของวงจรที่ได้จากการสังเคราะห์ในระดับเกต



รูปที่ 2.5 Gate-level Representation ของ Multiply-Accumulate Function

คุณสมบัติหลักของภาษา VHDL คือสามารถที่จะบรรยายฮาร์ดแวร์ ได้ในทุก ๆ ระดับของภาพรวมทั้งระบบ ฉะนั้นผู้ออกแบบจึงสามารถใช้เครื่องมือ (ภาษา) เพียงอันเดียวในการบรรยายทั้งระบบ ซึ่งก็เช่นเดียวกันกับเครื่องมือจำลองการทำงาน

2.4 Terminology and Conventions

การเขียนรูปแบบของระบบดิจิตอลด้วยภาษา VHDL นั้น จะมีศัพท์เทคนิคเฉพาะ ฉะนั้นจะอธิบายศัพท์บางคำที่จะต้องพบ

1. ลักษณะของรูปแบบ (model styles) : ลักษณะของการเขียนรูปแบบด้วยภาษา VHDL สามารถแบ่งได้เป็น

– Behavioral Model : หรือเรียกอีกอย่างได้ว่า algorithmic description เป็นรูปแบบที่บรรยายพฤติกรรมของระบบดิจิตอล ในส่วนที่บรรยายมีโครงสร้างคล้ายกับภาษาชั้นสูง (High Level Language) ทั่ว ๆ ไป เช่น PASCAL หรือ C เป็นต้น ในการจำลองการทำงานคำสั่งแต่ละคำสั่ง (Statement) จะถูกประเมินผลเป็นไปตามลำดับ (Sequential) จากบนลงล่าง ยกเว้นในกรณีของคำสั่ง LOOP หรือการเรียกใช้โปรแกรมย่อย รูปแบบนี้จะไม่ให้รายละเอียดที่เกี่ยวกับผลิต หรือโครงสร้างของฮาร์ดแวร์ แต่ในทางตรงข้ามจะให้รายละเอียดเกี่ยวกับความสัมพันธ์ระหว่าง Input กับ Output

– Dataflow Model : เรียกอีกอย่างหนึ่งได้ว่า "Register Transfer Level" (RTL) เป็น รูปแบบที่ถูกเขียนขึ้นเพื่อจุดประสงค์ที่จะใช้เครื่องมือสำหรับสังเคราะห์วงจรอัตโนมัติ รูปแบบลักษณะนี้ส่วนใหญ่จะเป็น Procedural Constructs และ Functional Operators ดูรูปที่ 2.1

– Structural Model : เป็นรูปแบบที่แสดงการเชื่อมต่อกันระหว่างอุปกรณ์ต่าง ๆ ที่ประกอบกันขึ้นเป็นวงจรหรือระบบดิจิตอลและสามารถเรียกอีกอย่างได้ว่า "Netlist Representation" เป็นการเขียนที่แสดงให้เห็นโครงสร้างของฮาร์ดแวร์

– Mixed-Level Model :จากความอ่อนตัวของภาษา VHDL จึงสามารถที่จะเขียนรูปแบบโดยใช้ลักษณะต่าง ๆ ที่กล่าวมาแล้วข้างต้น บรรยายวงจรหรือระบบดิจิตอลเดียวกันได้ ฉะนั้นรูปแบบเช่นนี้จึงมีการเขียมแบบผสม

2. Concurrency : ในภาษา VHDL นั้นชุดคำสั่งแต่ละชุดจะทำงานในเวลาเดียวกันและอิสระต่อกันซึ่งเป็นคุณสมบัติที่เป็นความจริงทางฟิสิกค์ของวงจรอิเล็กทรอนิกส์ชุดคำสั่งนี้เรียกว่า "Concurrent Statement" และจะทำงานก็ต่อเมื่อมีการเปลี่ยนแปลงค่าของสัญญาณ

3. Sequential : นอกจากความสามารถที่จะทำงานแบบ Concurrent แล้ว บางครั้งการเขียนรูปแบบในลักษณะที่บรรยายพฤติกรรมของวงจรมีความจำเป็นที่จะต้องให้ชุดคำสั่งทำงานเป็นลำดับขั้นเรียงกันจากบนลงล่าง อย่างเช่นการเขียนแบบ Behavioral Model เป็นต้น ชุดคำสั่งที่เป็น Sequential นี้จะใช้ในโปรแกรมย่อยและ Process Statement

4. Driver : สัญญาณต่าง ๆ ใน VHDL นั้นจะถูกควบคุมด้วยตัวขับหรือ "Driver" สัญญาณเหล่านี้จะรับค่าใหม่ (ระดับของสัญญาณ) ได้ด้วยตัวขับนี้เอง

5. Transaction : การเกิด Transaction กับ Signal นั้นเกิดขึ้นเมื่อมีการกำหนดค่า ๆ หนึ่งให้กับสัญญาณนั้นค่าใหม่ที่สัญญาณได้รับอาจจะมีหรือไม่มีผลทำให้เกิดการเปลี่ยนแปลงของระดับสัญญาณ (event) เช่นการเปลี่ยนจากค่า Logic '0' เป็นค่า Logic '1' เป็นต้น

6. Event : คือการเปลี่ยนระดับค่าของ SIGNAL จากระดับหนึ่งไปสู่ระดับอื่น อย่างเช่นในระบบดิจิตอลการเปลี่ยนจาก Logic '0' เป็น Logic '1' หรือในทางตรงกันข้ามถือว่า SIGNAL นั้นเกิด "Event" ฉะนั้นจะเห็นได้ว่า การที่จะเกิด Event ได้นั้นจะต้องเกิด Transaction ด้วย แต่ในทางตรงข้ามการเกิด Transaction ไม่จำเป็นต้องเกิด Event ทุกครั้ง

7. Sensitivity List : คือรายชื่อของ Signal ต่าง ๆ ที่มีผลให้เกิดการทำงานของ Concurrent Statement เมื่อเกิด Event ขึ้นกับ Signal ตัวใดตัวหนึ่งหรือหลายตัวพร้อมกันในรายชื่อนั้น

8. Objects : ในภาษา VHDL นั้นคำว่า Object ใช้เขียนเพื่อบ่งบอกถึงองค์ประกอบส่วนหนึ่งของรูปแบบ ซึ่งเปรียบได้เหมือนกับภาชนะที่มีไว้สำหรับบรรจุค่าต่าง ๆ สามารถแบ่งออกได้เป็นสามชั้นด้วยกันคือ

– CONSTANT : ได้แก่ Object ประเภทหนึ่งที่เมื่อกำหนดค่าเริ่มต้นให้แล้วจะคงค่านั้นไว้ตลอดไม่สามารถดัดแปลงหรือแก้ไขได้สามารถประกาศใช้ได้ในส่วนที่เป็นส่วนประกาศของรูปแบบ

– SIGNAL : หมายถึง Object ประเภทหนึ่งที่สามารถกำหนดค่าที่สัมพันธ์กับเวลาให้ได้ นั้นหมายความว่า SIGNAL สามารถรับค่าได้เพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง SIGNAL จะรับค่า ๆ หนึ่งได้จากตัวขับสัญญาณหรือ Driver ซึ่งตัวขับนี้อาจจะเก็บค่าในอนาคตสำหรับ

SIGNAL ไว้ด้วย SIGNAL สามารถประกาศใช้ได้ในส่วนที่เป็นเนื้อที่ของ Concurrent Body เท่านั้น ดังนั้น SIGNAL จึงสามารถถูกนำไปใช้ได้ตลอดโครงสร้างของรูปแบบหรือที่เรียกว่า Gobal Object

– VARIABLE : หรือตัวแปรได้แก่ Object ที่สามารถกำหนดค่าใด ๆ ได้ และสามารถที่จะเปลี่ยนแปลงค่าได้ตลอดการจำลองการทำงาน แต่จะเก็บค่าเพียงค่าเดียวเท่านั้น เนื่องจาก VARIABLE สามารถประกาศใช้ได้ในส่วนที่เป็นส่วนประกาศของ PROCESS, FUNCTION หรือ PROCEDURE ดังนั้น VARIABLE จึงสามารถนำไปใช้ได้เฉพาะในขอบเขตที่ถูกประกาศใช้

VHDL นั้นประกอบด้วยส่วนต่าง ๆ ที่สำคัญและเป็นพื้นฐานของการเขียนรูปแบบระบบดิจิตอลที่สำคัญ 4 หน่วยคือ

1. Entity Design Unit
2. Architecture Design Unit
3. Package Design Unit
4. Configuration Design Unit

2.5 Entity Design Unit

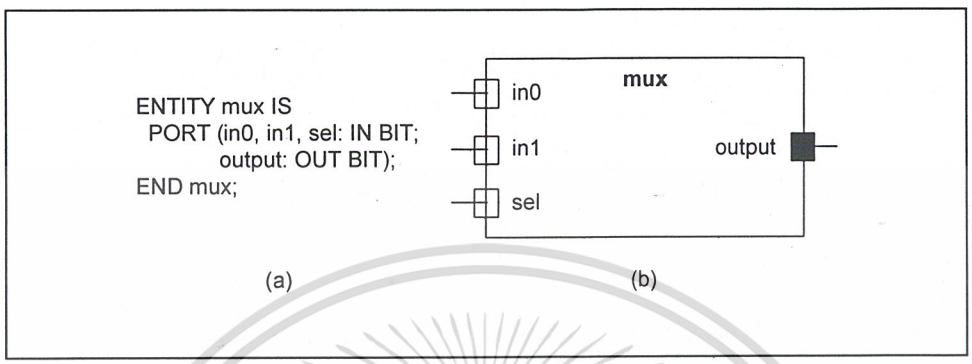
หน่วยของแบบ (Design Unit) ใช้สำหรับติดต่อระหว่างภายนอกกับรูปแบบที่จะเขียนขึ้น เรียกว่า "Entity Design Unit" ในส่วนนี้ใช้กำหนดจุดต่อ (Connection Point) ของรูปแบบกำหนดการไหลของสัญญาณและประเภทของค่าที่สามารถกำหนดสัญญาณตามจุดต่าง ๆ ของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น รูปที่ 2.6 แสดงโครงสร้างอย่างง่ายๆ ของ Entity Design Unit

```
ENTITY component_name IS
        input and output ports
        physical and other parameters
END [component_name] ;
```

รูปที่ 2.6 โครงสร้างอย่างง่าย ๆ ของ Entity Design Unit

ส่วนนี้จะขึ้นต้นด้วยคำ ENTITY และ IS ซึ่งระหว่างคำทั้งสองนี้เป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน (component_name) สำหรับกฎการตั้งชื่อนั้นจะต้องเป็นไปตามกฎเกณฑ์ของภาษา หลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้า-ออกของข้อมูล (Input-Output) รวมทั้งพารา

มิเตอร์อื่น ๆ ส่วนนี้เรียกว่าส่วนหัว (Entity Header) และที่สำคัญคือ Entity Design Unit จะต้องปิดท้ายด้วยคำว่า END และเครื่องหมายอัฒภาค หรือ Semicolon (;)



รูปที่ 2.7 รูปแบบของ 2:1 Multiplexer, (a) VHDL Entity Design, (b) มุมมองของ Interface

ในรูปที่ 2.7 เป็น Entity Design Unit ที่บรรยายอุปกรณ์ที่มีชื่อว่า Mux ในส่วนหัวของ Entity (Header) มีการกำหนดจุดต่อ 4 จุดภายใต้ชุดคำสั่ง PORT โดยที่ 3 จุดแรกเป็นจุดให้ข้อมูลไหลผ่านเข้า (Input) ได้แก่ in0, in1 และ sel ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอกเป็นการไหลเข้า (IN) ที่แสดงด้วยรูปสี่เหลี่ยมโปร่งในรูปที่ 2.1 ส่วนจุด Output เป็นจุดให้ข้อมูลไหลออก (Output) ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอกเป็นการไหลออก (OUT) ที่แสดงด้วยรูปสี่เหลี่ยมทึบในรูปที่ 2.7 ส่วนประเภท (Type) ของข้อมูลที่จะไหล เข้า-ออก นั้น เป็นประเภท BIT ที่สามารถมีค่าได้เพียงสองค่าคือ '0' และ '1' เท่านั้น

2.6 Architecture Design Unit

คือส่วนที่ใช้เขียนบรรยายกำหนดพฤติกรรมของรูปแบบในมุมมองของการจำลองการทำงานพฤติกรรมต่าง ๆ ที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่าน เข้า-ออก ตรงช่องทางตลอดจนพารามิเตอร์ต่าง ๆ (Ports and Generics) ที่กำหนดใน Entity Design Unit รูปที่ 2.8 แสดง ให้เห็นโครงสร้างอย่างง่ายๆ ของ Architecture Design Unit

```
ARCHITECTURE identifier OF component_name IS
        [ declaration ]
BEGIN
        specification of the functionality of the
        component in terms of its input lines and as
        influenced by physical and other parameters
END [identifier] ;
```

รูปที่ 2.8 โครงสร้างอย่างง่าย ๆ ของ Architecture Design Unit

ส่วนของ Architechture Design Unit นั้น เริ่มต้นด้วยคำ ARCHITECTURE และตามด้วยชื่อ สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า Architecture นั้นใช้บรรยาย Entity Design Unit ใด (OF <*entity design unit*> IS) ส่วนที่อยู่ระหว่าง ARCHITECTURE และ BEGIN เป็นส่วนประกาศกำหนด (Architecture Declarative Area) ที่เป็นเพียงส่วนเผื่อเลือก (Option) ในบริเวณนี้สามารถใช้เขียนประกาศกำหนดค่าต่าง ๆ ที่จะนำไปใช้ภายใน Architecture นั้นได้ อาทิเช่นประเภท ต่าง ๆ (ตัวอย่างเช่น BIT, BIT_VECTOR), สัญญาณ (SIGNAL), ตัวคงที่ (CONSTANT), โปรแกรมย่อย (ได้แก่ FUNCTION และ PROCEDURE) และอุปกรณ์ (COMPONENT) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า และไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง PORT) นั้นจะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า BEGIN กับ END ของ Architecture Design Unit และนอกจากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (Concurrent Statement) เท่านั้น Architecture Design Unit จะต้องปิดท้ายด้วยคำสั่ง END และชื่อของ Architecture นั้น ๆ ที่เป็นส่วนเผื่อเลือก โดยทั่วไปการเขียนรูปแบบระบบดิจิตอลด้วยภาษา VHDL สามารถเขียนได้ในลักษณะต่าง ๆ ดังนี้

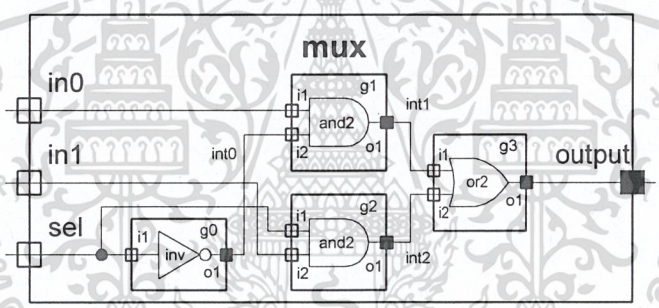
1. Dataflow Description
2. Behavioral Description
3. Structure Description
4. Mixed Model Description

```
ARCHITECTURE data_flow OF mux IS
BEGIN
        output  <=((NOT sel) AND in0) OR (sel AND in1);
END data_flow;
```

รูปที่ 2.9 Architecture design unit ของ 2:1 mux ตาม boolean expression

$$output = (\overline{sel} \cdot in0) + (sel \cdot in1)$$

รูปที่ 2.9 ส่วนที่บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า (in0, in1) กับข้อมูลที่ไหลออก ประกอบด้วยชุดคำสั่งแบบแข่งขนานเพียงชุดเดียว ซึ่งเป็นการบรรยายพฤติกรรมของ 2:1 Mux การบรรยายลักษณะนี้เรียกว่า Dataflow Description หรือ Register Transfer Level (RTL)



รูปที่ 2.10 Architecture description ของ 2:1 Multiplexer (Structural Description)

รูปที่ 2.10 เป็น Architecture ของการบรรยาย 2:1 Mux ในลักษณะของ Structural Description โดยใช้ Inverter (inv ที่ตำแหน่ง g0) , AND-gate 2 inputs จำนวน 2 gates (and2 ที่ตำแหน่ง g1 และ g2) และ OR-gate 2 inputs (or2 ที่ตำแหน่ง g3) มาสร้างตาม Boolean Expression ของรูปที่ 2.9

2.7 Package Design Unit

ข้อมูลต่าง ๆ และโปรแกรมย่อยจะเก็บใน Package และข้อมูลเหล่านี้ถูกเรียกไปใช้ได้โดย Entity Design Unit, Architecture Design Unit หรือจาก Package Design Unit อื่นๆ ด้วยชุดคำสั่ง

USE statement นอกจากนั้นสิ่งที่นิยมทำกันมากอย่างหนึ่งคือ รูปแบบมาตรฐานต่าง ๆ เช่น Standard Components จะเก็บไว้ใน Package ที่สามารถเข้าถึงและนำไปใช้ได้ สิ่งที่ประกาศใน Package ได้แก่

1. Subprogram
2. Types
3. Constants
4. Signals
5. Aliases
6. Attributes
7. Component
8. Disconnection Specification

โดยปกติแล้ว Package จะแบ่งเป็นสองส่วนคือ

1. Package declaration
2. Package body

เนื่องจาก Package ถูกสร้างเป็นส่วนแยกออกจากรูปแบบที่เขียนอยู่ การที่จะนำ Package ไปใช้นั้น จะต้องมีการเชื่อมโยงหรืออ้างอิงก่อน ซึ่งในภาษา VHDL ใช้ชุดคำสั่ง USE statement

2.7.1 Package declaration

ส่วนสำคัญที่สุดของ Package ได้แก่ Package Declaration เพราะจะกำหนดชื่อของสิ่งที่ประกาศอยู่ภายใน Package สำหรับนำไปใช้ภายนอก Package ถ้าสิ่งที่ประกาศในส่วนของ Package Body แต่ไม่ถูกประกาศใน Package Declaration จะไม่สามารถนำค่าและพฤติกรรมไปใช้ส่วนนอก โดยทั่วไปแล้ว Package สามารถสร้างขึ้นได้โดยไม่ต้องมีส่วน Body และยังถูกนำไปใช้จากรูปแบบภายนอกได้เช่น ใช้สำหรับประกาศ TYPE หรือ Signal แต่ Package นั้นจะนำไปใช้จากรูปแบบอื่นไม่ได้ การเขียน Package Declaration มีการเขียนดังแสดงในรูปที่ 2.11

```
PACKAGE package_name IS
        package_declarative_part
END package_name ;
```

รูปที่ 2.11 โครงสร้างของ package declaration

รูปที่ 2.12 เป็นตัวอย่างหนึ่งของการเขียน Package Declaration ที่มีการประกาศ TYPE, CONSTANT, COMPONENT และ SIGNAL

```
PACKAGE example IS
        TYPE cd IS ('C', 'D');
        CONSTANT pi : REAL := 3.14159;
        COMPONENT ttl_74163 IS
                PORT (    a, b: IN BIT;
                          c: OUT BIT );
                END COMPONENT;
                SIGNAL gobal_clock: BIT;
END example;
```

รูปที่ 2.12 ตัวอย่างของ package declaration

2.7.2 Package body

โครงสร้างที่ประกอบด้วยคำสั่งลำดับที่บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย ที่ชื่อของโปรแกรมย่อยถูกประกาศในส่วนของ Package Declaration แล้ว จะถูกเก็บไว้ใน Package Body รวมทั้ง Deferred Constants อันได้แก่ตัวคงที่ที่ถูกประกาศชื่อก่อนในส่วนของ Delaration แต่ถูกกำหนดค่าในส่วนของ Body ของ Package ฉะนั้นส่วน Package Bodyจึงไม่จำเป็นต้องมีถ้าในส่วน Package Declaration ไม่มีการประกาศชื่อในส่วนโปรแกรมย่อยหรือ Deferred Constant การเขียน Package Body นั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 2.13

```
PACKAGE BODY package_name IS
                declarative part
END package_name ;
```

รูปที่ 2.13 โครงสร้างของ Package Body

ชื่อของ package_name ที่ใช้ใน Package Body จะต้องเป็นชื่อเดียวกับชื่อที่กำหนดไว้ใน Package Declaration ในรูปที่ 2.14 แสดงตัวอย่างของการเขียน Package โดยนำการกำหนดโปรแกรมย่อยประเภท FUNCTION

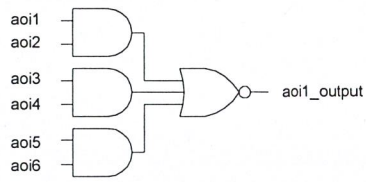
```
-- package declaration
PACKAGE pack_funct IS
        FUNCTION mean (a, b, c : REAL) RETURN REAL;
END pack_funct;
-- package body
PACKAGE BODY pack_funct IS
        FUNCTION mean (a, b, c : REAL) RETURN REAL IS
        BEGIN
                RETURN (a + b + c)/3.0;
        END mean;
END pack_funct;
```

รูปที่ 2.14 ตัวอย่างการเขียน Package

ในส่วนของ Package Declaration จะบรรจุส่วนที่เรียกว่า Function Declaration ในการประกาศชื่อของโปรแกรมย่อยและส่วนที่บรรยายการทำงานของโปรแกรมย่อย Mean (เรียกว่า Function Body) จะถูกเก็บไว้ในส่วน Package Body สิ่งสำคัญอย่างหนึ่งของการเขียน Package คือ ก่อนที่จะนำ Package ไปใช้ Package นั้นจะต้องถูกวิเคราะห์เสียก่อนว่าถูกต้อง หรือพูดง่าย ๆ ได้ว่า จะต้องผ่านการ Compile ก่อนนั้นเอง

2.8 ตัวอย่างการออกแบบ

หลังจากได้ศึกษาส่วนประกอบของภาษา VHDL และความสัมพันธ์ระหว่างส่วนต่าง ๆ ของรูปแบบ และการอ้างอิงซึ่งกันและกันแล้ว จะแสดงให้เห็นรูปแบบของวงจรดิจิตอลอย่างง่าย ๆ โดยอยู่ในรูปของรูปแบบที่เขียนด้วยภาษา VHDL เพื่อเปรียบเทียบให้เห็นกับโครงสร้างที่อยู่ในรูปของ Schematic ของอุปกรณ์ตามรูปที่ 2.15



รูปที่ 2.15 วงจรดิจิตอลลักษณะของ Schematic Diagram

จากรูปที่ 2.15 จะสังเกตได้ว่า ทั้งวงจรประกอบด้วยอุปกรณ์พื้นฐานเพียงสองประเภทได้แก่ 2_input AND-gate (จำนวน 3 ชิ้น) และ 3_input NOR-gate (จำนวน 1 ชิ้น) โดยสมมุติว่าอุปกรณ์เหล่านี้ถูกเก็บไว้ในตู้เก็บอุปกรณ์ชื่อ ieee ลิ้นชักใส่ของชื่อ std_logic_1164 การออกแบบโดยวิธีการเขียนรูปแบบด้วยภาษา VHDL เริ่มต้นด้วยการทำให้ตู้เก็บอุปกรณ์ (สำหรับระบบ VHDL หมายถึง library) ชื่อสัญลักษณ์ ieee ให้ Visible และบ่งบอกถึงลิ้นชัก (สำหรับระบบ VHDL หมายถึง Package) ชื่อ std_logic_1164 ที่เก็บอุปกรณ์ (Compnent) ทั้งสองโดยชุดคำสั่ง LIBRARY และ USE ตามที่แสดงในรูปที่ 2.16

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY and2 IS
        PORT (    in1, in2: IN std_logic;
                  output: OUT std_logic  );
END and2;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY nor3 IS
        PORT (    in1, in2, in3: IN std_logic;
                  output: OUT std_logic  );
END nor3;
```

รูปที่ 2.16 ส่วนที่บรรยายการติดต่อกับภายนอกของเกต

หลังจากที่ทำให้ทุก ๆ อย่าง ที่อยู่ใน package ชื่อ std_logic_1164 ภายใต้ library ชื่อ ieee สำหรับการออกแบบต่อไป การเขียนรูปแบบจะเริ่มในระดับล่างสุดของวงจร ซึ่งได้แก่ส่วนที่ติดต่อ

กับภายนอกของอุปกรณ์ สิ่งที่ต้องบรรยายต่อมาได้แก่ พฤติกรรมของวงจร หรือความสัมพันธ์ระหว่างสัญญาณ Output และ Input ซึ่งหมายถึง Architecture Design Unit ตามที่แสดงในรูปที่ 2.17

```
ARCHITECTURE and2_behave OF and2 IS
BEGIN
   output <= in1 AND in2 AFTER 3 NS;
END and2_behave;
ARCHITECTURE nor3_behave OF nor3 IS
BEGIN
   output <= NOT(in1 OR(in2 OR in3)) AFTER 4 NS;
END nor3_behave;
```

รูปที่ 2.17 ส่วนที่บรรยายพฤติกรรมของเกต

สำหรับการบรรยายวงจรสมบูรณ์นั้น จำเป็นที่ต้องกำหนดส่วน entity ใหม่ โดยที่ชื่อของสัญญาณต่าง ๆ เป็นไปตามวงจร Schematic ที่เขียนได้ตามรูปที่ 2.18

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY aoi IS
PORT (aoi1, aoi2, aoi3, aoi4, aoi5, aoi6: IN std_logic;
      aoi_output: OUT std_logic  );
END aoi;
```

รูปที่ 2.18 ส่วน entity ของวงจร

เนื่องจากส่วนที่เป็น Architecture ของรูปแบบเป็นหน่วยรองของส่วนที่เป็น entity ดังนั้นข้อมูลทั้งหลายที่ Visible สำหรับหน่วยหลัก จะสามารถนำไปใช้ได้กับหน่วยรอง หลังจากที่สร้างรูปแบบในระดับล่างสุดแล้ว หมายถึงการบรรยายส่วนที่เป็น entity และ Architecture ของอุปกรณ์ 2_input AND-gate และ 3_input NOR-gate ขั้นตอนต่อไปคือการออกแบบในระดับสูงขึ้นอีกชั้น ได้แก่การนำอุปกรณ์มาประกอบกันตามโครงสร้างของวงจร ตามที่แสดงในรูปที่ 2.19

```
ARCHITECTURE struct OF aoi IS
    COMPONENT and2
            PORT (    in1, in2: IN std_logic;
                      output: OUT std_logic );
    END COMPONENT;
    COMPONENT nor3
            PORT (    in1, in2, in3: IN std_logic;
                      output: OUT std_logic  );
    END COMPONENT;
    SIGNAL internal1, internal2, internal3: std_logic;
BEGIN
    u1:     and2    PORT MAP (aoi1, aoi2, internal1);
    u2:     and2    PORT MAP (aoi3, aoi4, internal2);
    u3:     and2    PORT MAP (aoi5, aoi6, internal3);
    u4:     nor3    PORT MAP (internal1, internal2, internal3, aoi_output);
END struct;
```

รูปที่ 2.19 การเชื่อมต่อสัญญาณภายในของวงจร

ภาษา VHDL ให้ผู้ออกแบบเขียน Architecture จำลองการทำงานหรือสังเคราะห์วงจรเพื่อกำหนดการเชื่อมหน่วยหลักกับหน่วยรองคือเขียนส่วนที่เป็นConfiguration ดังแสดงในรูปที่ 2.20

```
LIBRARY ieee;
CONFIGURATION top_level OF aoi IS
    FOR struct
            FOR u1: and2 USE ENTITY WORK.and2(and2_behave);
            END FOR;
            FOR u2: and2 USE ENTITY WORK.and2(and2_behave);
            END FOR;
            FOR u3:   and2 USE ENTITY WORK.and2(and2_behave);
            END FOR;
            FOR u4: nor3 USE ENTITY WORK.nor3(nor3_behave);
            END FOR;
    END FOR;
END top_level;
```

รูปที่ 2.20 โครงสร้างระดับบนสุดของรูปแบบ

2.9 โครงสร้างภายในของอุปกรณ์ FPGAs

FPGAs เป็นวงจรรวมที่สามารถโปรแกรมเป็นวงจรเชิงเลขโดยโปรแกรมลงบนสแตติกแรมภายในด้วยข้อมูลที่อยู่ภายนอก และสามารถโปรแกรมใหม่ได้โดยการรีเซตด้วยสัญญาณไฟฟ้า FPGAs ยังประหยัดไฟและมีความจุวงจรสูง (เกตมาก)

วงจรรวมผลิตโดยบริษัทไซลิงค์ (Xilinx) ซึ่งค้นคว้าร่วมกับบริษัทเอ็มเอ็มไอ (MMI) สร้างเป็นกลุ่มของเกทจำนวน 600-25000 เกท ดังในตารางที่ 2.1

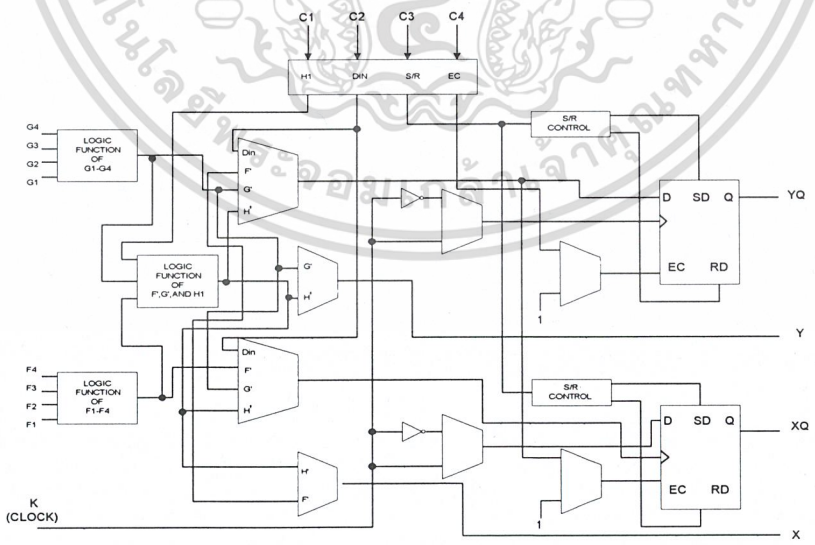
ตารางที่ 2.1 คุณสมบัติของ FPGAs ตระกูลต่าง ๆ

FPGAs	Appr.gate Count	Max I/Os	Flip-Flop	RAM bits	Available CLBs
XC2064	1,000	58	122	0	64
XC2018	1,500	74	174	0	100
XC3020/3120	1,800	64	256	0	64
XC3030/3130	2,700	80	360	0	100
XC3042/3142	3,700	96	480	0	144
XC3064/3164	5,500	120	688	0	244
XC3090/3190	7,500	144	928	0	320
XC3195	9,000	176	1,320	0	484
XC4002A	2,000	64	256	2,048	64
XC4003/4003A	3,000	80	360	3,200	100
XC4003H	3,000	160	200	3,200	100
XC4004	4,000	960	480	4,608	144
XC4005/4005A	5,000	122	616	6,072	196
XC4005H	5,000	192	392	6,272	196
XC4006	6,000	128	768	8,192	256
XC4008	8,000	144	936	10,368	324
XC4010	10,000	160	1,120	12,800	400
XC4013	13,000	192	1,536	18,432	576
XC4025	25,000	256	2,560	32,768	1,024

FPGAs มีโครงสร้างภายในใกล้เคียงกับสถาปัตยกรรมของเกตอาเรย์ (Gate Array Logic, GAL) มาก สามารถโปรแกรมและลบคอนฟิกกูเรชั่น (Configuration) ภายในสแตติกแรม (Static RAM) ได้โดยใช้กระแสไฟฟ้า ซึ่งทำการโปรแกรมได้โดยดึงข้อมูลฐานสิบหกมาจากภายนอก เช่น Parallel EPROM หรือ Serial PROM ต่างกับ EPLD , PAL ที่มี EPROM อยู่ในตัว ภายใน FPGAs จะจัดเรียงเป็นลอจิกเซลล์ล้อมรอบภายนอกด้วยอินพุต , เอาต์พุตเซลล์ FPGAs ตัวแรกที่ผลิตโดยบริษัทไซลิงค์คือเบอร์ XC2064(2000 Family) ประกอบด้วยเซลล์เรียงกันเป็นเมตริกซ์ (Matrix) จำนวน 64 เซล หลังจากนั้นผลิต FPGAs ตระกูล 3000 และ 4000 ซึ่งมีโครงสร้างซับซ้อนขึ้น สามารถเพิ่มจำนวนเกตได้มากขึ้นและดีขึ้นแต่ละเซลเรียกว่า CLB (ConfigurableLogic Block)

2.9.1 ส่วนที่เป็นองค์ประกอบของลอจิก (Configurable Logic Block)

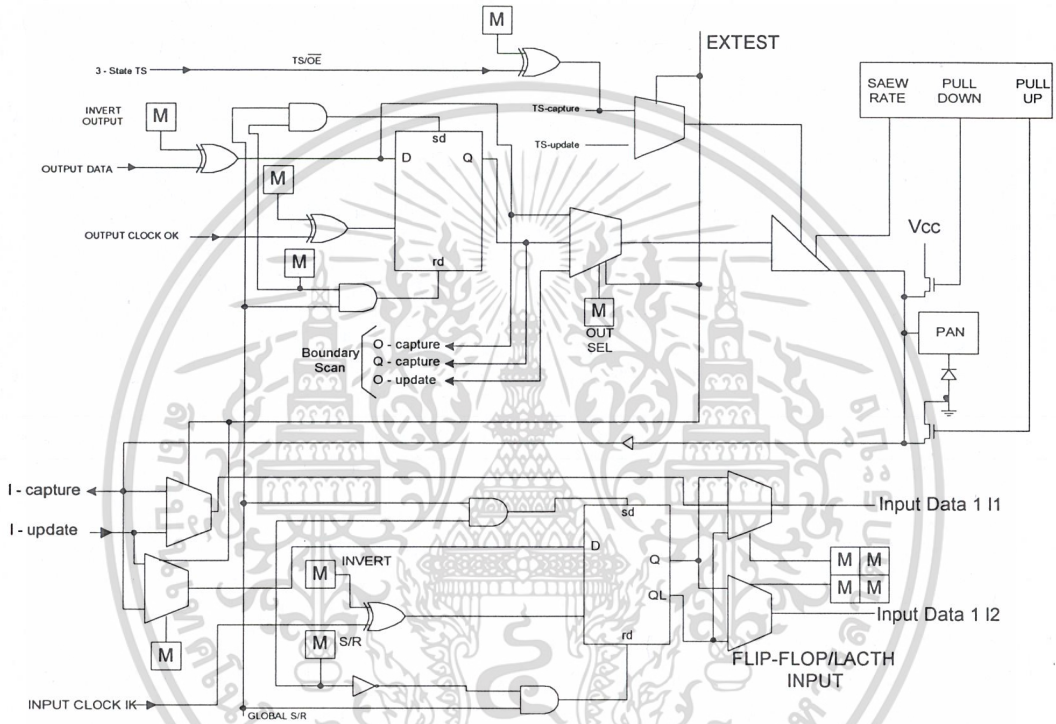
CLB จะจัดเรียงกันเป็นแบบเมตริกซ์แบบอาเรย์ขนาด M*N การออกแบบนั้นสามารถทำได้ โดยการจัดวาง CLB และต่อเชื่อมขาของ CLB ให้ต่อถึงกัน เราสามารถจัด CLB ให้เชื่อมต่อถึงกันได้โดยการทำด้วยมือหรือให้โปรแกรมที่สนับสนุน FPGAs ทำให้โดยอัตโนมัติ โดยวิธีของมันเองสำหรับไฟล์ที่ได้จากโปรแกรมเหล่านี้ เราเรียกว่า ไฟล์ที่กำหนดการวางอุปกรณ์ (Configuration File) ซึ่งจะบรรจุโครงร่างภายในของ CLB ตามความเหมาะสม อีกด้านหนึ่งไฟล์กำหนดการวางอุปกรณ์นั้นจะเป็นไฟล์กระแสข้อมูล (Bit Stream) ซึ่งสามารถใช้โปรแกรมหน่วยความจำภายในของ FPGAs ได้ สำหรับรูปที่ 2.21 แสดง CLB ของ FPGAs ตระกูล 4000



รูปที่ 2.21 แผนผัง CLB ของอุปกรณ์ FPGAS ตระกูล XC4000

2.9.2 ส่วนอินพุตและเอาต์พุตของอุปกรณ์ FPGAs

รอบนอกของ FPGAs จะประกอบด้วย IOBs ประมาณ 64 ถึง 144 ตัว ซึ่งขึ้นอยู่กับตระกูลของ FPGAs ซึ่ง IOBs จะเป็นตัวเชื่อมต่อระหว่างภายในกับภายนอกของวงจรลอจิกของ FPGAs ลักษณะของ IOBs จะมีลักษณะ 2 ทิศทาง สามารถโปรแกรมให้เป็นอินพุตหรือเอาต์พุตก็ได้ สำหรับรูปที่ 2.22 แสดง IOBs ของ FPGAs ตระกูล 4000



รูปที่ 2.22 แผนผัง IOBs ของอุปกรณ์ FPGAs ตระกูล XC4000

2.10 รายละเอียดการใช้งานอุปกรณ์ FPGAs

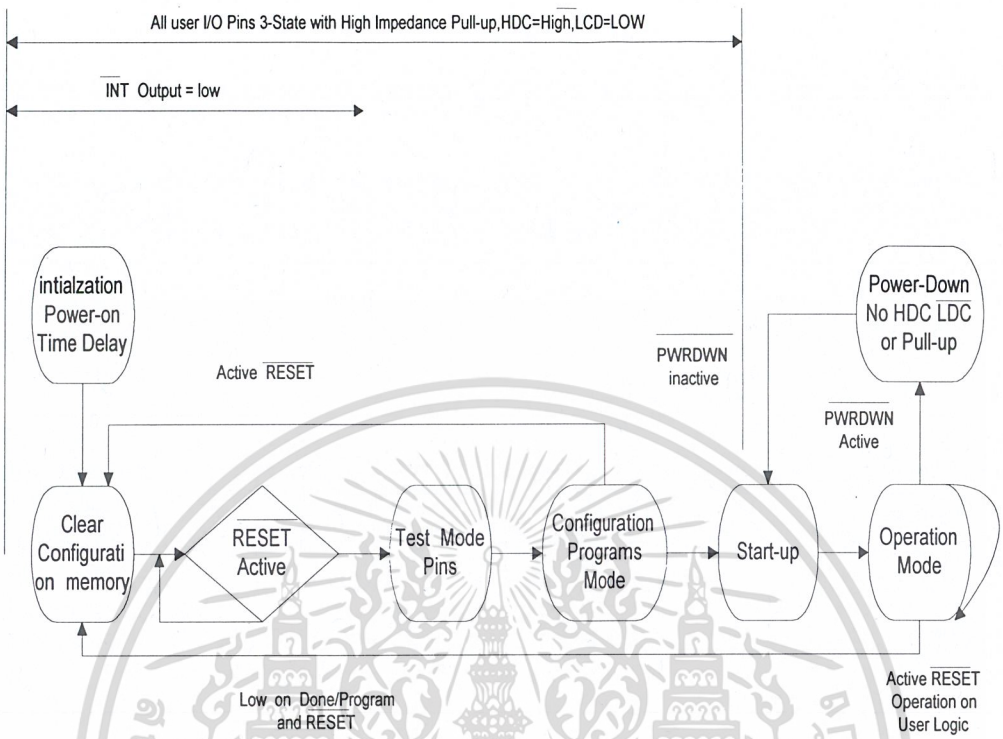
FPGAs สามารถทำงานได้หลายลักษณะโดยกำหนดได้ที่ขาสัญญาณ M0 M1 M2 ดังแสดงในตารางที่ 2.2 ในลักษณะมาสเตอร์พาราเรล (Master Parallel Mode) รันโปรแกรมคอนฟิกทีละ 1 ไบต์ (Byte) จากหน่วยความจำภายนอกที่เป็นแบบขนาน โดยสามารถรันโปรแกรมคอน (Config) จากแอดเดรส (Address) ต่ำหรือสูงก่อนก็ได้ การต่อลักษณะเพอริเฟอรัล (Peripheral) จะรับโปรแกรมคอนฟิกทีละ 1 ไบต์ จากไมโครโปรเซสเซอร์ โดยสามารถโต้ตอบกันได้ว่าพร้อมหรือไม่ที่จะรับข้อมูลต่อไป การต่อลักษณะสเลฟซีเรียล (Slave Serial) จะรับโปรแกรมคอนฟิดทีละ 1 บิต

จากไมโครโปรเซสเซอร์ตามสัญญาณอินพุต CCLK ส่วนการต่อลักษณะมาสเตอร์ซีเรียล (Master Serial) จะรับโปรแกรมคอนฟิกทีละ 1 บิตจากหน่วยความจำภายนอกที่เป็นแบบอนุกรม

ตารางที่ 2.2 โหมดต่าง ๆ ของการคอนฟิกกูเรชั่น

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

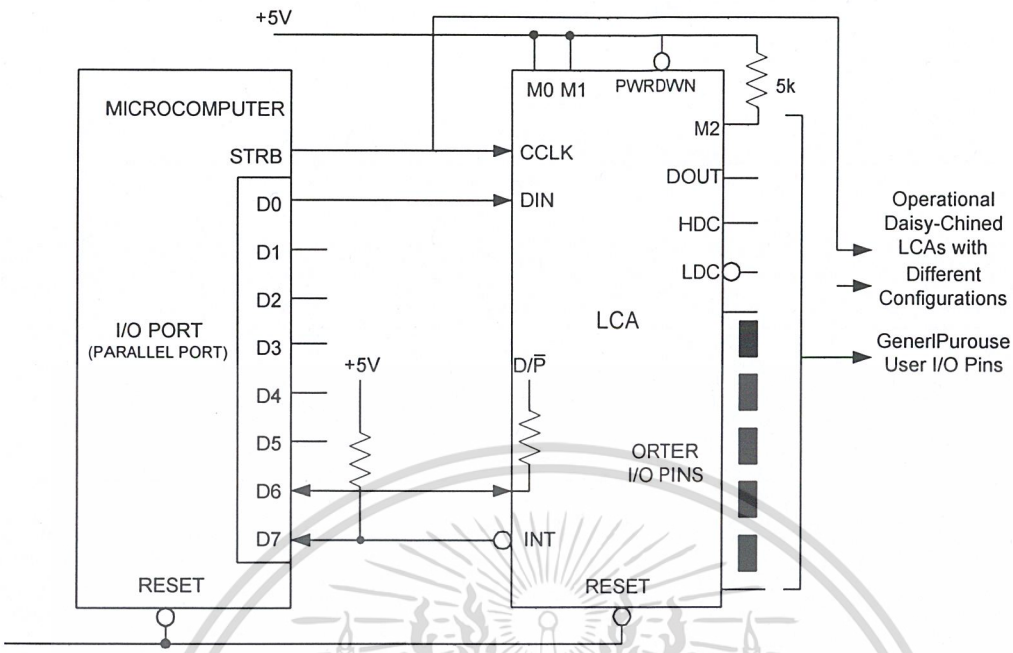
การใช้งาน FPGAs ในการต่อลักษณะสเลฟซีเรียลและมาสเตอร์ซีเรียล เมื่อเริ่มจ่ายไฟเข้าตัว FPGAs จะทำการลบข้อมูลหน่วยความจำที่ใช้ในคอนฟิก (Configuration Memory) ตรวจสอบลักษณะการคอนฟิกว่าเป็นลักษณะใดในตารางที่ 2.2 ว่าเป็นแบบอนุกรมหรือขนาน หลังจานนั้นจะเริ่มทำการโปรแกรมคอนฟิกสัญญาณ DONE/PROGRAM เป็น “0” ซึ่งอยู่ระหว่างโปรแกรมและเมื่อข้อมูลตรงกับที่ส่วนหัวของข้อมูลคอนฟิกสัญญาณ DONE/PROGRAM จะเป็น “1” ซึ่งหมายถึงโปรแกรมทำการคอนฟิกเสร็จสิ้น ดูรูปที่ 2.23 ประกอบ



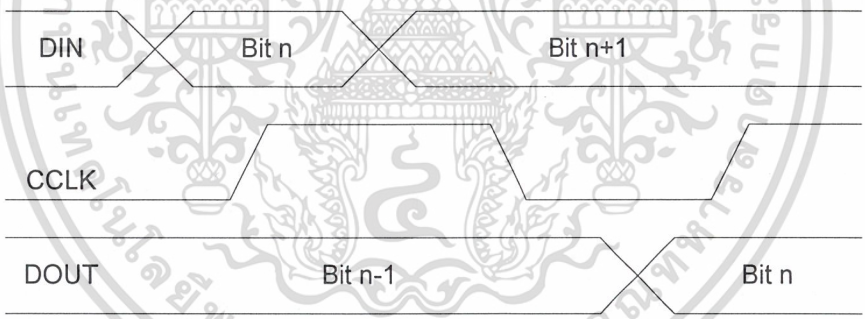
รูปที่ 2.23 ลำดับไดอะแกรมในการคอนฟิกเมื่อเริ่มป้อนแหล่งจ่ายไฟเข้าไอซีและการโปรแกรมใหม่

2.10.1 การใช้งานในลักษณะสเลฟซีเรียล

การต่อในลักษณะนี้เหมาะสมกับวงจรที่ออกแบบมาเพื่อทำงานร่วมกับไมโครคอมพิวเตอร์ ทั้งนี้เพราะ FPGAs ได้ใช้ความสามารถของไมโครคอมพิวเตอร์ในการเก็บและส่งข้อมูลคอนฟิกให้ เพียงแต่ต้องเขียนโปรแกรมเพื่อส่งโปรแกรมคอนฟิกให้เพิ่มลักษณะการต่อในลักษณะนี้เป็นดังรูปที่ 2.24 ซึ่งไมโครคอมพิวเตอร์จะสร้างสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGAs การป้อนโปรแกรมคอนฟิกให้ FPGAs ทำได้โดยต่อสัญญาณ Strobe เข้ากับขา CCLK และพอร์ต D0 เข้ากับขา DIN สร้างสัญญาณคล็อกป้อนที่ขา CCLK และป้อนโปรแกรมคอนฟิกแบบอนุกรมเข้าที่ขา DIN ดังแผนภูมิรูปที่ 2.24



รูปที่ 2.24 การต่อใช้งานในลักษณะสเลฟซีเรียล

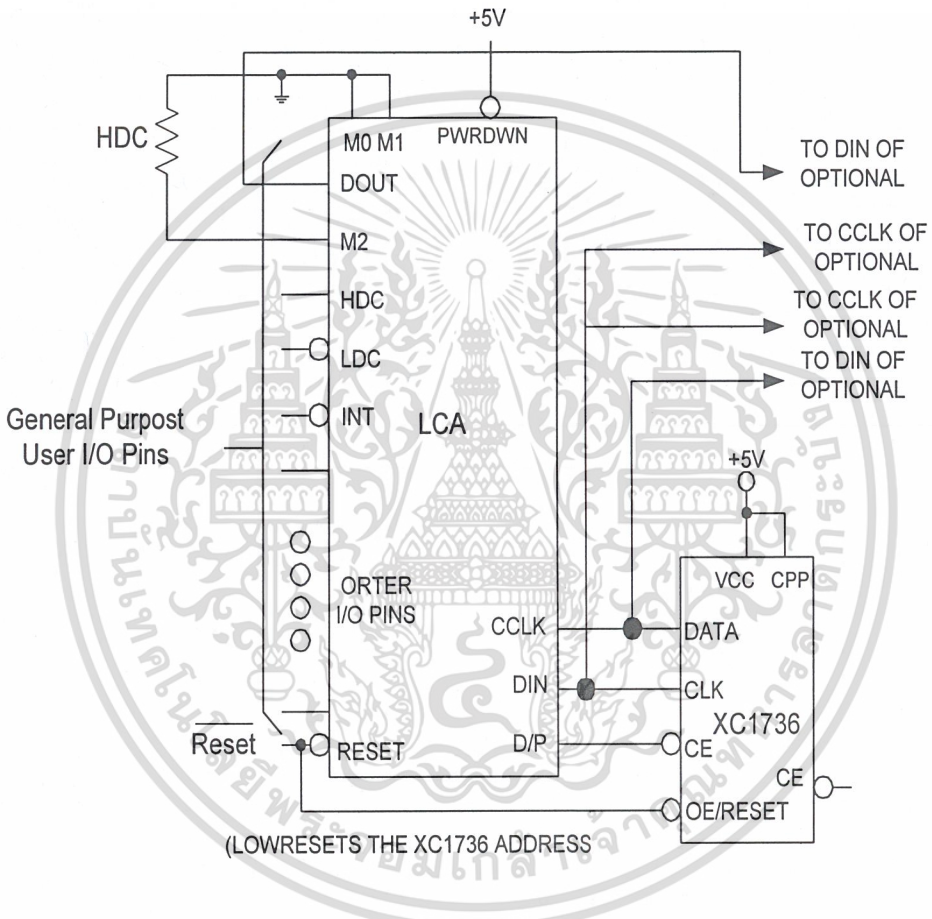


รูปที่ 2.25 แผนภูมิเวลาการป้อนข้อมูลโปรแกรมคอนฟิกในลักษณะสเลฟซีเรียล

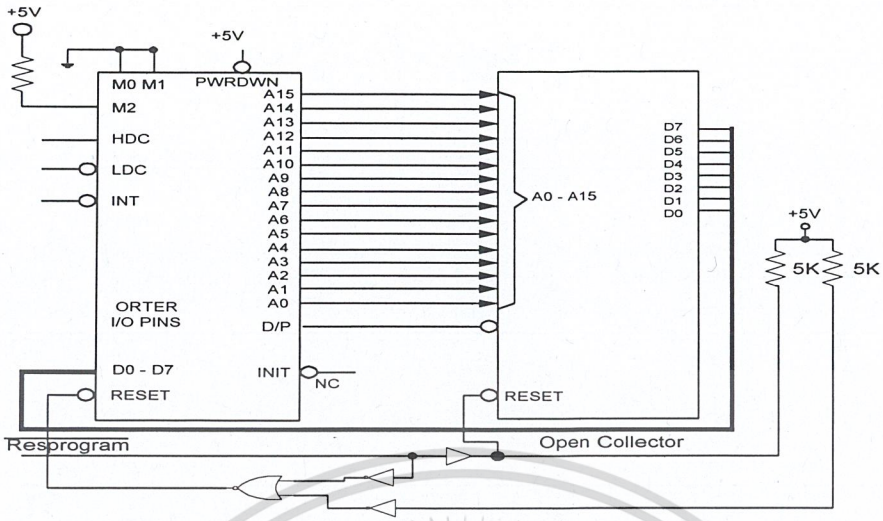
2.10.2 การใช้งานลักษณะมาสเตอร์ซีเรียล

การต่อใช้งานในลักษณะนี้ส่วนที่เก็บโปรแกรมคอนฟิกจะต่างจากการต่อลักษณะสเลฟซีเรียล คือใช้ PROM เบอร์ XC17XXX เป็นตัวเก็บโปรแกรม ทำให้ไม่ต้องเสียเวลาเขียนโปรแกรมเพื่อทำการคอนฟิก ซึ่งวิธีการอัดโปรแกรมคอนฟิกลง PROM ทำตามขั้นตอนดังนี้ คือ เมคบิต (MakeBits) สร้างไฟล์ .BIT จากวงจรที่ออกแบบ และใช้โปรแกรม MakePROM สร้าง Hex ไฟล์แล้วทำการอัดโปรแกรมลง PROM ด้วยอุปกรณ์อัด PROM ที่มาพร้อมกับตัวโปรแกรมของ

ไซลิงค์ PROM XC17XXX จะส่งสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGAs ดังแสดงในรูป 2.27 D0-D7 เป็นขารับข้อมูลที่ใช้ในการคอนฟิกแบบขนาน A0-A15 เป็นแอดเดรสที่ FPGAs สร้างให้กับ EPROM เพื่ออ่านข้อมูลจากหน่วยความจำ มาเก็บไว้ในสแตติกแรม (StaticRAM) แอดเดรสทั้ง 16 เส้น ไม่จำเป็นต้องต่อให้ครบก็ได้ ขึ้นอยู่กับขนาดหน่วยความจำ EPROM ที่ใช้และสามารถกำหนดให้นับขึ้นหรือลงได้



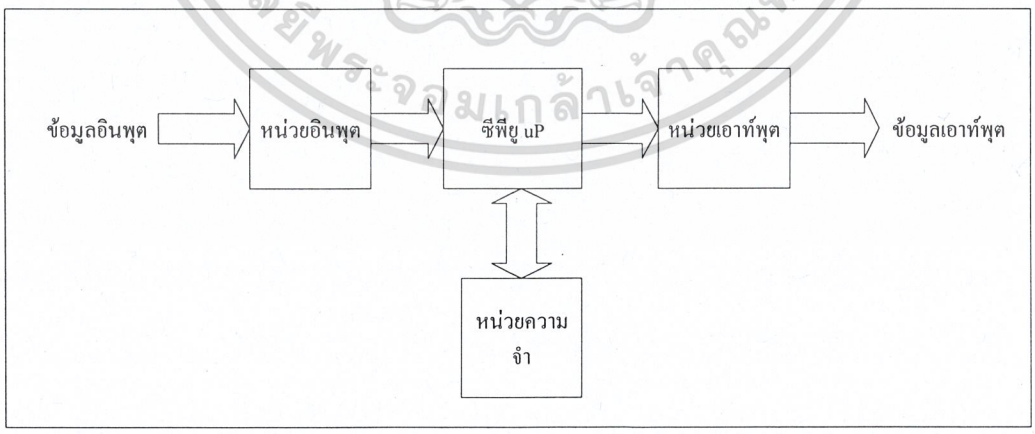
รูปที่ 2.26 การต่อใช้งานในลักษณะมาสเตอร์ซีเรียล



รูปที่ 2.27 การต่อใช้งานในลักษณะมาสเตอร์พาราแรล

2.11 ทฤษฎีไมโครโปรเซสเซอร์

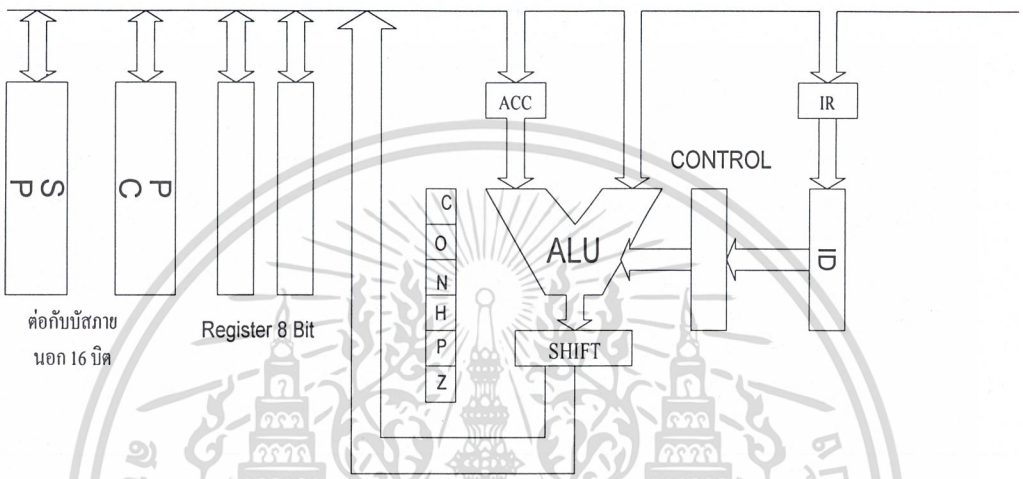
ไมโครโปรเซสเซอร์ คือ อุปกรณ์ที่รวบรวมหน่วยต่าง ๆ ของคอมพิวเตอร์อัน ได้แก่ หน่วยควบคุม (Control Unit) หน่วยความจำบางส่วน หน่วยคำนวณ (ALU) วงจรควบคุม I/O บางส่วน ทั้งหมดนี้รวมไว้ในแผ่นวงจรเดียวกัน ซึ่งสามารถทำหน้าที่ประมวลข้อมูลและควบคุมหน่วยอื่น ๆ ให้ทำงานด้วยกันได้ เช่น ใช้เป็นตัวประมวลผลกลางในคอมพิวเตอร์ เรียกคอมพิวเตอร์นั้นว่า ไมโครคอมพิวเตอร์ (Microcomputer) ดังมีรูปร่างตามแผนผังนี้



รูปที่ 2.28 แผนผังระบบไมโครคอมพิวเตอร์

2.11.1 โครงสร้างภายในไมโครโปรเซสเซอร์

โครงสร้างของไมโครโปรเซสเซอร์ขนาด 8 บิต โดยทั่วไปในปัจจุบันจะมีสถาปัตยกรรมภายในที่ใช้คล้าย ๆ กัน ดังแสดงในรูปที่ 2.29



รูปที่ 2.29 สถาปัตยกรรมภายในของไมโครโปรเซสเซอร์ 8 บิต

ส่วนประกอบภายในที่สำคัญของไมโครโปรเซสเซอร์ประกอบด้วย วงจรคณิตศาสตร์และลอจิก (Arithemetic Logic Unit หรือ ALU) รีจิสเตอร์คำสั่ง (Instruction Regester หรือ IR) วงจรถอดรหัสคำสั่ง (Instruction Decode หรือ ID) หน่วยควบคุม (Cotrol Unit) และรีจิสเตอร์ตำแหน่ง (Address Register) ส่วนประกอบที่สำคัญภายในเหล่านี้จะติดต่อกันด้วยบัสภายใน (Internal Bus) และบัสภายในนี้ต่อกับบัสภายนอกโดยผ่านบัฟเฟอร์ เพื่อเป็นการติดต่อกับวงจรภายนอกอีกทีหนึ่ง ซึ่งทำหน้าที่ของหน่วยต่าง ๆ อธิบายได้ดังนี้

1. หน่วยควบคุม (Control Unit)

หน่วยควบคุมเป็นหน่วยที่ใช้สร้างสัญญาณเพื่อควบคุมการทำงานภายในของไมโครโปรเซสเซอร์ ให้ทำงานอย่างมีระเบียบและสัมพันธ์กันโดยที่หน่วยควบคุมนี้จะรับสัญญาณมาจากวงจรถอดรหัสคำสั่ง ซึ่งจะทำการถอดรหัสคำสั่งจากรีจิสเตอร์คำสั่ง ดังนั้นการทำงานภายในไมโครโปรเซสเซอร์จึงขึ้นอยู่กับคำสั่งที่ป้อนเข้ามาให้กับไมโครโปรเซสเซอร์

2. หน่วยคณิตศาสตร์และลอจิก (Aritric Logic Unit)

หน่วยคณิตศาสตร์และลอจิกหรือเรียกย่อ ๆ ว่าALUหน่วยนี้มีหน้าที่กระทำทางคณิตศาสตร์และลอจิก หน่วย ALU นี้จะใช้รีจิสเตอร์พิเศษตัวหนึ่งเป็นอินพุต ซึ่งเรียกว่า แอคคิวมูเลเตอร์ จะเป็นได้ทั้งอินพุตและเอาต์พุตของ ALU นอกจากนี้ยังสามารถใช้ในการเลื่อน (Shift) และหมุน (Rotate) ข้อมูลได้อีกด้วย

3. รีจิสเตอร์ (Register)

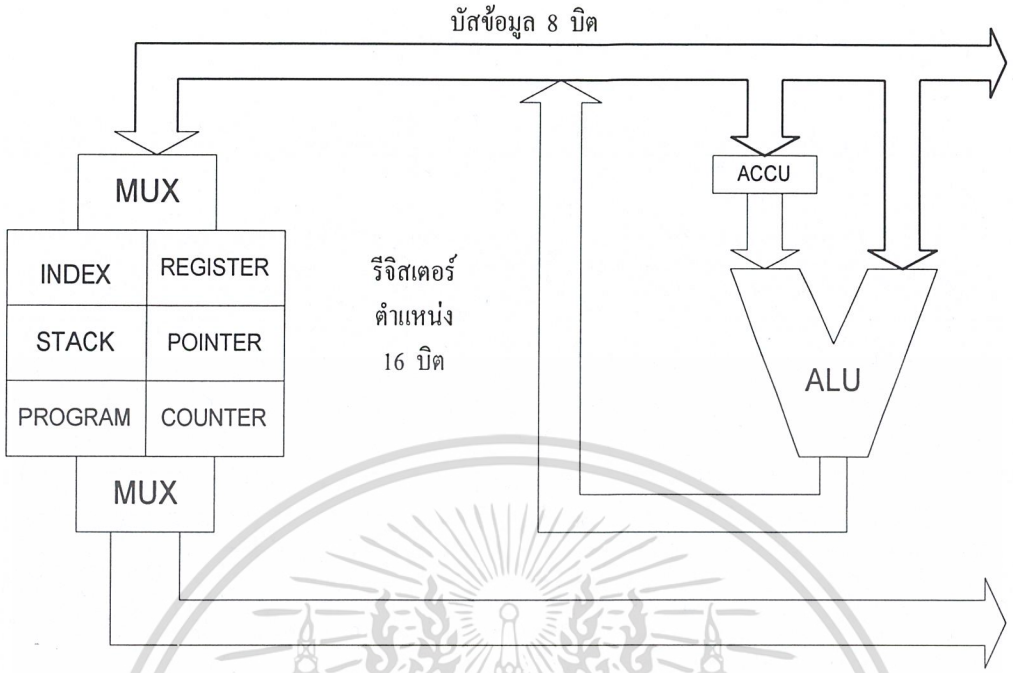
จากสถาปัตยกรรมของไมโครโปรเซสเซอร์ดังรูปที่ 2.29 เราอาจแบ่งรีจิสเตอร์ที่มีอยู่ภายในไมโครโปรเซสเซอร์ออกได้เป็น 2 กลุ่ม คือรีจิสเตอร์เพื่อใช้งานทั่วไป (General purpose register) และรีจิสเตอร์สำหรับการอ้างตำแหน่ง (Address register) ซึ่งรีจิสเตอร์ทั้ง 2 อย่างเราสามารถอธิบายการทำงานได้ดังนี้

- รีจิสเตอร์ใช้งานทั่วไป

ในไมโครโปรเซสเซอร์ขนาด 8 บิต รีจิสเตอร์นี้จะมีขนาด 8 บิต หน้าที่ของรีจิสเตอร์เหล่านี้ไม่ได้ถูกกำหนดเฉพาะเจาะจงลงไป แต่โดยทั่วไปจะใช้สำหรับเก็บข้อมูลเพื่อให้ ALU กระทำกับข้อมูลต่าง ๆ เหล่านั้นด้วยความเร็วสูง เนื่องจากไม่ต้องติดต่อกับหน่วยความจำที่อยู่ภายนอก นอกจากนี้ไมโครโปรเซสเซอร์บางตัว ยังสามารถนำรีจิสเตอร์ 2 ตัวมาต่อร่วมกันได้ เรียกว่า คู่รีจิสเตอร์ (Register pair) โดยที่คู่รีจิสเตอร์นี้จะกลายเป็นรีจิสเตอร์ขนาด 16 บิต หรือใช้เป็นตัวชี้ตำแหน่งข้อมูลในหน่วยความจำก็ได้ ซึ่งจะขึ้นอยู่กับคำสั่งที่ใช้

- รีจิสเตอร์สำหรับการอ้างตำแหน่ง

หน้าที่ของรีจิสเตอร์นี้ ใช้สำหรับเก็บตำแหน่งของหน่วยความจำที่ต้องการอ้างถึง ซึ่งอาจเป็นการอ้างถึงโดยคำสั่ง หรือการอ้างถึงโดยระบบก็ตาม ขนาดของรีจิสเตอร์นี้อาจเป็น 8 บิต หรือ 16 บิต ก็ได้ ขึ้นอยู่กับชนิดของไมโครโปรเซสเซอร์นั้น ๆ บางครั้งอาจเรียกรีจิสเตอร์เหล่านี้ว่า Data Counter หรือ Pointer ในไมโครโปรเซสเซอร์ทั่ว ๆ ไป ควรจะมีรีจิสเตอร์สำหรับการอ้างตำแหน่งอย่างน้อย 2 ตัว คือ โปรแกรมเคาท์เตอร์ (PC) และแสตคพอยเตอร์ (SP) ส่วนรีจิสเตอร์ตัวอื่น ๆ เช่น อินเด็กรีจิสเตอร์ (IX) อาจจะมีหรือไม่ก็ได้ รีจิสเตอร์นี้จะต่อไว้กับบัสตำแหน่งดังแสดงไว้ในรูป เอาต์พุตของรีจิสเตอร์เหล่านี้จะต่อไว้กับบัสตำแหน่งโดยมีตัวเลือกข้อมูล (Muliplexer) เพื่อทำหน้าที่เลือกว่าจะนำข้อมูลมาจากรีจิสเตอร์ใดเพื่อไปกำหนดตำแหน่งที่ต้องการ



รูปที่ 2.30 รีจิสเตอร์ตำแหน่งขนาด 16 บิต ที่ใช้สร้างข้อมูลบนบัสตำแหน่ง

2.11.2 หน้าที่เฉพาะของรีจิสเตอร์สำหรับการอ้างตำแหน่งต่าง ๆ

1. โปรแกรมเคาท์เตอร์ (Program Coumter หรือ PC) โปรแกรมเคาท์เตอร์จะต้องมีอยู่ในทุกโปรเซสเซอร์ ข้อมูลในโปรแกรมเคาท์เตอร์คือ ตำแหน่งของคำสั่งต่อไปที่โปรเซสเซอร์จะต้องอ่านเพื่อปฏิบัติ กลไกการปฏิบัติตามรหัสคำสั่ง และลำดับขั้นของการทำงานตามคำสั่งที่วางไว้จะกำหนดโดยข้อมูลที่อยู่ในโปรแกรมเคาท์เตอร์นี้เอง กล่าวโดยสรุปคือการปฏิบัติโปรแกรมจะเป็นไปแบบเรียงลำดับ และเพื่อที่จะดึงคำสั่งต่อไป โปรเซสเซอร์จำเป็นต้องดึงคำสั่งมาจากหน่วยความจำ ซึ่งกระบวนการนี้ ข้อมูลในโปรแกรมเคาท์เตอร์จะส่งมาบนบัสตำแหน่ง และส่งต่อไปยังหน่วยความจำ หน่วยความจำจะอ่านข้อมูลจาตำแหน่งที่ถูกอ้างถึง และส่งข้อมูลที่อ่านได้ไปบนบัสข้อมูล โปรเซสเซอร์จะอ่านข้อมูลบนบัสข้อมูลนั้น ซึ่งข้อมูลหรือคำที่อ่านได้คือคำสั่ง Operation Code

2. สแตคพอยท์เตอร์ (Stack Pointer หรือ SP) ไมโครโปรเซสเซอร์ที่มีความสามรถสูง โดยทั่วไปจะต้องมีสแตค ซึ่งสแตคอาจเป็นรีจิสเตอร์ที่อยู่ในโปรเซสเซอร์เอง หรือใช้หน่วยความจำภายนอกส่วนหนึ่ง เพื่อกำหนดให้เป็นสแตคในการเก็บรักษาตำแหน่งสูงสุดของสแตคที่อยู่ในหน่วยความจำ จะใช้รีจิสเตอร์ที่เรียกว่า สแตคพอยท์เตอร์ (SP) สแตคและสแตคพอยท์เตอร์เป็นสิ่งที่จำเป็นและขาดไม่ได้ในการทำโปรแกรมย่อย (Subroutine) และการทำโปรแกรมเกี่ยวกับการอินเตอร์รัพต์

3. อินเด็กรีจิสเตอร์ (Index Register หรือ IX) อินเด็กรีจิสเตอร์ เป็นรีจิสเตอร์สำหรับการอ้างตำแหน่งอีกตัวหนึ่งที่มีในไมโครโปรเซสเซอร์บางชนิดเท่านั้น และใช้กับคำสั่งที่มีการเข้าถึงข้อมูลแบบอินเด็กซ์ (Index Addressing mode) ซึ่งอินเด็กรีจิสเตอร์นี้จะทำให้การเข้าถึงหน่วยความจำเป็นกลุ่มได้โดยสะดวก ข้อมูลในอินเด็กรีจิสเตอร์อาจเป็นค่าระยะห่าง (Displacement) ซึ่งจะนำไปบวกกับค่าตำแหน่งฐาน (Base addres) เพื่อใช้ชี้ตำแหน่งของหน่วยความจำที่ต้องการอ้างอิง หรือข้อมูลในอินเด็กรีจิสเตอร์อาจเป็นค่าตำแหน่งฐานเพื่อนำไปบวกกับค่าระยะห่างที่กำหนดมากับคำสั่งก็ได้ ทั้งนี้แล้วแต่ไมโครโปรเซสเซอร์ตัวนั้น ๆ

2.11.3 สัญญาณติดต่อกับหน่วยความจำ

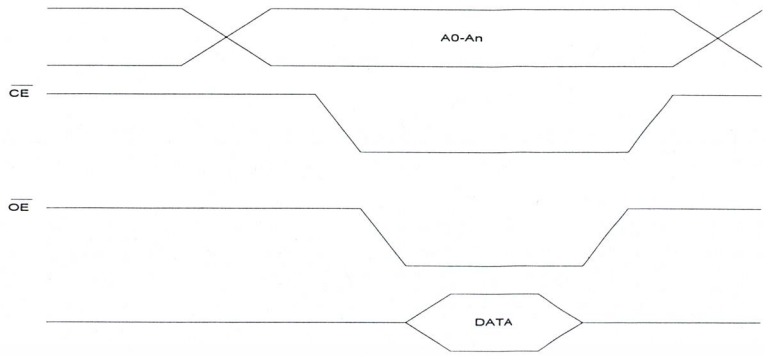
การเชื่อมต่อกับหน่วยความจำจะประกอบด้วยสัญญาณที่จำเป็น 4 กลุ่มคือ

1. ขาสัญญาณไฟเลี้ยง
2. ขาบัสแอดเดรส
3. ขาบัสข้อมูล
4. ขาบัสควบคุม

การนำข้อมูลที่จัดเก็บอยู่ภายในหน่วยความจำออกมาใช้งานนั้น จะเรียกว่าเป็น การอ่าน (Read) และการนำข้อมูลเข้าไปเก็บไว้ในหน่วยความจำ เรียกว่า การเขียน (Write) กระบวนการนำข้อมูลออกมาจากหน่วยความจำและนำข้อมูลเข้าไปเก็บไว้ในหน่วยความจำนั้น จะกระทำอยู่ภายในช่วงเวลาหนึ่งมารวมเรียกว่า รอบเวลาการอ่าน (Read Cycle) และรอบเวลาการเขียน (Write Cycle) ซึ่งก็คือ ช่วงเวลาหนึ่งที่ใช้ในการอ่านและเขียนข้อมูลภายในหน่วยความจำ

ลำดับสัญญาณการติดต่อเพื่ออ่านข้อมูลจาก RAM

1. ไมโครคอนโทรลเลอร์ส่งค่าแอดเดรสที่ต้องการออกมาทางบัสแอดเดรส
2. สัญญาณ CE เป็นลอจิกต่ำเพื่อเลือกให้ RAM ทำงาน
3. สัญญาณ OE หรือ RD เป็นลอจิกต่ำ เพื่อระบุว่าต้องการอ่านข้อมูลภายใน RAM
4. ไมโครคอนโทรลเลอร์หยุดรอในช่วงระยะเวลาหนึ่ง เพื่อให้วงจรภายใน RAM ทำการถอดรหัสแอดเดรสและอ่านข้อมูล
5. ข้อมูลถูกส่งออกมาที่บัสข้อมูล ไมโครคอนโทรลเลอร์สามารถอ่านข้อมูลนี้ไปประมวลผลต่อไป ช่วงต่อมาสัญญาณ OE กลับเป็นระดับลอจิกสูง เช่นเดิม เพื่อเป็นการทำให้สภาวะขาสัญญาณบัสข้อมูลมีอิมพีแดนซ์สูง และไม่มีผลต่อการทำงานอื่น ๆ ที่เกี่ยวข้องกับบัสข้อมูลของระบบ



รูปที่ 2.31 ลำดับสัญญาณการติดต่อเพื่ออ่านข้อมูลจาก RAM

ลำดับสัญญาณการติดต่อเพื่อเขียนข้อมูลลง RAM

1. ไมโครคอนโทรลเลอร์ส่งค่าแอดเดรสที่ต้องการ ออกมาทางบัสแอดเดรส
2. ส่งข้อมูลที่ต้องการเขียนไปบนบัสข้อมูล
3. สัญญาณ CE เป็นลอจิกต่ำ เพื่อเลือกให้ RAM ทำงาน
4. ไมโครคอนโทรลเลอร์หยุดรอในช่วงระยะเวลาหนึ่ง เพื่อให้วงจรภายใน RAM ทำการถอดรหัสแอดเดรสและเตรียมการเขียนข้อมูล
5. สัญญาณ WR เป็นลอจิกต่ำ เพื่อสั่งงานให้มีการนำข้อมูลในบัสข้อมูลเขียนลงหน่วยความจำ
6. ช่วงต่อมา สัญญาณ WR กลับเป็นระดับลอจิกสูง เช่นเดิม เพื่อเป็นการทำให้สภาวะขาสัญญาณบัสข้อมูลมีอิมพีแดนซ์สูง และไม่มีผลต่อการทำงานอื่น ๆ ที่เกี่ยวข้องกับบัสข้อมูลของระบบ

รูปที่ 2.32 ลำดับสัญญาณการติดต่อเพื่อเขียนข้อมูลลง RAM

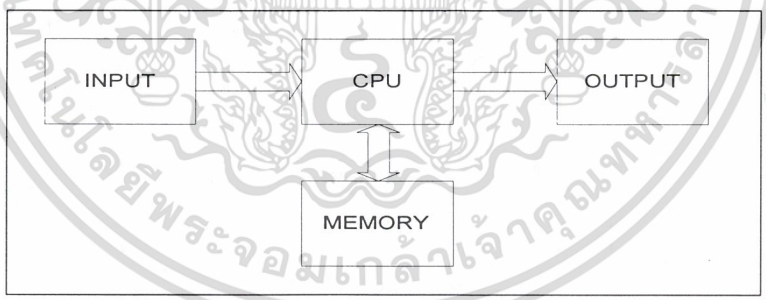
บทที่ 3
การออกแบบและการสร้าง

3.1 ลักษณะการออกแบบ

การออกแบบและการสร้างหน่วยประมวลผลกลางขนาด 4 บิต โครงงานนี้ได้ทำการเลือกการออกแบบจากบนลงล่าง โดยวิธีการกำหนดส่วนย่อยแต่ละส่วนแล้วทดลองการทำงานของส่วนย่อยนั้น ๆ จนเป็นที่น่าพอใจแล้วค่อยนำส่วนย่อยที่เราได้ทำการทดลองเหล่านั้นมารวมให้เป็นส่วนใหญ่ ซึ่งการทำงานในลักษณะนี้ทำให้ไม่มีโอกาสที่รูปแบบการทำงานจะผิดไปจากวัตถุประสงค์ เพราะสามารถจำลองการทำงานของวงจรที่ออกแบบ จนแน่ใจว่าทำงานได้ตามวัตถุประสงค์ที่ต้องการ และเมื่อเกิดข้อผิดพลาดก็สามารถแก้ไขได้ทันที หลังจากทำการตรวจสอบจนเป็นที่น่าพอใจแล้ว จึงนำโปรแกรมที่ได้ไปลงบนอุปกรณ์ FPGAs

3.2 โครงสร้างของระบบไมโครคอมพิวเตอร์

โครงสร้างระบบไมโครคอมพิวเตอร์แสดงได้ดังรูปที่ 3.1



รูปที่ 3.1 โครงสร้างของระบบไมโครคอมพิวเตอร์

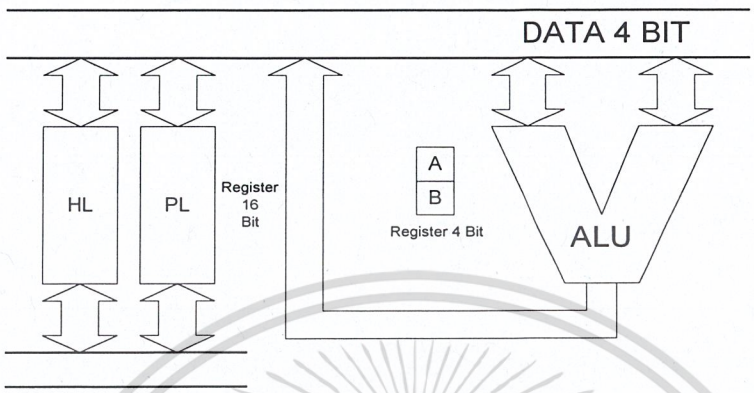
3.2.1 คุณสมบัติหน่วยประมวลผลกลาง

หน่วยประมวลผลกลางขนาด 4 บิต มีคุณสมบัติดังนี้

- มี DATA BUS ขนาด 4 บิต
- มี ADDRESS BUS ขนาด 16 บิต
- กระทำคำสั่งต่าง ๆ ได้ 16 คำสั่ง
- ใช้สัญญาณนาฬิกาขนาด 2 เมกะเฮริตซ์

3.2.2 โครงสร้างของหน่วยประมวลผลกลาง

โครงสร้างของหน่วยประมวลผลกลางที่ได้ออกแบบไว้ ดังแสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 โครงสร้างหน่วยประมวลผลกลาง

จากรูปที่ 3.2 จะเห็นได้ว่าโครงสร้างภายในของหน่วยประมวลผลกลาง ประกอบด้วยหน่วยประมวลทางคณิตศาสตร์และลอจิก (ALU) รีจิสเตอร์ขนาด 4 บิต และรีจิเตอร์ขนาด 16 บิต ซึ่งแต่ละส่วนมีหน้าที่ต่าง ๆ ดังนี้

1. หน่วยประมวลผลทางคณิตศาสตร์และลอจิก

มีหน้าที่ในการคำนวณทางคณิตศาสตร์ เช่น การบวกและลบเลขขนาด 4 บิต และการกระทำทางลอจิก เช่น การ AND หรือ การ OR กัน

2. รีจิสเตอร์ขนาด 4 บิต

หน่วยประมวลผลกลางมีรีจิสเตอร์ใช้งานขนาด 4 บิต อยู่ 2 ตัวคือ

- A เป็นแอคคิวมูเลเตอร์รีจิสเตอร์
- B เป็นรีจิสเตอร์ใช้งานขนาด 4 บิต

3. รีจิสเตอร์ขนาด 16 บิต

หน่วยประมวลผลกลางมีรีจิสเตอร์ใช้งานขนาด 16 บิต อยู่ 2 ตัวคือ

- PC เป็นรีจิสเตอร์ที่เก็บตำแหน่งของคำสั่งต่อไปที่หน่วยประมวลผลกลางจะทำงานต่อ
- HL เป็นรีจิสเตอร์ที่เก็บตำแหน่งขนาด 16 บิต

3.2.3 รายละเอียดของคำสั่ง

โครงงานนี้ได้จัดแบ่งกลุ่มของคำสั่งต่าง ๆ ไว้ดังนี้

- กลุ่มคำสั่งโอนย้ายข้อมูล

  LOAD

- กลุ่มคำสั่งคณิตศาสตร์และลอจิก

  ADD , SUB , AND , OR , INC

- กลุ่มคำสั่งการกระโดด

  JP , DJNZ

- กลุ่มคำสั่งรับส่งข้อมูลกับอุปกรณ์ภายนอก

  IN , OUT

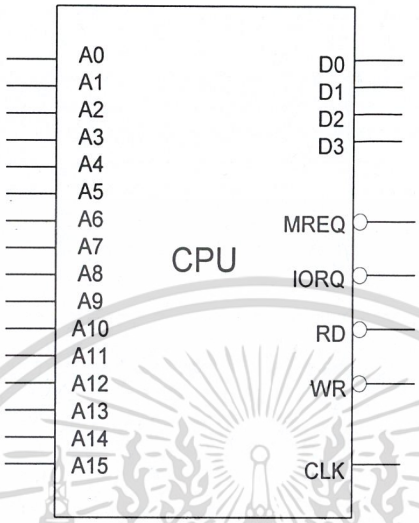
กลุ่มของคำสั่งที่ได้ทำการแบ่งไว้นั้น สามารถนำมาแจกแจงรายละเอียดของคำสั่งต่าง ๆ ตามกลุ่มที่ได้แบ่งไว้ ดังแสดงได้ตามตารางที่ 3.1

ตารางที่ 3.1 รายละเอียดคำสั่งต่าง ๆ

คำสั่ง	Op-Code	การทำงาน
NOP	0H	หยุดการทำงาน
JP	1H	กระโดดไปตำแหน่งที่ HL เก็บค่าไว้
LD A,B	2H	นำข้อมูลที่ B ไปเก็บไว้ที่ A
LD B,A	3H	นำข้อมูลที่ A ไปเก็บไว้ที่ B
LD A,(HL)	4H	นำข้อมูลที่ HL ไปเก็บไว้ที่ A
LD (HL),A	5H	นำข้อมูลที่ A ไปเก็บไว้ที่ HL
LD A,n	6nH	นำข้อมูลไปเก็บไว้ที่ A
LD HL,nnnn	7nnnnH	นำข้อมูลที่ได่ไปเป็น Addressที่ HL ชี้
ADD	8H	เป็นคำสั่งบวก
SUB	9H	เป็นคำสั่งลบ
AND	0AH	การนำข้อมูลมา AND กับข้อมูลที่ A
OR	0BH	การนำข้อมูลมา OR กับข้อมูลที่ A
INC	0CH	การเพิ่มค่าที่ A
DJNZ	0DnnnnH	กระโดดเมื่อ HL ไม่เท่ากับ “0”
IN	0EH	การนำข้อมูลเข้าที่ A ณ ตำแหน่งของ HL
OUT	0FH	การนำข้อมูลออกที่ A ณ ตำแหน่งของ HL

3.2.4 หน้าที่การทำงานของขาสัญญาณแต่ละขา

ขาสัญญาณต่าง ๆ ในหน่วยประมวลผลกลางดังแสดงได้ในรูปที่ 3.3



รูปที่ 3.3 ขาใช้งานของหน่วยประมวลผลกลาง

ขา CLK เป็นขาอินพุททำงานที่ขอบขาขึ้น ทำหน้าที่รับสัญญาณฐานเวลาให้แก่วงจรของหน่วยประมวลผลกลางขนาด 4 บิต

ขา $A_{15} - A_0$ เป็นขาสัญญาณกำหนดตำแหน่ง (Address Bus) โดยที่ A_0 เป็นบิตทางด้านต่ำ (LSB) และ A_{15} เป็นบิตทางด้านสูง (MSB) จะทำงานที่สภาวะ "1" บัสนี้มีด้วยกัน 16 เส้น สามารถติดต่อกับหน่วยความจำได้ $2^{16} = 65,536$ ตำแหน่ง (64 Kbytes)

ขา $D_4 - D_0$ เป็นขาสัญญาณข้อมูล (Data Bus) D_0 เป็นบิตทางด้านต่ำ และ D_4 เป็นบิตทางด้านสูง ลักษณะเป็นบัสแบบสองทิศทาง ขนาด 4 บิต ทำงานที่สภาวะ "1" ใช้เป็นเส้นทางผ่านของข้อมูลระหว่างไมโครโปรเซสเซอร์กับหน่วยความจำหรืออุปกรณ์อินพุต/เอาท์พุตต่าง ๆ

ขา MREQ (Memory Reguest) ทำงานที่สภาวะ "0" เมื่อสายสัญญาณนี้ทำงาน แสดงว่าขณะนี้ไมโครโปรเซสเซอร์ต้องการติดต่อกับหน่วยความจำเพื่ออ่านหรือเขียนข้อมูลโดยที่ตำแหน่งของหน่วยความจำจะปรากฏอยู่บนบัสแอดเดรสแล้ว

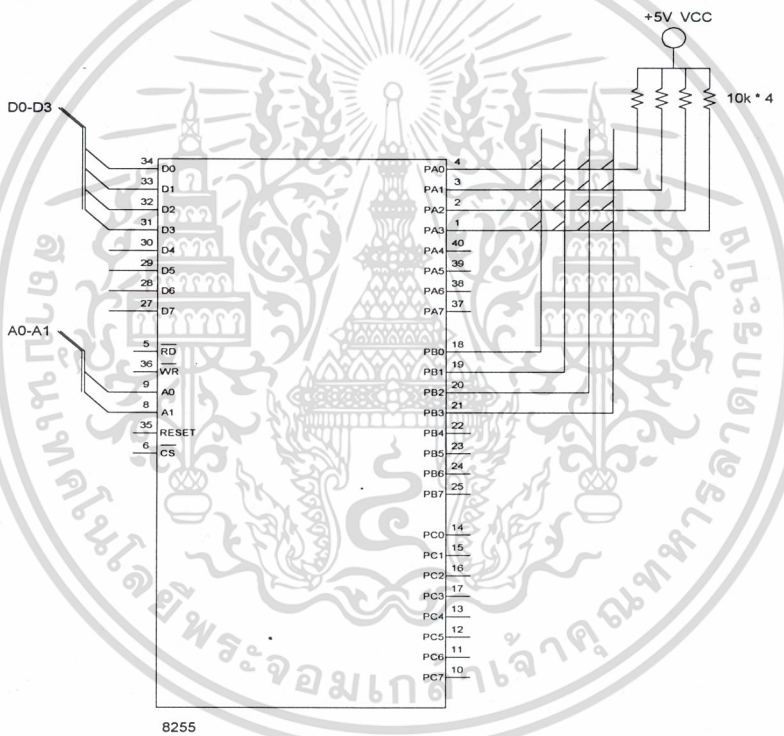
ขา IORQ (Input/Output Reguest) ทำงานที่สภาวะ "0" เมื่อสายสัญญาณนี้ทำงาน แสดงว่าขณะนี้อุปกรณ์อินพุต/เอาต์พุต ต้องการติดต่อกับไมโครโปรเซสเซอร์หรือหน่วยความจำเพื่ออ่านหรือเขียนข้อมูล

ขา RD (Memory Read) เป็นขาเอาต์พุต ทำงานที่สภาวะ “0” สัญญาณนี้ทำงานขณะไมโครโปรเซสเซอร์ต้องการอ่านข้อมูลจากหน่วยความจำ หรือจากอุปกรณ์อินพุต/เอาต์พุต

ขา WR (Memory Write) เป็นขาเอาต์พุต ทำงานที่สภาวะ “0” เมื่อสัญญาณนี้แอคตีฟ แสดงว่าไมโครโปรเซสเซอร์ต้องการเขียนข้อมูลเข้าหน่วยความจำหรือเข้าอุปกรณ์อินพุต/เอาต์พุต

3.3 ส่วนรับข้อมูล (Input)

ส่วนรับข้อมูลจะทำหน้าที่รับข้อมูลมาจากเมตริกซ์สวิตซ์ โดยตรวจสอบการกดคีย์ผ่านทาง 8255 ว่าคีย์ที่ทำการกดเป็นคีย์ใด และได้กำหนดคีย์ที่ใช้กดไว้ทั้งหมด 16 คีย์ คือ 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , A , B , C , D , E , F ดังแสดงในรูปที่ 3.4

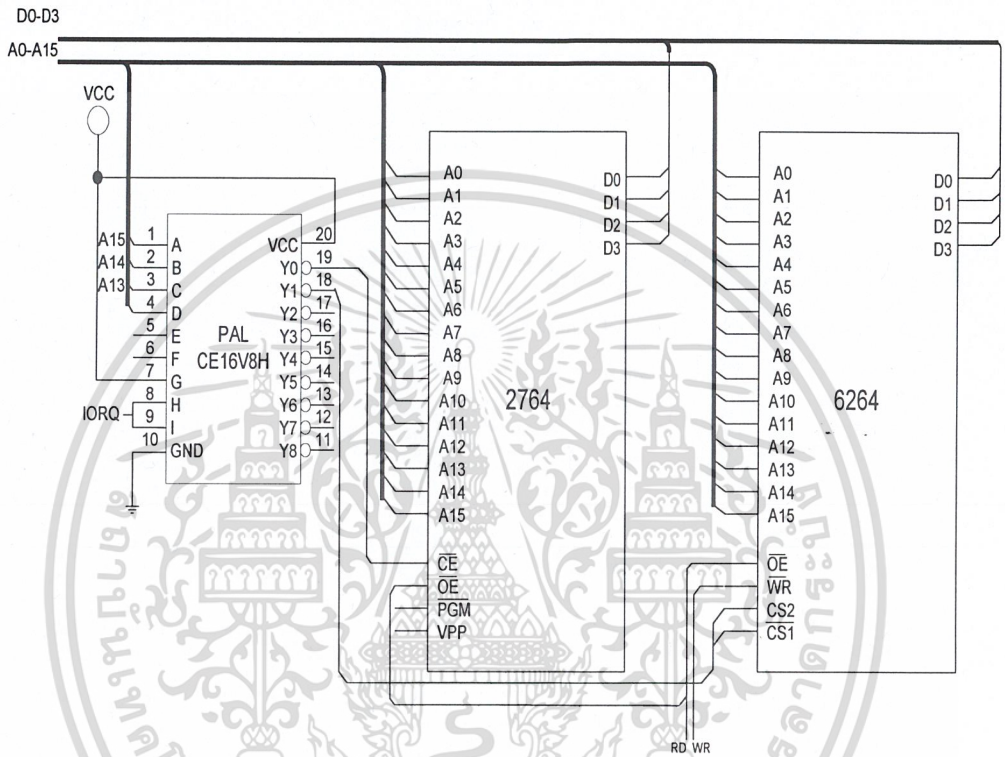


รูปที่ 3.4 วงจรส่วนรับข้อมูล

3.4 หน่วยความจำ (Memory Unit)

หน่วยความจำสามารถแยกออกเป็น หน่วยความจำภายในและหน่วยความจำภายนอก ซึ่งหน่วยความจำภายในออกแบบโดยใช้ EPROM เบอร์ 2764 มีความจุ 8 กิโลไบต์ ทำหน้าที่เก็บส่วนของคำสั่งภายในหน่วยประมวลผลกลาง และหน่วยความจำภายนอกใช้ RAM เบอร์ 6264 มีความจุ 8 กิโลไบต์

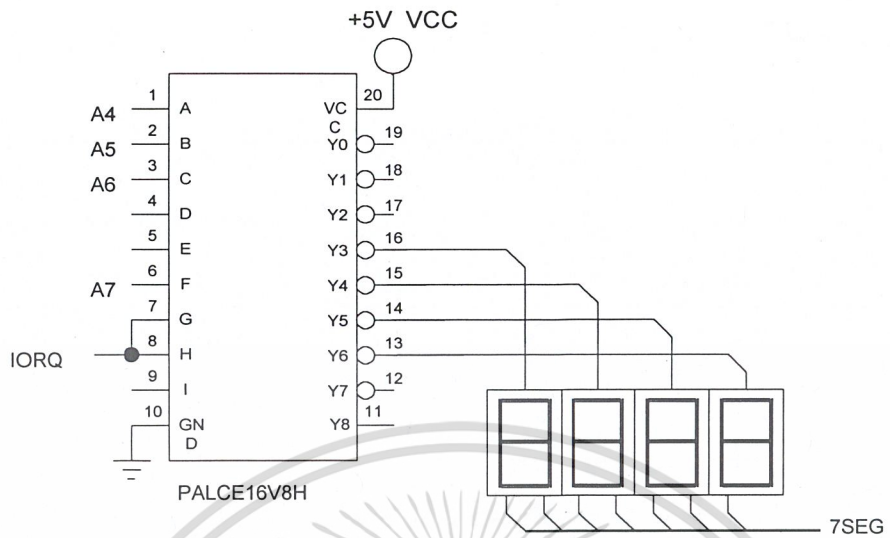
การต่อใช้งาน EPROM และ RAM นั้น จะต้องใช้สัญญาณที่ได้มาจากการดีโค๊ดแอดเดรสของ EPROM และ RAM โดยกำหนดให้แอดเดรสของ EPROM คือ 0000H-1FFFH แอดเดรสของ RAM คือ 2000H-3FFFH เมื่อดีโค๊ดได้แล้วได้เลือกใช้ PAL ที่โปรแกรมเป็น IC74LS138 เพื่อเลือกขาที่จะไปต่อใช้งานดังแสดงในรูปที่ 3.5



รูปที่ 3.5 การต่อ EPROM และ RAM

3.5 หน่วยแสดงผล (Output)

หน่วยแสดงผลจะใช้หน่วยแสดงผลเจ็ดส่วน(7-segment) จำนวน 4 หลัก เป็นตัวแสดงผลโดยการดีโค๊ดแอดเดรสผ่านโดยใช้ PAL ซึ่งได้โปรแกรมให้ PAL ทำหน้าที่เป็นตัวเลือกตำแหน่งให้หน่วยแสดงผลเจ็ดส่วน แสดงผลดังแสดงไว้ในรูปที่ 3.6



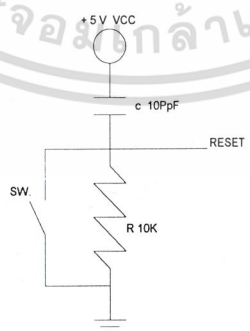
รูปที่ 3.6 วงจรภาคแสดงผลหน่วยแสดงผลเจ็ดส่วน

3.6 ส่วนประกอบร่วมของวงจร

เพื่อให้วงจรสามารถทำงานได้สมบูรณ์ขึ้นจึงต้องมีส่วนประกอบต่าง ๆ ดังนี้

3.6.1 วงจรรีเซ็ต

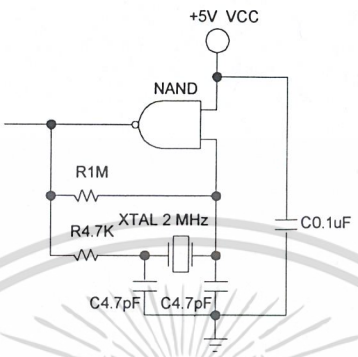
วงจรรีเซ็ตใช้ในการรีเซ็ตระบบไมโครโปรเซสเซอร์ โดยทำการรีเซ็ตที่ 8255 และตัว CPU เอง ให้พร้อมที่จะเริ่มทำงาน โดยเมื่อกดรีเซ็ตจะส่งสภาวะ "0" ไปทำการรีเซ็ตระบบ วงจรรีเซ็ตแสดงได้ดังรูปที่ 3.7



รูปที่ 3.7 วงจรรีเซ็ต

3.6.2 วงจรกำเนิดสัญญาณนาฬิกา

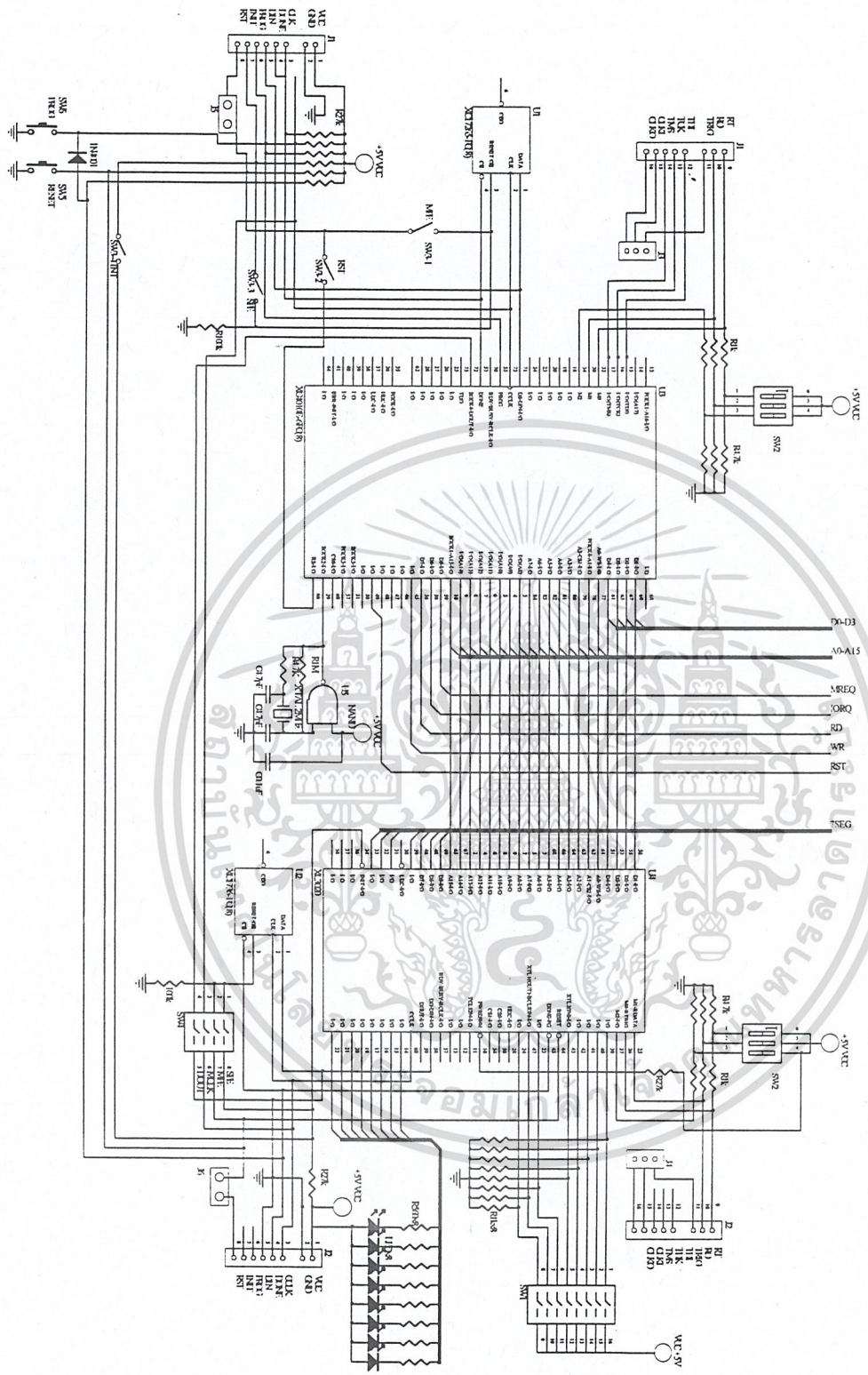
วงจรกำเนิดสัญญาณนาฬิกาใช้กำเนิดสัญญาณนาฬิกาที่ความถี่ 2 MHz โดยใช้คริสตรัลความถี่ 2 MHz และอุปกรณ์อื่นเป็นตัวช่วยกำเนิดสัญญาณนาฬิกา ตามรูปที่ 3.8



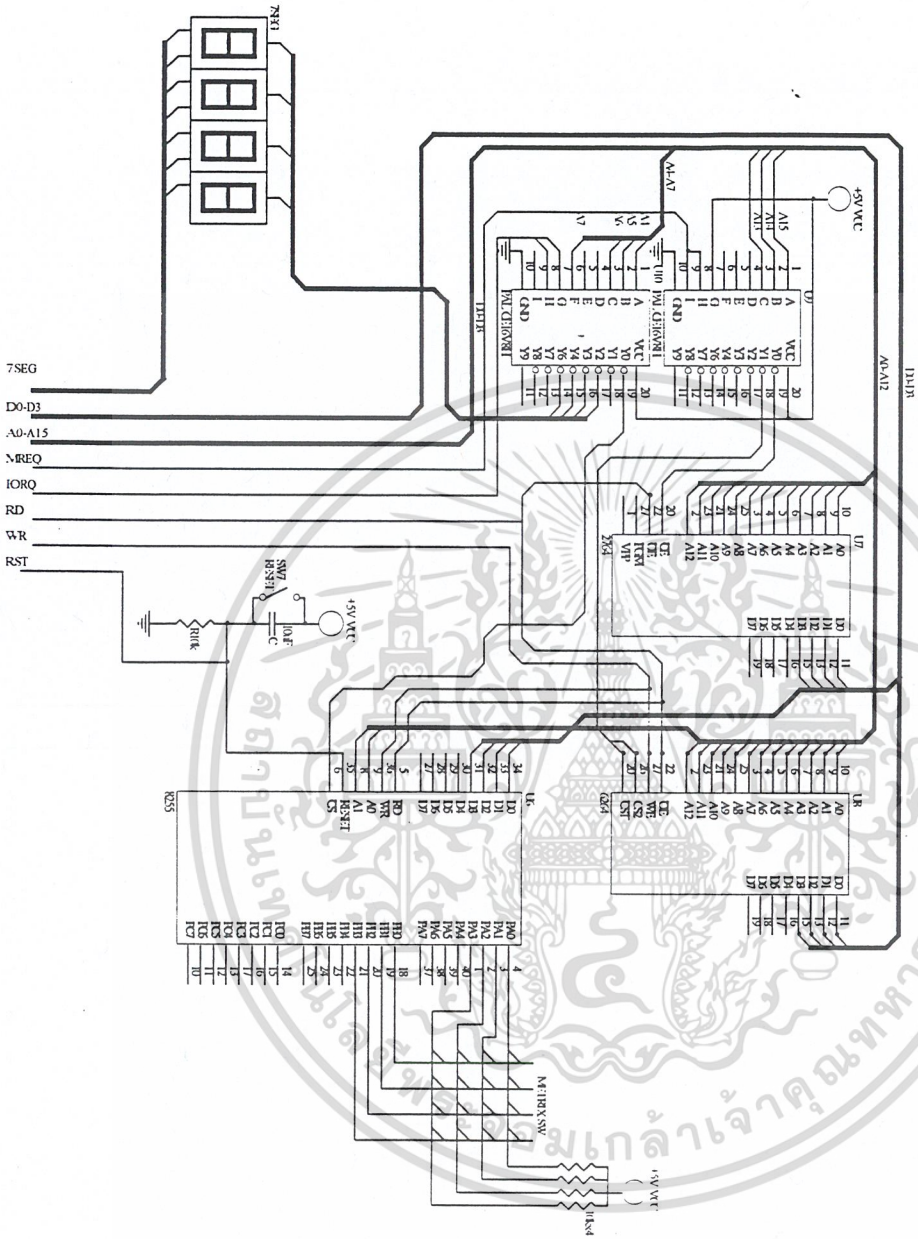
รูปที่ 3.8 วงจรกำเนิดสัญญาณนาฬิกา

3.7 วงจรประยุกต์ใช้งาน

จากวงจรที่ได้ออกแบบมาแล้วทั้งหมดข้างต้น สามารถนำมารวม และสร้างเป็นวงจรใหม่ดังรูปที่ 3.9 เพื่อนำมาประยุกต์ใช้งานได้หลายรูปแบบ

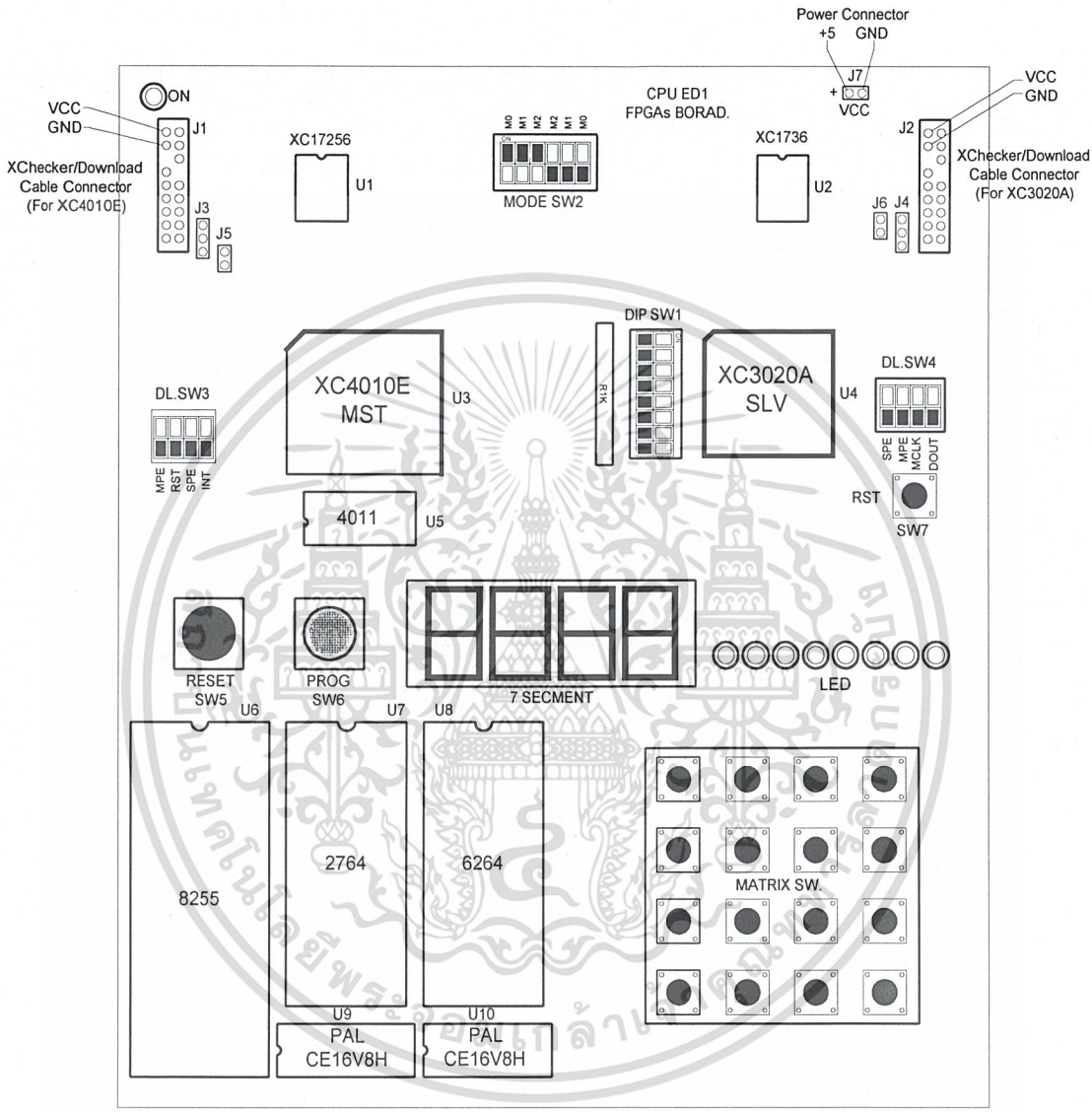


รูปที่ 3.9 วงจรประยุกต์ใช้งาน



รูปที่ 3.9 (ต่อ) วงจรประยุกต์ใช้งาน

จากวงจรที่ได้ออกแบบไว้ในรูปที่ 3.9 ได้ออกแบบและกำหนดตำแหน่งการวางอุปกรณ์บนบอร์ด CPU ED1 ดังรูปที่ 3.10



รูปที่ 3.10 ตำแหน่งการวางอุปกรณ์บนบอร์ด CPU ED1

3.8 ลำดับการสร้างและทดลอง

3.8.1 การเขียนโปรแกรมด้วยภาษา VHDL

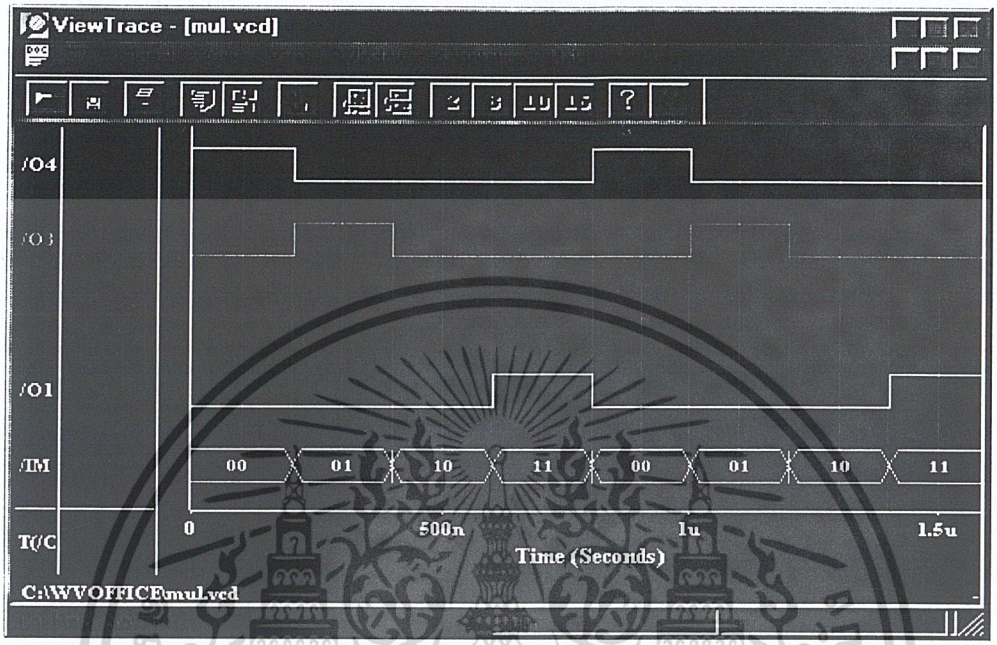
ขอยกตัวอย่างโปรแกรมเพียงบางส่วน เช่น โปรแกรม VHDL อธิบายการมัลติเพล็ก

```
-------------------------
--  Multiplex 2 lines to 4  --
-------------------------
library IEEE;
use IEEE.Std_logic_1164.all;
entity i_mul is
port(im : in std_logic_vector( 0 to 1 );
        o1,o2,o3,o4 : out std_logic );
end i_mul;
architecture a_i_mul of i_mul is
begin
        process( im )
begin
        if( im = "00" ) then
                o1 <= '0';  o2 <= '0';  o3 <= '0' ;  o4 <= '1' ;
        elsif( im = "01" ) then
        o1 <= '0';  o2 <= '0';  o3 <= '1' ;  o4 <= '0' ;
        elsif( im = "10" ) then
                o1 <= '0';  o2 <= '1';  o3 <= '0' ;  o4 <= '0' ;
        elsif( im = "11" ) then
                o1 <= '1';  o2 <= '0';  o3 <= '0' ;  o4 <= '0' ;
        end if;
end process;
end a_i_mul;
```

3.8.2 การจำลองการทำงาน

หลังจากที่เขียนโปรแกรมหน่วยประมวลผลกลางเสร็จเรียบร้อยแล้ว และได้ทำการแปลงภาษาจนไม่พบข้อผิดพลาดขั้นต้น ในขั้นต่อมาก็จะเป็นการทดสอบการทำงานของโปรแกรมที่เขียนขึ้นมาเพื่อดูว่าการทำงานเป็นไปตามวัตถุประสงค์ที่ได้ออกแบบไว้หรือไม่ ในการจำลองการ

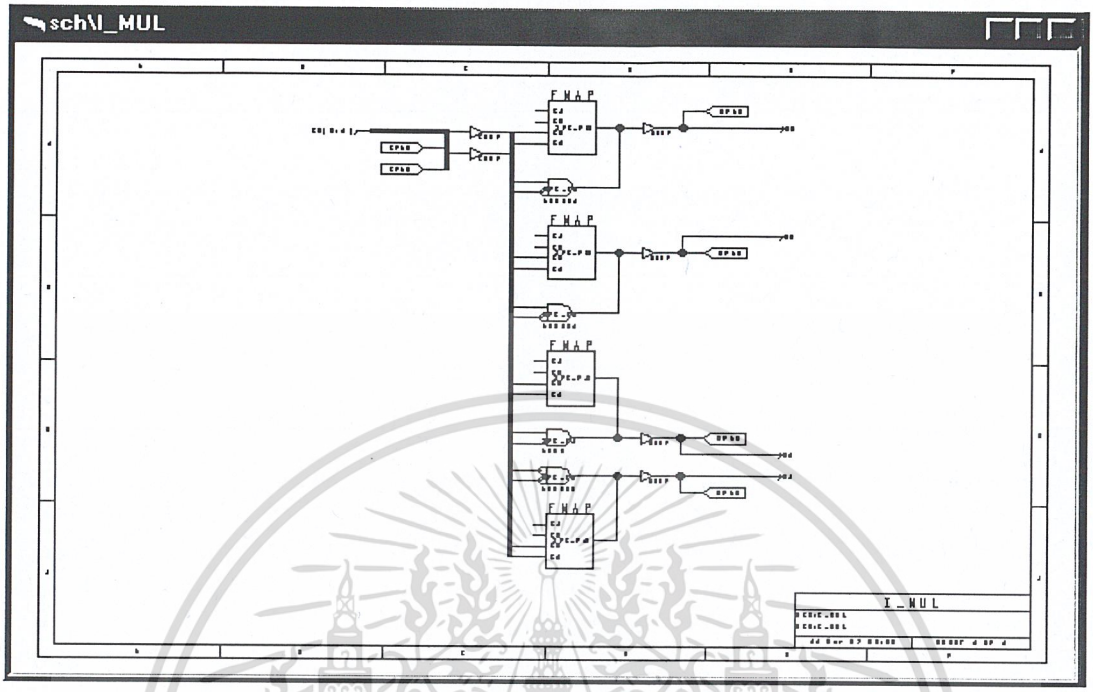
ทำงานของโปรแกรมจะต้องสร้างสัญญาณทดสอบวงจรในทุกสภาวะ เพื่อให้แน่ใจว่าผลลัพธ์ที่ได้นั้นตรงตามที่ได้ออกแบบไว้ โดยการใช้โปรแกรมวิวเทรซเป็นตัวแสดงผลของสัญญาณต่าง ๆ ดังแสดงได้ดังรูปที่ 3.11



รูปที่ 3.11 การจำลองการทำงานของการ Mul

3.8.3 การสังเคราะห์เป็นวงจรระดับเกท

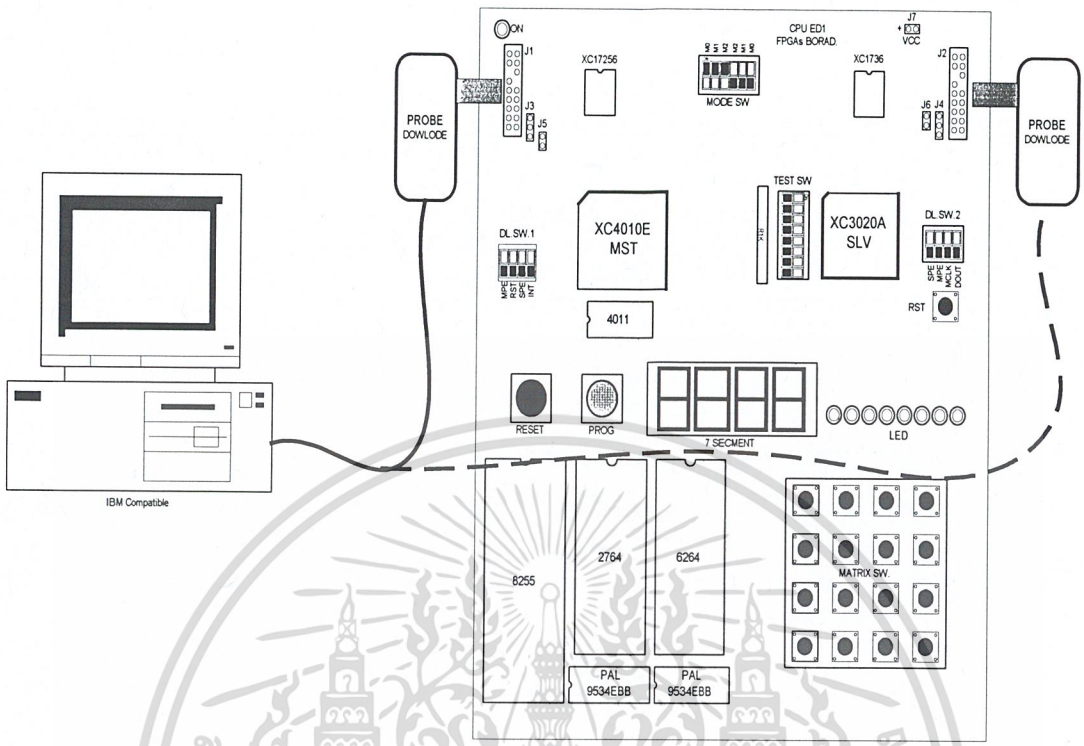
หลังจากจำลองผลการทำงานจนได้ผลการทำงานตรงตามวัตถุประสงค์ที่ออกแบบไว้แล้ว ขั้นตอนต่อมาคือ แปลงโปรแกรมของวงจรของหน่วยประมวลผลกลางขนาด 4 บิต ให้เป็นวงจรซึ่งประกอบด้วยอุปกรณ์ดิจิตอลพื้นฐานจำพวกเกท และฟลิปฟลอปต่าง ๆ ในการสังเคราะห์เป็นวงจรระดับเกทนี้ต้องกำหนดด้วยว่าจะใช้อุปกรณ์พื้นฐานของบริษัทใด ซึ่งโครงงานนี้ได้เลือกทำการพัฒนาวงจรที่ได้ลงบนอุปกรณ์ FPGAs เบอร์ XC4010E และ XC3020 ของบริษัทไซลิงค์ หลังจากการจำลองผลการทำงาน และสังเคราะห์ให้เป็นระดับเกทแล้วจะแสดงได้ดังรูปที่ 3.12



รูปที่ 3.12 การสังเคราะห์ระดับเกตของการ Mul

3.8.4 การโปรแกรมวงจรลงบนไอซี FPGAs

หลังจากสังเคราะห์วงจรระดับเกทของวงจรของหน่วยประมวลผลกลางขนาด 4 บิต แล้ว ขั้นตอนต่อมาเป็นการนำเอาวงจรที่ได้มาทำการดาวน์โหลดลงไอซี FPGAs โดยใช้ซอฟแวร์ XACT Development Tools ของบริษัทไซลิงค์ในการดาวน์โหลดวงจรหน่วยประมวลผลกลางขนาด 4 บิต ที่ได้ลงบนเซลล์ต่าง ๆ ภายในไอซี FPGAs ผลสุดท้ายจะได้ออกมาเป็นวงจรหน่วยประมวลผลกลางขนาด 4 บิตบนไอซี FPGAs ซึ่งสามารถนำไปสร้างวงจรต้นแบบได้ทันที

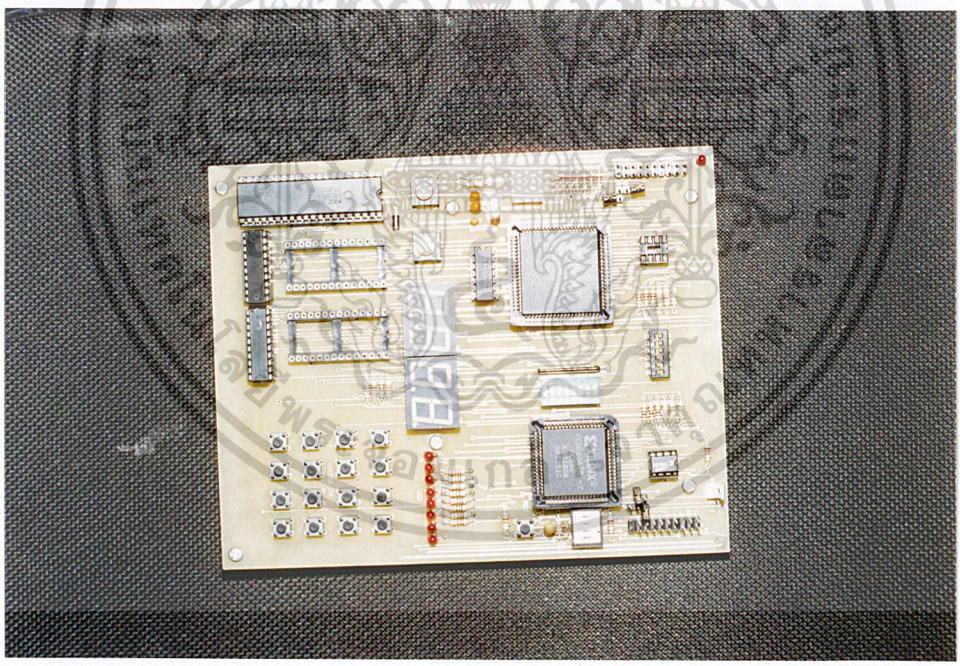


รูปที่ 3.13 การเชื่อมต่อกันระหว่างคอมพิวเตอร์กับบอร์ดตัวอย่างของ FPGAs

บทที่ 4

การทดลองและผลการทดลอง

การออกแบบหน่วยประมวลผลกลางขนาด 4 บิต คือ เขียนโปรแกรมภาษา VHDL ในแต่ละส่วนแล้วนำไปจำลองการทำงานของแต่ละส่วนจนได้ผลตามที่ออกแบบแล้วนำไปรวมเป็นส่วนกลาง นำโปรแกรมไปสังเคราะห์เป็นวงจรระดับเกท เพื่อทำการดาวน์โหลดบนอุปกรณ์ FPGAs โดยทำการเลือก FPGAs เบอร์ 4010E เป็นหน่วยประมวลผลกลาง และเลือก FPGAs เบอร์ 3020A ทำหน้าที่เป็นอุปกรณ์ส่วนทดลองทางด้าน โปรแกรมภาษา VHDL บอร์ด CPU ED1 ที่ได้ทำการออกแบบนั้น มีลักษณะดังรูปที่ 4.1



รูปที่ 4.1 บอร์ดทดลอง CPU ED1

4.1 การเขียนโปรแกรมส่วนย่อย

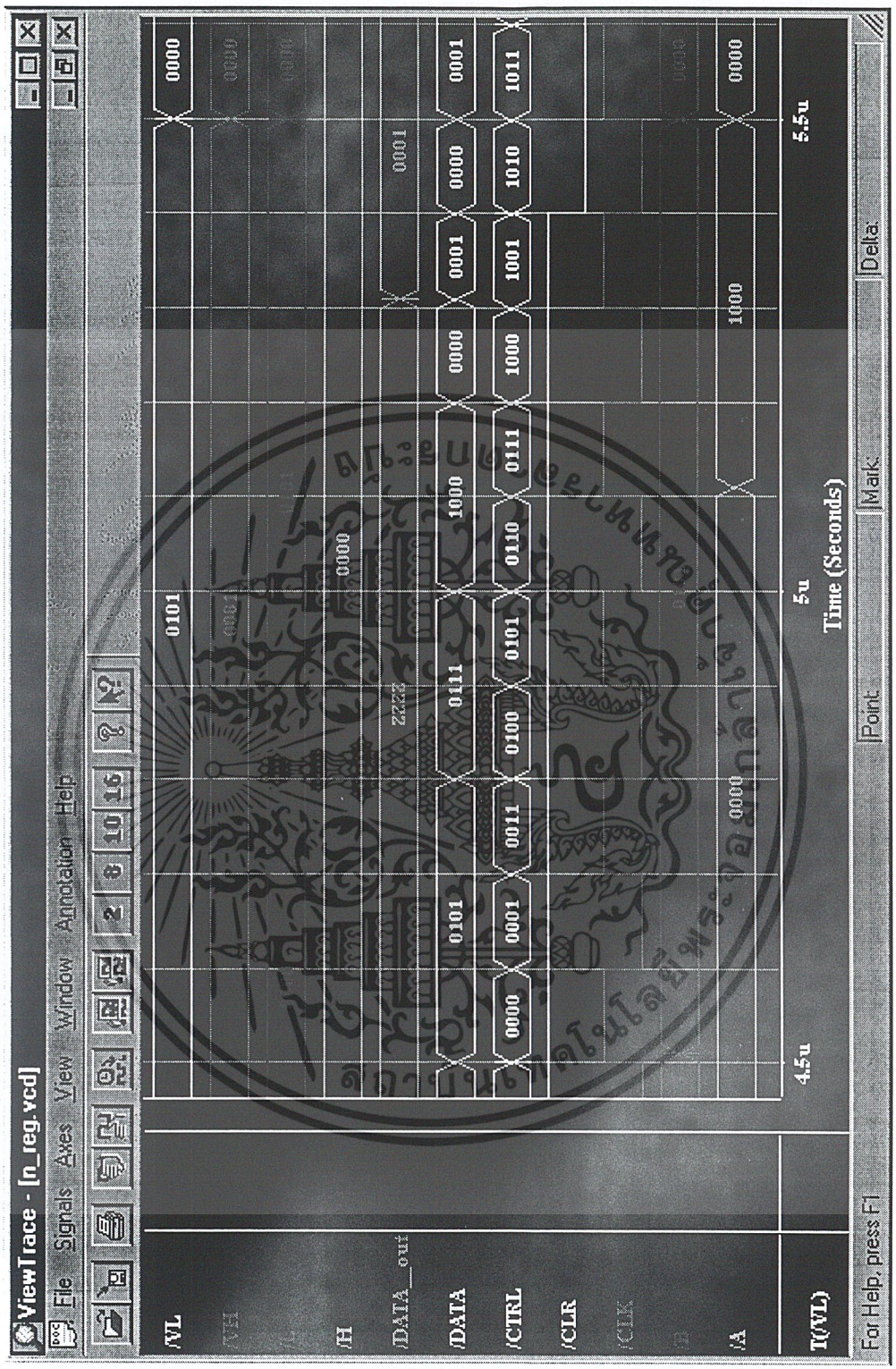
4.1.1 ส่วนโปรแกรมสร้างรีจิสเตอร์

ส่วนเขียนโปรแกรมภาษา VHDL ที่เป็นการสร้างรีจิสเตอร์นี้สามารถดูการเขียนทั้งหมดได้จาก entity n reg ในภาคผนวก ก ดังแสดงไว้ในรูปที่ 4.2

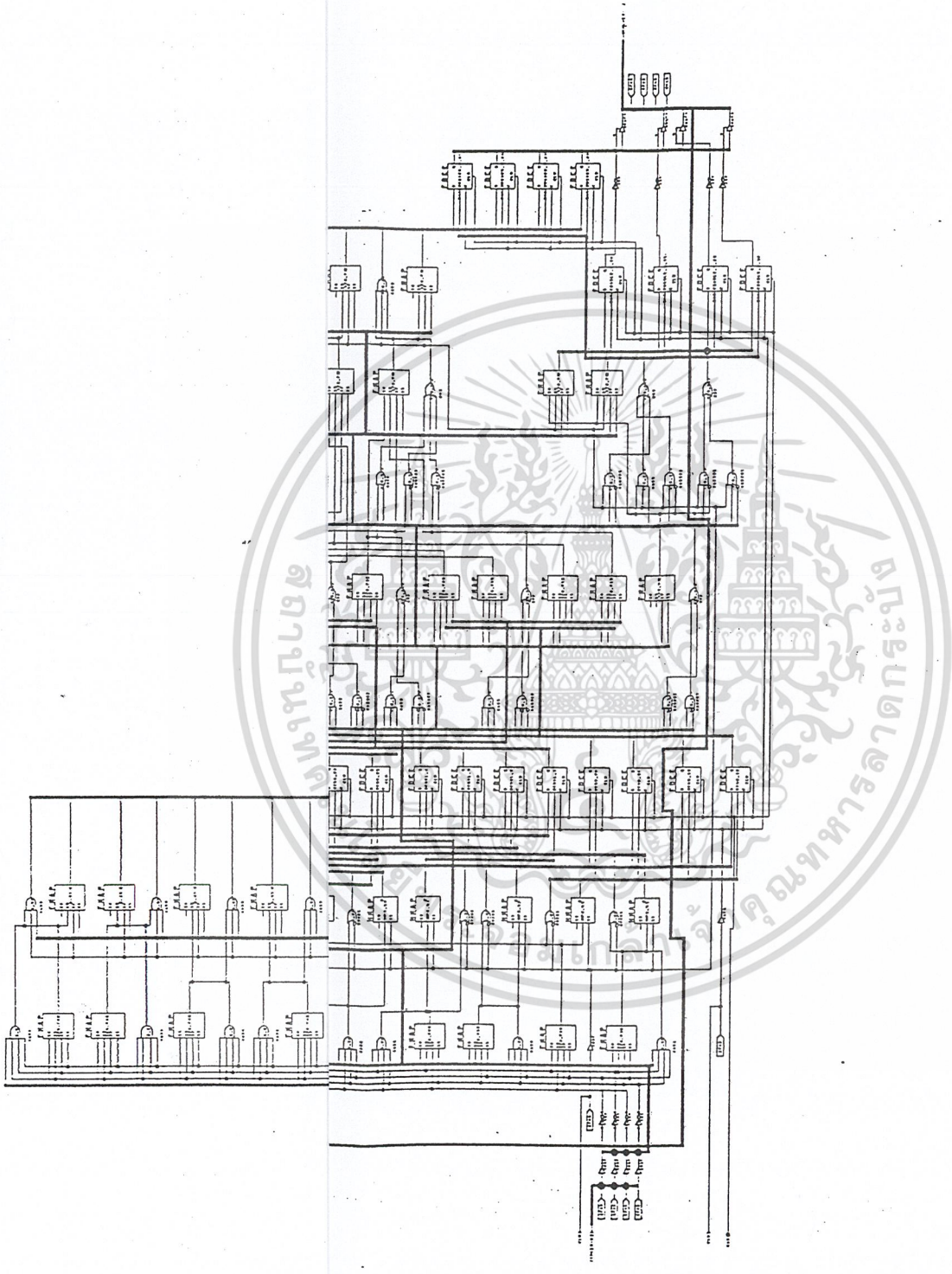
```
entity n_reg is
port(CLR, CLK     :  in std_logic;
Ctrl   : in std_logic_vector( 3 downto 0);
DATA : inout std_logic_vector( 3 downto 0 )bus    );
end  n_reg;
```

รูปที่ 4.2 entity การสร้างรีจิสเตอร์

หลังจากการเขียนโปรแกรมในส่วน entity n reg แล้วนำโปรแกรมนี้ไปผ่านการ Simulate เพื่อดูระดับสัญญาณที่ได้ออกมาแล้ววิเคราะห์ว่าสัญญาณนั้นทำงานตามที่ได้ออกแบบไว้หรือไม่ ซึ่งสัญญาณที่ได้จากการ Simulate ของ entity n_reg ดังแสดงไว้ในรูปที่ 4.3

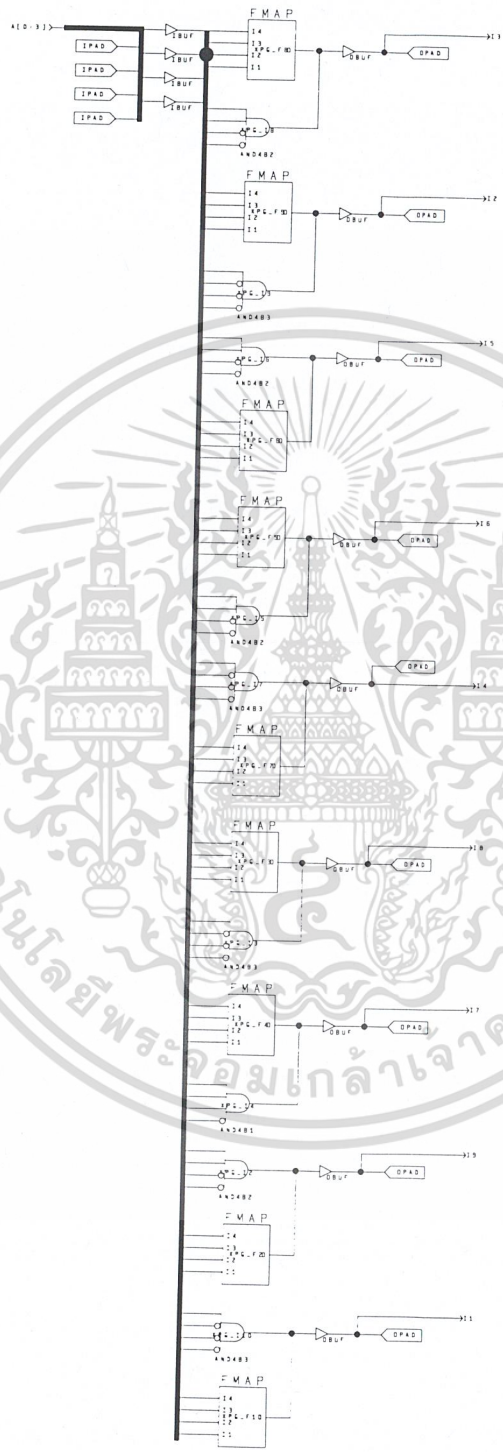


รูปที่ 4.3 ผลการ Simulate entity n_seg



รูปที่ 4.6 ผลการ Simulate ของ entity mil l

ขั้นตอนการ Synthesis entity mul l เพื่อให้ได้วงจรระดับเกตดังแสดงไว้ในรูปที่ 4.7



รูปที่ 4.7 ผลการ Synthesis ของ entity mul 1

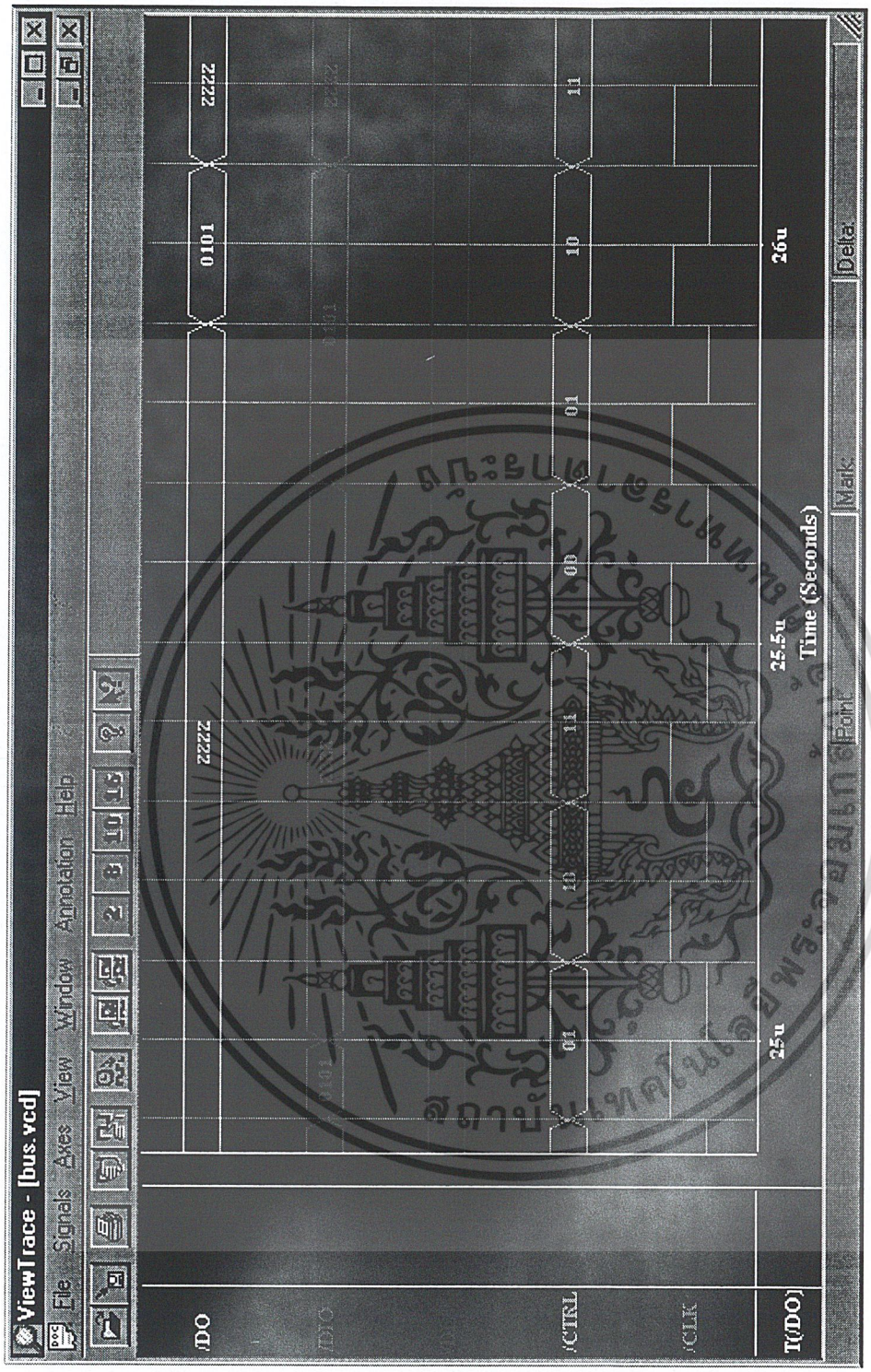
4.1.3 บัส

บัสเป็นเส้นทางที่ใช้สำหรับติดต่อทั้งข้อมูลอินพุตและเอาต์พุต โดยใน entity bus4 นี้จะสร้างบัสขึ้นมา 4 เส้นคือ คอนโทรลบัส ดาต้าบัสแบบอินพุต เอาต์พุต และทั้งสองทาง ดังแสดงไว้ในรูปที่ 4.8 และสามารถดูโครงสร้างภายในได้ในภาคผนวก ก

```
entity bus4 is
port(clk     : in std_logic;
     ctrl    : in std_logic_vector( 1 downto 0);
     Di      : in std_logic_vector( 3 downto 0);
     Do      : out std_logic_vector( 3 downto 0);
     Dio     : inout std_logic_vector( 3 downto 0) );
end bus4;
```

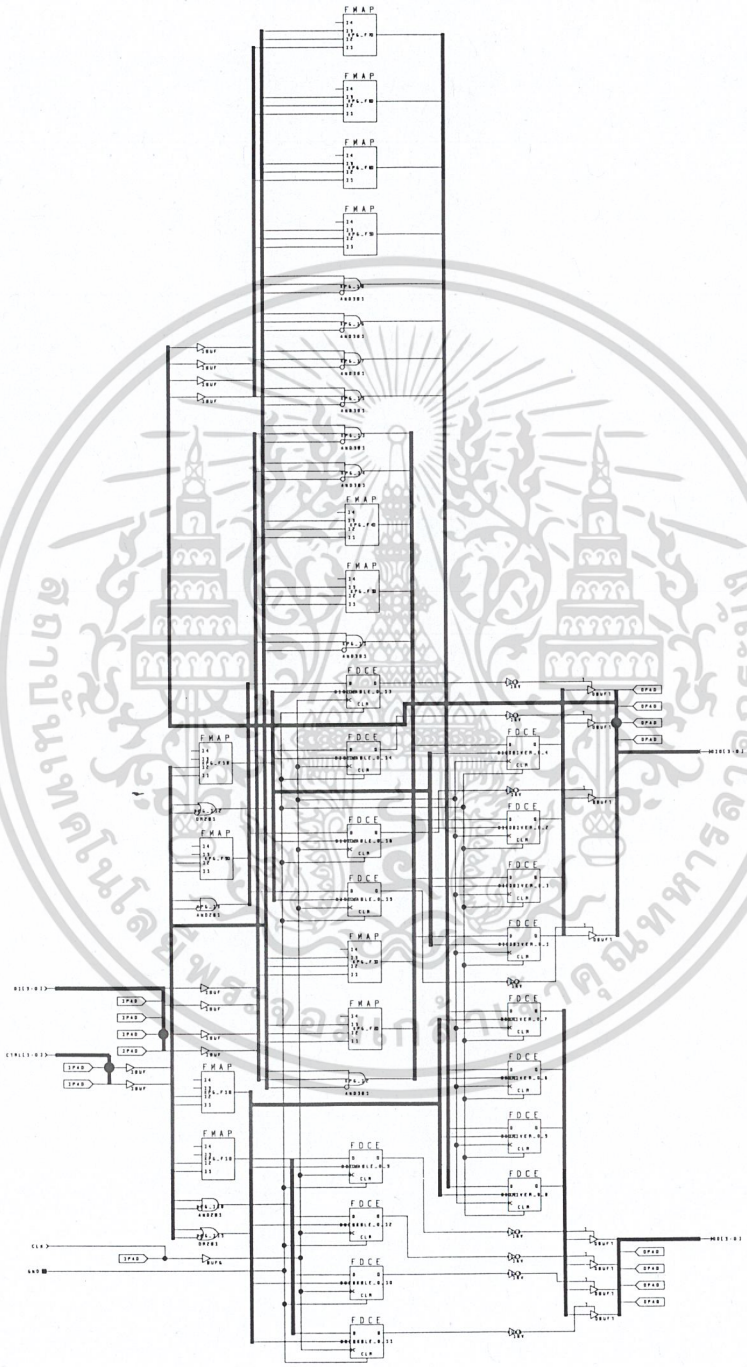
รูปที่ 4.8 entity การสร้างบัส

entity การสร้างบัสหลังจากเขียนขึ้นมาแล้วให้นำมาทำการ Simulate จะได้ผลดังแสดงไว้ในรูปที่ 4.9



รูปที่ 4.9 ผลการ Simulate entity bus4

เมื่อผ่านการ Simulate เป็นสัญญาณแล้ว ให้นำมาทำการ Synthesis เป็นระดับเกท ซึ่งแสดงได้ดังรูปที่ 4.10



รูปที่ 4.10 ผลการ Synthesis entity bus4

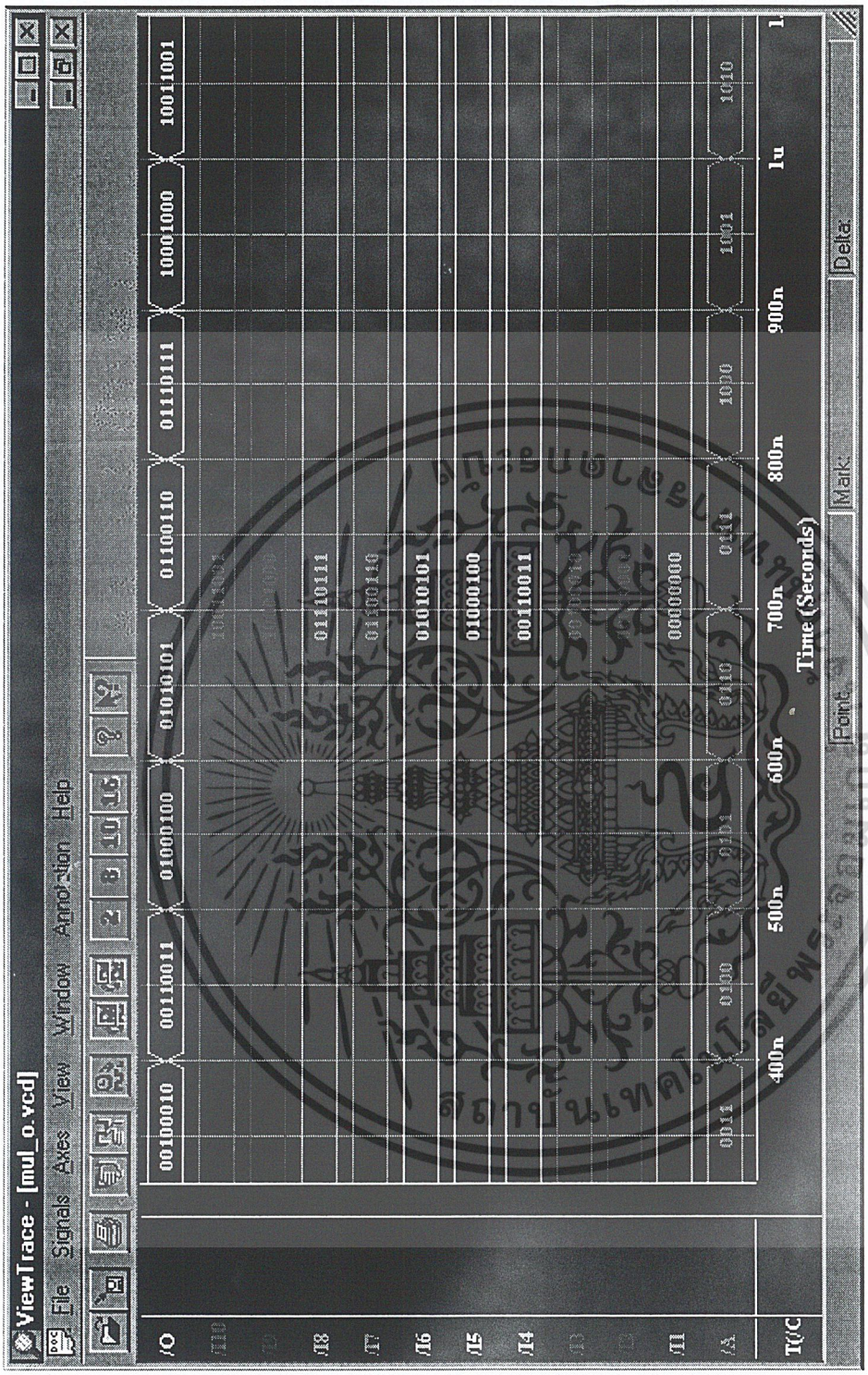
4.1.4 การมัลติเพล็กเอาต์พุต

การมัลติเพล็กเอาต์พุต สร้างขึ้นเพื่อใช้มัลติเพล็กสัญญาณอินพุตที่เข้ามา ออกเป็นเส้นสัญญาณให้เลือกใช้ได้ 10 เส้นสัญญาณ โดย entity mux o ดูโครงสร้างได้ใน ภาคผนวก ก ดังรูปที่

```
entity  mux_O is
port(A  : in std_logic_vector( 0 to 3 );
      signal  i1,i2,i3,i4,i5,i6,i7,i8,i9,i10 :  in std_logic_vector( 0 to 7);
                  O : buffer std_logic_vector( 0 to 7)  );
end mux_O;
```

รูปที่ 4.11 การมัลติเพล็กเอาต์พุต

เมื่อนำโปรแกรมที่ได้มาทำการ Simulate จะได้ผลดังแสดงไว้ในรูปที่ 4.12



รูปที่ 4.12 ผลการ Simulate ของ entity mux o