สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การออกแบบซีพียูโดยใช้ภาษาวีเอชดีแอล

THE DESIGN OF CPU USING VHDL

T046299

นายเฉลิมพงษ์ กล้าขยัน

นายชลเดช สิริพุฒิพรกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

เลขหมู่.....
เลขทะเบียน 46299
วัน, เดือน, ปี 21 มี.ค. 2546

.b.....

.i.....

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การออกแบบซีพียูโดยใช้ภาษาวีเอชดีแอล

THE DESIGN OF CPU USING VHDL

โดย

นายเฉลิมพงษ์	กล้าขยัน	41014083
นายชลเดช	สิริพุฒิพรกุล	41014091

อาจารย์ที่ปรึกษา

ผศ.ดร.อภินันท์ ธนชยานนท์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ ปีการศึกษา 2544

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบซีพียูโดยใช้ภาษาวีเอชดีแอล

THE DESIGN OF CPU USING VHDL

ผู้จัดทำ

1. นายเฉลิมพงษ์ กล้าขยัน 41014083
2. นายชลเดช สิริพุฒิพรกุล 41014091

_____ อาจารย์ที่ปรึกษา

(ผศ.ดร.อภินันท์ ธนชยานนท์)

วันที่ 21 / มี.ค. / 2545

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การออกแบบซีพียู โดยใช้ภาษาวีเอชดีแอล

นายเฉลิมพงษ์ กล้าขยัน 41014083
นายชลเดช สิริพุฒิพรกุล 41014091
ผศ.ดร.อภินันท์ ธนชยานนท์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

โครงงานนี้เป็นการออกแบบซีพียูโดยใช้ภาษาวีเอชดีแอลเป็นภาษาบรรยายการทำงานของวงจร ในการออกแบบจะใช้หลักการของ Top Down Design โดยทำการแบ่งซีพียูออกเป็นส่วนๆ จากนั้นจึงเขียนแต่ละส่วนด้วยภาษา VHDL และตรวจสอบการทำงานของแต่ละส่วนด้วยซอฟท์แวร์ซิมูเลชั่น จนกระทั่งแน่ใจว่าแต่ละส่วนที่เขียนขึ้นมีการทำงานที่ถูกต้อง จึงนำแต่ละส่วนที่เขียนขึ้นนี้มารวมเข้าด้วยกัน และทำการทดสอบการทำงานจนแน่ใจว่าแต่ละส่วนที่สร้างขึ้นนั้นสามารถทำงานร่วมกันได้อย่างไม่มีปัญหา แล้วนำ VHDL Code ของ CPU ที่ได้ เข้าสู่กระบวนการ Synthesis เพื่อแปลงให้เป็นวงจรในระดับ Gate-level และนำวงจรที่ได้จากการ Synthesis นี้ไปทำการ Place & Route บันทึกลง FPGA และทดสอบการทำงานกับอุปกรณ์ฮาร์ดแวร์

ซีพียูที่ได้ทำการออกแบบสามารถประมวลผลข้อมูลแบบ 8 บิต อ้างอิงหน่วยความจำได้ 64 กิโลไบต์ ทำคำสั่งได้ 35 คำสั่ง มีความเร็วในการทำงานสูงสุด 5.8 MHz และ FPGA ที่ได้ก็สามารถทำงานได้ผลที่น่าพอใจ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

THE DESIGN OF CPU USING VHDL

Mr.Chalermpong Klakayan 41014083
Mr.Chonladeth Siriputipornkul 41014091
Asst.Prof.Dr.Apinunt Thanachayanont Advisor
2001

Abstract

This project is the use of VHDL as a language for modeling and design of CPU. In the design process, the researchers use top-down design model to develop the CPU. The process of design begins with writing each part in VHDL language and test each part with simulation software to ensure that the these parts are working correctly, then combine them together and test the corrective of the cooperation between these parts. Then bring the derived VHDL Code into the synthesis process to convert into gate-level netlist and into the place-and-route process to program this CPU into FPGA. When deriving the CPU in FPGA, bring this FPGA into the circuit board to test the corrective operation of it.

The CPU derived in this project can process 8 bit data , is capable of addressing 64 kbyte of memory and has capability of 35 instructions. The Maximum frequency which this CPU can run is 5.8 MHz and the drived FPGA are working correctly.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ในครั้งนี้สำเร็จลุล่วงไปด้วยดี เพราะได้รับความกรุณาจากอาจารย์ อภินันท์ ธนชยานนท์ อาจารย์ที่ปรึกษา ที่ได้ให้คำแนะนำและความช่วยเหลือ ขอขอบพระคุณเป็นอย่างยิ่ง

ขอขอบคุณ คุณพ่อ คุณแม่ ที่เอาใจใส่และให้กำลังใจตลอดมา

ขอขอบคุณ พี่นเรศ ห้อง Hardware ภาควิชาวิศวกรรมคอมพิวเตอร์ ที่ให้ความกรุณาในการยืม FPGA และคำแนะนำต่างๆ

ขอบคุณจริงๆ สำหรับ ก้อง เต้ย และเพื่อนๆ ภาคคอมที่คอยช่วยเหลือยามวิกฤต

ขอบคุณชมรมบริดจ์ ชุมนุมวิชาการ ชุมนุมโทรคมนาคม ที่เอื้อเฟื้อสถานที่ทำ project และสถานที่หลับนอน

ขอบคุณ เปรี้ยว ต่อ ทัส มด เป้ ท็อป ก้อง ต๊ะ บอล เตียว จิ๊บ เล็ก ที่คอยสอบถามว่า project เป็นอย่างไรบ้าง ทั้งๆ ที่ของตัวเองก็ยังไม่เสร็จเหมือนกัน

ขอบคุณเพื่อนๆ พี่ๆ น้องๆ ทุกคนที่ให้กำลังใจ

ขอบคุณไฟตึกกิจฯที่เทอมนี้ไม่ค่อยตกใส่

สุดท้ายนี้ขอบคุณภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ ที่ให้ความรู้ และประสบการณ์ ต่างๆ มากมาย

เฉลิมพงษ์ กล้าขยัน

ชลเดช สิริพุฒิพรกุล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

	หน้า
บทคัดย่อภาษาไทย	I
Abstract	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญรูป	V
สารบัญตาราง	VIII
บทที่ 1 บทนำ	1
บทที่ 2 Central Processing Unit (CPU)	4
บทที่ 3 ภาษาวีเอชดีแอล (VHDL)	15
บทที่ 4 เอฟพีจีเอ (FPGA)	30
บทที่ 5 ขั้นตอนและรายละเอียดการออกแบบ CPU	37
บทที่ 6 การทดสอบการทำงานของ CPU ด้วยการ Simulation	64
บทที่ 7 การ Synthesis และการบันทึกโปรแกรมลง FPGA	66
บทที่ 8 สรุปและวิจารณ์	83
ภาคผนวก ก VHDL Code	85
ภาคผนวก ข Wave form ที่ได้จากการ Simulation	119
หนังสืออ้างอิง	135

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

หน้า

รูปที่ 2.1 Block Diagram ของระบบ Microcomputer 4
รูปที่ 2.2 แสดงรอบคำสั่ง (Instruction cycle) 5
รูปที่ 2.3 แสดงพฤติกรรมการทำงานของ CPU 6
รูปที่ 2.4 แสดงโครงสร้างพื้นฐานของ CPU 7
รูปที่ 2.5 แสดงสถาปัตยกรรมภายในของ CPU 8 bit 7
รูปที่ 2.6 แสดงรีจิสเตอร์ตำแหน่ง ที่ใช้สร้างข้อมูลบนบัสตำแหน่ง 9
รูปที่ 2.7 ระบบไมโครคอมพิวเตอร์พื้นฐาน 13
รูปที่ 3.1 สิ่งต่างๆ ที่สามารถอธิบายด้วย VHDL ได้ 19
รูปที่ 3.2 แสดงขั้นตอนของ Top Down Design 21
รูปที่ 3.3 แสดงการแบ่งย่อยในระดับราบของการออกแบบฮาร์ดแวร์ 22
รูปที่ 3.4 แสดงฮาร์แวร์โมดูลซึ่งสร้างจากการสร้างบล็อกของ VHDL 22
รูปที่ 3.5 แสดงให้เห็นถึงลำดับชั้น(Hierachy) ในการอธิบายอุปกรณ์ 23
รูปที่ 3.6 ตัวอย่างการอธิบาย ROM ในระดับต่างๆ 24
รูปที่ 3.7 แสดงการซ่อนรายละเอียดที่ไม่จำเป็นในระดับ NAND gate 25
รูปที่ 3.8 แสดงโครงสร้างอย่างง่ายๆ ของ entity design unit 27
รูปที่ 3.9 แสดงโครงสร้างอย่างง่ายๆ ของ architecture design unit 27
รูปที่ 3.10 แสดงโครงสร้างของ package declaration 28
รูปที่ 3.11 แสดงโครงสร้างของ package body 28
รูปที่ 3.12 แสดงโครงสร้างของ configruration 28
รูปที่ 4.1 แสดงวงจรพื้นฐานของอุปกรณ์พีแอลดีซึ่งอยู่ในรูปของ Sum of product 31
รูปที่ 4.2 แสดงวงจรพื้นฐานภายใน Xilinx FPGA 32
รูปที่ 4.3 แสดงผังวงจรภายในของ CLB ของ FPGA 33
รูปที่ 4.4 แสดงผังวงจรของ IOB ภายใน FPGA 34
รูปที่ 4.5 แสดง Interconnect ระหว่าง IOB กับ CLB ของ FPGA 34
รูปที่ 5.1 Flowchart แสดงขั้นตอนการออกแบบ CPU โดยใช้ VHDL 38
รูปที่ 5.2 แสดงส่วนต่างๆ ของซีพียูด้วย Block Diagram 42

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

หน้า

รูปที่ 5.3 แสดงขาอินพุทเอาท์พุทของ CPU ที่ออกแบบ 44
รูปที่ 5.4 แสดง Component หลักภายใน CPU 45
รูปที่ 5.5 (ก) แสดงการเขียน Accumulator ด้วย VHDL ส่วน entity design 46
รูปที่ 5.5 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Accumulator 46
รูปที่ 5.6 (ก) แสดงการเขียน Instruction Register ด้วย VHDL ส่วน entity design 47
รูปที่ 5.6 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Instruction Register 47
รูปที่ 5.7 (ก) แสดงการเขียน program counter ด้วย VHDL ส่วน entity design 48
รูปที่ 5.7 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ program counter 48
รูปที่ 5.8 (ก) แสดงการเขียน memory address register ด้วย VHDL ส่วน entity design 49
รูปที่ 5.8 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ memory address register 49
รูปที่ 5.9 (ก) แสดงการเขียน Status register ด้วย VHDL ส่วน entity design 50
รูปที่ 5.9 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Status register 50
รูปที่ 5.10 (ก) แสดงการเขียน Stack pointer ด้วย VHDL ส่วน entity design 51
รูปที่ 5.10 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Stack pointer 51
รูปที่ 5.11 (ก) แสดงการเขียน General purpose register ด้วย VHDL ส่วน entity design 52
รูปที่ 5.11 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ General purpose register 52
รูปที่ 5.12 (ก) แสดงการเขียน Arithmetic Logic unit ด้วย VHDL ส่วน entity design 53
รูปที่ 5.12 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Arithmetic Logic unit 53
รูปที่ 5.13 (ก) แสดงการเขียน Shifter unit ด้วย VHDL ส่วน entity design 54
รูปที่ 5.13 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Shifter unit 54
รูปที่ 5.14 (ก) แสดงการเขียน Data path ด้วย VHDL ส่วน entity design 55
รูปที่ 5.14 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Data path 56
รูปที่ 5.15 (ก) แสดงการเขียน Control unit ด้วย VHDL ส่วน entity design 57
รูปที่ 5.15 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Control unit 58
รูปที่ 5.16 แสดง state machine ของซีพียู 62

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

หน้า

รูปที่ 6.1 แสดงหน้าจอของโปรแกรม ModelSim SE-EE5.4 64
รูปที่ 6.2 แสดง Wave form ที่ได้จากการเขียน Test bench 65
รูปที่ 7.1 แสดงหน้าจอของโปรแกรม Leonardo Spectrum 66
รูปที่ 7.2 แสดงหน้าจอของโปรแกรม Xilinx Foundation 67
รูปที่ 7.3 แสดงหน้าจอของโมดูล Flow Engine ใน Xilinx Foundation 68
รูปที่ 7.4 แสดงหน้าจอของโมดูล Hardware Debugger ใน Xilinx Foundation 68
รูปที่ 7.5 แสดงรูปวงจรที่ได้จากการ Synthesis ของ accumulator และ General purpose register 70
รูปที่ 7.6 แสดงรูปวงจรที่ได้จากการ Synthesis ของ instruction register 71
รูปที่ 7.7 แสดงรูปวงจรที่ได้จากการ Synthesis ของ status register 71
รูปที่ 7.8 แสดงรูปวงจรที่ได้จากการ Synthesis ของ stack pointer 72
รูปที่ 7.9 แสดงรูปวงจรที่ได้จากการ Synthesis ของ memory address register 73
รูปที่ 7.10 แสดงรูปวงจรที่ได้จากการ Synthesis ของ shifter unit 74
รูปที่ 7.11 แสดงรูปวงจรที่ได้จากการ Synthesis ของ program counter 75
รูปที่ 7.12 แสดงรูปวงจรที่ได้จากการ Synthesis ของ alu 76
รูปที่ 7.13 แสดงรูปวงจรที่ได้จากการ Synthesis ของ datapath 77
รูปที่ 7.14 แสดงรูปวงจรที่ได้จากการ Synthesis ของ control unit 78
รูปที่ 7.15 แสดงรูปวงจรที่ได้จากการ Synthesis ของซีพียู 78
รูปที่ 7.16 แสดงรูปของ สาย XChecker 79
รูปที่ 7.17 แสดงรูป XILINX84 Demo board 80
รูปที่ 7.18 แสดง Lay out ของ XILINX84 Demo Board 80
รูปที่ 7.19 Schematic ของ XILINX84 Demo Board 81

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

หน้า

ตารางที่ 5.1 แสดง opcodes ของชุดคำสั่ง 63



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

ปัจจุบัน การใช้ชีวิตประจำวันต้องการความสะดวกสบายมากกว่าสมัยก่อน ซึ่งความสะดวกสบายเหล่านี้ ล้วนมาจากเทคโนโลยีที่ก้าวไปข้างหน้าอย่างไม่หยุดยั้ง และเทคโนโลยีทางด้านอิเล็กทรอนิกส์ก็เป็นเทคโนโลยีหนึ่งที่มีความก้าวหน้าไปอย่างรวดเร็ว จะเห็นได้จากเครื่องมือเครื่องใช้ต่างๆ ที่ต้องอาศัยชิพอิเล็กทรอนิกส์ในการสั่งให้ปฏิบัติงานให้ได้ผลตามที่ต้องการ เราจึงควรตามเทคโนโลยีที่ก้าวหน้าไปอย่างรวดเร็วนี้ให้ทัน จึงควรส่งเสริมให้มีการศึกษา วิจัย คิดค้นและออกแบบอุปกรณ์หรือวงจรอิเล็กทรอนิกส์ ซึ่งการออกแบบวงจรรวม (IC Design) ก็จัดเป็นวิธีหนึ่ง

ซึ่งในอดีตนักออกแบบวงจรดิจิตอลได้ใช้วิธีการออกแบบวงจรด้วยวิธี Bottom-up Design โดยทำกันที่ระดับ Gate – level สร้างความยุ่งยากเป็นอย่างมากให้แก่ผู้ออกแบบ ซึ่งต่อมาการออกแบบก็เริ่มซับซ้อนมากขึ้น ผู้ออกแบบจะต้องออกแบบไอซีที่มีจำนวนเกทมากกว่าแสนเกทภายในระยะเวลาอันจำกัด เนื่องจากแรงกดดันในการที่จะต้องนำผลิตภัณฑ์ออกสู่ท้องตลาดให้ได้ภายในระยะเวลาอันสั้น วิธีการออกแบบที่เคยใช้กันมาเริ่มไม่ใช่วิธีที่เหมาะสมอีกต่อไป เนื่องจากยุ่งยาก เสียเวลาและมักเกิดข้อผิดพลาดขึ้นบ่อยๆ จนกระทั่งปัจจุบัน เทคนิคการออกแบบวงจรก้าวหน้าไปมาก ผู้ออกแบบไม่จำเป็นต้องออกแบบในระดับ Gate-level เหมือนดังแต่ก่อน วิธีการออกแบบที่นิยมคือการเขียนวงจรที่ต้องการด้วยภาษาบรรยายการทำงานของวงจร (Hardware Description Language : HDL) ซึ่งสามารถตรวจสอบความถูกต้องได้ด้วยการใช้ Software Simulation จนแน่ใจว่างานที่ออกแบบนั้นมีการทำงานที่ถูกต้อง จึงนำงานนั้นไปเข้าสู่กระบวนการ Synthesis เพื่อแปลง HDL Code ที่เขียนให้อยู่ในระดับเกทซึ่งขั้นตอนดังกล่าวนั้นหากเป็นวิธีการออกแบบวงจรแบบเก่าแล้วจะเป็นงานที่เสียเวลามาก หลังจากได้วงจรในระดับเกทแล้วจึงนำวงจรไปทำเป็นวงจรรวมหรือบันทึกลงบนอุปกรณ์ประเภท Programmable Logic Devices หรือ PLD ต่อไป

ด้วยวิธีการออกแบบดังกล่าว ทำให้การออกแบบ Hardware ในปัจจุบันสามารถทำได้อย่างสะดวกรวดเร็ว ผู้ออกแบบสามารถสร้างวงจรที่มีฟังก์ชันการทำงานตามที่ต้องการได้ภายในระยะเวลาอันสั้น

ภาษาบรรยายการทำงานของวงจร(HDL) ในปัจจุบันที่นิยมใช้คือ VHDL พัฒนามาจากภาษา ADA ผู้พัฒนาคือ กระทรวงกลาโหมของสหรัฐอเมริกา มีโครงสร้างทางภาษาคล้ายภาษา PASCAL ทุกหน่วยงานที่ต้องการออกแบบหรือทำการพัฒนาให้กับกระทรวงนี้จะต้องใช้ภาษานี้ จึงทำให้ภาษานี้กลายเป็นภาษามาตรฐานในการนำไปออกแบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาษา VHDL มีลักษณะที่ผสมผสานกันระหว่าง Object Oriented Language และ Concurrent Programming Language โดยมองส่วนต่างๆ เป็นโมดูล(Module) หรือออบเจ็ก(Object) และมีสัญญาณเชื่อมต่อระหว่างกัน การส่งสัญญาณต่างๆ ในระบบเกิดขึ้นพร้อมๆ กันได้เช่นเดียวกับกระแสไฟฟ้าที่ไหลไปยังส่วนต่างๆ ของวงจร

ภาษา VHDL สามารถนำเอามาใช้ในการออกแบบวงจรได้ตั้งแต่วงจร Combination ขนาดเล็กไปจนถึงวงจรขนาดใหญ่ ซึ่งจะทำให้สามารถออกแบบวงจรที่มีความซับซ้อนได้ง่ายขึ้น อันจะส่งผลให้เกิดการพัฒนาทางด้านเทคโนโลยี ในการที่จะสร้างอุปกรณ์ Hardware ใหม่ๆ ต่อไป

1.1 ความสำคัญและที่มา

การออกแบบวงจรดิจิตอลขนาดใหญ่ที่ระดับ Gate-Level นั้น เป็นงานที่ยุ่งยากและซับซ้อนมาก เป็นผลให้งานที่พัฒนามีความล่าช้า และเสร็จไม่ตรงตามกำหนด การนำภาษาบรรยายการทำงานของวงจรมาใช้ในการออกแบบจะช่วยแก้ปัญหานี้ได้

Central Processing Unit (CPU) หรือหน่วยประมวลผลกลาง ทำหน้าที่ในการคำนวณและจัดการข้อมูล ซึ่งเป็นสิ่งสำคัญที่สุดในทุกๆ ระบบ หากสามารถออกแบบซีพียูโดยใช้ภาษาบรรยายการทำงานของวงจรได้ ก็จะทำผู้ออกแบบมีความเชี่ยวชาญในการออกแบบวงจรดิจิตอลขนาดใหญ่ได้เป็นอย่างดี ซึ่งจะเป็นพื้นฐานในการออกแบบวงจรที่ซับซ้อนต่อไป

1.2 วัตถุประสงค์

เพื่อออกแบบให้ได้ซีพียูในรูปของ FPGA ซึ่งสามารถเลียนแบบการทำงานของซีพียู ขนาด 8 bit โดยกระทำคำสั่งพื้นฐานได้ 35 คำสั่ง ซึ่งใช้ภาษา VHDL เป็นภาษาบรรยายการทำงานของวงจรในการออกแบบ

1.3 ขอบเขตของโครงงาน

ซีพียูที่ทำการออกแบบเป็นแบบ 8 บิต โดยสามารถอ้างหน่วยความจำได้ 64 กิโลไบต์ โดยมีสถาปัตยกรรมภายในประกอบด้วย

1. Arithmetic Logic Unit
2. Control Unit
3. Shifter Unit
4. Accumulator
5. Program Counter
6. Status Register
7. Instruction Register
8. Memmory Address Register
9. General purpose Register
10. Stack Pointer

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1.4 ขั้นตอนในการทำงาน

1. ศึกษาการใช้ภาษา VHDL ซึ่งเป็นภาษาที่ใช้ในการอธิบายการทำงาน รวมถึงการใช้ Software Tools ในการออกแบบ อันได้แก่ ModelSim SE-EE 5.4, Leonardo Spectrum, และโปรแกรม Xilinx Foundation
2. ศึกษาหลักการทำงานและส่วนประกอบของซีพียู
3. ออกแบบภาพรวมของระบบ และทำการเขียนการทำงานของระบบ โดยแบ่งงานที่ออกแบบเป็นส่วนๆ และทดสอบแต่ละส่วนโดยการใช้ Test bench ที่เขียนขึ้นมาเอง
4. ทำการรวมแต่ส่วนเข้าด้วยกัน ทดสอบการทำงานจนกระทั่งแน่ใจว่าสามารถทำงานได้ถูกต้อง
5. นำ Code ของ CPU ที่ได้เข้าสู่กระบวนการ Synthesis เพื่อแปลง VHDL Code ให้เป็นวงจรในระดับ Gate – level และนำไปเข้าสู่กระบวนการ Place & Route เพื่อบันทึกลงสู่บอร์ด FPGA
6. ทดสอบบอร์ด FPGA ที่บันทึกโปรแกรมกับอุปกรณ์ Hardware ว่าสามารถทำงานได้อย่างถูกต้องหรือไม่

5. ประโยชน์ที่คาดว่าจะได้รับ

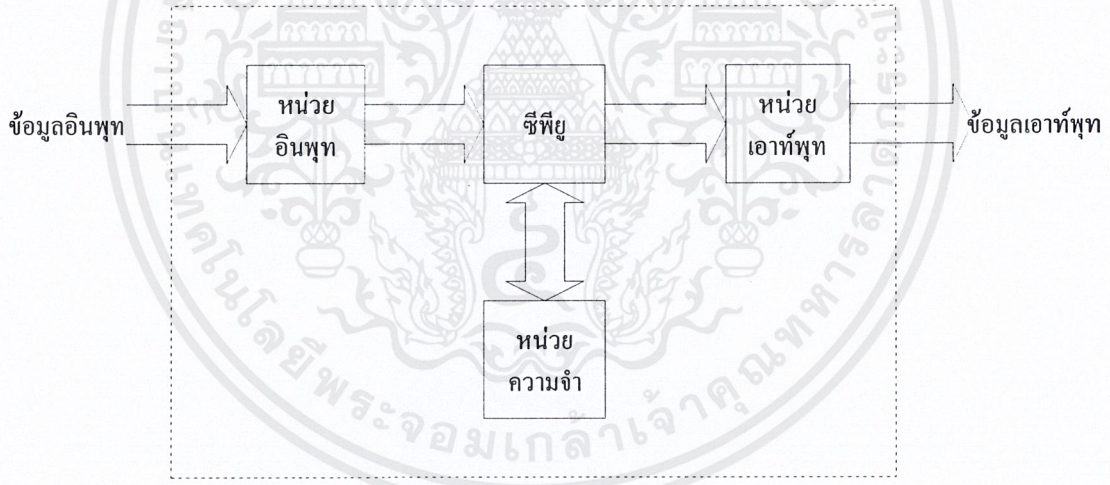
1. เข้าใจขั้นตอนการออกแบบ การทำงาน และโครงสร้างภายในของซีพียู
2. สามารถใช้ภาษา VHDL ในการออกแบบวงจรดิจิตอลได้เป็นอย่างดีและเป็นพื้นฐานในการออกแบบวงจรรวมอื่นๆ ต่อไป
3. มีความเชี่ยวชาญในการใช้ Software Tools ต่างๆ ในการออกแบบวงจรดิจิตอล
4. ได้รู้จักขั้นตอนการทำงาน การวางแผน การตรวจสอบการทำงานและแนวทางการแก้ไขปัญหาที่เกิดขึ้น
5. ได้ฝึกการทำงานร่วมกับผู้อื่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2

Central Processing Unit (CPU)

นับตั้งแต่ระบบคอมพิวเตอร์แบบอิเล็กทรอนิกส์เครื่องแรกได้กำเนิดขึ้น เพื่อใช้แก้ปัญหาทางวิทยาศาสตร์ที่ยุ่งยากซับซ้อน ซึ่งต่อมาได้มีความเจริญก้าวหน้าอย่างรวดเร็ว จนกระทั่งในปัจจุบันมีความสามารถสูงกว่าในยุคแรกมาก และยังถูกประยุกต์ใช้งานต่างๆ ในชีวิตประจำวันอย่างมากมาย ซึ่งการที่พัฒนาอย่างรวดเร็วนี้เนื่องมาจากการพัฒนาสิ่งประดิษฐ์ที่เป็นสารกึ่งตัวนำขึ้น เกิดเป็นทรานซิสเตอร์ ซึ่งนำมาแทนหลอดสูญญากาศในคอมพิวเตอร์และมีการเปลี่ยนแปลงอีกเมื่อมีวงจรรวม (Integrated Circuit : IC)ใช้แทนทรานซิสเตอร์บางตัว ซึ่งต่อมาได้ใช้เทคโนโลยีของการผลิตวงจรรวมแบบ Large Scale Integrated Circuit หรือ LSI ทำการรวมเอาวงจรที่ใช้เป็นหน่วยประมวลผลกลาง (CPU) ของคอมพิวเตอร์บรรจุอยู่ในแผ่นไอซีเพียงตัวเดียว เมื่อนำเอา CPU นี้มารวมกับหน่วยความจำ (Memory) หน่วยอินพุท และหน่วยเอาท์พุท (Input Output Unit) ก็จะทำให้ได้ระบบคอมพิวเตอร์ที่มีขนาดเล็กขึ้นมา ที่เรียกว่า ไมโครคอมพิวเตอร์



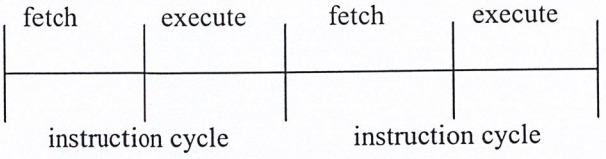
รูปที่ 2.1 Block Diagram ของระบบ Microcomputer

Central Processing Unit (CPU) คือหน่วยประมวลผลกลางทำหน้าที่ประมวลผลข้อมูลซึ่งจะรวมทั้งการคำนวณและการจัดการข้อมูล ลักษณะการทำงานของซีพียูคือจะปฏิบัติ(execute) คำสั่งที่เก็บไว้ในหน่วยความจำหลัก(Main Memory) ตามลำดับของคำสั่ง ซึ่งหน่วยความจำนี้โดยปกติจะอยู่นอกซีพียู ในขั้นแรกซีพียูต้องอ่านคำสั่ง (fetch) มาจากหน่วยความจำก่อน จากนั้นจะถอดรหัสคำสั่ง และปฏิบัติตามคำสั่งที่อ่านมาได้ ลำดับของการกระทำนี้ก่อให้เกิดขบวนการทำงานของคำสั่งที่เรียกว่า รอบคำสั่ง(instruction cycle) ซึ่งสามารถแบ่งออกเป็น 2 ส่วนใหญ่ๆ ได้คือ ช่วงอ่านคำสั่ง(fetch cycle) และช่วงการปฏิบัติคำสั่ง(execution cycle)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Fetch หมายถึง ขบวนการที่ CPU นำคำสั่งจากหน่วยความจำหลักและถอดรหัสคำสั่ง

Execute หมายถึง ขบวนการที่ CPU ทำคำสั่งนั้นจนเสร็จ

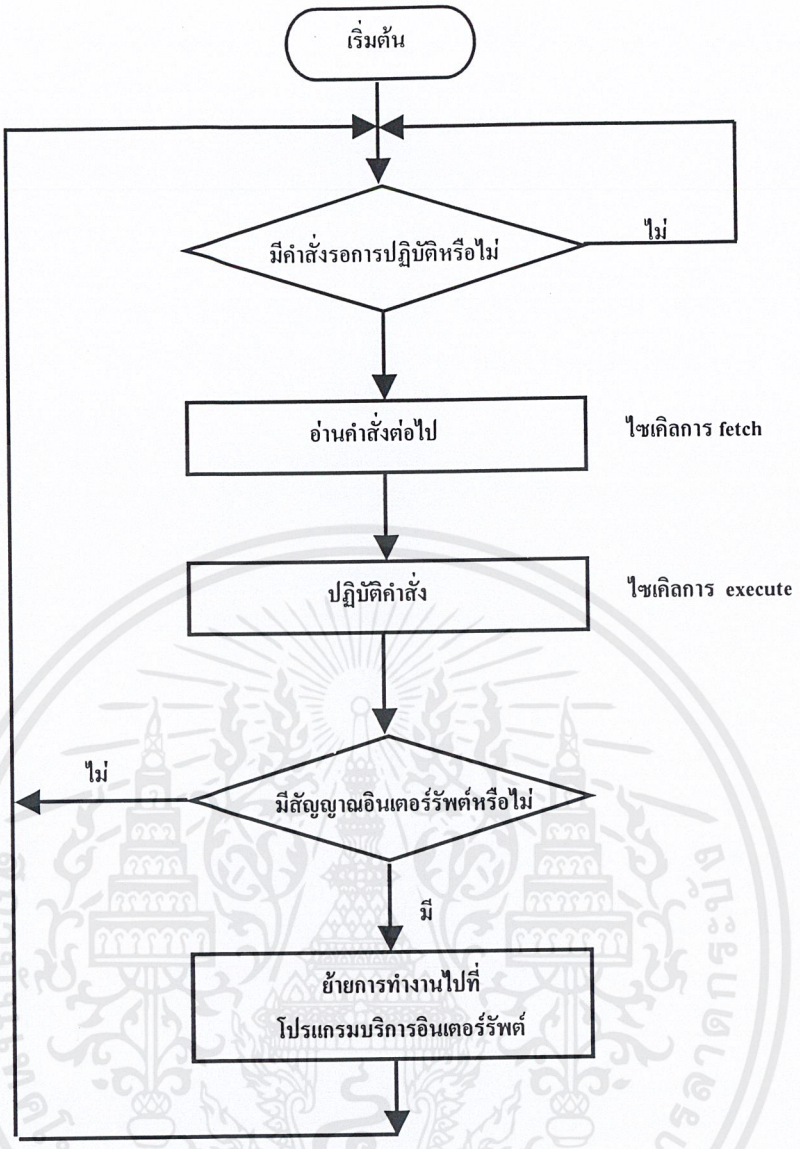


รูปที่ 2.2 แสดงรอบคำสั่ง (Instruction cycle)

เมื่อทำคำสั่งหนึ่งเสร็จเรียบร้อยแล้วก็จะ fetch และ execute คำสั่งถัดไปตามลำดับ พฤติกรรมของซีพียู ใน 1 รอบคำสั่งจะกำหนดโดยลำดับของการทำงานย่อย (Micro Operation) หรือ Machine Cycle ซึ่งในแต่ละ micro operation นี้ก่อให้เกิดการส่งถ่ายข้อมูลในรีจิสเตอร์ภายใน และการเกิดสัญญาณควบคุมต่างๆ เวลาที่ซีพียูใช้ในการทำงานของ micro operation ต่างๆ เรียกว่า Cycle Time และค่านี้เป็นค่าพื้นฐานสำหรับการคำนวณหาเวลาทำงานของโปรแกรมเวลาในการทำงานขึ้นอยู่กับความถี่ของสัญญาณนาฬิกา(Clock Rate) ซึ่งโดยทั่วไปใช้เป็นเมกกะเฮิร์ท ความถี่ของสัญญาณนาฬิกานี้ ขึ้นอยู่กับเทคโนโลยีของการผลิตอุปกรณ์ที่มาใช้สร้าง CPU

ในการ execute โปรแกรม ซีพียูยังต้องคอยตรวจสอบการทำงานของระบบ อื่นๆ อีกด้วย ซึ่งโดยทั่วไปจะใช้สัญญาณควบคุมพิเศษตัวอย่างเช่น เมื่อซีพียูควบคุมการทำงานของอุปกรณ์อินพุท/เอาท์พุท ที่ต้องการการตอบสนองอย่างรวดเร็วจากซีพียู อุปกรณ์อินพุท/เอาท์พุท จะมีสัญญาณในการขอร้องการบริการที่เรียกว่าสัญญารณการขัดจังหวะ (interrupt) เพื่อส่งมาที่ซีพียู ในการเกิดการอินเตอร์รัพต์นี้ เมื่อซีพียูได้รับสัญญาณอินเตอร์รัพต์ ซีพียูจะหยุดการทำงานตามโปรแกรมที่กำลังทำอยู่ชั่วขณะ และย้ายไปทำงานที่โปรแกรมบริการการอินเตอร์รัพต์ ตามตำแหน่งที่กำหนดไว้ด้วยวิธีของการอินเตอร์รัพต์ ซึ่งโดยทั่วไป CPU จะตรวจสอบสัญญาณการอินเตอร์รัพต์ที่ Cycle สุดท้ายของแต่ละคำสั่ง ซึ่งพฤติกรรมการทำงานของซีพียูแสดงได้ดังรูปที่ 2.3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

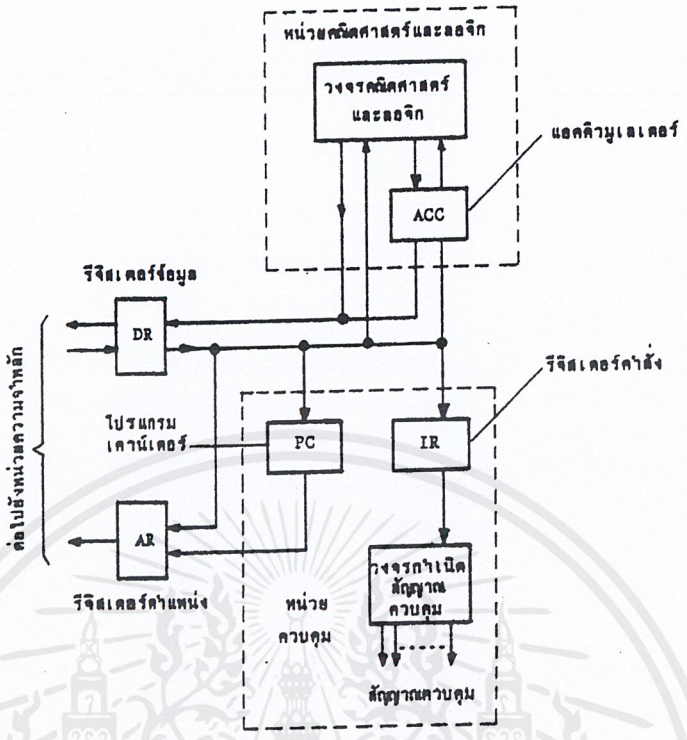


รูปที่ 2.3 แสดงพฤติกรรมการทำงานของ CPU

2.1 โครงสร้างพื้นฐานของซีพียู

โครงสร้างพื้นฐานของซีพียูแสดงได้ดังรูปที่ 2.4 โดยที่ซีพียูจะใช้รีจิสเตอร์แอคคิวมูเลเตอร์(Acumulator) เป็นที่เก็บตัวกระทำหลัก(Main Operand) ของหน่วยกระทำทางคณิตศาสตร์และลอจิก (Arthimetic Logic Unit:ALU) และยังใช้เก็บผลลัพธ์ที่เกิดจากการกระทำของ ALU ด้วย ส่วนรีจิสเตอร์ข้อมูล (Data Register) ทำหน้าที่เป็นบัฟเฟอร์ระหว่างซีพียู และหน่วยความจำหลัก ซึ่งรีจิสเตอร์นี้ใช้เพื่อเป็นอินพุตของตัวกระทำอีกตัวหนึ่งที่จะมากระทำกับแอคคิวมูเลเตอร์ ส่วนโปรแกรมเคาน์เตอร์(Program Counter) มีหน้าที่คือใช้เก็บตำแหน่งของคำสั่งต่อไป รีจิสเตอร์คำสั่ง(Instruction Register) ใช้เก็บรหัสของคำสั่งขณะนั้นๆ เพื่อนำไปตีความ (decode) รีจิสเตอร์ตำแหน่ง(Address Register) เป็นรีจิสเตอร์ที่ใช้เก็บตำแหน่งของหน่วยความจำที่จะถูกเข้าถึง

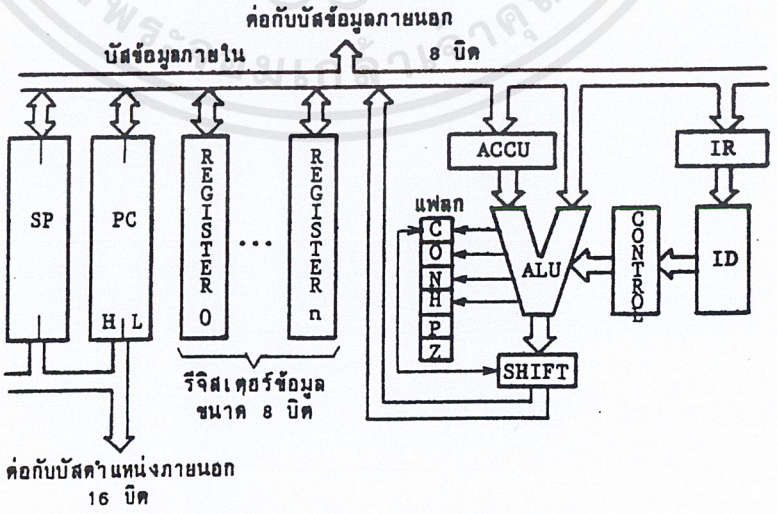
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.4 แสดงโครงสร้างพื้นฐานของซีพียู

2.2 สถาปัตยกรรมภายในของซีพียูขนาด 8 บิต

โครงสร้างของซีพียูขนาด 8 บิตโดยทั่วไปในปัจจุบันจะมีสถาปัตยกรรมที่ใช้คล้ายๆ กัน ดังแสดงในรูปที่ 2.5



รูปที่ 2.5 แสดงสถาปัตยกรรมภายในของซีพียูขนาด 8 bit

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ส่วนประกอบภายในที่สำคัญของซีพียูประกอบด้วย วงจรคณิตศาสตร์และลอจิก(ALU) รีจิสเตอร์คำสั่ง (IR) วงจรถอดรหัสคำสั่ง(ID) หน่วยควบคุม (Control Unit) และ รีจิสเตอร์ตำแหน่ง (Address Register) ส่วนประกอบที่สำคัญภายในเหล่านี้จะติดต่อกันด้วยบัสภายใน (internal Bus) และ บัสภายในนี้จะต่อกับบัสภายนอกโดยผ่านบัฟเฟอร์ เพื่อเป็นการติดต่อกับวงจรภายนอกอีกทีหนึ่ง ซึ่งหน้าที่ของหน่วยต่างๆ อธิบายได้ดังนี้

หน่วยควบคุม (Control Unit)

เป็นหน่วยที่ใช้สร้างสัญญาณเพื่อควบคุมการทำงานภายในของซีพียู ให้ทำงานอย่างมีระเบียบและสัมพันธ์กัน โดยที่หน่วยควบคุมนี้จะรับสัญญาณมาจากวงจรถอดรหัสคำสั่งซึ่งจะถอดรหัสคำสั่งจากรีจิสเตอร์คำสั่ง ดังนั้นการทำงานภายในซีพียูจึงขึ้นอยู่กับคำสั่งที่ป้อนเข้ามาให้กับซีพียู

หน่วยคณิตศาสตร์และลอจิก (Arithmetic Logic Unit)

หน่วยนี้มีหน้าที่กระทำทางคณิตศาสตร์และลอจิก หน่วย ALU นี้จะใช้รีจิสเตอร์พิเศษตัวหนึ่งเป็น อินพุท ซึ่งเรียกว่า แอคคิวมูเลเตอร์ แอคคิวมูเลเตอร์นี้จะเป็นได้ทั้ง อินพุทและเอาท์พุทของ ALU นอกจากนี้ยังสามารถใช้ในการเลื่อน(Shift) และหมุน(Rotate) ข้อมูลได้อีกด้วย

วงจรเลื่อนข้อมูล(Shifter Unit)

ใช้ในการเคลื่อนย้ายข้อมูลใน 1 byte ไปทางซ้ายหรือทางขวา 1 ตำแหน่งหรือมากกว่า การเลื่อนข้อมูลนี้อาจเลื่อนผ่านแฟลกตัวทด(Carry Flag) หรือไม่ก็ได้

รีจิสเตอร์แสดงสถานะ(Status register)

รีจิสเตอร์แสดงสถานะหรือ แฟลก ใช้เก็บเงื่อนไขพิเศษต่างๆ ที่เกิดขึ้นในซีพียู ข้อมูลในแฟลกสามารถตรวจสอบได้โดยคำสั่งบางคำสั่ง คำสั่งประเภทนี้มีการตรวจสอบสถานะที่แฟลก

รีจิสเตอร์(Register)

จากสถาปัตยกรรมของซีพียู ดังรูปที่ 2.2 เราอาจแบ่งรีจิสเตอร์ที่อยู่ในซีพียูออกเป็น 2 กลุ่ม คือ รีจิสเตอร์ใช้งานทั่วไป(General purpose register) และรีจิสเตอร์สำหรับการอ้างตำแหน่ง (Address register) ซึ่งรีจิสเตอร์ทั้ง 2 อย่างเราสามารถ อธิบายการทำงานได้ดังนี้คือ

รีจิสเตอร์ใช้งานทั่วไป

ในซีพียุขนาด 8 บิต รีจิสเตอร์จะมีขนาด 8 บิต หน้าที่ของรีจิสเตอร์เหล่านี้ไม่ได้ถูกกำหนดให้เฉพาะเจาะจงลงไป แต่โดยทั่วไป จะใช้สำหรับเก็บข้อมูลเพื่อให้ ALU กระทำกับข้อมูลต่างๆ เหล่านั้นด้วยความเร็วสูง เนื่องจากไม่ต้องติดต่อกับหน่วยความจำที่อยู่ภายนอก นอกจากนี้ซีพียูบางตัว ยังสามารถนำรีจิสเตอร์ 2 ตัวมาต่อร่วมกันได้เรียกว่า คู่รีจิสเตอร์ (Register pair) โดยที่คู่รีจิสเตอร์นี้จะกลายเป็นรีจิสเตอร์ขนาด 16 บิต หรือใช้เป็นตัวชี้ตำแหน่งข้อมูลในหน่วยความจำก็ได้ ซึ่งจะขึ้นอยู่กับคำสั่งที่ใช้

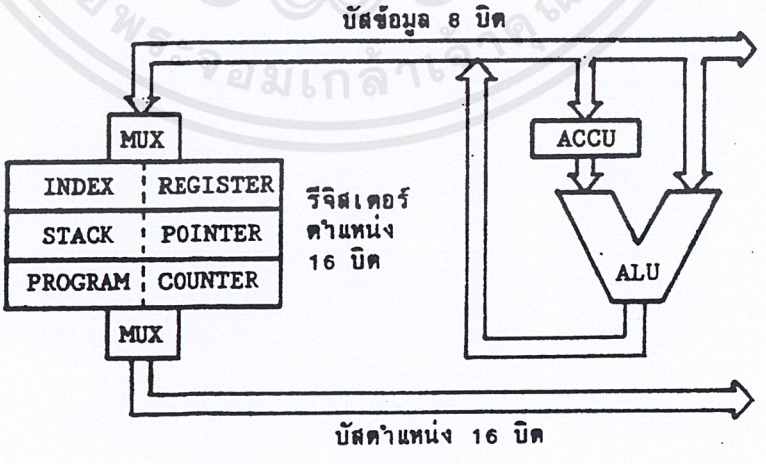
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รีจิสเตอร์สำหรับการอ้างตำแหน่ง

หน้าที่ของรีจิสเตอร์นี้ ใช้สำหรับเก็บตำแหน่งของหน่วยความจำที่ต้องการอ้างถึง ซึ่งอาจเป็นการอ้างถึงโดยคำสั่งหรือการอ้างถึงโดยระบบก็ตาม ขนาดของรีจิสเตอร์นี้อาจเป็น 8 บิต หรือ 16 บิตก็ได้ ขึ้นอยู่กับชนิดของซีพียูนั้นๆ ในซีพียูทั่วๆ ไปควรจะต้องมีรีจิสเตอร์สำหรับอ้างตำแหน่งอย่างน้อย 2 ตัว คือ โปรแกรมเคาน์เตอร์(PC) และสแตคพอยท์เตอร์ (SP) ส่วนรีจิสเตอร์ตัวอื่นๆ อาจมีหรือไม่ก็ได้

โปรแกรมเคาน์เตอร์(Program counter หรือ PC) ต้องมีอยู่ในซีพียูทุกตัว ข้อมูลในโปรแกรมเคาน์เตอร์คือตำแหน่งของคำสั่งต่อไป ที่ซีพียูจะต้องอ่านมาเพื่อปฏิบัติ กลไกการปฏิบัติตามรหัสคำสั่ง และลำดับขั้นของการทำงานตามคำสั่งที่วางไว้ จะกำหนดโดยข้อมูลที่อยู่ใน PC นี้เอง กล่าวโดยสรุปคือ การปฏิบัติโปรแกรมจะเป็นไปแบบเรียงลำดับ และเพื่อที่จะดึงคำสั่งต่อไปซีพียูจำเป็นต้องดึงคำสั่งมาจากหน่วยความจำ ซึ่งขบวนการนี้ข้อมูลใน PC จะส่งมาบนบัสตำแหน่งและส่งต่อไปยังหน่วยความจำ หน่วยความจำจะอ่านข้อมูลจากตำแหน่งที่ถูกอ้างอิง และส่งข้อมูลที่อ่านได้บนบัสข้อมูลซีพียูจะอ่านข้อมูลบนบัสข้อมูลนั้น ซึ่งข้อมูลหรือคำสั่งที่อ่านได้นี้คือคำสั่ง (Operation Code) นั่นเอง

สแตคพอยท์เตอร์(Stack pointer หรือ SP) ซีพียูที่มีความสามารถสูงโดยทั่วไปจะต้องมีสแตค ซึ่งสแตคอาจเป็นรีจิสเตอร์ที่อยู่ในซีพียูนั้นเองหรือใช้หน่วยความจำภายนอกส่วนหนึ่ง เพื่อกำหนดให้เป็นสแตค ในการเก็บรักษาตำแหน่งสูงสุดของสแตคที่อยู่ในหน่วยความจำ จะใช้รีจิสเตอร์ที่เรียกว่า สแตคพอยท์เตอร์ (SP) สแตคและสแตคพอยท์เตอร์เป็นสิ่งที่จำเป็น และขาดไม่ได้ในการทำโปรแกรมย่อย (Subroutine) และการทำโปรแกรมเกี่ยวกับการอินเตอร์รัพต์



รูปที่ 2.6 แสดงรีจิสเตอร์ตำแหน่ง ที่ใช้สร้างข้อมูลบนบัสตำแหน่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สแตค(Stack)

สแตค คือ กลุ่มของรีจิสเตอร์ หรือส่วนหนึ่งของหน่วยความจำที่ถูกจัดเตรียมไว้ สามารถจัดแบ่งออกได้เป็น 2 อย่าง

1. ฮาร์ดแวร์สแตค (Hardware Stack) สแตคแบบนี้เป็นรีจิสเตอร์ที่กำหนดไว้คงที่ภายในตัวซีพียูเอง ซึ่งมีข้อดีในเรื่องของความเร็วในการทำงาน แต่มีข้อเสียคือ จำนวนของรีจิสเตอร์ที่ใช้เป็นสแตคจะจำกัด

2. ซอฟต์แวร์สแตค (Software Stack) ซีพียูที่ใช้งานโดยทั่วๆ ไป จะใช้ซอฟต์แวร์สแตค เพราะจำนวนของสแตคไม่ได้ถูกจำกัดด้วยจำนวนของรีจิสเตอร์ที่อยู่ภายในแต่จะใช้ส่วนหนึ่งของหน่วยความจำ และใช้คำสั่งในการกำหนดตำแหน่งเริ่มต้นของสแตคตำแหน่งบนสุดของสแตค จะชี้โดยข้อมูลที่มีอยู่ในสแตคพอยท์เตอร์

ลักษณะโครงสร้างของสแตคทั้งสองจะเป็นแบบเรียงลำดับ (Chronological Structure) คือ ข้อมูลที่ใส่ลงไปในสแตคข้อมูลแรกจะอยู่ที่ชั้นใต้สุดของสแตค และข้อมูลที่ใส่เข้ามาทีหลังจะวางทับด้านบนต่อไปเรื่อยๆ ส่วนการนำข้อมูลออก ข้อมูลที่อยู่บนสุด หรือข้อมูลที่ใส่เข้าไปทีหลังที่สุดจะถูกนำออกมาก่อน ดังนั้นโครงสร้างในลักษณะนี้จึงเรียกได้ว่าเป็นโครงสร้างแบบ LIFO (Last – In – First - Out)

สแตคมีประโยชน์ในการทำโปรแกรม 3 ประเภท คือ โปรแกรมย่อย (Subroutine) โปรแกรมการบริการขัดจังหวะ (Interrupt Routine) และการใช้ที่เก็บข้อมูลชั่วคราว (Temporary Data Register) ของโปรแกรมต่างๆ

คำสั่งโดยทั่วไปสำหรับซีพียู ที่ใช้สำหรับการนำข้อมูลเข้า และออกจากสแตคโดยตรงมีอยู่ 2 คำสั่ง คือ push และ pop (หรือ pull) คำสั่ง push มีผลทำให้ใส่ข้อมูลลงไปในสแตค ส่วนคำสั่ง pop ทำการดึงข้อมูลออกจากสแตค และตำแหน่งบนสุดของสแตคจะแสดงที่สแตคพอยท์เตอร์รีจิสเตอร์เสมอ ซึ่งโดยทั่วไปตำแหน่งบนสุดของสแตค หมายถึง ตำแหน่งหน่วยความจำที่มีค่าน้อยที่สุดของหน่วยความจำที่ใช้เป็นสแตค

2.3 การอินเตอร์รัพต์

บริการขัดจังหวะ (Interrupt Service Routine) อาจเรียกทับศัพท์ว่า การอินเตอร์รัพต์ หมายถึง การมีเหตุการณ์อื่นเข้ามาแทรกซ้อนงานที่กำลังทำอยู่ตามปกติ ในการทำงานของซีพียูนั้น เมื่อซีพียูได้รับสัญญาณอินเตอร์รัพต์จากอุปกรณ์ภายนอก อาจทำให้ซีพียูหยุดการกระทำโปรแกรมที่กำลังทำอยู่ แล้วไปทำบริการอุปกรณ์ที่ส่งสัญญาณการอินเตอร์รัพต์เข้ามา และเมื่อทำโปรแกรมบริการอินเตอร์รัพต์นั้นๆ เสร็จเรียบร้อยแล้ว ก็จะกลับไปทำงานที่โปรแกรมเดิมต่อไป การทำโปรแกรมบริการการอินเตอร์รัพต์มีลักษณะคล้ายกับการเรียกใช้โปรแกรมย่อย(Subroutine) แต่จะ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

มีข้อแตกต่างกัน คือ โปรแกรมย่อยจะถูกเรียกใช้โดยคำสั่ง call ซึ่งผู้เขียนโปรแกรมได้กำหนดขึ้นว่าจะเรียกใช้เมื่อใด แต่การอินเตอร์รัพต์ จะมีผลทำให้ข้ามไปทำงานอีกโปรแกรมหนึ่ง เมื่อซีพียูได้รับการอินเตอร์รัพต์จากภายนอก ซึ่งกำหนดไม่ได้ว่าสัญญาณการอินเตอร์รัพต์นี้จะเกิดขึ้นเมื่อใด

การอินเตอร์รัพต์ คือการติดต่อเพื่อรับส่งข้อมูลกันระหว่างอุปกรณ์ภายนอกต่างๆ ของคอมพิวเตอร์ เช่น จอภาพ แป้นพิมพ์ เครื่องพิมพ์ เมาส์ และอื่นๆ กับซีพียู ซึ่งจะมีการติดต่อกันอยู่เสมอๆ การที่จะทำให้ระบบมีประสิทธิภาพมากที่สุดนั้น ก็คือ การมีการติดต่อ หรือการอินเตอร์รัพต์ที่ดีนั่นเอง

ในระบบที่มีอุปกรณ์ต่างๆ หลายชนิดนั้น จะสังเกตเห็นว่าในขณะที่นาฬิกาของเครื่องเดินไปได้ตลอดเวลาพร้อมๆ กับที่ใช้งานอย่างอื่น โดยที่เวลาเดินไปอย่างไม่ผิดพลาด หรือว่าเครื่องได้อ่านแผ่นดิกส์อยู่ ในขณะเดียวกับที่พิมพ์ข้อมูลผ่านทางแป้นพิมพ์ได้ ซึ่งเสมือนว่าเครื่องสามารถทำงานได้หลายๆ อย่างในเวลาเดียวกัน ซึ่งการที่จะทำเช่นนี้ได้นั้น ซีพียูจำเป็นต้องมีวิธีการติดต่อกับอุปกรณ์ภายนอกต่างๆ อย่างมีประสิทธิภาพ วิธีหนึ่ง ก็คือ ให้ซีพียูใช้เวลาส่วนใหญ่ในการทำโปรแกรมหลัก และหันมาสนใจอุปกรณ์ภายนอก ก็ต่อเมื่ออุปกรณ์ภายนอกส่งสัญญาณเข้ามาขัดจังหวะการทำงานที่ซีพียูทำอยู่ในขณะนั้น ซึ่งซีพียูอาจจะยอมรับ หรือปฏิเสธการขัดจังหวะนี้ก็ได้ หากว่าซีพียูยอมรับการขัดจังหวะนี้ ก็จะหยุดจากการทำงานในโปรแกรมหลักและไปทำงานในโปรแกรมย่อยต่อไป ลักษณะเช่นนี้เรียกว่า การอินเตอร์รัพต์ (interrupt)

การอินเตอร์รัพต์ นี้สามารถแบ่งออกได้เป็น 2 ประเภทใหญ่ๆ คือ

1. Non – Maskable Interrupt (NMI) คือการอินเตอร์รัพต์ที่ซีพียูไม่สามารถปฏิเสธได้ ซีพียูจะต้องหยุดการทำงานของโปรกรมที่ทำอยู่ในขณะนั้นโดยไม่มีข้อแม้ และเปลี่ยนไปทำงานให้อุปกรณ์ที่มีการส่งอินเตอร์รัพต์เข้ามา
2. Maskable Interrupt(INT) คือการอินเตอร์รัพต์ที่ซีพียูสามารถปฏิเสธได้ โดยปกติซีพียูจะมีการกำหนดว่าในขณะนั้น ซีพียูจะอยู่ในภาวะ Disable Interrupt (DI) หรือ Enable Interrupt (EI) ถ้าหากว่า ในขณะที่มีการอินเตอร์รัพต์เข้ามานั้น ซีพียูอยู่ในภาวะ Disable Interrupt แล้ว การอินเตอร์รัพต์นั้นก็จะได้รับการปฏิเสธไป แต่ถ้าเป็น Enable Interrupt การอินเตอร์รัพต์นั้นก็จะได้รับการสนองตอบ

ถึงแม้ว่าจะมีวงจรจากอุปกรณ์ภายนอกมากมายที่สามารถสร้าง และส่งสัญญาณการอิน-เตอร์รัพต์มาให้ซีพียูได้ แต่ซีพียูก็สามารถแยกแยะได้ว่าวงจรใดเป็นผู้สร้างสัญญาณอินเตอร์รัพต์นั้นขึ้นมา เนื่องจากแต่ละวงจรจะให้อินเตอร์รัพต์เวกเตอร์ที่ต่างกัน ทำให้ซีพียูกระโดไปทำงานอินเตอร์รัพต์ต่างๆ กันได้ และแม้ว่าซีพียูจะสามารถแยกแยะได้ว่าอุปกรณ์ใดเป็นผู้ส้งสัญญาณอินเตอร์รัพต์ แต่ในทางปฏิบัติแล้ว เมื่อมีอุปกรณ์มากกว่าหนึ่งอุปกรณ์ที่ต้องการจะติดต่อกับซีพียูพร้อมๆ กันจะเกิดปัญหาที่ว่าซีพียูจะติดต่อกับอุปกรณ์ใดก่อน จึงต้องจัดความสำคัญของอุปกรณ์ให้ไม่เท่ากัน (Priority Arbitration) อุปกรณ์ภายนอกจะไม่ต่อสัญญาณอินเตอร์รัพต์ของตนเข้าโดยตรงกับ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ซีพียู แต่จะผ่านวงจรจัดลำดับความสำคัญ ถ้ามีอุปกรณ์หลายตัวต้องการที่จะส่งสัญญาณการอินเตอร์รัพต์ (แบบ INT) เข้าซีพียูพร้อมๆ กัน วงจรจัดลำดับความสำคัญจะส่งผ่านสัญญาณอินเตอร์รัพต์ และอินเตอร์รัพต์เวกเตอร์ของอุปกรณ์ที่มีความสำคัญสูงสุด(Priority) ไปยังซีพียู ซีพียูก็จะตอบรับ และให้บริการอุปกรณ์ที่ส่งอินเตอร์รัพต์นั้นมาให้เสร็จเสียสิ้นก่อน แล้วจึงเริ่มให้ความสนใจแก่อินเตอร์รัพต์ของอุปกรณ์ที่มีความสำคัญรองลงไป

เราสามารถควบคุมการอินเตอร์รัพต์ได้ คือ สามารถควบคุมได้ว่าต้องการให้ซีพียูตอบรับอินเตอร์รัพต์หรือไม่ เนื่องจากในบางขณะที่ซีพียูกำลังทำงานที่ต่อเนื่องอยู่ เช่น โปรแกรมนาฬิกา ซึ่งหากว่ามีการอินเตอร์รัพต์เข้ามาก็จะก่อให้เกิดความผิดพลาดของนาฬิกา ดังนั้นจึงต้องมีการบังคับไม่ให้มีการตอบสนองได้ คำสั่งที่ใช้ในการเซต หรือรีเซตแฟลกอินเตอร์รัพต์ เรียกว่า คำสั่งอินเตอร์รัพต์ ซึ่งได้แก่

DI (Disable Interrupt) เป็นการเซตให้ซีพียูอยู่ในภาวะที่ไม่ตอบสนอง

EI (Enable Interrupt) เป็นการเซตให้ซีพียูอยู่ในภาวะที่ตอบสนอง

ในซีพียูนั้น การอินเตอร์รัพต์จะเกิดขึ้นโดยอุปกรณ์อินพุท/เอาท์พุท ได้ส่งสัญญาณอิเล็กทรอนิกส์ เพื่อไปเปลี่ยนระดับสัญญาณที่ขาใดขาหนึ่งของซีพียู ซึ่งขานี้เรียกว่า ขาอินเตอร์รัพต์ (Interrupt Pin) เพื่อเป็นการแจ้งการแก่ซีพียูถึงการอินเตอร์รัพต์ โดยมีขั้นตอนดังนี้

1. วงจรภายนอกส่งสัญญาณอินเตอร์รัพต์มายังซีพียู
2. หากซีพียูอยู่ในสภาวะ Enable Interrupt ก็จะตอบรับการอินเตอร์รัพต์ด้วยการส่งสัญญาณ Interrupt Acknowledge กลับไปยังวงจรภายนอก
3. เซตซีพียูให้อยู่ในสภาวะ Disable Interrupt เพื่อป้องกันการอินเตอร์รัพต์ซ้อน
4. หยุดการทำงานในโปรแกรม
5. วงจรภายนอกส่งอินเตอร์รัพต์เวกเตอร์มาให้ซีพียู เพื่อบอกตำแหน่งที่จะให้ซีพียูกระโดดไป
6. ซีพียูส่งค่าในรีจิสเตอร์ PC และรีจิสเตอร์ทั่วไป ไปเก็บไว้ในสแตค
7. ไปทำงานในโปรแกรมย่อย เพื่อบริการอุปกรณ์ที่ส่งอินเตอร์รัพต์เข้ามาจนเสร็จ
8. รับค่าที่เก็บไว้ในสแตคคืนสู่รีจิสเตอร์ PC และรีจิสเตอร์ทั่วไปตามเดิม
9. เซตซีพียูให้กลับสู่ภาวะ Enable Interrupt
10. ทำงานในโปรแกรมที่ค้างอยู่ต่อไป

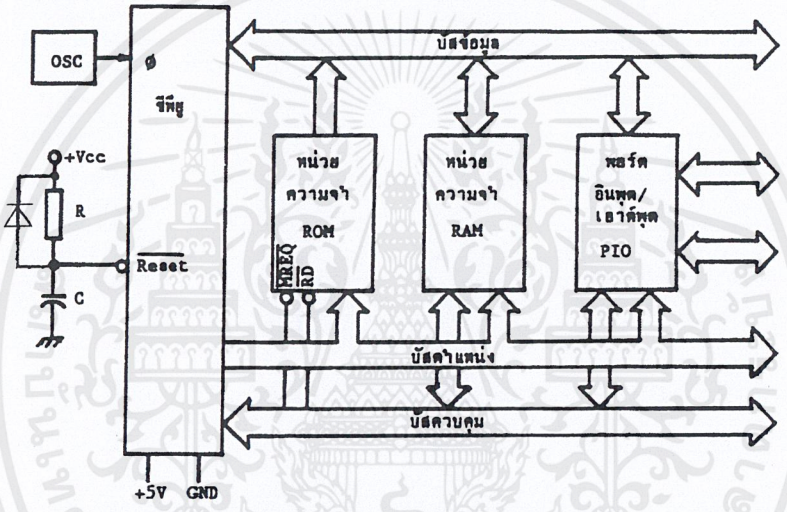
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4 สถาปัตยกรรมของระบบไมโครคอมพิวเตอร์

ไมโครคอมพิวเตอร์ จะมีทางเดินของสัญญาณต่างๆ ซึ่งเราเรียกเส้นทางเดินของสัญญาณที่ใช้เชื่อมโยงระหว่างหน่วยต่างๆ ของระบบนี้ว่า บัส (Bus) บัสที่มีใช้ระบบไมโครคอมพิวเตอร์นี้มีด้วยกัน 3 ชนิด คือ

- บัสข้อมูล (Data Bus)
- บัสตำแหน่ง (Address Bus)
- บัสควบคุม (Control Bus)

หน้าที่ของบัสต่างๆ จะอธิบายได้ดังนี้



รูปที่ 2.7 ระบบไมโครคอมพิวเตอร์พื้นฐาน

บัสข้อมูล (Data Bus) เป็นบัส 2 ทิศทาง ที่ใช้เป็นเส้นทางสำหรับนำพาข้อมูลที่เคลื่อนย้ายระหว่าง ส่วนต่างๆ ของระบบ ซึ่งส่วนมากจะเป็นการเคลื่อนย้ายข้อมูล ระหว่างหน่วยความจำกับซีพียู หรือระหว่างซีพียูกับอุปกรณ์อินพุท/เอาท์พุท

บัสตำแหน่ง (Address Bus) เป็นบัสทิศทางเดียว เป็นเส้นทางที่ใช้สำหรับนำพาสัญญาณเพื่อกำหนดตำแหน่งที่สร้างขึ้นโดยซีพียู สัญญาณนี้ใช้ในการกำหนดตำแหน่งของหน่วยความจำเพื่อจะนำข้อมูลออกมาสู่ซีพียู หรือนำข้อมูลจากซีพียูเข้าไปเก็บ หรือใช้ในการกำหนดตำแหน่งของอุปกรณ์ อินพุท/เอาท์พุท

บัสควบคุม (Control Bus) เป็นเส้นทางผ่านของสัญญาณควบคุมต่างๆ ที่ใช้ในระบบ เพื่อให้การทำงานของระบบสอดคล้องกัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ระบบไมโครคอมพิวเตอร์ โดยทั่วไปจะต้องประกอบด้วย บัสทั้ง 3 นี้เสมอ

ซีพียูทุกตัว ต้องการสัญญาณเพื่อการเปรียบเทียบเวลาการทำงาน ซึ่งสัญญาณนี้คือสัญญาณนาฬิกา ซึ่งสร้างขึ้นโดยวงจรสร้างสัญญาณนาฬิกา โดยอาจใช้วงจรคริสตรอลออสซิลเลเตอร์หรือวงจรออสซิเลเตอร์แบบใดๆ ก็ได้ที่ให้สัญญาณเป็นรูปสี่เหลี่ยมและมีความถี่เท่าที่ต้องการ

ROM (Read Only Memory) เป็นหน่วยความจำที่เก็บข้อมูลแบบถาวรและสามารถนำข้อมูลภายในไปใช้ได้อย่างเดียว ข้อมูลใน ROM นี้จะคงอยู่ตลอดไปไม่สูญหายแม้ว่าไม่มีแหล่งจ่ายไฟ (Non Volatile) หน่วยความจำแบบ ROM นี้ใช้สำหรับเก็บข้อมูลที่ใช้เป็นโปรแกรมควบคุมการทำงานของระบบหรือที่เรียกว่า โปรแกรมมอนิเตอร์ ซึ่งโปรแกรมส่วนนี้เป็นโปรแกรมเพื่อทำให้ระบบไมโครคอมพิวเตอร์เริ่มทำงาน (bootstrap) ในการใช้ระบบไมโครคอมพิวเตอร์ในงานควบคุมขบวนการ (Process Control) ซึ่งส่วนมากโปรแกรมควบคุมนี้จะคงที่ ดังนั้นโปรแกรมควบคุมนี้จะอยู่ใน ROM เพื่อป้องกันปัญหาที่เกิดเนื่องจากไฟฟ้าดับ

RAM (Random Access Memory) เป็นหน่วยความจำของระบบที่สามารถอ่านข้อมูลออกมาหรือเขียนข้อมูลเข้าไปก็ได้ ดังนั้นข้อมูลใน RAM นี้เป็นข้อมูลที่ไม่ถาวรสามารถเปลี่ยนแปลงได้ RAM เป็นหน่วยความจำที่ต้องการไฟเลี้ยงตลอดเวลา (Volatile) คือ เมื่อไฟดับข้อมูลใน RAM จะหายไป หน้าที่ของหน่วยความจำแบบ RAM นี้ ใช้สำหรับเก็บข้อมูลที่มีการเปลี่ยนแปลงตลอดเวลา เช่น ใช้เก็บข้อมูลชั่วคราว(Temporary) ต่างๆ ใช้ทำเป็นสแตค เป็นต้น ในงานควบคุมขบวนการจะใช้หน่วยความจำแบบ RAM น้อยมาก เนื่องจากโปรแกรมควบคุมคงที่ แต่ในงานพัฒนาระบบ (Program Development) จะใช้หน่วยความจำแบบ RAM นี้มาก เนื่องจากง่ายต่อการเปลี่ยนแปลงแก้ไข

PIO (Parallel Input/output) เป็นอุปกรณ์ที่ใช้เชื่อมต่อสื่อสาร ระหว่างซีพียูกับอุปกรณ์ต่างๆ ที่อยู่ภายนอกอุปกรณ์ ในการต่อระหว่างซีพียูกับหน่วยความจำ และตัวเชื่อมต่อต่างๆ จะต้องมีวงจรลอจิกภายนอกร่วมอยู่ด้วย

การทำงานของไมโครคอมพิวเตอร์ คำสั่งที่จะให้คอมพิวเตอร์ทำงานต้องถูกเตรียมมาก่อนแล้ว ลำดับของคำสั่งนี้เรียกว่า โปรแกรม ซึ่งโปรแกรมจะต้องถูกนำเข้ามาเก็บไว้ในหน่วยความจำด้วยวิธีการต่างๆ คำสั่งเหล่านี้จะถูกเก็บไว้ในส่วนใดส่วนหนึ่งของหน่วยความจำ และเมื่อจะให้คอมพิวเตอร์ทำงานตามโปรแกรม ต้องทำให้ซีพียู เริ่มทำคำสั่งตั้งแต่จุดเริ่มต้นของโปรแกรมนั้น โดยการทำให้ค่าโปรแกรมเคาน์เตอร์ (PC) มีค่าเท่ากับ ตำแหน่งเริ่มต้นของคำสั่ง ที่เราต้องการให้ทำแล้วให้ซีพียูเริ่มทำงาน ซีพียูก็จะทำงานตามลำดับขึ้นไปเรื่อยจนกว่าจะจบโปรแกรมนอกจากว่าซีพียู ถูกขัดจังหวะ(interrupt) จากสัญญาณภายนอก ซึ่งจะมีผลให้การทำงานไม่เป็นไปตามลำดับของคำสั่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
ภาษาวีเอชดีแอล (VHDL)

วีเอชดีแอล(VHDL) ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC ย่อมาจาก Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง(High Level Language) ซึ่งใช้อธิบายการทำงานของระบบดิจิตอลฮาร์ดแวร์ สามารถใช้อธิบายฟังก์ชันการทำงานได้หลายๆ ระดับตั้งแต่ระดับบล็อก จนถึงระดับเกท ความซับซ้อนของระบบสามารถจะเขียนได้ตั้งแต่ระดับเกท ประกอบกันจนเป็นระบบที่สมบูรณ์ รูปแบบของภาษาวีเอชดีแอลนั้น จะประกอบไปด้วย 2 ส่วนใหญ่ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล(Sequential Language) และภาษาคอนเคอร์เรนท์(Concurrent Language) การโปรแกรมด้วยภาษาวีเอชดีแอล สามารถเขียนได้ทั้ง 2 รูปแบบรวมกัน เพราะในการทำงานของระบบใดๆ ย่อมจะมีการทำงานในแบบซีเควนเชียลและคอนเคอร์เรนท์ อยู่ร่วมกัน นอกจากนี้ตัวภาษาวีเอชดีแอลยังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยๆ เข้าไว้ด้วยกันเพื่อให้เป็นระบบใหญ่ได้ ตัวภาษาวีเอชดีแอลนอกจากจะกำหนดรูปแบบไวยากรณ์(Syntax) ของตัวภาษาแล้ว ยังมีการตรวจสอบความหมายของตัวภาษาว่าจะซิมูเลชั่น(Simulation) ได้หรือไม่ เพราะโปรแกรมที่เขียนโดยวีเอชดีแอลต้องผ่านการ Simulation เพื่อตรวจดูการทำงาน ฉะนั้นในการคอมไพล์ (Compile) จะมีการตรวจสอบทั้งไวยากรณ์และซิมูเลชั่นซีเมนติด (Simulation Semantics) อย่างไรก็ตามแม้ตัวภาษาจะมีความซับซ้อนในรูปแบบและกฎเกณฑ์ของภาษา แต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำไปใช้งานโดยไม่จำเป็นต้องศึกษารายละเอียดทั้งหมด เนื่องจากตัวภาษาวีเอชดีแอลออกแบบมาให้ใช้สำหรับการออกแบบตั้งแต่วงจรที่มีขนาดเล็กจนถึงวงจรที่มีขนาดใหญ่และซับซ้อน

3.1 ประวัติความเป็นมาของภาษาวีเอชดีแอล

วิวัฒนาการของภาษาวีเอชดีแอล นั้นเริ่มต้นประมาณปี ค.ศ.1981 โดยที่กระทรวงกลาโหมสหรัฐอเมริกาหรือ Department of Defence (DOD) มองเห็นว่าอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร เป็นอุปกรณ์ที่ได้รับการพัฒนาเมื่อประมาณ 20 ปีก่อน เพราะเทคโนโลยีในขณะนั้นทำให้การพัฒนาอุปกรณ์อิเล็กทรอนิกส์เป็นไปอย่างล่าช้า ซึ่งเป็นสภาพที่ไม่ค่อยยอมรับในปัจจุบัน เพราะเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์ ได้รับการพัฒนาไปอย่างรวดเร็ว ดังที่จะเห็นได้ว่ามีวงจรดิจิตอลอิเล็กทรอนิกส์หลายวงจร ที่แต่เดิมถูกสร้างขึ้นมาจากชิ้น ถูกนำประกอบกันอยู่บนแผงวงจรไฟฟ้า (Printed Circuit Board หรือ PCB) ที่มีขนาดใหญ่ แต่ในปัจจุบันสามารถที่จะใช้เทคโนโลยีการออกแบบ และผลิตวงจรรวมขนาดใหญ่มาก (VLSI: Very

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Large Scale Integration) รวมอุปกรณ์ต่างๆ เหล่านั้นให้อยู่บนชิ้นอุปกรณ์สารกึ่งตัวนำ ที่มีขนาดประมาณ 1 - 2 ตร.ซม. ซึ่งเป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้น(ความเร็วในการทำงานของวงจร) ตลอดจนความน่าเชื่อถือในการทำงาน และความคงทนต่อสภาพแวดล้อมสูง ขณะเดียวกันนั้นในวงการทหารได้มีการนำระบบคอมพิวเตอร์และอิเล็กทรอนิกส์ มาใช้ในระบบอาวุธอย่างแพร่หลาย ดังนั้นอุปกรณ์ที่มีใช้อยู่ จึงไม่เหมาะสมกับเทคโนโลยีด้านอาวุธของประเทศคู่แข่งขัน การที่จะเปลี่ยนอุปกรณ์ใหม่เป็นสิ่งที่ต้องใช้งบประมาณมาก และก็จะประสบกับปัญหาเช่นเดิม คือ อุปกรณ์ใหม่ได้รับการพัฒนามานานแล้วเช่นกัน เพราะในขณะนั้นขั้นตอนของการออกแบบ การผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ต้องใช้วิศวกร และเวลาสำหรับดำเนินการมาก ฉะนั้นทาง DOD จึงตั้งโครงการขึ้นมาเพื่อศึกษา วิธีการที่จะช่วยพัฒนาวงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำไปผลิตได้เร็วขึ้น และโครงการดังกล่าวมีชื่อว่า Very High Speed Integrated Circuits หรือ VHSIC ในระยะแรกนั้นโครงการเป็นความลับทางด้านความมั่นคงของประเทศ และอยู่ในความดูแลควบคุมของ United States International Traffic and Arms Regulations (ITAR) ในปี ค.ศ.1983 ตามคำแนะนำของคณะทำงาน ("Woods Hole" workshop) ทาง DOD ได้ออกความต้องการมาตรฐานของภาษาที่ใช้สำหรับบรรยายพฤติกรรมของวงจรหรือฮาร์ดแวร์ของระบบสำหรับ โครงการ VHSIC ซึ่งมีสาระสำคัญพอสรุปได้ดังนี้

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถจะเข้าใจได้ทั้งคนและเครื่องโดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง (High Level Language) เช่นเดียวกับภาษา PASCAL, FORTRAN และ ADA หรือภาษา C ซึ่งในทางวิศวกรรมการออกแบบฮาร์ดแวร์เรียกว่า "Hardware Description Language" หรือ HDL ดังนั้นภาษามาตรฐานนี้จึงมีชื่อว่า VHSIC – HDL หรือ VHDL นั่นเอง

เริ่มต้น DOD ได้มอบหมายให้บริษัท IBM, TEXAS INSTRUMENT, INTERMETRICS 3 บริษัทร่วมกันเป็นผู้ศึกษาและพัฒนา การดำเนินการได้กระทำไปอย่างต่อเนื่อง และได้ผลเป็นที่น่าพอใจจนกระทั่งปี ค.ศ.1985 ทาง ITAR ได้ยกเลิกข้อจำกัดในการถ่ายทอดเทคโนโลยีทางทหาร ออกจากโครงการนี้ ดังนั้น VHDL จึงเริ่มเป็นที่รู้จักกันโดยทั่วไป จนกระทั่งทาง IEEE ได้รับภาษานี้เข้ามาศึกษา

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

และประมาณปี ค.ศ.1987 ได้ยอมรับกำหนดมาตรฐานของภาษา โดยให้ชื่อว่า IEEE1076-1987 และมีชื่อเรียกว่า VHDL มาตรฐานนี้ก็ได้รับการปรับปรุงจนปัจจุบัน ได้ชื่อว่า IEEE1076-1993 หรือ VHDL 1993 การที่ทาง DOD ขณะนั้น เป็นลูกค้ารายใหญ่ของอุตสาหกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ จึงมีผู้รับโครงการต่างๆ จาก DOD ไปดำเนินการด้านวิจัยและพัฒนามากเพื่อที่จะให้เป็นมาตรฐานเดียวกันหมด ทาง DOD จึงกำหนดว่า ในการส่งโครงการนั้นจะต้องเขียนอยู่ในรูปของภาษา VHDL เท่านั้น ซึ่งทำให้เกิดข้อดีต่อ DOD เองที่เป็นมาตรฐานเดียวกัน สามารถนำไปจำลองกับเครื่องคอมพิวเตอร์ได้หลายๆ ระบบ

3.2 ความสามารถของภาษาวีเอชดีแอล

- ตัวภาษาวีเอชดีแอลสามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิปกับผู้ออกแบบ (CAD Tools)
- ใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างซีเออี(CAE) และซีเอดีทูลส์(CAD Tools) เช่นตัวภาษา Source Code ของวีเอชดีแอล สามารถคอมไพล์โดยใช้ Compiler และ Simulator ได้หลายตัวต่างกัน
- ภาษาวีเอชดีแอลสนับสนุนการออกแบบ แบบ Top Down Design และแบบ Bottom Up Design หรือผสมกันทั้ง 2 แบบ
- ภาษาวีเอชดีแอลเป็นแบบทั่วไป คือไม่อิงเทคโนโลยีอันใดอันหนึ่งสามารถอิงเทคโนโลยีใดก็ได้ และในขณะเดียวกันก็สามารถสนับสนุนหลายๆ เทคโนโลยี
- สนับสนุนการออกแบบทั้งระบบ Synchronous และ Asynchronous
- สนับสนุนการออกแบบระบบดิจิตอลในหลายๆ เทคนิค เช่น Finite State Machine, Algorithmic, Boolean Equation
- ตัวภาษาวีเอชดีแอลสามารถอ่านและทำความเข้าใจได้โดยมนุษย์
- ภาษาวีเอชดีแอลเป็นมาตรฐานรับรองโดย IEEE และ ANSI ทำให้โมเดลที่ออกแบบโดยภาษาวีเอชดีแอล สามารถเคลื่อนย้ายไปยังระบบใดๆ ก็ได้ และสามารถนำกลับมาใช้ใหม่ได้
- ภาษาวีเอชดีแอลสนับสนุนรูปแบบการเขียนถึง 3 รูปแบบ ได้แก่ แบบ Behavioral Style, แบบ Structural Style, แบบ Data Flow หรือสามารถเขียนรวมกันทั้ง 3 รูปแบบ
- สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของส่วนประกอบ (Component) , Function procedure, package
- ไม่จำเป็นต้องศึกษา Software Simulator เพราะ Simulation Model สามารถเขียนได้โดยใช้ภาษาวีเอชดีแอล เช่นกัน

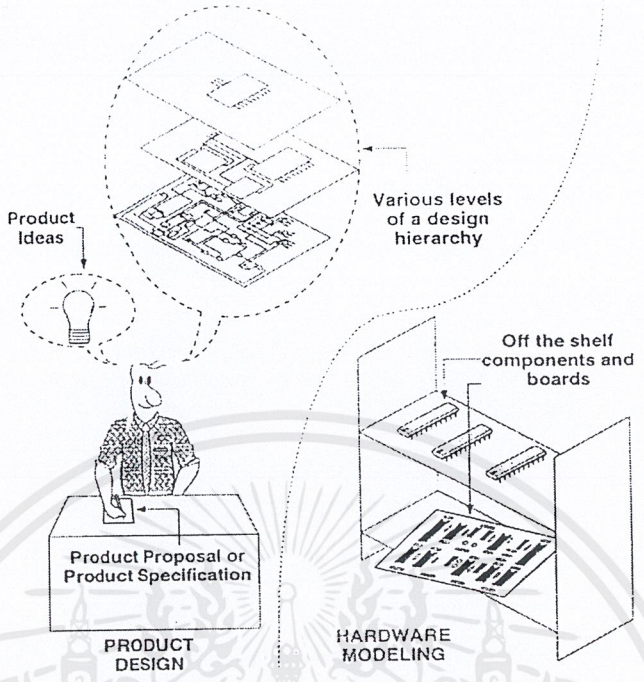
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

46299

- สามารถเขียนโมเดลได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของโมเดล(ขึ้นอยู่กับ Software)
- สามารถอธิบายตัวแปรที่เกี่ยวกับฟังก์ชั่นทางด้านเวลา เช่น Propagation Delay, Min-Max Delay, Setup, Holding time, Spike Detection สามารถอธิบายได้ภายในตัวภาษา
- Generics ช่วยให้เราสามารถสร้างตัวแปรของรูปแบบ
- โมเดลที่สร้างด้วยภาษาวีเอชดีแอลนั้น ไม่เพียงแต่จะอธิบายฟังก์ชั่นการทำงานเท่านั้นยัง สามารถอธิบายถึงรายละเอียดของตัวโมเดล เช่น Total Area และ Speed ของโมเดล
- ภาษาวีเอชดีแอลเป็นมาตรฐานที่ใช้โดยบริษัทและผู้ออกแบบหลายๆ แห่ง ฉะนั้นจึงเป็นการง่าย ที่จะทำความเข้าใจ ถึงแม้มาจากแหล่งต่างๆ
- โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะตัวแปลได้ตรวจสอบไวยากรณ์ทางด้าน ซิมูเลชันซีแมนติคไว้ด้วย
- การอธิบายโมเดลด้วยแบบ Behavior สามารถ Synthesis ไปเป็นระดับ Gate-Level ได้ ถ้าทำ ตามกฎของ Synthesis Guideline
- มีความสามารถที่ให้เราออกแบบข้อมูลใหม่ๆ ได้ ทำให้วีเอชดีแอลโมเดล เป็นการออกแบบ ระดับสูงที่ไม่ต้องคำนึงถึงว่าจะสร้างตัวโมเดลนั้นขึ้นมาได้อย่างไร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3 หลักการสร้างโมเดลโดยใช้ภาษาวิเอชดีแอล



รูปที่ 3.1 แสดงสิ่งต่างๆ ที่สามารถอธิบายด้วย VHDL ได้

วีเอชดีแอลเป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจง่ายซึ่งช่วยในการสร้างและออกแบบวงจรดิจิตอล และส่วนประกอบต่างๆ อาจใช้อธิบายระบบทั้งระบบ หรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของ Component Block จากนั้นก็ทำการจำลองการทำงานโดยที่รูปแบบนั้นยังไม่ได้สร้างขึ้นจริง หรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังจากการจำลองจนได้ตามที่ต้องการจึงนำไปทำการ Synthesis เพื่อให้ได้วงจร Gate-Level ต่อไป

ประโยชน์จริงของการใช้ VHDL เป็น Design Tools แทนการสร้างต้นแบบ (Prototype) ขึ้นมาจริง คือเราสามารถอธิบาย Product Idea, Product Proposal, Product Specification เป็นลักษณะในรูปของ Text จากนั้นก็นำไปคอมไฟล์เพื่อดู Timing การทำงาน จากนั้นก็ทำการ Refine แก้ไขจนกว่าจะได้ Specification ตามต้องการ เมื่อ Product ได้ผลตามที่ต้องการแล้วจึงนำไปเข้าสู่การ Synthesis เพื่อให้ได้ Schematic แล้วนำไปสร้างเป็นต้นแบบจริงต่อไป ซึ่งต้นแบบที่สร้างนั้นทำงานได้จริงเพราะได้ทำการ Simulate เรียบร้อยแล้ว เป็นการลดเวลาและค่าใช้จ่ายในการสร้างต้นแบบได้มาก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เนื่องจากว่าภาษา VHDL เป็นภาษาที่มีประสิทธิภาพสูง เราจึงใช้ภาษา VHDL อธิบายฮาร์ดแวร์ เพราะว่ามีข้อดี 2 ประการ คือ

1. เข้าใจง่าย
2. สามารถแก้ไขได้ง่าย

การเข้าใจได้ง่ายมีผลประโยชน์ต่อใครก็ได้ซึ่งมีความจำเป็นที่จะต้องอ่านรหัสที่ได้ออกแบบมาแล้วโดยไม่จำเป็นต้องให้ผู้ออกแบบมาอธิบายให้ฟัง ตัวภาษาวีเอชดีแอลอธิบายการทำงานภายในตัวอยู่แล้วส่วนอีกประการหนึ่งคือ ความต้องการในการเปลี่ยนแปลงฮาร์ดแวร์ ที่ได้ออกแบบแล้ว คือว่า หลังจากทดสอบแล้วพบข้อผิดพลาดซึ่งต้องแก้ไข หรือว่า ระหว่างพัฒนามีการเปลี่ยนแปลงความต้องการของระบบ หรือต้องการเพิ่มการทำงานบางส่วนลงไป ตัวภาษาวีเอชดีแอลนั้นสนับสนุนหลักการต่างๆ ให้เขียนแก้ไข และบำรุงรักษาวงจรดิจิตอล ที่มีความซับซ้อนเป็นไปอย่างรวดเร็ว และมีประสิทธิภาพ ซึ่งหลักการมีดังนี้

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

3.3.1 Top Down Design

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน เช่น ASIC (Application Specific Integrated Circuit) วิศวกรหรือผู้ออกแบบมักมองรูปแบบให้อยู่ในรูปของ Block Diagram เสียก่อน ก่อนที่จะย่อยรูปแบบ ให้ระลึกถึงรายละเอียดต่อไป ซึ่งภาษาวีเอชดีแอลนั้นอนุญาตให้

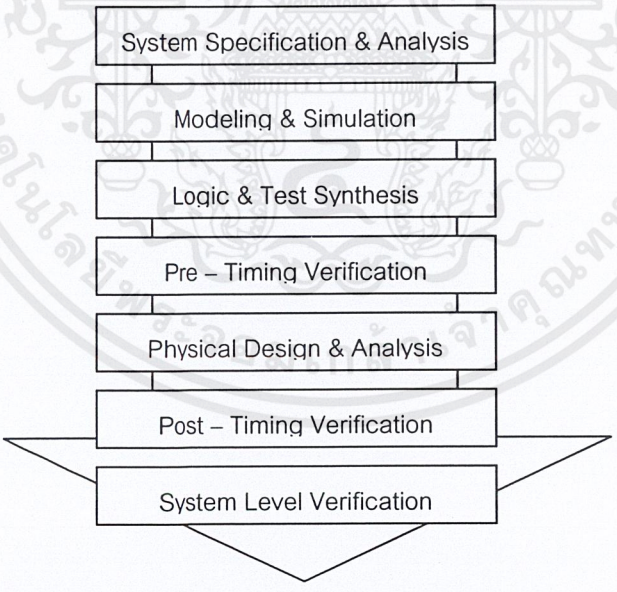
- อธิบายการทำงานของแต่ละ Block
- วิเคราะห์การทำงาน (Analyze)
- จัดการแก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ ก่อนที่จะทำการออกแบบให้ละเอียดลึกลงในขั้นตอนต่อไป การแก้ไขขั้นตอนนี้จะทำให้ลดค่าใช้จ่ายกว่าการแก้ไขในช่วงของการพัฒนาในระดับสร้างซิลิกอนชิป(Silicon Chip)

การเริ่มต้นด้วยวิธีการเขียนรูปแบบจากแนวความคิดอย่างสังเขป พร้อมทั้งการจำลองการทำงานของรูปแบบที่เขียนขึ้น เพื่อตรวจสอบความถูกต้อง ประกอบกับการกลั่นกรองเพิ่มเติมรายละเอียดลงสู่ระบบดิจิตอลที่สมบูรณ์ในรูปของวงจรไฟฟ้าทีละขั้น นั้นเป็นกระบวนการของ Top Down Design การที่เริ่มต้นด้วยการเขียนรูปแบบ (modeling) ในระดับบน (top-level) ของแนวความคิดอย่างสังเขปนั้น วิศวกรออกแบบสามารถที่จะพัฒนาสภาพแวดล้อมต่างๆ เพื่อการตรวจ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สอบการทำงานของวงจร (test environment) ได้ตั้งแต่ในระยะแรกๆ ของการออกแบบ เพื่อใช้สำหรับตรวจสอบความถูกต้องของแนวความคิดกับสิ่งที่ต้องการจริงหรือ specification ของงาน ดังนั้นจึงเป็นไปได้ยากที่ในระดับล่างลงมา โดยที่ได้เพิ่มรายละเอียดของรูปแบบให้มากขึ้นตามลำดับ จะเกิดข้อผิดพลาด หรือเบี่ยงเบนไปจากจุดประสงค์เดิม เพราะในแต่ละขั้นตอนย่อยจะมีการสร้างสภาพแวดล้อมเพื่อการตรวจสอบขึ้นใหม่ โดยอ้างอิงจากระดับที่สูงกว่าขึ้นไปเสมอ จากรูปที่ 3.2 แสดงให้เห็นขั้นตอนของการออกแบบในลักษณะของ Top Down Design ทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย ก็เนื่องมาจากขั้นตอนของการผลิต (implementation) สามารถกระทำได้ในหลายๆ เทคโนโลยี เช่น Programmable Logic Devices อันได้แก่ PLA, FPGA หรือ CPLD เป็นต้น นอกจากนั้นยังมี Semi-Custom IC (Gate Array,Standard Cell) และ Fill Custom IC

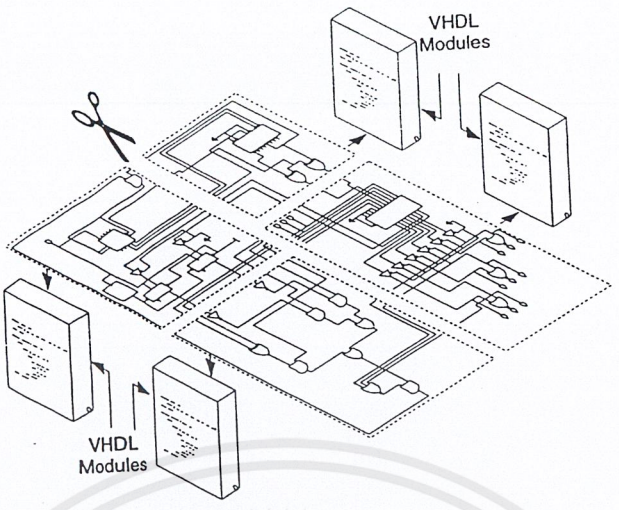
จากความอ่อนตัวของภาษา และความสามารถที่จะเขียนรูปแบบได้หลายลักษณะนี้เอง VHDL จึงเป็นเครื่องมือที่ใช้สำหรับออกแบบตั้งแต่ขั้นตอนบนสุด คือแนวความคิดที่จะแก้ปัญหาลงไปทีละขั้นจนถึงขั้นตอนของการผลิตวงจรจริง ซึ่งข้อดีนี้ทำให้วิศวกรออกแบบสามารถสร้างรูปแบบและจำลองการทำงาน เพื่อให้ตรงกับ specification ที่กำหนดไว้ ทำให้ข้อผิดพลาดที่อาจจะเกิดขึ้นถูกตรวจพบและแก้ไขให้ถูกต้อง ก่อนที่จะทำงานในขั้นตอนต่อๆ ไปของขบวนการออกแบบ



รูปที่ 3.2 แสดงขั้นตอนของ Top Down Design

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

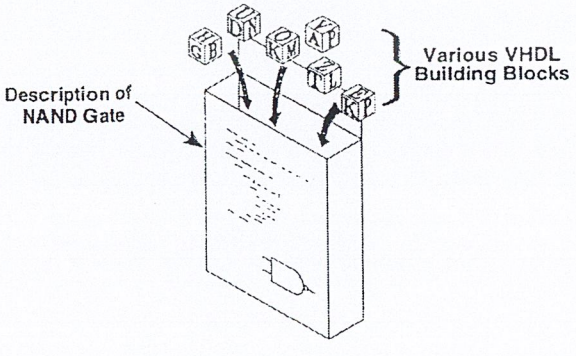
3.3.2 Modularity



รูปที่ 3.3 แสดงการแบ่งย่อยในระดับราบของการออกแบบฮาร์ดแวร์

Modularity คือหลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ ออกเป็นส่วนย่อยเล็กๆ ลงไป ซึ่งปกติการทำงานของฮาร์ดแวร์ใหญ่ต้องประกอบด้วยฮาร์ดแวร์ย่อยๆ ลงไป ดังรูปที่ 3.3 แสดงรูปแบบ ซึ่งแสดงวงจรทั้งหมดในรูปเดียว(Flatten Design) หลังจากนั้นตัดออกเป็นส่วนย่อยๆ เล็กลงมา เมื่อเราออกแบบโดยใช้ภาษาวีเอชดีแอลหน้าที่การทำงานของแต่ละส่วนสามารถอธิบายได้โดย Module ของ Code (คล้าย Function หรือ Procedure) ซึ่งแสดงการทำงานของส่วนย่อยนั้นอย่างชัดเจน ซึ่งการแยกรูปแบบใหญ่ๆ ออกเป็นส่วนย่อยๆ นี้ทำให้ง่ายต่อการจัดการและง่ายต่อการทำความเข้าใจ

ตัวภาษาวีเอชดีแอลประกอบขึ้นมาด้วย Language Building Block ซึ่งประกอบไปด้วย 75 Reserve Word และมากกว่า 200 Combination words รูปที่ 3.4 แสดงให้เห็นว่า ภาษาวีเอชดีแอลแต่ละ Module นั้นประกอบด้วย Language Building Block อะไรบ้าง และอธิบายการทำงานของ NAND gate

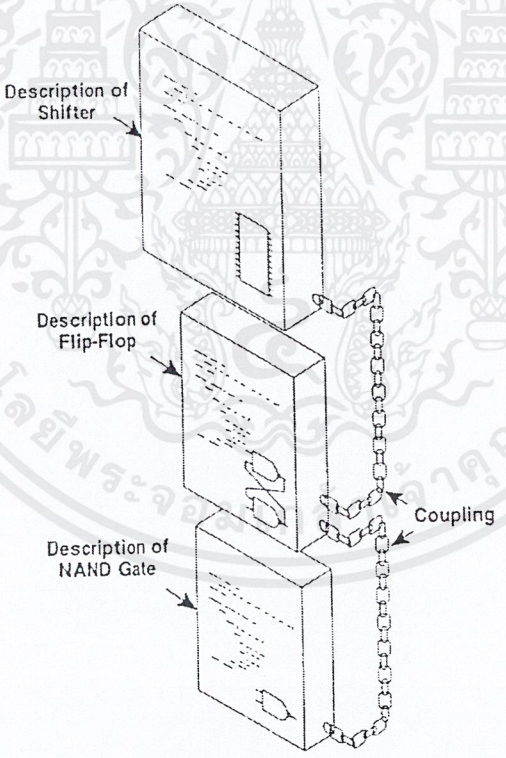


รูปที่ 3.4 แสดงฮาร์ดแวร์โมดูลซึ่งสร้างจากการสร้างบล็อกของ VHDL

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูปที่ 3.5 แสดง Hierarchy Method โดยการแยกส่วนรูปแบบออกเป็นส่วนย่อยๆ ส่วนบนสุดอธิบายการทำงานของ Shifter ส่วนล่างๆ ลงมาคือการแยกส่วนของ Shifter ออกเป็น ฟลิปฟลอป จาก ฟลิปฟลอป แยกเป็น NAND gate ภายใน Shifter ได้อธิบายการทำงานโดยการใช้การต่อกันของ ฟลิปฟลอปในระดับต่ำลงมา ฟลิปฟลอปก็เกิดจากการใช้ NAND gate 2 ตัวต่อกัน ในระดับต่ำลงมาอีกก็เป็น NAND gate ซึ่งมีการอธิบายการทำงานอยู่ภายใน ซึ่งแต่ละ Module จะมีคำอธิบายการทำงานในตัวของมันอยู่แล้ว คำอธิบายในแต่ละ Module มีไว้เพื่อให้สามารถใช้ฟลิปฟลอป Module ได้ ส่วน ฟลิปฟลอป Module ก็อธิบายเชื่อมต่อไว้อย่างดีทำให้สามารถเชื่อมต่อกับ NAND gate ในระดับล่างสุดได้

ประโยชน์อย่างหนึ่งของการแยกส่วนฟลิปฟลอป และ NAND gate ออกจากกัน เนื่องจากว่าทำให้ง่ายในการใช้ NAND gate ตัวนี้ในรูปแบบไฮเลเวล(High Level) ตัวอื่นๆ ทำให้นำออกมาใช้ได้อีก และลดความซับซ้อนในการใช้อุปกรณ์ส่ง เป็นการง่ายที่จะแก้ไขการทำงานของ Shifter โดยปราศจากการแก้ไข Flip-Flop และ NAND gate จากประโยชน์ที่ได้ของ Modularity นี้ทำให้รูปแบบที่เราออกแบบนั้นง่ายต่อการเข้าใจและแก้ไขได้เสมอ



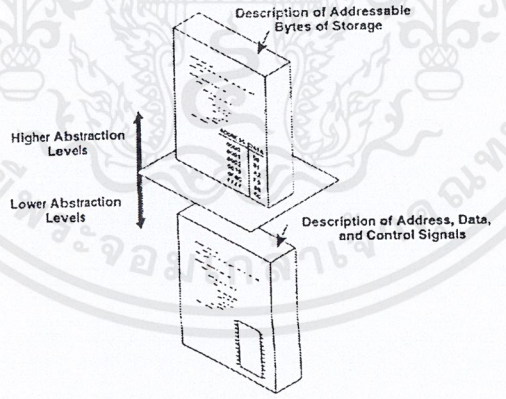
รูปที่ 3.5 แสดงให้เห็นถึงลำดับชั้น (Hierachy) ในการอธิบายอุปกรณ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3 Abstaction

คำนิยามของรูปแบบจะอธิบายการทำงานของตัวรูปแบบ มากกว่าที่จะอธิบายถึงว่าจะพัฒนา รูปแบบนั้นอย่างไร หลักการนี้จะมีความสัมพันธ์อย่างใกล้ชิดกับหลักการ Modularity ในรูปที่ 3.5 จะเห็นว่าตัว Flip-Flop นั้นถูกอธิบายในรูปของ NAND gate แต่ Shifter นั้นถูกอธิบายในระดับของ Flip-Flop ดังนั้นจะเห็นได้ว่าแต่ละ Level ของการออกแบบนั้นถูกสร้างขึ้นมาจาก Level ในระดับล่างขึ้นมาทั้งสิ้น

รูปที่ 3.6 แสดงอีกวิธีการหนึ่งซึ่งแสดงถึงการอธิบายการทำงานของรูปแบบโดยใช้ VHDL ในหลายๆ ระดับของการนิยาม ROM(Read Only Memory) อธิบายโดยใช้ภาษาระดับสูง แสดงถึงตำแหน่ง (Address) ต่างๆ ซึ่งเก็บข้อมูลไว้ในตำแหน่งนั้นๆ ที่ระดับนี้ไม่ต้องสนใจถึง Address Line, Data Line, Control Line เราสามารถเพ่งจุดสนใจไปที่ขนาดของข้อมูล โดยไม่ต้องคิดถึงสัญญาณควบคุมต่างๆ มากมายภายใน เพราะว่าส่วนนั้นจะถูกจัดการเองในระดับต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ในการจัดการสัญญาณภายในทุกเส้นภายในการที่จะอ่านข้อมูลหรือโปรแกรมข้อมูลใน ROM ถ้าต้องการเปลี่ยนค่าข้อมูลภายใน ROM เราควรขึ้นมาแก้ไขในระดับที่สูงขึ้นมา (High Level) จะทำให้ง่ายกว่าในการที่จะควบคุมสัญญาณภายใน ซึ่งเราจะเห็นว่าในแต่ละระดับความเหมาะสมแตกต่างกันไป และตรงจุดนี้เองทำให้รูปแบบที่เราออกแบบง่ายต่อการแก้ไข โดยการใช้ประโยชน์ของ Abstraction

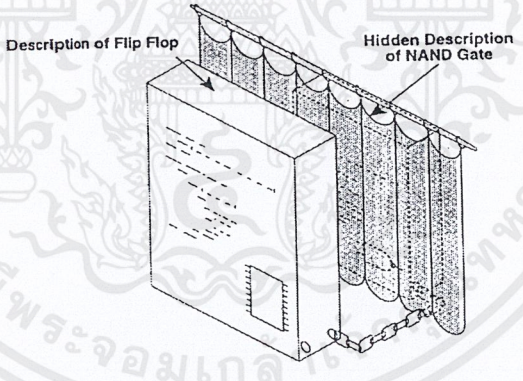


รูปที่ 3.6 แสดงตัวอย่างแสดงการอธิบาย ROM ในระดับต่างๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4 Information Hiding

เมื่อเราทำการเขียน VHDL Code ขึ้นมาเพื่ออธิบายการทำงานของฮาร์ดแวร์ตัวหนึ่ง บางครั้งเราอาจต้องการที่ซ่อนรายละเอียดการพัฒนา Module นั้นๆ โดยไม่ต้องการให้ส่วน Module อื่นๆ รู้การทำงานภายใน Information Hiding มีประโยชน์คือ ทำให้รูปแบบภาษาวีเอชดีแอลนั้นสามารถจัดการได้ง่าย และสามารถอ่านและทำความเข้าใจได้ง่ายกว่า หลักการนี้จะใช้สนับสนุนหลักการ Abstraction คือจะสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่ารูปแบบนั้นถูกสร้างขึ้นมาอย่างไร มีวงจรอย่างไรบ้าง เป็นต้น การซ่อนรายละเอียดภายใน Module ทำให้ความสนใจของผู้ออกแบบสนใจไปในส่วนที่สำคัญมากกว่า ในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ ในรูปที่ 3.5 คำอธิบายของ NAND gate นั้น จะถูกปิดบังเอาไว้จากคนที่เขียนอธิบายฟลิปฟลอป ดูรูปที่ 3.7 คนที่เขียนอธิบายการทำงานของ ฟลิปฟลอป ไม่ต้องสนใจเลยว่า NAND gate จะทำงานอย่างไน จะต่อกันภายในอย่างไร โดย NAND gate จะถูกสร้างและพัฒนามาอย่างไร และประโยชน์อีกอย่างของ Information Hiding ก็คือป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย วีเอชดีแอลโมเดล ไปยังที่อื่นๆ เช่น ส่งไปให้อีกบริษัทใช้พัฒนาร่วมกับวีเอชดีแอลอื่นๆ โดยเป็นการแจกจ่าย อาจส่งไปแค่ภาษาวีเอชดีแอลที่คอมไพล์แล้ว ไม่ต้องส่งตัว Source Code ไป ทำให้เราป้องกันทรัพย์สินทางปัญญาได้ในอีกระดับหนึ่ง



รูปที่ 3.7 แสดงการซ่อนรายละเอียดที่ไม่จำเป็นในระดับ NAND gate

3.5 Uniformity

Uniformity เป็นหลักการอีกอย่างที่ช่วยในการอธิบายฮาร์ดแวร์ด้วยภาษาวีเอชดีแอล หมายถึงการสร้าง Module ของรหัสในลักษณะคล้ายกันโดยใช้ตัวภาษา VHDL Building Block ทำให้เกิดการเขียนรหัสที่ดีอย่างเช่น มีการใช้ย่อหน้า มีการใช้คำอธิบาย(Comment) เป็นต้น ทำให้มีการพัฒนา Module อ่านและทำความเข้าใจง่าย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4 ลักษณะรูปแบบของการเขียนโมเดลด้วยภาษาวีเอชดีแอล

ลักษณะของการเขียนรูปแบบ (model) ด้วยภาษา VHDL สามารถแบ่งได้เป็น

1. **Behavioral Style** หรือเรียกอีกอย่างว่า algrorithmic description เป็นรูปแบบที่บรรยายพฤติกรรมของระบบดิจิตอล ในส่วนที่บรรยายมีโครงสร้างคล้ายกับภาษาชั้นสูง (HDL) ทั่วๆ ไป เช่น PASCAL หรือ C เป็นต้น ในการจำลองการทำงาน (simulation) คำสั่งแต่ละคำสั่ง (statement) จะถูกประเมินผลเป็นไปตามลำดับ (sequential) จากบนลงล่าง ยกเว้นในกรณีของคำสั่ง LOOP หรือการเรียกใช้โปรแกรมย่อย รูปแบบลักษณะนี้จะไม่ให้รายละเอียดที่เกี่ยวกับการผลิต หรือโครงสร้างของ hardware แต่จะให้รายละเอียดเกี่ยวกับความสัมพันธ์ระหว่าง input กับ output ที่ดี
2. **Dataflow Style** เรียกอีกอย่างหนึ่งได้ว่า "Register Transfer Level (RTL)" เป็นรูปแบบที่ถูกเขียนขึ้น เพื่อจุดประสงค์ที่จะใช้เครื่องมือสำหรับการสังเคราะห์วงจรอัตโนมัติ รูปแบบลักษณะนี้ส่วนใหญ่จะเป็น procedural constructs และ functional operators
3. **Structural Style** เป็นรูปแบบที่แสดงการเชื่อมต่อกันระหว่างอุปกรณ์ต่างๆ ที่ประกอบกันขึ้นเป็นวงจรหรือระบบดิจิตอล และสามารถเรียกอีกอย่างได้ว่า "netlist representation" เป็นการเขียนที่แสดงให้เห็นโครงสร้างของ hardware
4. **Mixed-Level Style** จากคุณสมบัติที่อ่อนตัวของภาษา VHDL จึงสามารถที่จะเขียนรูปแบบโดยใช้ลักษณะต่างๆ ที่กล่าวมาแล้วข้างต้น บรรยายวงจรหรือระบบดิจิตอลเดียวกันได้ ฉะนั้นรูปแบบเช่นนี้จึงมีการเขียนแบบผสม

3.5 ส่วนต่างๆ ของแบบ (Design Unit)

ภาษา VHDL นั้นประกอบด้วยส่วนต่างๆ ที่สำคัญและเป็นพื้นฐานของการเขียนรูปแบบระบบดิจิตอลที่สำคัญ 4 หน่วยคือ

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

3.5.1 Entity Design Unit

หน่วยของแบบส่วนที่ใช้สำหรับติดต่อระหว่างโลกภายนอกกับรูปแบบ(model) ที่จะเขียนขึ้น ส่วนนี้เรียกว่า "entity design unit" ในส่วนนี้ใช้กำหนดจุดต่อ (connection point) ของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ (mode) และประเภทของค่า (type of value) ที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆ (PORT) ของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้นรูปที่ 3.8 แสดงให้เห็นโครงสร้างอย่างง่ายๆ ของ entity design unit

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

รูปที่ 3.8 แสดงโครงสร้างอย่างง่ายๆ ของ entity design unit

3.5.2 *Architecture Design Unit*

คือส่วนที่ใช้เขียนบรรยายกำหนดพฤติกรรมของรูปแบบ ในมุมมองของการจำลองงาน (simulation) พฤติกรรมต่างๆ ที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าออก ตรงช่องทางตลอดจนพารามิเตอร์ต่างๆ (ports and generics) ที่กำหนดใน entity design unit รูปที่ 3.9 แสดงให้เห็นโครงสร้างอย่างง่ายๆ 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 3.9 แสดงโครงสร้างอย่างง่ายๆ ของ architecture design uinit

3.5.3 Package Design Unit

ข้อมูลต่างๆ ตลดจนโปรแกรมย่อย (subprogram) ที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบดิจิตอล สามารถเก็บไว้ในส่วนที่เรียกว่า package ได้ และข้อมูลเหล่านี้สามารถนำเอาไปใช้ได้โดย entity design unit , architecture design unit หรือจาก package design unit อื่นๆ โดยปกติแล้ว package จะแบ่งเป็น 2 ส่วนคือ

1. Package Declaration

ส่วนที่มีความสำคัญที่สุดของ package คือ package declaration เพราะจะเป็นส่วนกำหนดชื่อ (identifier) ของสิ่งที่ประกาศอยู่ภายใน package สำหรับนำไปใช้ภายนอกตัวของ package เอง ถ้าสิ่งใดๆ ถูกประกาศในส่วนของ package body แต่ไม่ถูกประกาศใน package declaration จะไม่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สามารถถูกนำค่า และพฤติกรรมไปใช้จากส่วนนอกได้ การเขียน package declaration มีเกณฑ์ตามที่แสดงในรูปที่ 3.10

```
PACKAGE package_name IS
    Package_declarative_part
END package_name ;
```

รูปที่ 3.10 แสดงโครงสร้างของ package declaration

2. Package body

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ (sequential statement) ที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย (subprogram) ทั้งหลายที่ชื่อของโปรแกรมย่อยนั้นๆ ที่ถูกประกาศไปในส่วนของ package declaration แล้วจะถูกเก็บไว้ใน package body การเขียนเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 3.11

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

รูปที่ 3.11 แสดงโครงสร้างของ package body

3.5.4 Configuration Design Unit

รูปแบบหนึ่งของระบบดิจิตอลไม่ว่าจะเป็นอะไรก็ตาม จะมี entity design unit ได้เพียงหน่วยเดียวเท่านั้น แต่ในขณะที่ entity design unit หนึ่งหน่วยนี้อาจมี architecture design unit เป็นหน่วยรองได้หลายหน่วย ทำให้ต้องใช้ configuration ประกอบ entity กับ architecture design unit ที่ต้องการเข้าด้วยกัน เพื่อบอกให้ simulator ทราบ โดยแสดงกฎเกณฑ์การเขียนดังรูปที่ 3.12

```
CONFIGURATION identifier OF entity_name IS
    Configuration_declarative_part
END ;
```

รูปที่ 3.12 แสดงโครงสร้างของ configuration

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สรุปได้ว่าส่วน entity มีหน้าที่กำหนดการติดต่อข้อมูล (interface) กับภายนอกของอุปกรณ์ ส่วน architecture กำหนดพฤติกรรมของอุปกรณ์ เนื่องจากอาจมีการบรรยายพฤติกรรมได้หลายลักษณะจึงมีความจำเป็นที่ต้องประกอบ architecture ที่ต้องการเข้ากับ entity ดังนั้น configuration จึงทำหน้าที่นี้ด้วยการเชื่อม entity กับ architecture เข้าด้วยกัน ส่วน package นั้นคือที่รวมของฟังก์ชันต่างๆ ที่สามารถนำไปใช้ได้จากรูปแบบ (model) อื่นๆ สิ่งที่อยู่ใน package อาจเป็น TYPE declaration หรือ Subprogram ตลอดจนอุปกรณ์ต่างๆ และสามารถเข้าได้ด้วยคำสั่ง USE statement

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4
เอฟพีจีเอ (FPGA)

ความก้าวหน้าของอุตสาหกรรมอิเล็กทรอนิกส์ปัจจุบันทำให้เกิดพัฒนาความสามารถของอุปกรณ์ต่างๆ มากมาย ซึ่งทำให้เกิดการลดค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาด ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพ และระดับความน่าเชื่อถือของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครโพรเซสเซอร์และหน่วยความจำปัจจุบันทุกๆ ครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างวงจรรวมและไอซีมาตรฐานมากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวน ฟังก์ชันลอจิกที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตให้ขนาดมากๆ และการผลิตวงจรรวม (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวม จะแบ่งตามการสร้างออกเป็น 2 กลุ่มคือ Filed programmable และ Mask programmable ซึ่งในที่นี้จะกล่าวถึง Filed programmable เท่านั้น

4.1 Field Programmable

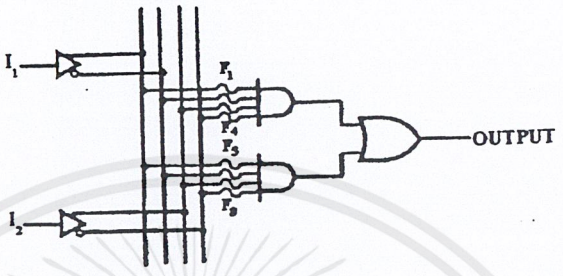
อุปกรณ์วงจรรวม ASIC แบบ field programmable มีอยู่มากมายหลายชนิด แต่มีลักษณะการสร้างหรือกำหนดการทำงานของวงจรที่เหมือนกัน กล่าวคือ ผู้ใช้งานสามารถออกแบบและสร้างวงจรที่ต้องการใช้ลงในตัวอุปกรณ์ได้เองโดยไม่ต้องไปโรงงานเพื่อผลิต โดยเฉพาะอย่างยิ่งในปัจจุบันนี้มีเครื่องมือที่ใช้ช่วยในการออกแบบ และสร้างวงจรร่วมกับไมโครคอมพิวเตอร์ที่มีความสามารถสูงในการพัฒนาตั้งแต่ขั้นการออกแบบ การจำลองการทำงาน จนถึงจัดสร้างวงจรลงในอุปกรณ์ รวมทั้งอุปกรณ์ field programmable เหล่านี้สามารถหาซื้อได้ง่ายทำให้การสร้างวงจรอิเล็กทรอนิกส์จนถึงระบบไมโครโพรเซสเซอร์หันมาใช้อุปกรณ์จำพวกนี้ เป็นอุปกรณ์ประกอบในวงจรแทนอุปกรณ์ย่อยๆ แยกชิ้น (Discrete component) มาก โดยอุปกรณ์ประเภทนี้จะแบ่งเป็น 2 ประเภทคือ PLD และ FPGA

4.1.1 พีแอลดี (PLD : Programmable Logic Device)

ภายในอุปกรณ์พีแอลดีถูกเตรียมเป็นวงจรพื้นฐานทางด้านลอจิกต่อกันอยู่เป็นกลุ่มมีทั้งวงจร Combination และแบบ Sequential ซึ่งมีส่วนประกอบเป็นวงจรภายใน เทคโนโลยีของวงจรที่ใช้สร้างพีแอลดีมีทั้ง TTL , ECL และ CMOS ตามความเหมาะสมของแต่ละระบบ อุปกรณ์พีแอลดีทุกชนิดมีหลักการพื้นฐานของวงจรภายในที่เหมือนกันโดยมีวงจรหลักเป็นวงจร Combination ที่ให้ผลเป็น Sum of product ประกอบไปด้วยชุดของแอนด์เกทที่ต่อร่วมกับออร์เกทการโปรแกรมคือ การเลือกว่าจะให้มีการต่ออินพุทภายในของแอนด์เกทกับสัญญาณอินพุทใดบ้างซึ่งมีทั้งจากภายนอก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

และสัญญาณป้อนกลับจากเอาท์พุทภายในเอง การติดต่ออินพุทของออร์เกทกับเอาท์พุทของแอนด์เกท ตัวต่างๆ วิธีการเลือกหรือการโปรแกรมทางกายภาพ อินพุทต่างๆ ของอุปกรณ์ทุกตัวจะถูกต่อผ่านฟิวส์เข้ากับแหล่งสัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใดจะตัดฟิวส์ทำให้สามารถโปรแกรมได้ครั้งเดียว อุปกรณ์พีแอลดีบางชนิดใช้มอสทรานซิสเตอร์แทนฟิวส์ทำให้สามารถโปรแกรมโดยใช้กระแสไฟฟ้า และสามารถลบและโปรแกรมใหม่เข้าไปได้อีก



รูปที่ 4.1 แสดงวงจรพื้นฐานของอุปกรณ์พีแอลดีซึ่งอยู่ในรูปของ Sum of product

4.1.2 เอฟพีจีเอ (FPGA : Field Programmable Gate Array)

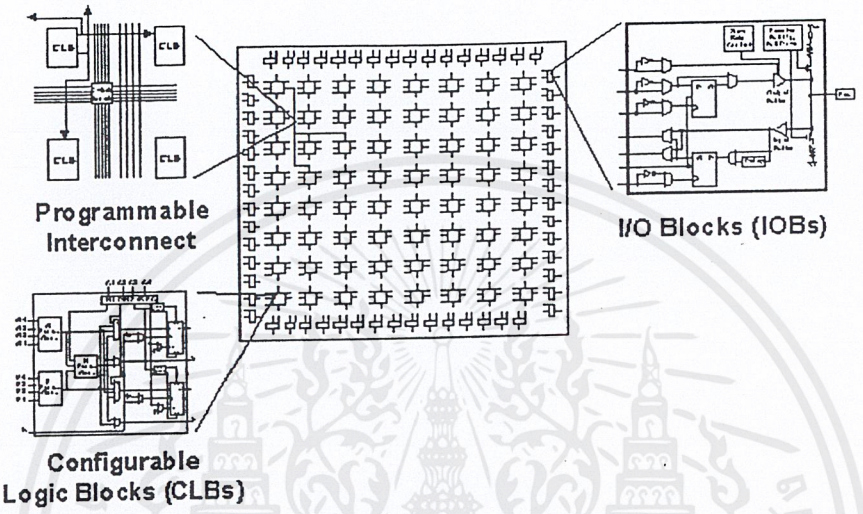
เป็นอุปกรณ์ที่ถูกพัฒนาต่อจากอุปกรณ์แอลซีเอของบริษัทไซลิงซ์ (Xilinx Inc.) โดยมีประสิทธิภาพการทำงานและมีปริมาณความหนาแน่นของเกทสูง สามารถจะกำหนดฟังก์ชั่นการทำงานได้ความต้องการของผู้ใช้โดยผ่านการโปรแกรมเอฟพีจีเอได้รวบรวมข้อดีทั้งหมดของการทำคัสตัมวีแอลเอสไอ(Custom VLSI) มารวมไว้ทั้งหมดได้แก่ การออกแบบการผลิต,ลดเวลาที่จะส่งตัวผลิตภัณฑ์ออกตลาด ซึ่งเป็นประโยชน์ต่อการผลิตวงจรเป็นอย่างมาก นักออกแบบเพียงกำหนดฟังก์ชั่นการทำงานของวงจร ดังนั้นการออกแบบวงจรโดยใช้เอฟพีจีเอ สามารถออกแบบและทดสอบภายในเวลาเพียง 2-3 วันเท่านั้น ตรงกันข้ามกับการออกแบบโดยใช้ gate array ซึ่งใช้เวลาหลายสัปดาห์ การเปลี่ยนแปลงแก้ไขแบบก็เช่นเดียวกัน จากประโยชน์ของเอฟพีจีเอ ดังกล่าวมา ทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์ ลดค่าเอ็นอาร์อี (NRE: Nonrecurring Engineering Cost) ลงไปด้วย

4.2 สถาปัตยกรรมภายในของเอฟพีจีเอ

FPGA เป็นชิพที่มีสนามวงจรขนาดใหญ่อยู่ภายใน ที่เราสามารถนำมาใช้และออกแบบวงจรต่างๆ ได้ตามที่เราต้องการโดยโปรแกรมสำเร็จรูปที่ใช้ในการออกแบบอยู่แล้ว มีความเร็วสูง มีสถาปัตยกรรมการออกแบบคล้าย CPLD (Complex Programmable Logic Device) แต่มีส่วนประกอบที่ซับซ้อน และมีประสิทธิภาพมากกว่า ในการออกแบบวงจรสามารถทำได้ง่าย ทั้งในการเชื่อมต่อและการแก้ไข ดังนั้น FPGA จึงเหมาะสมสำหรับการออกแบบวงจรเป็นอย่างมาก สถาปัตยกรรมภายในของ FPGA แบ่งเป็น 3 ส่วนคือ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- CLB (Configuration Logic Block)
- IOB (Input Output Block)
- Interconnect



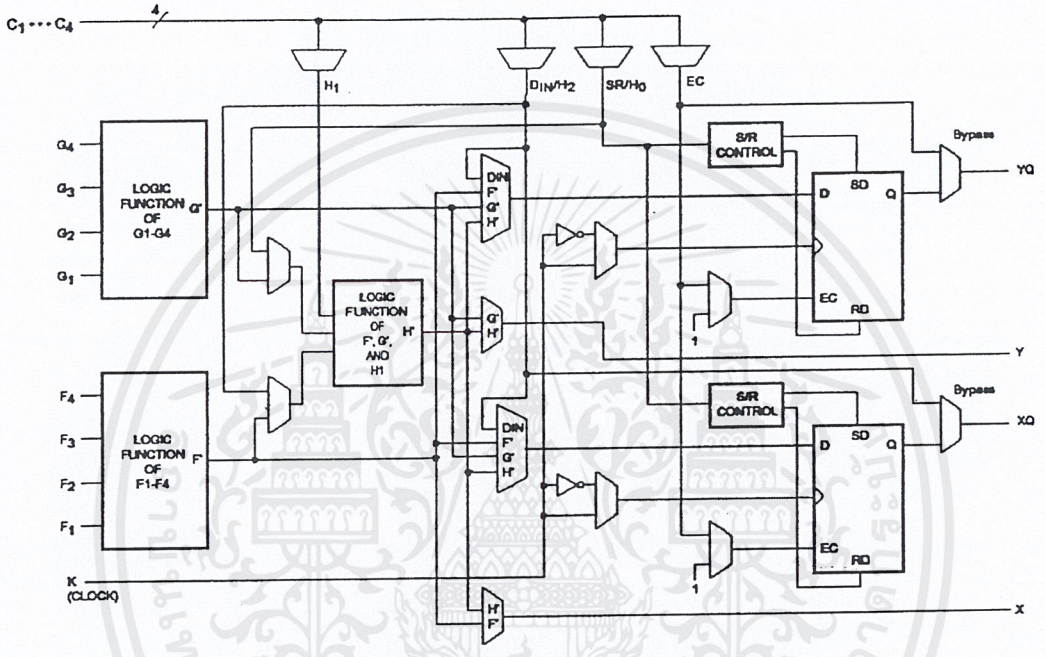
รูปที่ 4.2 แสดงวงจรพื้นฐานภายใน Xilinx FPGA

ซึ่งภายในมีสถาปัตยกรรมของ FPGA จะมีลักษณะเป็นตารางของลอจิคบล็อกและล้อมรอบไปด้วยบล็อกการเชื่อมต่อของไอโอ (I/O Interface block) การเชื่อมต่อระหว่าง CLB และ IOB ทำได้โดยผ่านช่องที่ว่างพาดผ่านแถว(Row) และคอลัมน์(Column) มีการทำงานเหมือนกันกับไมโครโพรเซสเซอร์ ตัว LCA(Logic Cell Array) จะทำงานได้ต้องใช้ Program-driven logic device หน้าที่ของ CLB และ IOB แต่ละตัว การเชื่อมต่อภายใน (Interconnect) ถูกกำหนดไว้ในโปรแกรม Configuration หรือเก็บไว้ใน EPROM ภายใน LCA โปรแกรมจะถูกโหลดเข้าสู่ LCA เมื่อมีการจ่ายไฟ (Power-up) โดยทางคำสั่ง(Command) ซึ่งเป็นส่วนหนึ่งของการเริ่มต้นระบบ (System initialization) ประสิทธิภาพของ LCA กำหนดโดย ความเร็วของลอจิกส่วนประกอบหน่วยความจำและการโปรแกรมการเชื่อมต่อต่างๆ ความเร็วของอัตราของระบบสัญญาณนาฬิกา (System Clock Rate) ถูกกำหนดด้วย Toggle F/F สำหรับการประยุกต์ใช้โดยทั่วไปจะอยู่ที่ประมาณ 1/3 ถึง 1/2 ค่าสูงสุดของ Toggle gate

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1. CLB (Configuration Logic Block)

ภายใน LCA คือเมทริกซ์ของซีแอลบีแต่ละตัวประกอบด้วยหน่วยของคอมบิเนชั่นลอจิกที่สามารถโปรแกรมได้ (Programmable combination logic) และส่วนของรีจิสเตอร์เก็บข้อมูล (Storage register) ส่วนของวงจรคอมบิเนชั่นลอจิกสามารถใช้สร้างวงจรทางด้านฟังก์ชั่นบูลีนของอินพุท ส่วนรีจิสเตอร์รับค่าจากส่วนคอมบิเนชั่นหรือโดยตรงจากเอาท์พุทของซีแอลบี สามารถขับวงจรคอมบิเนชั่นลอจิกโดยตรงผ่านเส้นทางเดินย้อนกลับ(Feedback path)

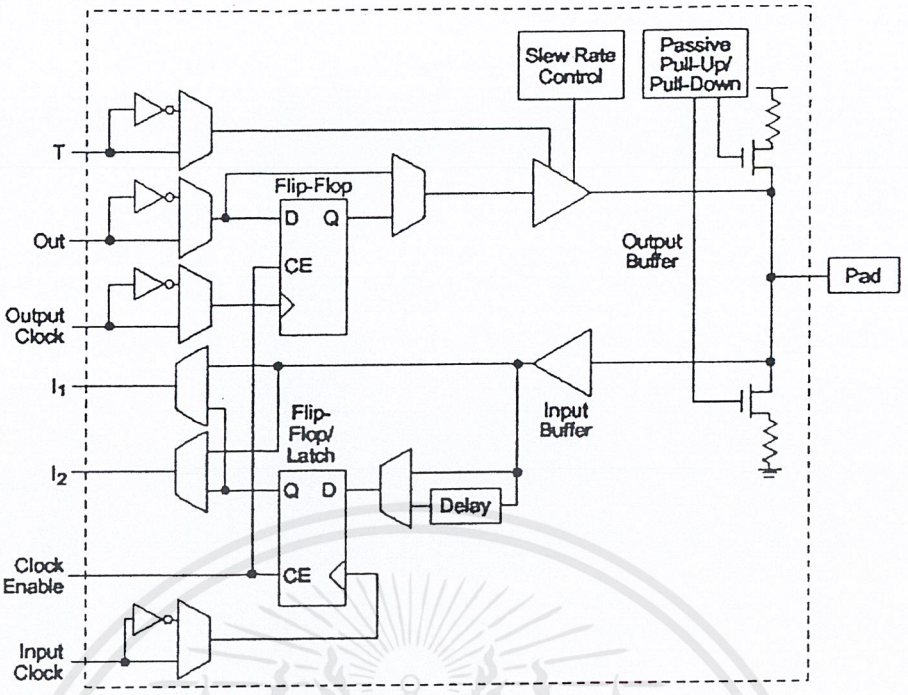


รูปที่ 4.3 แสดงผังวงจรภายในของ CLB ของ FPGA

2. IOB (Input Output Block)

เป็นส่วนติดต่อกับวงจรภายนอกของแอซีเอ สร้างมาจากส่วนของอุปกรณ์อินพุท/เอาท์พุทที่สามารถโปรแกรมได้ (Programmable Input/Output device) แต่ละตัวสามารถโปรแกรมได้อย่างอสระโดยจะให้เป็นอินพุท/เอาท์พุทแบบ 3 สถานะหรือไอโอแบบสองทิศทางก็ได้ โดยอินพุทสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณ TTL และซีมอสเทรดโฮลของไอโอบี แต่ลละตัวมีฟลิปฟลอปสามารถใช้เป็นบัฟเฟอร์สำหรับอินพุทและเอาท์พุท

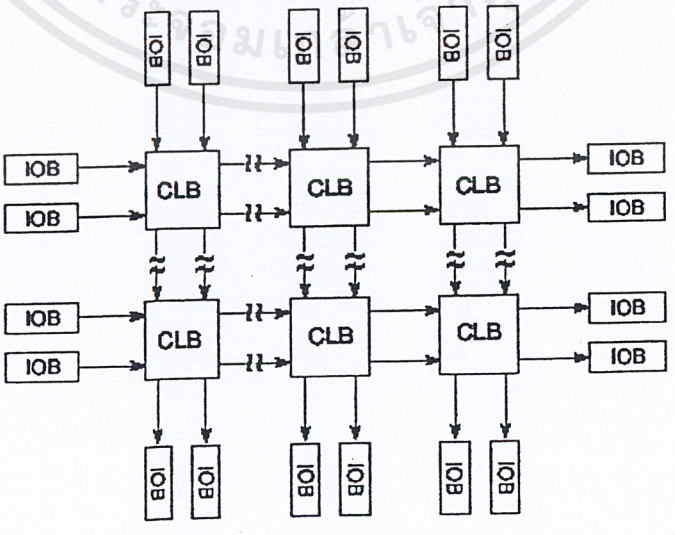
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.4 แสดงผังวงจรของ IOB ภายใน FPGA

3. Interconnect

ความยืดหยุ่นของการใช้แอลซีเอมาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรม ทรัพยากรต่างๆ ที่อยู่ภายในเข้าด้วยกันการที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายในชิปเหมือนกับเกทอาเรย์ทั่วๆ ไป การเชื่อมต่อภายในแอลซีเอประกอบด้วยเน็ตเวิร์ค 2 ทิศทางคือ แถวและคอลัมน์ ซึ่งวางอยู่ระหว่าง CLB Programmable switch จะทำการเชื่อมต่ออินพุทและเอาท์พุทของ IOB และ CLB ที่จุดต่อร่วมระหว่างแถวกับคอลัมน์สามารถสลับสัญญาณจากเส้นทางไปยังส่วนต่างๆ



รูปที่ 4.5 แสดง Interconnect ระหว่าง IOB กับ CLB ของ FPGA

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3 คุณสมบัติโดยทั่วไปของเอฟพีจีเอตระกูล XC4000

4.3.1 เป็นอุปกรณ์รุ่นที่สามของเอฟพีจีเอ

- มีฟลิปฟลอปเป็นจำนวนมาก
- ในการผลิตฟังก์ชั่นของการทำงานมีความยืดหยุ่นสูง
- มีจำนวนเกทภายในจำนวน 2,000 – 10,000 เกท
- เพิ่มความสามารถพิเศษของรีจิสเตอร์และอินพุท/เอาท์พุท
- มีค่า fan-out สูง
- มีบัสภายใน 3 สถานะ
- ทำงานกับสัญญาณทีทีแอลและซีมอส
- มีออสซิเลเตอร์แอมปลิฟลายเออร์ภายใน
- มีแรมภายในความเร็วสูง (< 25 ns)
- ใช้กับงานที่ต้องการความเร็วสูง (ใช้งานได้ที่ความถี่ 70/100/125 MHz)
- มี Wide edge decoder
- เส้นทางการเชื่อมต่อ(Interconnect line) เป็นแบบลำดับขั้น
- มีการกระจายกำลังงานของสัญญาณต่ำ

4.3.2 มีสถาปัตยกรรมภายในที่ยืดหยุ่น

- มีลอจิกบล็อกและไอโอบล็อกที่สามารถโปรแกรมได้
- มีอินเตอร์คอนเน็คและ Wide Decoder ที่โปรแกรมได้

4.3.3 ทำกระบวนการซับไมครอนชนิดซีมอสได้

- มีลอจิกและอินเตอร์คอนเน็คที่มีความเร็วสูง
- ใช้กำลังงานต่ำ

4.3.4 คุณลักษณะทาง System-Oriented

- รองรับมาตรฐาน IEEE 1149.1 ในการทำ boundary-scan logic
- สามารถโปรแกรมค่า output slew rate ได้
- สามารถโปรแกรมให้อินพุทมีลักษณะ pull-up หรือ pull-down รีจิสเตอร์ได้
- ให้กระแสเอาท์พุทได้ตั้งแต่ 12-24 มิลลิแอมป์(ขึ้นอยู่แต่ละรุ่น)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3.5 ทำการโหลดเอาแฟ้มข้อมูลประเภทไบนารี

- ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
- มีโหมดในการโปรแกรมให้เลือก 6 โหมด

4.3.6 มีโปรแกรมช่วยพัฒนาได้แก่ XACT Development System (ปัจจุบัน Foundation series) ที่ทำงานบนคอมพิวเตอร์รุ่นต่างๆ

- สามารถติดต่อกับโปรแกรมอื่นๆ ได้เช่น Viewlogic, Mentor Graphic และ OrCAD เป็นต้น
- มีโปรแกรมการวางและเชื่อมโยงอุปกรณ์ภายในแบบอัตโนมัติ(automatic place and routing) ที่ครบสมบูรณ์
- มี Interactive Design Editor ที่ใช้สำหรับการทำ optimization
- มี 288 มาโคร 34 ฮาร์ดมาโคร และ แรม/รอมคอมพายเลอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
ขั้นตอนและรายละเอียดการออกแบบ CPU

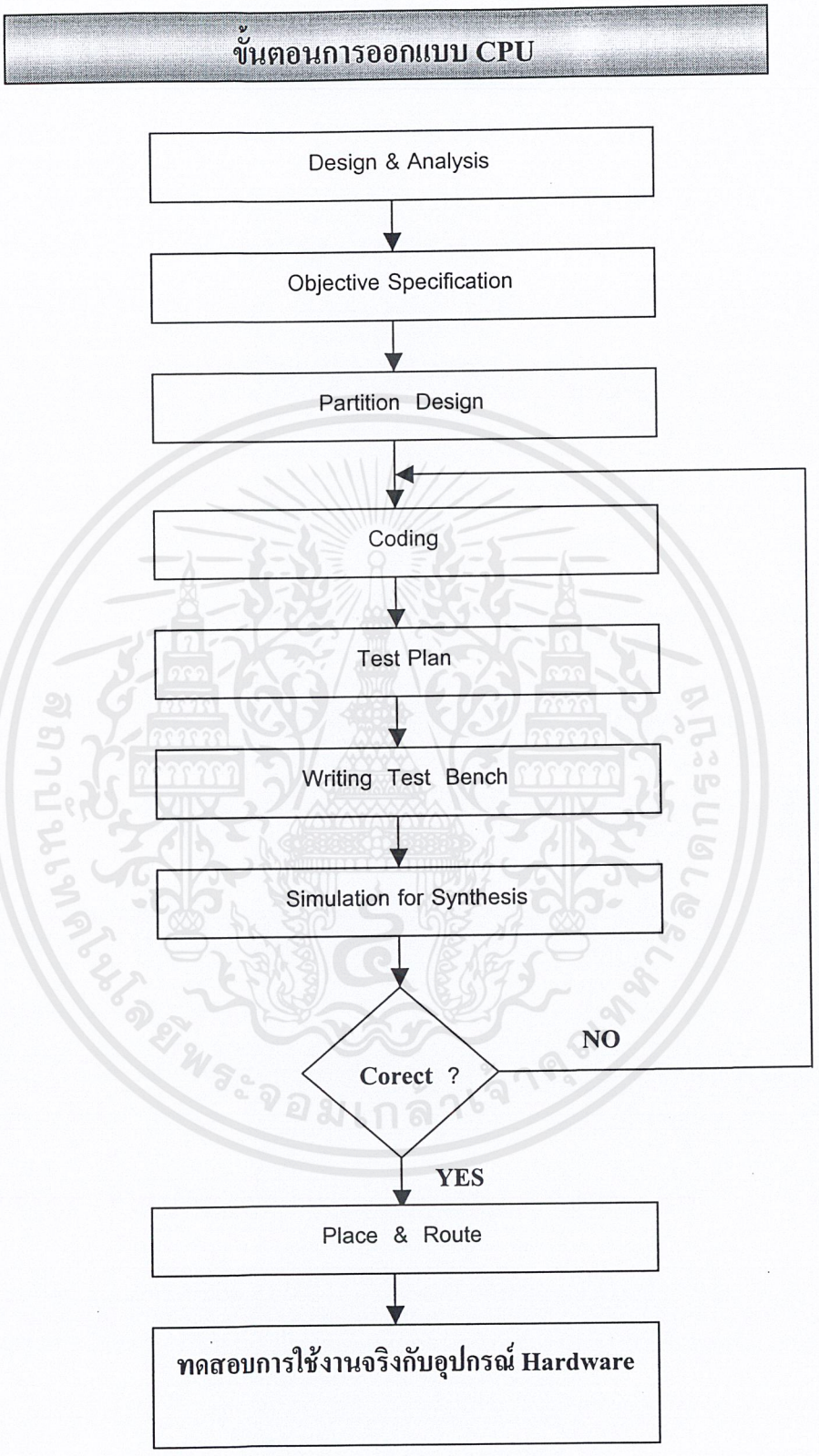
ในบทนี้จะอธิบายถึงการทำงานต่างๆ ที่ได้ทำในโครงงานว่ามีขั้นตอนในการปฏิบัติงานอย่างไร อะไรควรจะทำก่อน อะไรควรจะทำหลัง ตลอดจนรายละเอียดของแต่ละ Component ที่ทำการออกแบบในขั้นต้น รวมทั้งแนวทางในการทดสอบโครงงาน เพื่อทดสอบความถูกต้องในการทำงานของ CPU ที่ได้ทำการออกแบบ

5.1 ขั้นตอนการออกแบบ CPU

การที่จะออกแบบชิ้นงานอะไรขึ้นมานั้น ก่อนอื่นจะต้องมีความเข้าใจในชิ้นงานที่ต้องทำการออกแบบเสียก่อน ในการออกแบบซีพียูก็เหมือนกัน จะต้องศึกษาการทำงานของซีพียูอย่างละเอียดก่อนว่ามีสถาปัตยกรรมภายในอะไรบ้าง ทำงานอย่างไร เมื่อทราบรายละเอียดการทำงานเป็นอย่างดีแล้ว ก็จะมาทำขั้นตอนการกำหนด Specification ของซีพียูว่าเราต้องการทำอะไรบ้างเมื่อได้ Specification ที่เราต้องการแล้ว ก็มาเข้าสู่ขั้นตอนการแบ่งงานที่จะทำการออกแบบเป็นส่วนๆ เพื่อให้ง่ายต่อการออกแบบตามหลักการของ Top Down Design โดยจะระบุ Specification ของแต่ละส่วนอย่างละเอียด แล้วก็จะเข้าสู่ขั้นตอนของการเขียน Code ด้วยภาษา VHDL ของแต่ละ Block ที่ทำการออกแบบไว้ จากนั้นก็วางแผนการตรวจสอบและทำการเขียน Test bench ด้วยภาษา VHDL เพื่อทำการทดสอบแต่ละ Block ที่เขียนขึ้นมา แล้วก็นำ VHDL Code ของแต่ละ Block มาทดสอบเพื่อดูว่าทำงานได้อย่างถูกต้องหรือไม่ หากมีข้อผิดพลาดเกิดขึ้นก็ต้องกลับไปแก้ไข VHDL Code ของ Block นั้นใหม่ ทำการทดสอบแบบ Synthesis จนแน่ใจว่าทำงานได้อย่างถูกต้องแล้ว จึงนำ VHDL Code มาแปลงและทำขั้นตอนการ Place & Route เพื่อบันทึกลง FPGA ต่อไป

เมื่อได้ FPGA ที่ทำการโปรแกรมเรียบร้อยแล้ว ก็นำ FPGA นั้นไปทดสอบการทำงานด้วยการต่อ Hardware ภายนอก และทดสอบการทำงานต่อไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.1 Flowchart แสดงขั้นตอนการออกแบบ CPU ด้วยภาษา VHDL

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.1.1 การกำหนด Specification

เมื่อทราบถึงหลักการทำงานของซีพียูอย่างละเอียดแล้ว ขั้นตอนต่อมาคือการกำหนด Design Specification ของซีพียูที่จะทำการสร้างซึ่งได้ Specification ที่ต้องการดังนี้

1. การประมวลผล

เป็นการประมวลผลข้อมูลแบบ 8 bit

2. ความสามารถในการอ้างตำแหน่งหน่วยความจำ

สามารถอ้างหน่วยความจำภายนอกได้สูงสุด 64 กิโลไบต์

3. ชุดคำสั่ง

มีทั้งหมด 35 คำสั่ง

1. ld เป็นคำสั่งสำหรับดึงข้อมูลจากหน่วยความจำเข้าสู่รีจิสเตอร์
2. ldr เป็นคำสั่งสำหรับป้อนข้อมูลจากรีจิสเตอร์ไปสู่หน่วยความจำ
3. in เป็นคำสั่งสำหรับดึงข้อมูลจากอุปกรณ์ภายนอก
4. out เป็นคำสั่งสำหรับป้อนข้อมูลไปสู่อุปกรณ์ภายนอก
5. add เป็นคำสั่งให้บวกข้อมูลระหว่างรีจิสเตอร์
6. adc เป็นคำสั่งให้บวกข้อมูลระหว่างรีจิสเตอร์ พร้อม carry flag ด้วย
7. sub เป็นคำสั่งให้ลบข้อมูลระหว่างรีจิสเตอร์
8. suc เป็นคำสั่งให้ลบข้อมูลระหว่างรีจิสเตอร์ พร้อม carry flag ด้วย
9. inc เป็นคำสั่งให้เพิ่มค่าของรีจิสเตอร์อีก 1
10. dec เป็นคำสั่งให้ลดค่าของรีจิสเตอร์ลง 1
11. not เป็นคำสั่งทางตรรกะให้กลับบิตข้อมูลของรีจิสเตอร์
12. and เป็นคำสั่งทางตรรกะนำรีจิสเตอร์มา and กัน
13. or เป็นคำสั่งทางตรรกะนำรีจิสเตอร์มา or กัน
14. xor เป็นคำสั่งทางตรรกะนำรีจิสเตอร์มา xor กัน
15. set เป็นคำสั่ง set บิตของรีจิสเตอร์เป็น 1
16. res เป็นคำสั่ง reset บิตของรีจิสเตอร์เป็น 0
17. bit เป็นคำสั่งเปรียบเทียบบิตแต่ละบิตของรีจิสเตอร์ถ้าเป็น 0 ให้ set ค่า zero flag เป็น 1
18. cp เป็นคำสั่งเปรียบเทียบค่ากับข้อมูลของ acc ถ้าเท่ากัน ให้ set ค่า zero flag เป็น 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

19. ror เป็นคำสั่งหมุนข้อมูลไปทางขวา 1 บิต โดยนำ carry flag มาหมุน
20. rol เป็นคำสั่งหมุนข้อมูลไปทางซ้าย 1 บิต โดยนำ carry flag มาหมุน
21. rrc เป็นคำสั่งหมุนข้อมูลไปทางขวา 1 บิต โดยนำค่าบิตที่ 0 เก็บเข้า carry flag
22. rlc เป็นคำสั่งหมุนข้อมูลไปทางขวา 1 บิต โดยนำค่าบิตที่ 7 เก็บเข้า carry flag
23. srs เป็นคำสั่งเลื่อนข้อมูลไปทางขวา 1 บิต โดยนำค่าบิตที่ 0 เก็บเข้า carry flag และนำค่า 1 เก็บเข้าบิตที่ 7
24. sls เป็นคำสั่งเลื่อนข้อมูลไปทางซ้าย 1 บิต โดยนำค่าบิตที่ 7 เก็บเข้า carry flag และนำค่า 1 เก็บเข้าบิตที่ 0
25. srr เป็นคำสั่งเลื่อนข้อมูลไปทางขวา 1 บิต โดยนำค่าบิตที่ 0 เก็บเข้า carry flag และนำค่า 0 เก็บเข้าบิตที่ 7
26. slr เป็นคำสั่งเลื่อนข้อมูลไปทางซ้าย 1 บิต โดยนำค่าบิตที่ 7 เก็บเข้า carry flag และนำค่า 0 เก็บเข้าบิตที่ 0
27. jp เป็นคำสั่งกระโดดไปทำงานในตำแหน่งที่กำหนดอย่างไม่มีเงื่อนไข
28. jz เป็นคำสั่งกระโดดไปทำงานในตำแหน่งที่กำหนดเมื่อ zero flag เป็น 1
29. jnz เป็นคำสั่งกระโดดไปทำงานในตำแหน่งที่กำหนดเมื่อ zero flag เป็น 0
30. jc เป็นคำสั่งกระโดดไปทำงานในตำแหน่งที่กำหนดเมื่อ carry flag เป็น 1
31. jnc เป็นคำสั่งกระโดดไปทำงานในตำแหน่งที่กำหนดเมื่อ carry flag เป็น 0
32. call เป็นคำสั่งเรียกโปรแกรมย่อย
33. push เป็นคำสั่งให้เก็บข้อมูลลงสู่ Stack Pointer
34. pop เป็นคำสั่งให้ดึงข้อมูลจาก Stack Pointer
35. ret เป็นคำสั่งออกจากโปรแกรมย่อย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

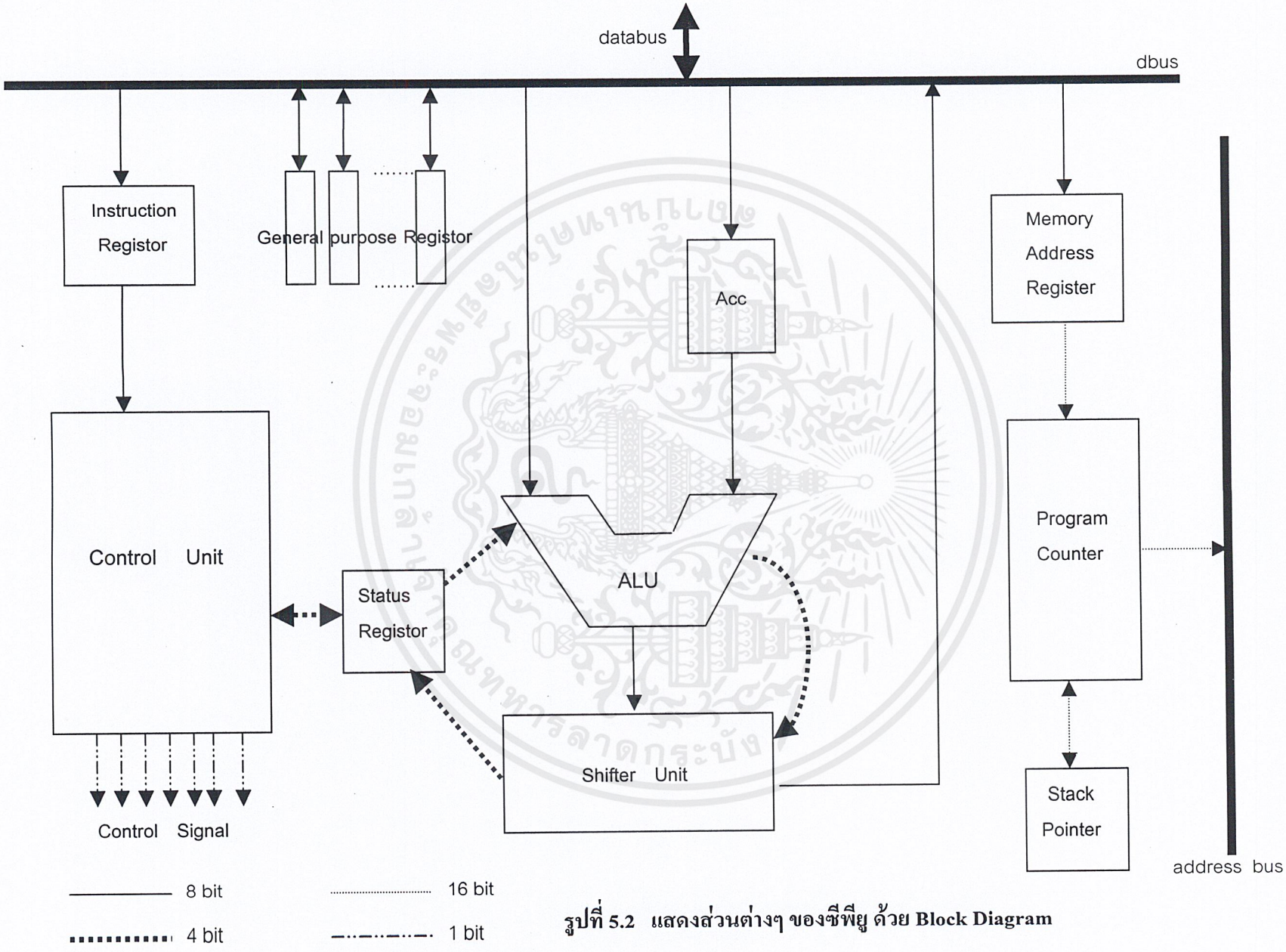
4. สถาปัตยกรรมภายใน

1. *ALU* สำหรับกระทำคำสั่งทางคณิตศาสตร์และลอจิก
2. *Shifter Unit* สำหรับกระทำคำสั่งเกี่ยวกับการ Shift ข้อมูล
3. *Status Register* หรือ แฟลก ใช้เก็บสถานะพิเศษต่างๆ VCZN
4. *Accumulator* ใช้เก็บข้อมูลสำหรับ ALU
5. *Instruction Register* ใช้เก็บรหัสคำสั่งเพื่อให้ Control Unit ตีความ
6. *Control Unit* เป็นตัวกำเนิดสัญญาณเพื่อให้ CPU กระทำตามคำสั่งต่างๆ
7. *Memory Address Register* ใช้เก็บค่า address ของ memory สำหรับอ้างอิง
8. *Program Counter* ใช้เก็บตำแหน่งของคำสั่งต่อไปที่ CPU ต้องอ่านเพื่อนำมาปฏิบัติ
9. *Stack Pointer* ใช้เก็บค่าข้อมูลชั่วขณะ เมื่อเกิดการ interrupt หรือคำสั่ง call
10. *General purpose Register* เป็นกลุ่มรีจิสเตอร์อิสระใช้เก็บข้อมูลต่างๆ

5.1.2 การทำ Partition Design

เมื่อทราบ Specification ของซีพียูที่ต้องการออกแบบแล้วก็ทำการแบ่งชิ้นงานที่จะออกแบบเป็นส่วนๆ เพื่อให้ง่ายต่อการออกแบบตามหลักการของ Top Down Design ได้ Block Diagram ของซีพียูที่จะออกแบบดังรูปที่ 5.2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.2 แสดงส่วนต่างๆ ของซีพียู ด้วย **Block Diagram**

5.1.3 แนวทางการทดสอบความถูกต้องของ VHDL Code

เมื่อเขียน VHDL Code เสร็จแล้ว ขั้นตอนต่อไปคือการนำเอา VHDL Code ดังกล่าวมาทำการทดสอบว่าสามารถทำงานได้อย่างถูกต้องหรือไม่ด้วยโปรแกรม Simulation ซึ่งในที่นี้เราจะใช้ ModelSim SE-EE5.4 เป็นตัวทดสอบ VHDL Code โดยทั่วไปทำได้ 2 วิธีคือ

1. การจำลองการทำงานโดยการกำหนดค่า

วิธีนี้จะต้องมีการกำหนดค่าให้กับสัญญาณด้วยการ Force input signal ซึ่งจะต้องทำทุกครั้งที่มีการจำลองการทำงาน วิธีนี้เหมาะกับการ Simulate วงจรขนาดเล็กเท่านั้น หากวงจรที่ออกแบบมีความซับซ้อนมาก การใช้วิธีนี้จะยุ่งยากและเสียเวลาเป็นอย่างมากเพราะต้องเสียเวลาในการสั่ง Force input signal ใหม่ทุกครั้งที่มีการจำลองการทำงาน เพื่อแก้ปัญหาดังกล่าว ซอฟท์แวร์ทูลล์บางตัวจึงอนุญาตให้สร้าง Test pattern file เก็บไว้เพื่อทดสอบได้ ซึ่งอำนวยความสะดวกให้ผู้ใช้เป็นอย่างมาก อย่างไรก็ตามการทำแบบนี้มีข้อเสียคือ Test pattern จะยึดติดกับทูลล์ตัวที่สร้างมันขึ้นมา นั่นหมายถึง Test pattern ของซอฟท์แวร์ตัวหนึ่งจะใช้ไม่ได้ในอีกซอฟท์แวร์ทูลล์ตัวอื่น

2. การจำลองการทำงานด้วย VHDL Code ที่เขียนขึ้นเพื่อทดสอบโดยเฉพาะ

วิธีนี้จะใช้วิธีเขียน Test Bench เราจะเขียน VHDL Code ขึ้นมาใหม่อีก 1 Entity โดยที่ Entity ที่เขียนขึ้นใหม่นี้จะทำการเรียกใช้ VHDL Code ที่ต้องการทดสอบในรูปของการเรียกใช้ Component ตัวหนึ่ง โดยตัว Testing Program จะทำหน้าที่ให้กำเนิดสัญญาณทดสอบและป้อนสัญญาณทดสอบนั้นเข้าไปที่ขาของ Component ที่ต้องการทดสอบ แล้วทำการทดสอบ Output ที่ออกมาโดยนำมาเปรียบเทียบกับค่า Output pattern ที่ควรจะเป็น แล้วรายงานความถูกต้องของผลลัพธ์ออกมา

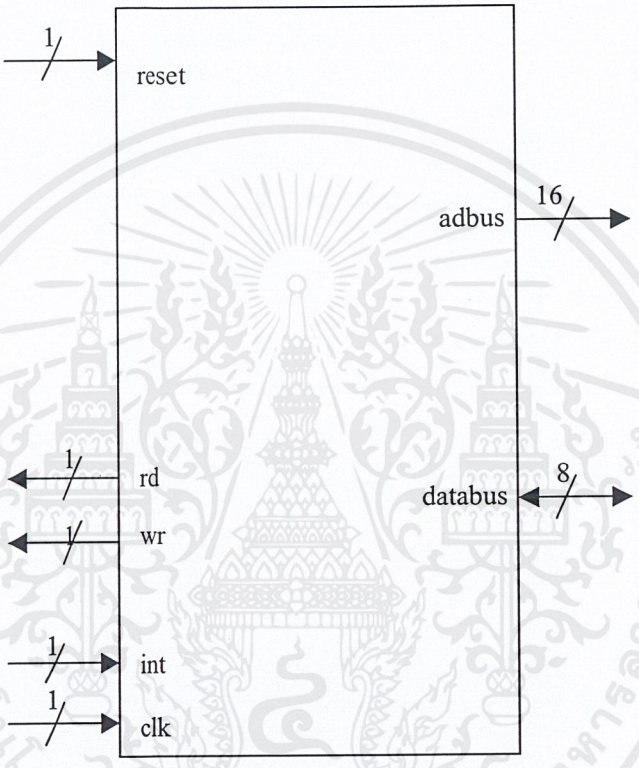
วิธีนี้มีข้อดีคือ สามารถนำไปจำลองการทำงานด้วย Software Tools ตัวใดก็ได้เพราะเนื่องจาก Software Tools ทุกตัวย่อมที่สามารถจะ Simulate การทำงานของ VHDL Code ได้อยู่แล้ว และขอดีอีกอย่างของวิธีนี้คือไม่ต้องเสียเวลาในการสั่ง Force input signal ใหม่ทุกครั้งที่มีการจำลองการทำงาน ทำให้สามารถทดสอบการทำงานของงจรได้เร็วกว่า ถึงแม้จะเสียเวลาในการเขียน Test Bench บ้าง แต่ถ้าวงจรที่ทำการทดสอบมีขนาดใหญ่แล้ว เวลาที่เสียไปในการเขียน Test bench น้อยมากเมื่อเทียบกับเวลาที่ต้องเสียไปในการสั่ง Force input signal ใหม่ทุกครั้งที่มีการจำลองการทำงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2 รายละเอียดของการออกแบบ

การออกแบบจะใช้หลักการของ Top Down Design คือการออกแบบโดยมองภาพรวมของซีพียูก่อนว่าควรประกอบด้วย Component อะไรบ้าง จากนั้นค่อยนำแต่ละส่วน มาแยกเป็น Component ย่อยๆ ต่อไป

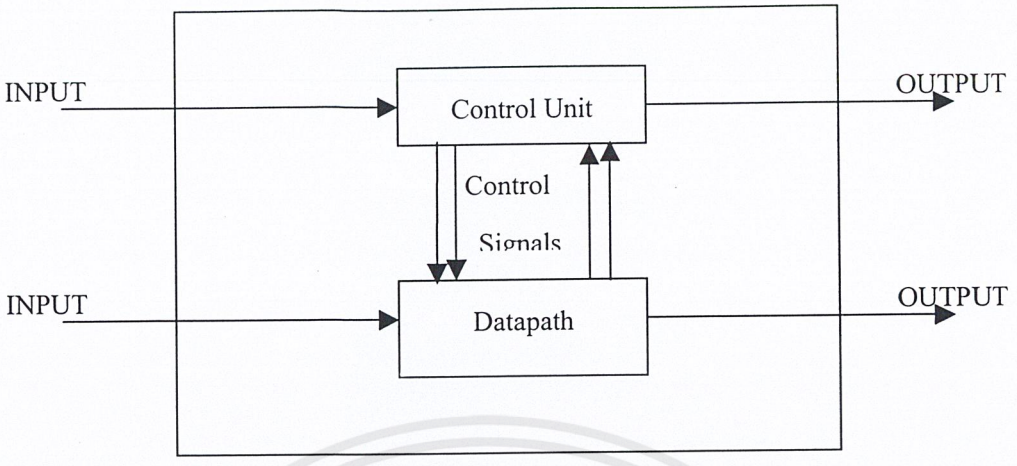
ซีพียูที่ทำการออกแบบกำหนดให้มีขาอินพุทเอาท์พุท ดังรูปที่ 5.3



รูปที่ 5.3 แสดงขา อินพุทเอาท์พุท ของ CPU ที่ออกแบบ

ทำการแบ่งซีพียูออกเป็น 2 ส่วนคือ ส่วนของ Data path และ Control unit โดยที่ภาค Data path จะประกอบด้วยหน่วยประมวลผลทางคณิตศาสตร์และลอกจิก(ALU) , Shifter unitและรีจิสเตอร์ต่างๆ ซึ่งจะถูกควบคุมการทำงานโดย Control signal ที่ส่งออกมา Control unit อีกทีหนึ่ง จากรูปที่ 5.4 แสดงถึงซีพียูที่ออกแบบ โดย Control unit จะมีการรับสัญญาณเข้าจาก Data path และจากสัญญาณที่เข้าซีพียูโดยตรง สัญญาณที่ Control unit รับมาจาก Data path คือ Instruction Register และ Status Register ส่วนสัญญาณที่ได้รับจากสัญญาณเข้า CPU โดยตรงคือ clk , reset ส่วนเอาท์พุทของ Control unit จะเป็น control signal ส่งให้ Data path เพื่อควบคุมการทำงานของรีจิสเตอร์ SHU และ ALU นอกจากนี้ Output Signal บางเส้นของ Control Unit ยังถูกส่งออกไปเป็นสัญญาณที่ออกจากซีพียูโดยตรงด้วยคือสัญญาณ rd และ wr

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.4 แสดง Component หลักภายใน CPU

ส่วน Data path ประกอบด้วย รีจิสเตอร์ Shifter_unit และ ALU โดยที่ Data path ที่ออกแบบประกอบด้วย Component ต่างๆ ดังนี้

1. Accumulator
2. Instruction Register
3. Program Counter
4. Memory Address Register
5. Status Register
6. Stack Pointer
7. General purpose Register
8. Arithmetic Logic Unit
9. Shifter Unit

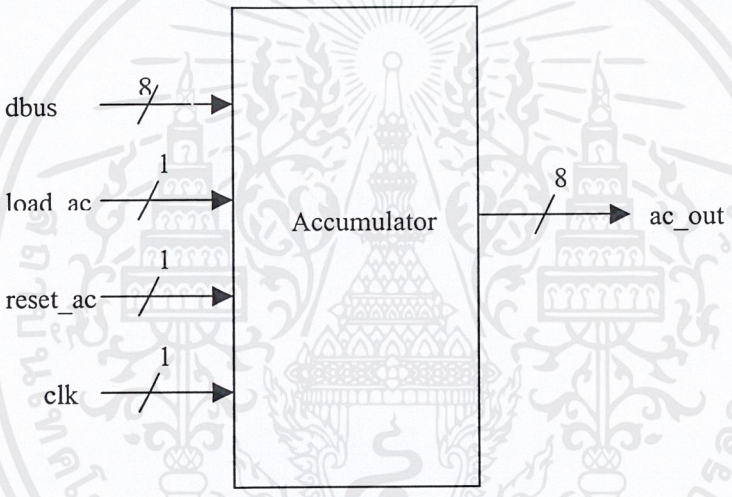
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2.1 รายละเอียดของแต่ละ Component ที่ทำการออกแบบ

1. Accumulator

```
ENTITY accumulator IS
 PORT ( dbus    : IN  byte;
        ac_out  : OUT  byte;
        load_ac, reset_ac, clk      : IN  std_logic);
END accumulator ;
```

(ก)



(ข)

รูปที่ 5.5 (ก) แสดงการเขียน Accumulator ด้วย VHDL ส่วน entity design
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ Accumulator

Accumulator เป็นรีจิสเตอร์ขนาด 8 bit ทำงานที่ขอบขาลงของสัญญาณนาฬิกา ใช้สำหรับเก็บข้อมูลที่จะส่งให้กับ ALU และเก็บผลลัพธ์ที่ได้จากการทำงานของ ALU

จะทำงานเมื่อ clk 'EVENT , clk= '0' และ load_ac= '1'

ถ้า reset_ac = '1' แล้ว ac_out= "00000000"

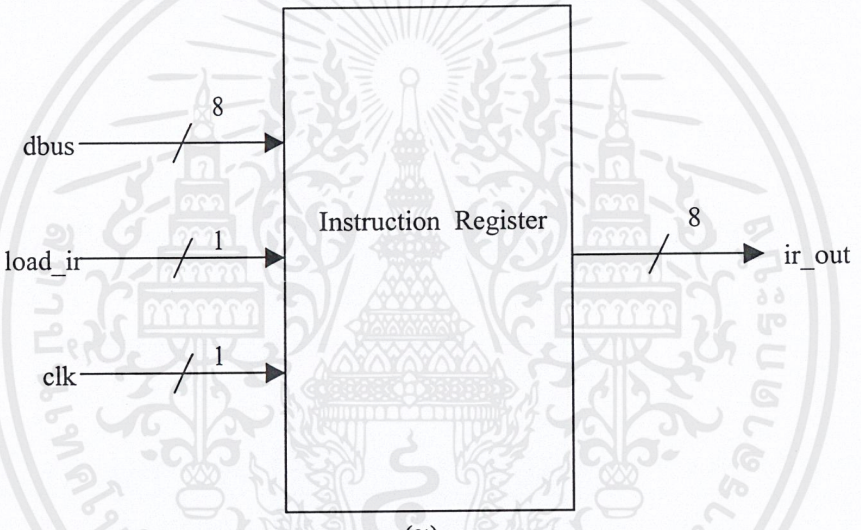
ถ้า reset_ac = '0' แล้ว ac_out= dbus

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. Instruction Register

```
ENTITY instruction_register IS
 PORT ( dbus :   IN byte;
        ir_out :  OUT byte;
        load_ir ,clk :  IN std_logic);
END instruction_register ;
```

(ก)



(ข)

รูปที่ 5.6 (ก) แสดงการเขียน instruction register ด้วย VHDL ส่วนของ entity design
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ instruction register

Instruction Register เป็น Register ขนาด 8 bit ทำงานที่ขอบขาลงของสัญญาณนาฬิกา ทำหน้าที่ใช้เก็บรหัสของคำสั่งขณะนั้นๆ เพื่อนำไปตีความ และส่งไปยัง Control Unit

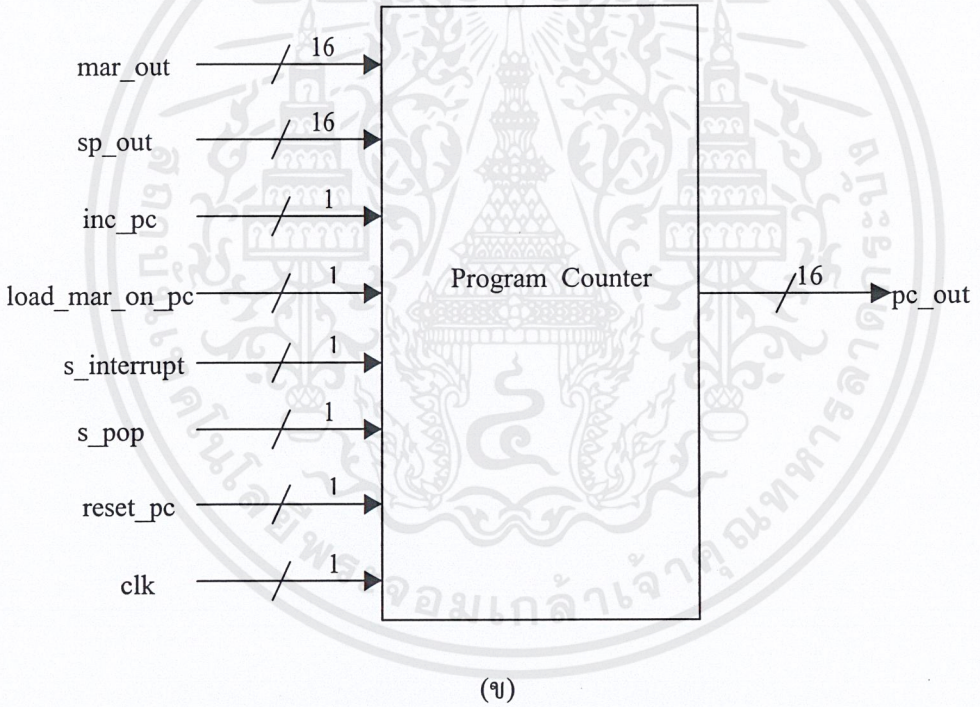
จะทำงานเมื่อ clk'EVENT , clk= '0' และ ถ้า load_ir = '1' ค่า ir_out = dbus

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. Program Counter

```
ENTITY program_counter IS
 PORT (mar_out :  IN std_logic_vector(15 downto 0);
        sp_out : IN std_logic_vector(15 downto 0);
        pc_out :  OUT std_logic_vector(15 downto 0);
        load_mar_on_pc, s_pop, inc_pc, s_interrupt,
        reset_pc , clk :  IN std_logic);
END program_counter ;
```

(ก)



(ข)

รูปที่ 5.7 (ก) แสดงการเขียน program counter ด้วย VHDL ส่วนของ entity design
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ Program Counter

Program Counter ใช้เก็บตำแหน่งของคำสั่งต่อไป ที่ CPU จะต้องอ่านมาเพื่อปฏิบัติ การปฏิบัติโปรแกรมจะเป็นไปแบบเรียงลำดับ และเพื่อที่จะดึงคำสั่งต่อไป CPU จำเป็นต้องดึงคำสั่งมาจากหน่วยความจำ ซึ่งขบวนการนี้ข้อมูลใน PC จะส่งมาบนบัสตำแหน่ง และส่งต่อไปยังหน่วยความจำ หน่วยความจำจะอ่านข้อมูลจากตำแหน่งที่ถูกอ้างอิง และส่งข้อมูลที่อ่านได้บนบัสข้อมูล CPU จะอ่านข้อมูลบนบัสข้อมูลนั้น ซึ่งข้อมูลหรือคำสั่งที่อ่านได้นี้คือคำสั่งนั่นเอง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Program Counter จะทำงานเมื่อ clk'EVENT และ clk= '0'

ถ้า reset_pc = '1' แล้ว pc_out = "0000000000000000"

ถ้า inc_pc = '1' แล้ว pc_out = pc_out + 1

ถ้า load_mar_on_pc = '1' แล้ว pc_out = mar_out

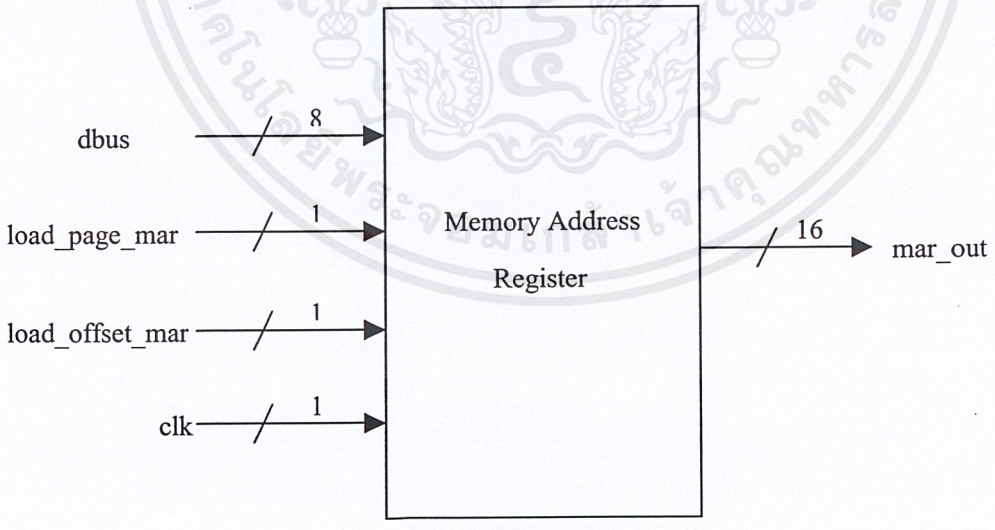
ถ้า s_pop = '1' แล้ว pc_out = sp_out

ถ้า s_interrupt = '1' ให้ pc_out = "0000000001000000" ค่า address 60 H

4. Memory Address Register

```
ENTITY memory_address_register IS
 PORT ( dbus :  IN   byte ;
          mar_out  :  OUT std_logic_vector(15 downto 0);
           load_page_mar ,load_offset_mar ,clk :  IN std_logic );
 END memory_address_register ;
```

(ก)



(ข)

รูปที่ 5.8 (ก) แสดงการเขียน memory address register ด้วย VHDL ส่วนของ entity design

(ข) แสดงสัญญาณอินพุทเอาท์พุทของ memory address register

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Memory Address Register เป็น register สำหรับเก็บตำแหน่งเพื่ออ้างอิง Address ของ memory ที่เราต้องการอ้างถึง จำพวก

จะทำงานเมื่อ clk'EVENT และ clk= '0'

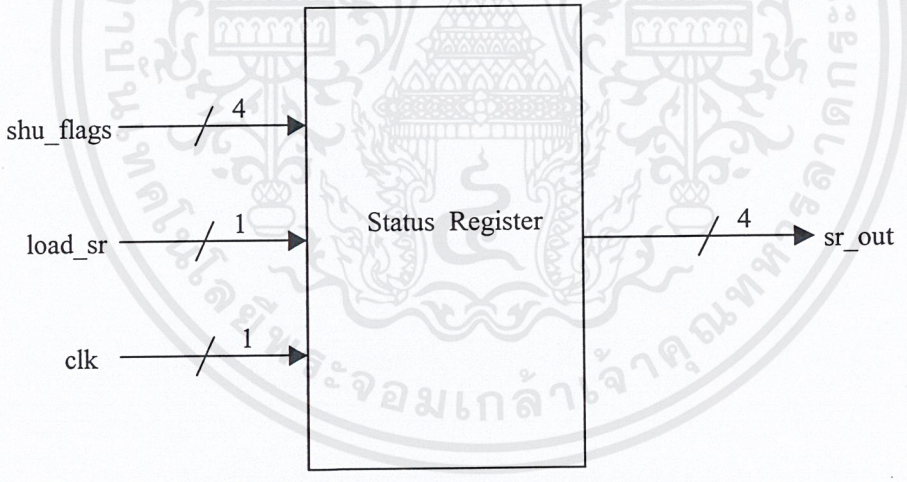
ถ้า load_page_pc = '1' แล้ว mar_out (15 downto 8) = dbus

ถ้า load_offset_pc = '1' แล้ว mar_out (7 downto 0) = dbus

5. Status Register

```
ENTITY status_register IS
 PORT ( shu_flags :  IN nibble;
          sr_out :  OUT nibble;
          load_sr , clk :  IN std_logic);
END status_register ;
```

(ก)



(ข)

รูปที่ 5.9 (ก) แสดงการเขียน Status Register ด้วย VHDL ส่วนของ entity design
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ Status Register

Status Register หรือ แฟลก รีจิสเตอร์ ใช้เก็บเงื่อนไขพิเศษต่างๆ ที่เกิดขึ้นใน CPU ซึ่งมี 4 แฟลกคือ VCZS

จะทำงานเมื่อ clk'EVENT และ clk= '0'

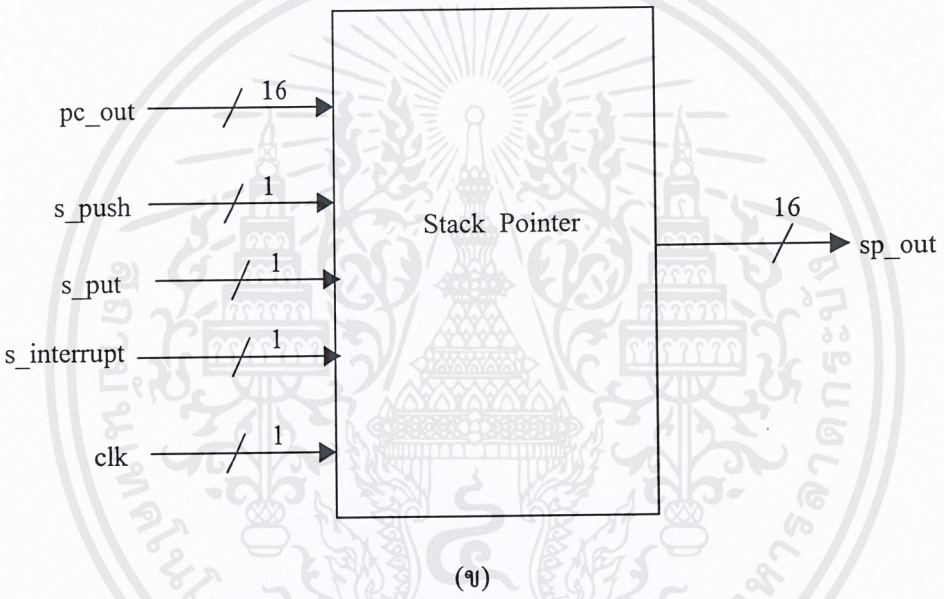
เมื่อ load_sr = '1' แล้ว ค่า sr_out = shu_flags

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6. Stack Pointer (SP)

```
ENTITY stack_pointer IS
 PORT ( pc_out :  IN std_logic_vectoe(15 downto 0);
          sp_out : OUT std_logic_vector(15 downto 0);
          s_push, s_put, s_interrupt ,clk :  IN std_logic);
 END stack_pointer ;
```

(ก)



(ข)

รูปที่ 5.10 (ก) แสดงการเขียน Stack Pointer ด้วย VHDL ส่วน entity design
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ Stack Pointer

stack pointer เป็น register ใช้สำหรับเก็บค่าตำแหน่ง address ขณะที่เกิดการ interrupt หรือมีการใช้คำสั่ง call และเมื่อมีคำสั่ง ret ก็จะส่งค่าจาก stack กลับไปยัง program counter

จะทำงานเมื่อ clk'EVENT และ clk='0'

เมื่อสัญญาณ s_push ='1' ค่า sp_out = pc_out - 1

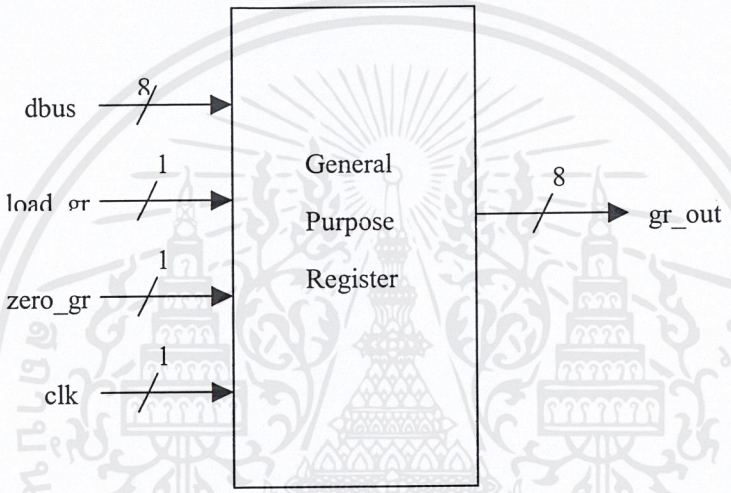
เมื่อสัญญาณ s_put , s_interrupt = '1' ค่า sp_out = pc_out

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7. General Purpose Register

```
ENTITY general_purpose_register IS
 PORT ( dbus     : IN  byte;
        gr_out  : OUT  byte;
        load_gr,reset_gr,clk        : IN  std_logic);
END general_purpose_registerr ;
```

(ก)



(ข)

รูปที่ 5.11 (ก) แสดงการเขียน General Purpose Register ด้วย VHDL ส่วน entity design
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ General Purpose Register

General Purpose Register เป็นรีจิสเตอร์ขนาด 8 bit ทำงานที่ขอบขาลงของสัญญาณนาฬิกา ใช้สำหรับเก็บข้อมูลที่จะส่งให้กับ dbus เพื่อไปกระทำทางคณิตศาสตร์และลอกจิกที่ ALU หรือ Memory Address Registor ต่อไป ในซีพียูที่ทำการออกแบบมี register ไม่รวม accumulator ทั้งหมด 7 ตัว

จะทำงานเมื่อ clk 'EVENT , clk='0' และ load_gr='1'

ถ้า reset_gr = '1' แล้ว gr_out= "00000000"

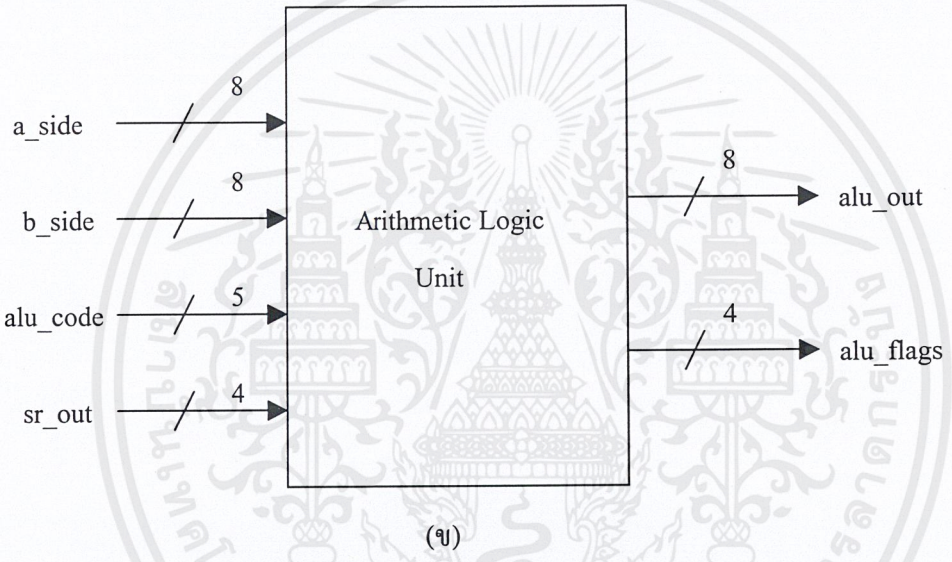
ถ้า zero_gr = '0' แล้ว gr_out= dbus

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8. Arithmetic Logic Unit(ALU)

```
ENTITY arithmetic_logic_unit IS
 PORT ( a_side, b_side :  IN byte;
          alu_code :  IN std_logic_vector(4 downto 0) ;
          sr_out :  IN nibble; alu_out : OUT byte;
          alu_flags :  OUT nibble);
END arithmetic_logic_unit;
```

(ก)



(ข)

รูปที่ 5.12 (ก) แสดงการเขียน Arithmetic Logic Unit ด้วย VHDL ส่วน entity unit
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ Arithmetic Logic Unit

Arithmetic Logic Unit หรือ ALU ใช้ในการประมวลผลการกระทำทางคณิตศาสตร์และลอจิก จะรับอินพุท 8 bit มา 2 ค่าและมา operate กัน โดยดูจาก alu_code ว่าทำอะไร

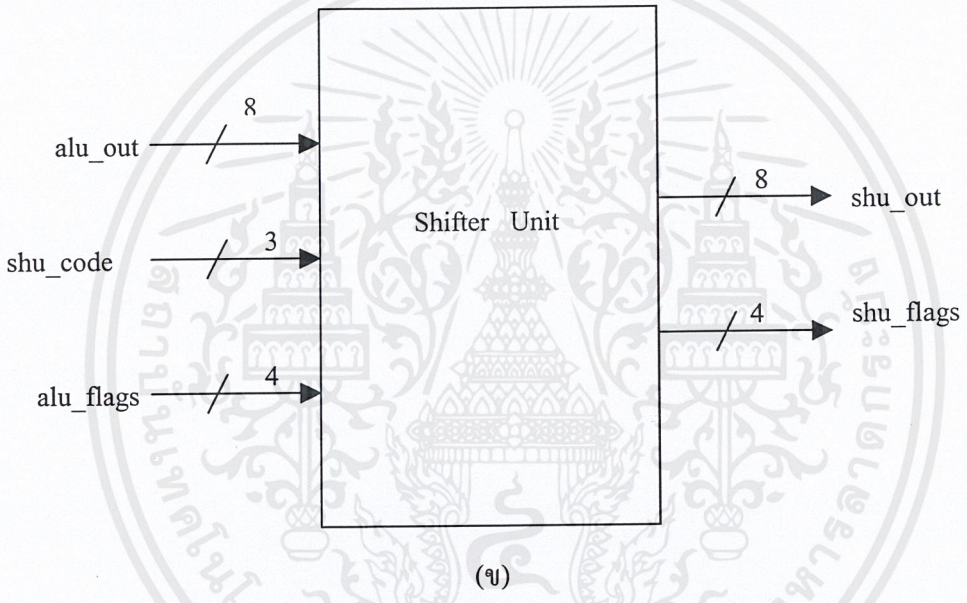
การทำงานของหน่วยนี้จะรับค่า temp กับ ac_out และ alu_code เป็นตัวบอกว่าจะนำ ทั้ง 2 มา operate กันอย่างไร เป็นส่วนหนึ่งของ opcode ซึ่งจะได้ alu_out ออกมาเป็นผลจากการกระทำ และ alu_flags เป็นสถานะพิเศษหลังจากผ่านการกระทำนั้นแล้ว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

9. Shifter Unit

```
ENTITY shifter_unit IS
 PORT ( alu_out  :  IN byte;
        shu_code :  IN std_logic_vector(2 downto 0);
        alu_flags :  IN nibble;
        shu_out :  OUT byte; shu_flags :  OUT nibble);
END shifter_unit;
```

(ก)



(ข)

รูปที่ 5.13 (ก) แสดงการเขียน Shifter Unit ด้วย VHDL ส่วนของ entity design
(ข) แสดงสัญญาณอินพุทเอาท์พุทของ Shifter Unit

Shifter Unit เป็นหน่วยสำหรับการ Shift ข้อมูลโดยรับข้อมูลมาแล้วทำการเลื่อนข้อมูลหรือหมุนข้อมูลตาม shu_code ที่สั่งเข้ามา

แล้ว dbus = shift_data และ sr_in = shu_flags

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

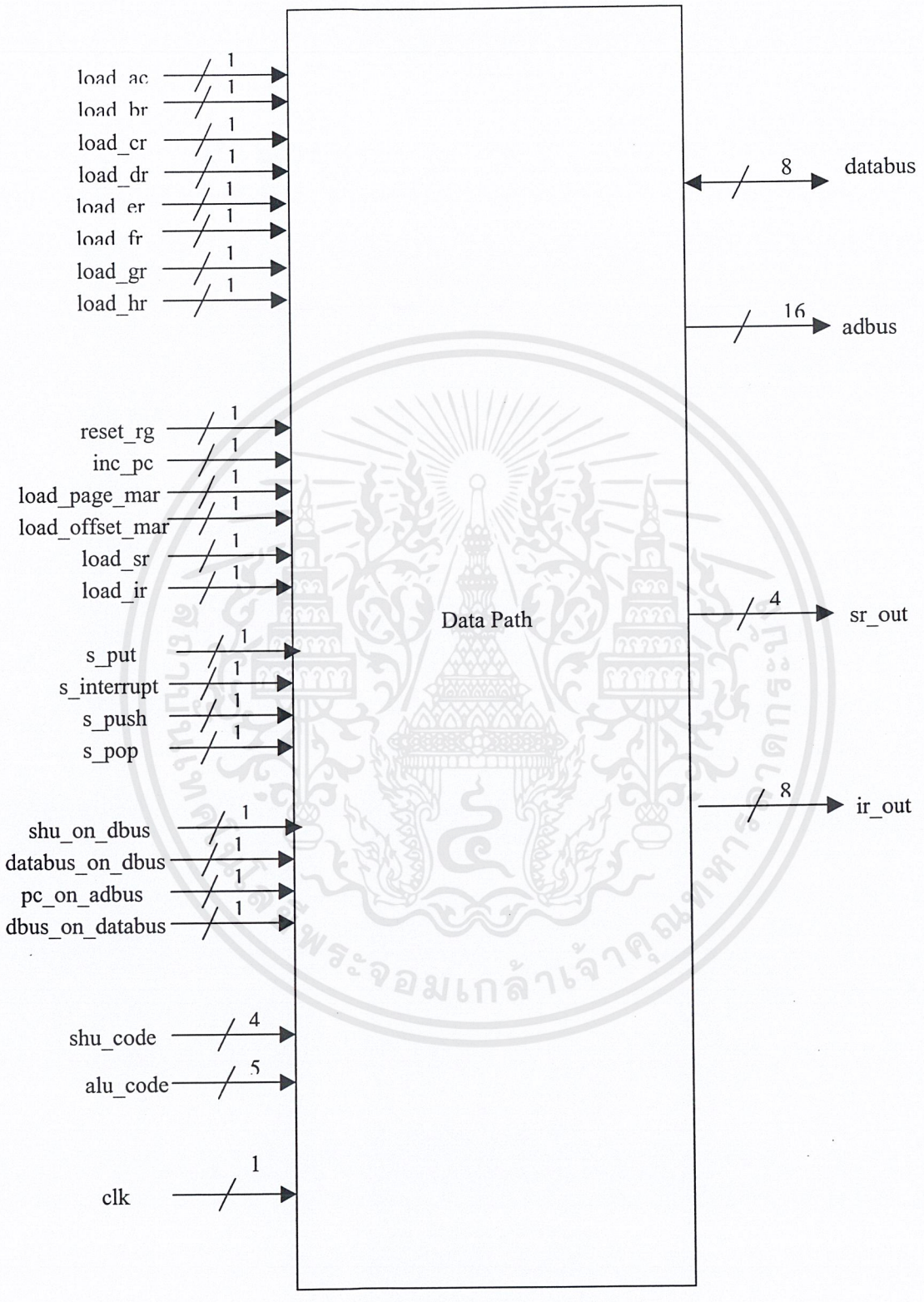
5.2.2 การนำแต่ละ Component มารวมกันเป็น Data path

เมื่อออกแบบแต่ละ Component เสร็จแล้วก็นำแต่ละ Component มา Map ลงบน Data path จากนั้นก็ทำการออกแบบ Control Unit ต่อไป

```
ENTITY data_path IS
 PORT(databus : INOUT byte;  adbus: OUT std_logic_vector(15 downto 0);
        clk : IN std_logic;
         load_ac, reset_rg, load_br load_cr, load_dr load_er load_fr
        load_gr load_hr, load_ir,
         inc_pc, s_put,s_push, s_pop,load_mar_on_pc,
         load_page_mar, load_offset_mar,
         load_sr, s_ interrupt,
         databus_on_dbus, pc_on_adbus, dbus_on_databus
        shu_on_dbus : IN std_logic;
         alu_code :  IN std_logic_vector(4 downto 0);
        shu_code : IN std_logic_vector(3 downto 0);
         ir_out :  OUT byte; sr_out :  OUT nibble);
END data_path;
```

รูปที่ 5.14 (ก) แสดงการเขียน Data path ด้วย VHDL ส่วน entity design

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.14 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Data path

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

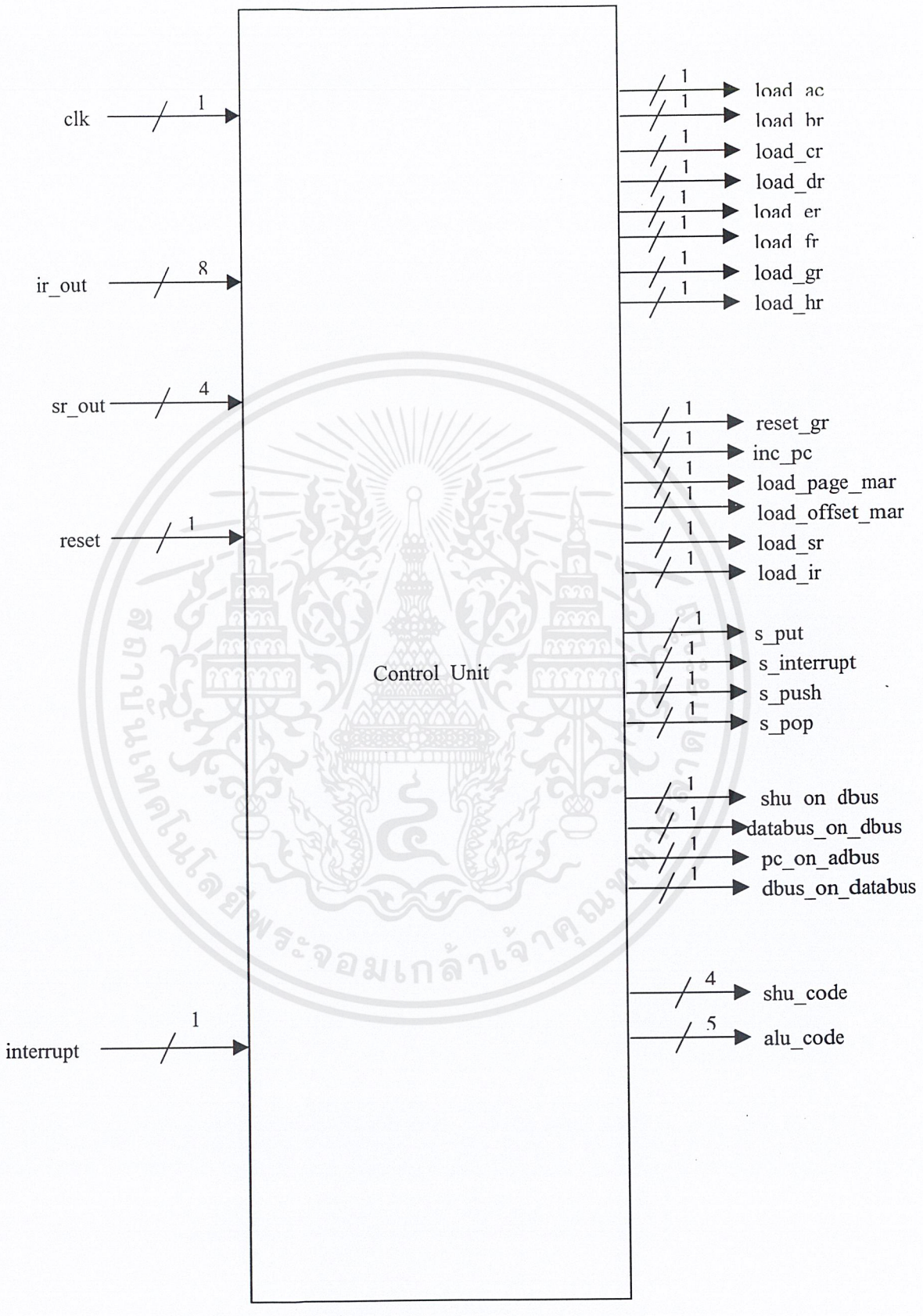
5.2.3 การออกแบบ Control Unit

Control Unit นั้นมีหน้าที่ส่งสัญญาณออกไปควบคุม Datapath และในขณะเดียวกันก็รับข้อมูลบางสัญญาณจาก Datapath เข้ามาเพื่อใช้ในการตัดสินใจในการทำคำสั่งต่างๆ

```
ENTITY control_unit IS
 PORT( clk : IN std_logic;
          load_ac, reset_rg, load_br load_cr, load_dr load_er load_fr
         load_gr load_hr, load_ir,
          inc_pc, s_put,s_push, s_pop,load_mar_on_pc,
          load_page_mar, load_offset_mar,
          load_sr, s_ interrupt,
          databus_on_dbus, pc_on_adbus, dbus_on_databus
         shu_on_dbus : OUT std_logic;
          alu_code :  OUT std_logic_vector(4 downto 0);
         shu_code : OUT std_logic_vector(3 downto 0);
          ir_out :  IN byte; sr_out :  IN nibble);
          rd , wr :  OUT std_logic ; reset,interrupt :  IN std_logic);
 END  control_unit;
```

รูปที่ 5.15 (ก) แสดงการเขียน control unit ด้วย VHDL ส่วนของ entity design

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.15 (ข) แสดงสัญญาณอินพุทเอาท์พุทของ Control Unit

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2.4 สัญญาณที่อินพุทเอาท์พุทของ Control Unit

Control Unit เป็นตัวให้กำเนิดสัญญาณต่างๆ เพื่อควบคุมการทำงานของ CPU ซึ่งจะแบ่งเป็นการควบคุมเป็นหน่วยต่างๆ คือ Register Control Signal , Bus Connection Control Signal , Arithmetic Logic Unit Control Signal ,Shifter Unit Control Signal และ Memory Control and external signals

Register Control Signal

load_ac : ทำให้ ACC เก็บข้อมูล

reset_rg : ทำให้รีจิสเตอร์ทุกตัวทำการ reset ข้อมูล

load_br : ทำให้ b_register เก็บข้อมูล

load_cr : ทำให้ c_register เก็บข้อมูล

load_dr : ทำให้ d_register เก็บข้อมูล

load_er : ทำให้ e_register เก็บข้อมูล

load_fr : ทำให้ f_register เก็บข้อมูล

load_gr : ทำให้ g_register เก็บข้อมูล

load_hr : ทำให้ h_register เก็บข้อมูล

load_ir : ทำให้ ir เก็บค่าข้อมูล

inc_pc : ทำให้ pc เพิ่มค่า address ภายในขึ้นเท่ากับ pc+1

load_page_mar : ทำให้ mar เก็บค่าส่วนที่เป็น page address

load_offset_mar : ทำให้ mar เก็บค่าส่วนที่เป็น offset address

load_sr : ทำให้ sr เก็บค่า status

Bus Connection Control Signal

shu_on_dbus : ส่งข้อมูลจาก shu_out ไปยัง dbus

databus_on_dbus : ส่งข้อมูลจาก databus ไปยัง dbus

dbus_on_databus : ส่งข้อมูลจาก dbus ไปยัง databus

pc_on_adbus : ส่งข้อมูลจาก pc ไปยัง adbus

dbus เป็น internal databus

databus เป็น bus ข้อมูลภายนอก

adbus คือ address bus

shu_out เป็นค่า output ของ shifter unit

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Arithmetic Logic Unit Control Signal

alu_code : เป็นสัญญาณที่กำหนดการทำงานของ ALU ว่าจะทำงานในลักษณะใด

1. a_and_cont: สั่งให้ ALU นำค่า accumulator มา and กับค่าคงที่
2. a_or_cont: สั่งให้ ALU นำค่า accumulator มา and กับค่าคงที่
3. a_xor_cont: สั่งให้ ALU นำค่า accumulator มา and กับค่าคงที่
4. a_not: สั่งให้ ALU นำค่า accumulator มากลับบิต
5. a_add_b: สั่งให้ ALU นำค่า input ทั้ง 2 มาบวกกัน
6. a_adc_b: สั่งให้ ALU นำค่า input ทั้ง 2 มาบวกกันโดยนำค่า carry flag ด้วย
7. a_sub_b: สั่งให้ ALU นำค่า input ทั้ง 2 มาลบกัน
8. a_suc_b: สั่งให้ ALU นำค่า input ทั้ง 2 มาลบกันโดยนำค่า carry flag มาคิดด้วย
9. incr_a: สั่งให้ ALU นำค่า accumulator มาบวก 1
10. inc: สั่งให้ ALU นำค่า register ที่ต้องการมาบวก 1
11. decr_a: สั่งให้ ALU นำค่า accumulator มาลบ 1
12. dec: สั่งให้ ALU นำค่า register ที่ต้องการมาลบ 1
13. set_a: สั่งให้ ALUนำค่า accumulator มา set บิตที่ต้องการ
14. res_a: สั่งให้ ALUนำค่า accumulator มา reset บิตที่ต้องการ
15. bit_a: สั่งให้ ALU นำค่า accumulator เช็ค bit ที่ต้องการว่าเป็น 0 หรือไม่
16. cp_a: สั่งให้ ALU นำค่า accumulator เช็คกับค่าคงที่ว่าเท่ากันหรือไม่
17. a_input: สั่งให้ ALU นำค่า accumulator ออกไปเพื่อไปกระทำที่ shifter unit

Shifter Unit control Signal

shu_code : เป็นสัญญาณที่กำหนดการทำงานของ SHU ว่าจะทำงานลักษณะใด

1. ror สั่งหมุนข้อมูลไปทางขวา 1 บิต โดยนำ carry flag มาหมุน
2. rol สั่งหมุนข้อมูลไปทางซ้าย 1 บิต โดยนำ carry flag มาหมุน
3. rrc สั่งหมุนข้อมูลไปทางขวา 1 บิต โดยนำค่าบิตที่ 0 เก็บเข้า carry flag
4. rlc สั่งหมุนข้อมูลไปทางขวา 1 บิต โดยนำค่าบิตที่ 7 เก็บเข้า carry flag
5. srs สั่งเลื่อนข้อมูลไปทางขวา 1 บิต โดยนำบิตที่ 0 เก็บเข้า carry flag และนำ1 เก็บเข้าบิตที่ 7
6. sls สั่งเลื่อนข้อมูลไปทางซ้าย 1 บิต โดยนำบิตที่ 7 เก็บเข้า carry flag และนำ1 เก็บเข้าบิตที่ 0
7. srr สั่งเลื่อนข้อมูลไปทางขวา 1 บิต โดยนำบิตที่ 0 เก็บเข้า carry flag และนำ 0 เก็บเข้าบิตที่ 7
8. slr สั่งเลื่อนข้อมูลไปทางซ้าย 1 บิต โดยนำบิตที่ 7 เก็บเข้า carry flag และนำ 0 เก็บเข้าบิตที่ 0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Memory Control and External Signals

rd : เป็นสัญญาณแสดงว่า CPU ต้องการอ่านข้อมูลจาก memory

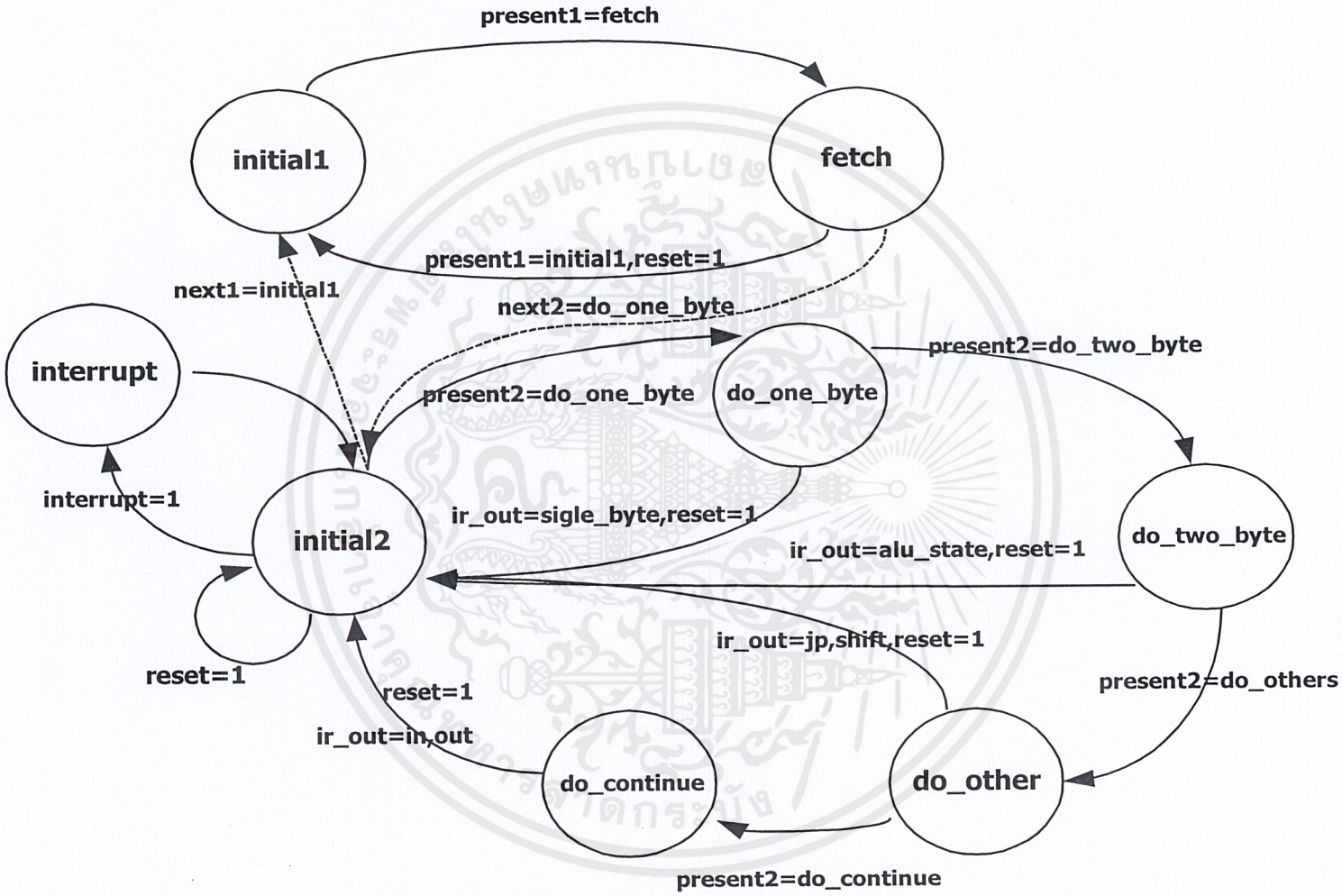
wr : เป็นสัญญาณแสดงว่า CPU ต้องการส่งข้อมูลไปยัง memory

reset : เป็นสัญญาณใช้สำหรับ reset

interrupt : เป็นสัญญาณสำหรับแจ้งว่ามีการ interrupt



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.16 แสดง state machine ของซีพียู

Instruction Mnermonic	Opcode 7 6 5 4 3 2 1 0
ld	0 0 0 0 x x x -
ldr	0 0 0 1 x x x -
push	0 0 1 0 - - - -
pop	0 0 1 1 - - - -
add	0 1 0 0 0 x x x
sub	0 1 0 0 1 x x x
inc	0 1 0 1 0 x x x
dec	0 1 0 1 1 x x x
adc	0 1 1 0 0 x x x
suc	0 1 1 0 1 x x x
or	0 1 1 1 0 0 0 -
xor	0 1 1 1 0 0 1 -
not	0 1 1 1 0 1 0 -
and	0 1 1 1 0 1 1 -
set	0 1 1 1 1 0 0 -
res	0 1 1 1 1 0 1 -
bit	0 1 1 1 1 1 0 -
cp	0 1 1 1 1 1 1 -

Instruction Mnermonic	Opcode 7 6 5 4 3 2 1 0
ror	1 0 0 0 0 0 - -
rol	1 0 0 0 0 1 - -
rrc	1 0 0 0 1 0 - -
rlc	1 0 0 0 1 1 - -
srs	1 0 0 1 0 0 - -
sls	1 0 0 1 0 1 - -
srr	1 0 0 1 1 0 - -
slr	1 0 0 1 1 1 - -
jp	1 1 0 0 0 0 - -
jz	1 1 0 0 0 1 - -
jnz	1 1 0 0 1 0 - -
jc	1 1 0 0 1 1 - -
jnc	1 1 0 1 0 0 - -
call	1 1 1 0 0 0 - -
ret	1 1 1 0 0 1 - -
in	1 1 1 0 1 0 - -
out	1 1 1 0 1 1 - -

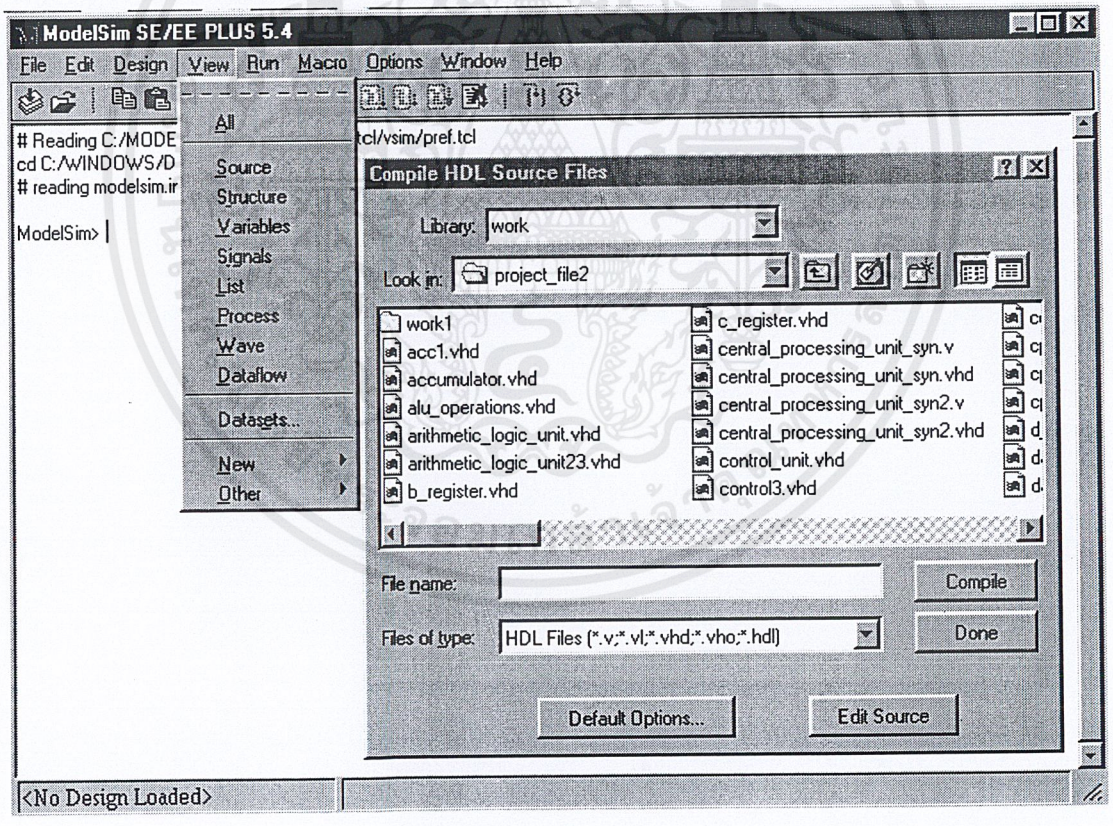
ตารางที่ 5.1 แสดง opcodes ของชุดคำสั่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6
การทดสอบการทำงานของซีพียูด้วยการ Simulation

6.1 ซอฟท์แวร์ทูลที่ใช้ : โปรแกรม ModelSim SE-EE5.4

โปรแกรม ModelSim SE-EE5.4 เป็นโปรแกรมที่ใช้ในการ Simulation การทำงานของ VHDL Code ว่าทำงานตรงตามที่ต้องการหรือไม่ โดยจะแสดงผลให้ผู้ออกแบบดูในรูปของ Wave form เนื่องจากก่อนที่จะนำ VHDL Code ไปทำการซิมูเลชั่นจะต้องมีการคอมไพล์ ตัว Code เพื่อตรวจสอบความถูกต้องของตัว Code ก่อนดังนั้น โปรแกรม ModelSim SE-EE5.4 จึงมีประโยชน์อีกประการหนึ่งคือสามารถใช้การตรวจสอบว่า VHDL Code ที่เขียนเสร็จแล้วนั้นถูกต้องตามหลักไวยากรณ์ของภาษา VHDL หรือไม่

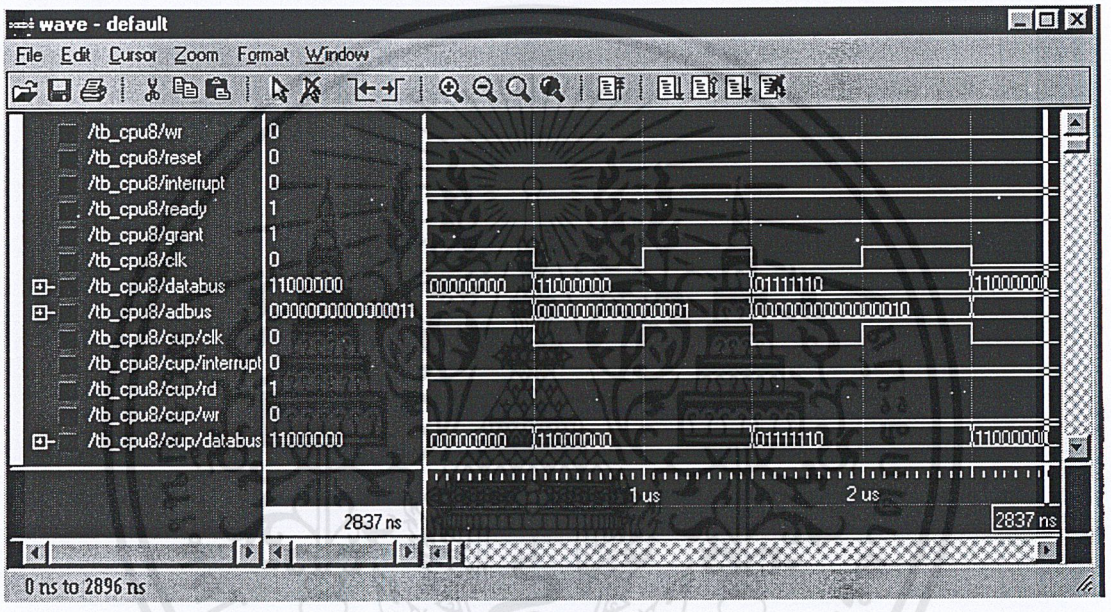


รูปที่ 6.1 แสดงหน้าจอของโปรแกรม ModelSim SE-EE5.4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.2 การสร้าง Test bench และการทดสอบ

ในการทดสอบการทำงานของโปรแกรมจะต้องเขียน VHDL Code สร้างวงจรจำลอง ROM ภายนอกและนำวงจรดังกล่าวมาต่อกับ CPU และทำการทดสอบโดยเขียน Entity ขึ้นมาใหม่อีก 1 ตัว ทำหน้าที่นำเอา Entity ที่ได้ดังกล่าวจากข้างต้นมาทำการ Map ให้เป็นวงจรและทดสอบ Simulation ดูผลลัพธ์ที่ออกมา



รูปที่ 6.2 แสดง wave form ที่ได้จากการเขียน Test bench

ซึ่งรูปกราฟของสัญญาณที่ได้จากการ Simulation จะนำเสนอในภาคผนวก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 7

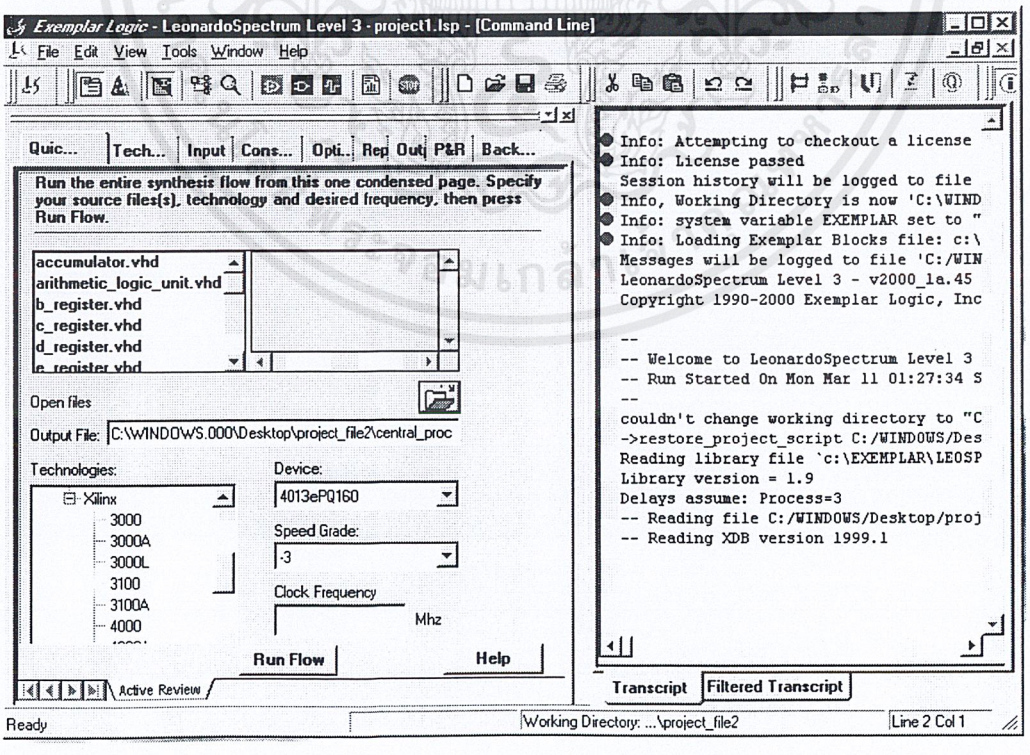
การ Synthesis และการบันทึกโปรแกรมลง FPGA

7.1 ซอฟท์แวร์ทูลล์ที่ใช้

7.1.1 โปรแกรม Leonardo Spectrum

เป็นโปรแกรมที่ใช้ในการ Synthesis วงจรเพื่อแปลง VHDL Code ให้เป็นวงจรระดับ gate-level ซึ่งในขบวนการ Synthesis นั้นผู้ใช้ต้องกำหนดว่าจะ Synthesis ลงฮาร์ดแวร์ตัวใด และยังสามารถกำหนดรูปแบบของการ Synthesis ได้ว่าต้องการวงจรแบบมีจำนวนเกทน้อยที่สุดหรือต้องการวงจรแบบมีความเร็วสูงสุด ผู้ใช้สามารถเลือกเอาท์พุทของการ Synthesis ได้หลายแบบเช่น สร้างไฟล์ นามสกุล .VHD, .V, .XNF หากเลือกเอาท์พุทเป็นไฟล์ .EDF,.XNF ก็จะได้ไฟล์ที่สามารถทำการ Mapping และ Place & Route เพื่อลง FPGA ต่อได้

โปรแกรม Leonardo Spectrum อนุญาตให้ผู้ใช้สามารถกำหนดค่า Timing Constraint ของวงจรได้ หากวงจรที่สร้างไม่สามารถบรรจุลงในฮาร์ดแวร์ที่กำหนดได้โปรแกรมจะแจ้งให้ทราบและพยายาม map กับอุปกรณ์ตัวที่มีความจุสูงกว่าซึ่งอยู่ในตระกูลเดียวกันให้



รูปที่ 7.1 แสดงหน้าจอของโปรแกรม Leonardo Spectrum

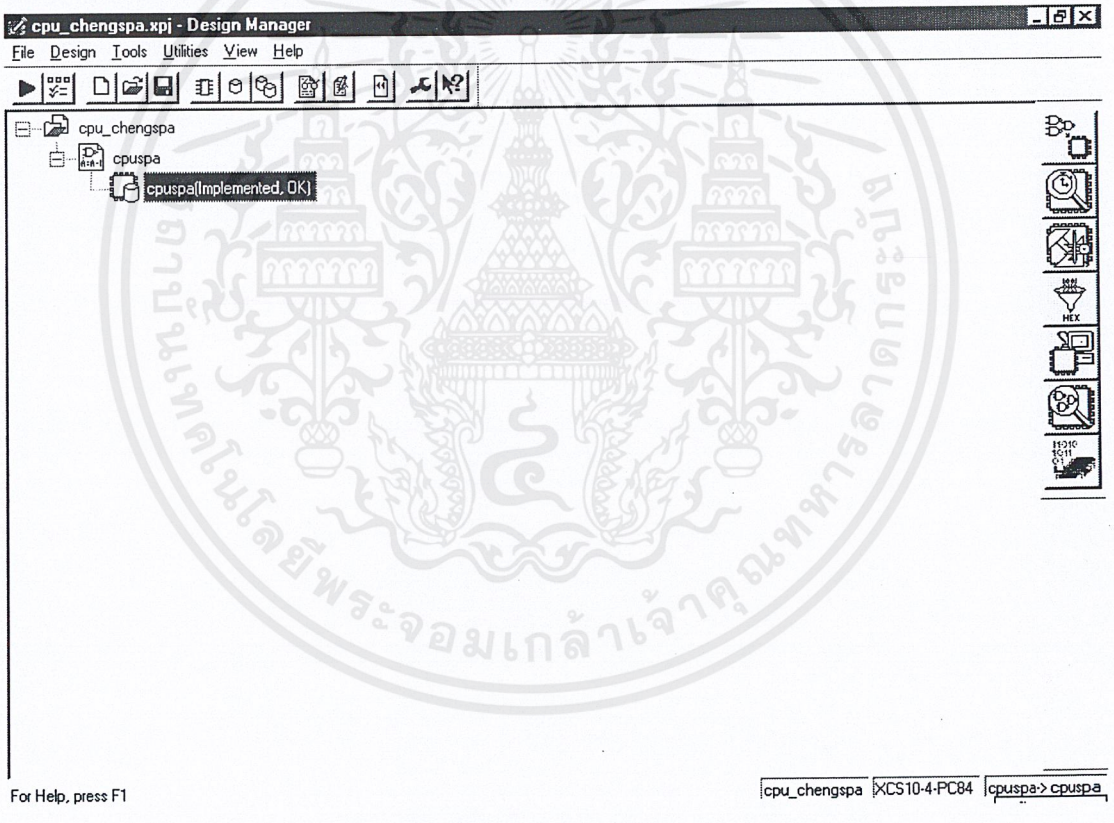
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.1.2 โปรแกรม Xilinx Foundation

ใช้ในการทำ Place and Route เพื่อทำการ map วงจรที่ Synthesis แล้วลงบน FPGA โดย Software ตัวนี้จะประกอบด้วยโมดูลหลายโมดูลด้วยกัน ได้แก่ Design Manager , Flow Engine, Timing Analyzer , EPROM File Formatter, Hardware Debugger , EPIC Design Editor และ JTAG Programmer

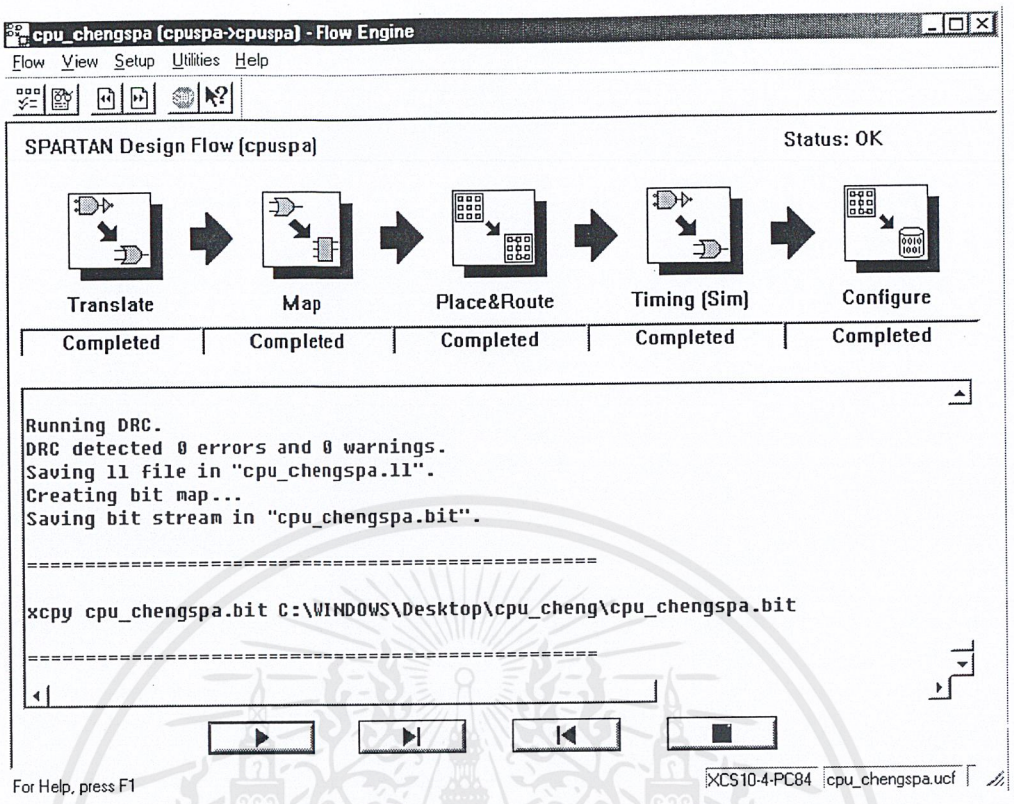
ผู้ใช้จะใช้โมดูล Design Manager ในการจัดการกับตัว Design ของผู้ใช้ และโมดูลอื่นๆ ใช้ประโยชน์ต่างๆ ไป

Flow Engine ใช้ในกระบวนการ Place and Route , Timing Analyzer และสร้าง file นามสกุล .BIT ซึ่งนำไปบันทึกลง FPGA โดย และจะใช้ Hardware Debugger ในการบันทึกลง FPGA

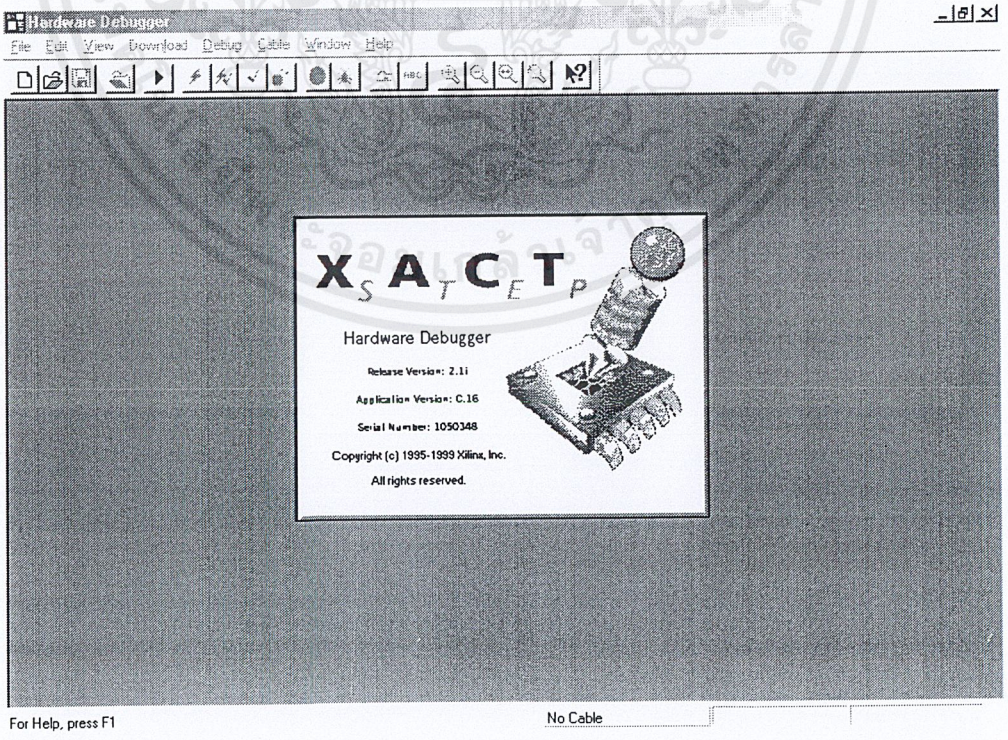


รูปที่ 7.2 แสดงหน้าจอของโปรแกรม Xilinx Foundation

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7.3 แสดงหน้าจอของโมดูล Flow Engine ใน Xilinx Foundation



รูปที่ 7.4 แสดงหน้าจอของโมดูล Hardware Debugger ใน Xilinx Foundation

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

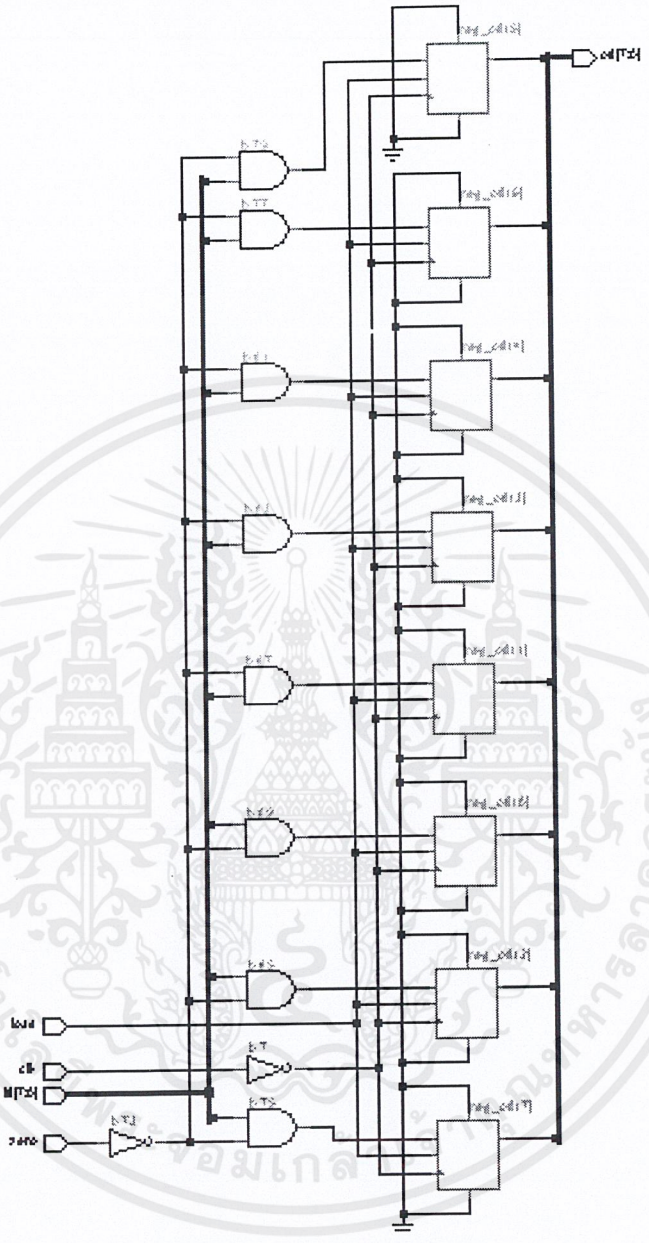
7.2 การ Synthesis วงจร

การ Synthesis คือการแปลง VHDL Code ที่ผ่านการ Simulation แล้วให้ได้วงจรระดับ Gate-level ซึ่งจะเป็น Netlist ของเกทต่างๆ ในวงจร การ Synthesis เราสามารถทำได้ 2 แบบ คือ

1. แบบ Technology-independent คือไม่อ้างอิงกับเทคโนโลยีใดๆ หมายถึงผู้ใช้สามารถนำ Netlist ที่ได้จากการ Synthesis ไปใช้ในการสร้างวงจรด้วยเทคโนโลยีอะไรก็ได้ เช่น FPGA หรือ PLD
2. แบบ Technology-dependent ที่ผู้ใช้ต้องระบุถึง Technology ที่ใช้ ชื่อผู้ผลิตฮาร์แวร์ และหมายเลขรุ่นของฮาร์ดแวร์ที่ใช้ด้วย

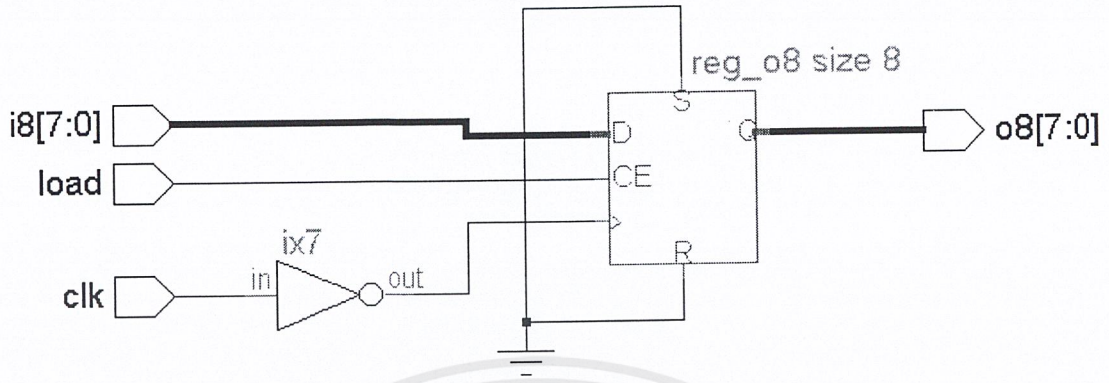
โปรแกรมจะนำข้อมูลดังกล่าวไปทำ Optimization ซึ่งผู้ใช้จะต้องระบุว่าต้องการทำ Optimization แบบใดระหว่าง Speed Optimization กับ Area Optimization ซึ่งหากเลือกวงจรที่ได้มีความเร็วสูงสุด(speed Optimization) แล้ว วงจรที่ได้มักมีขนาดใหญ่ แต่หากเลือกวงจรที่ได้มีขนาดเล็กสุด(Area Optimization) แล้วความเร็วของวงจรมักจะตกลง ถึงแม้ว่าทุก statement ในภาษา VHDL สามารถ Simulate ได้ แต่ในการ Simulate นั้น ไม่สามารถทำได้กับทุก statement เพราะไม่สามารถสร้างเป็นฮาร์ดแวร์ได้ เช่น Delay, wait เป็นต้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

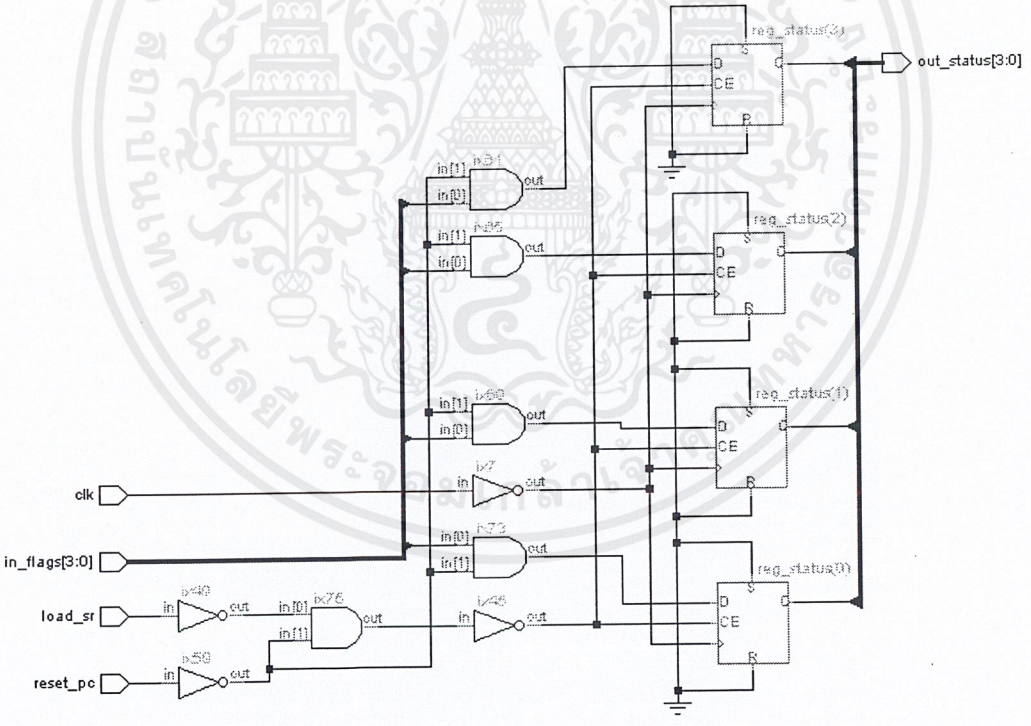


รูปที่ 7.5 แสดงรูปวงจรที่ได้จากการ Synthesis ของ accumulator และ General purpose register

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

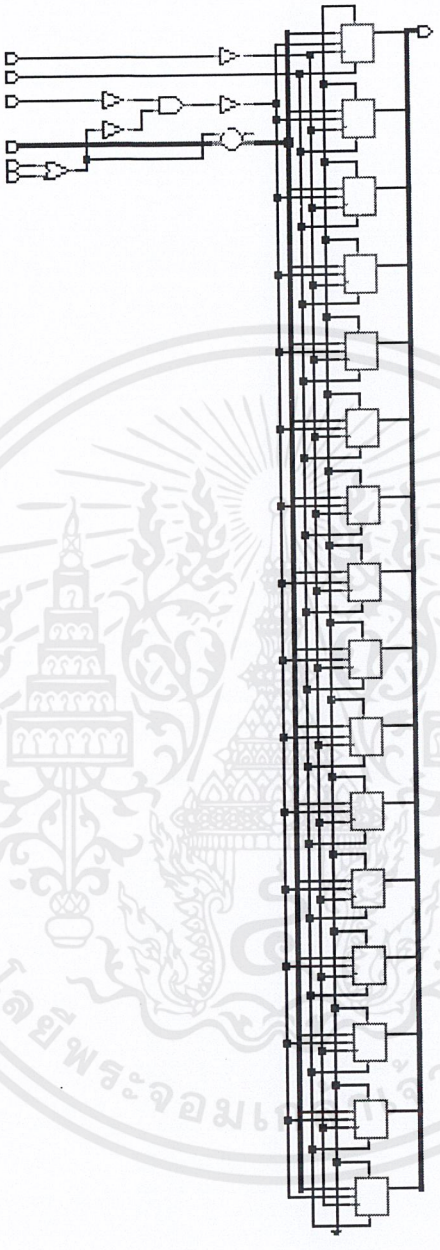


รูปที่ 7.6 แสดงรูปวงจรที่ได้จากการ Synthesis ของ instruction register



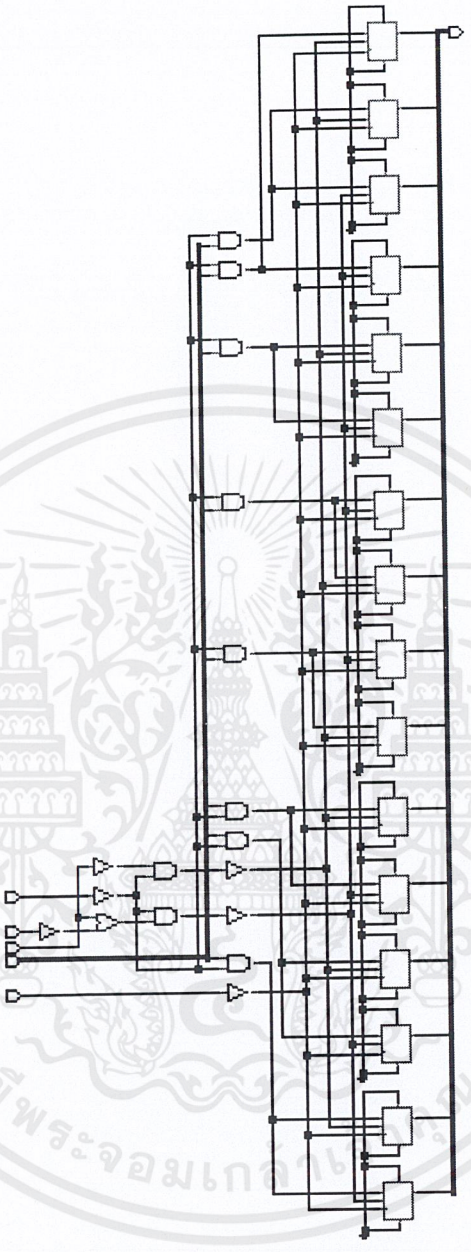
รูปที่ 7.7 แสดงรูปวงจรที่ได้จากการ Synthesis ของ status register

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



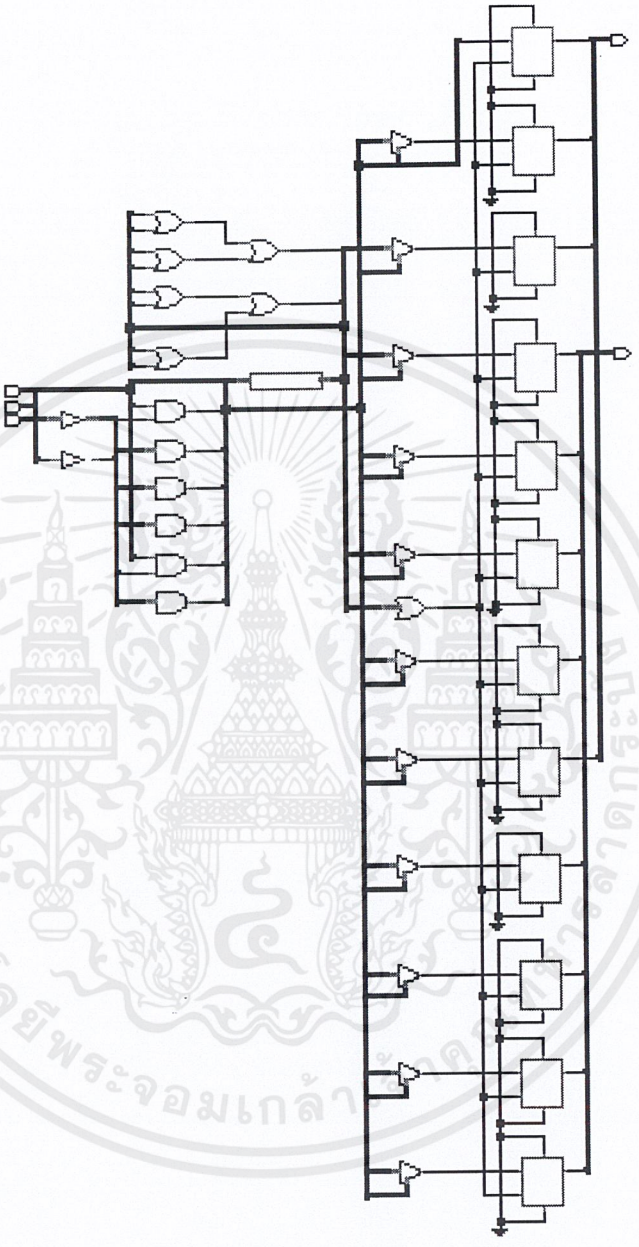
รูปที่ 7.8 แสดงรูปวงจรที่ได้จากการ Synthesis ของ stack pointer

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



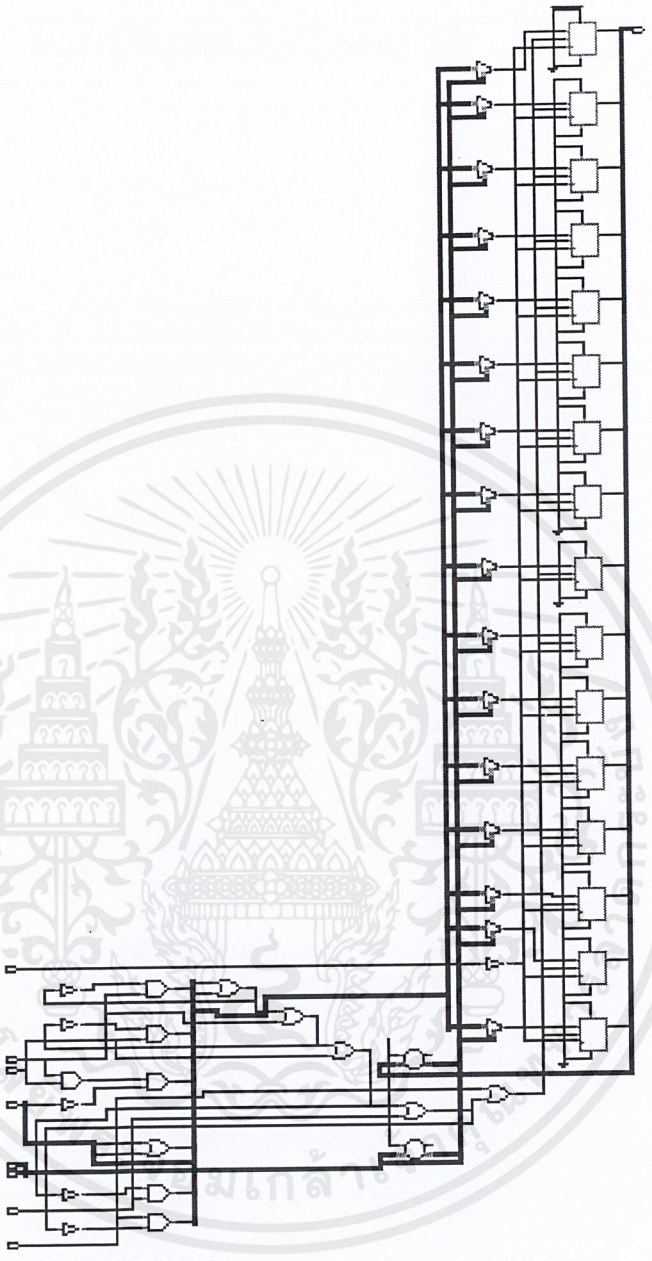
รูปที่ 7.9 แสดงรูปวงจรที่ได้จากการ Synthesis ของ memory address register

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



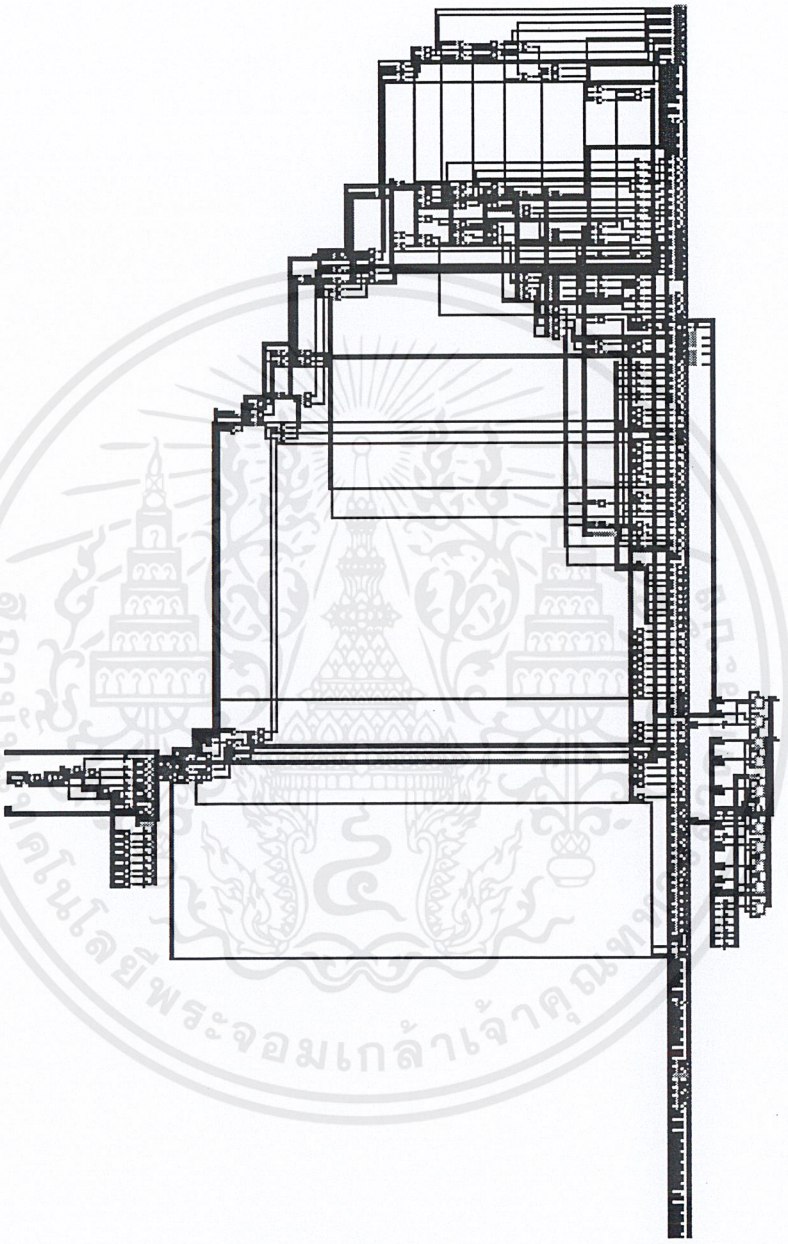
รูปที่ 7.10 แสดงรูปวงจรที่ได้จากการ Synthesis ของ shifter unit

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



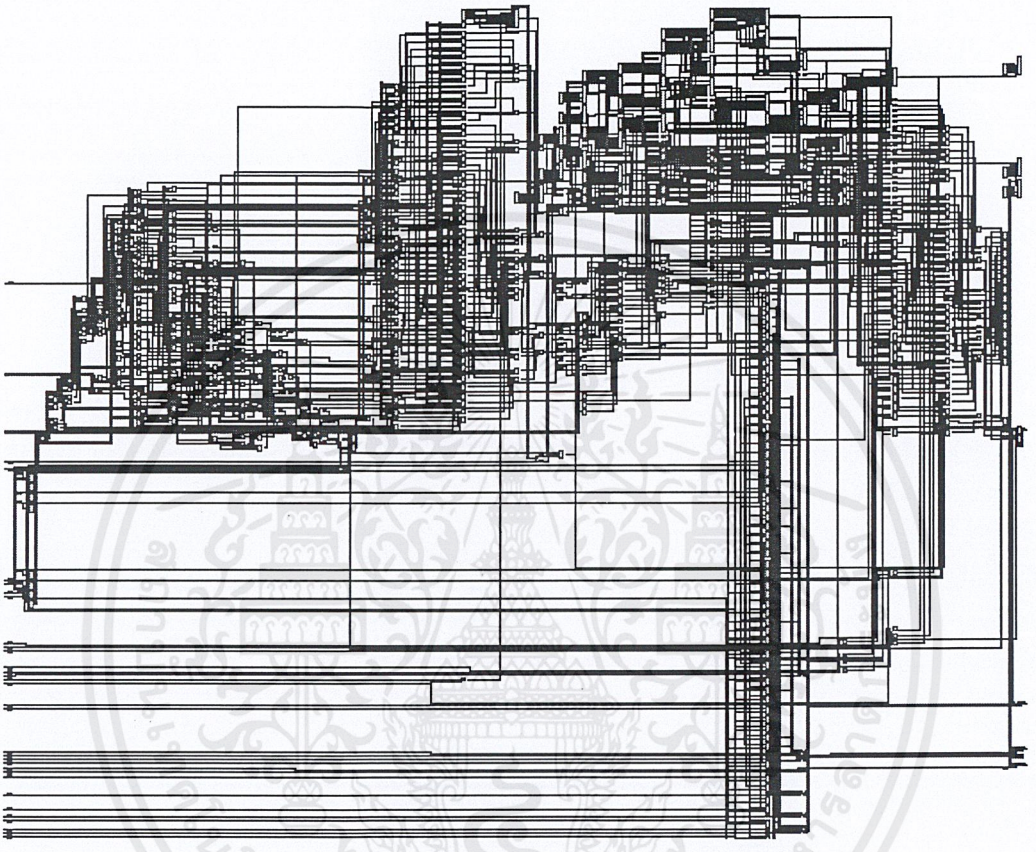
รูปที่ 7.11 แสดงรูปวงจรที่ได้จากการ Synthesis ของ program counter

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



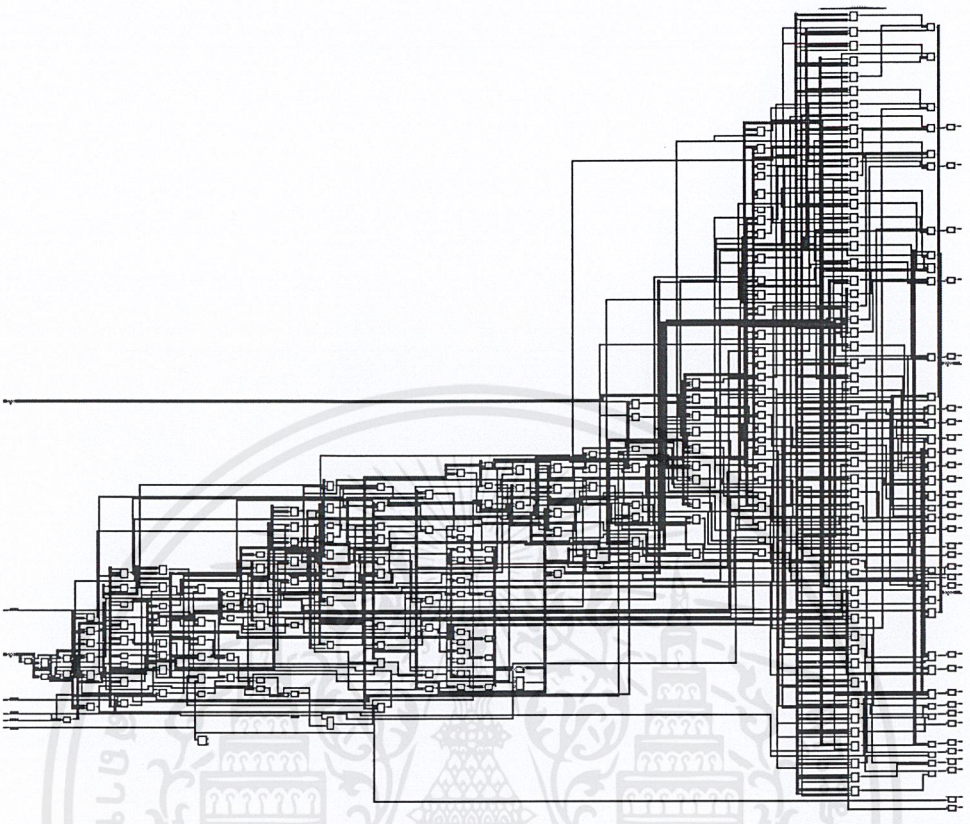
รูปที่ 7.12 แสดงรูปวงจรที่ได้จากการ Synthesis ของ alu

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

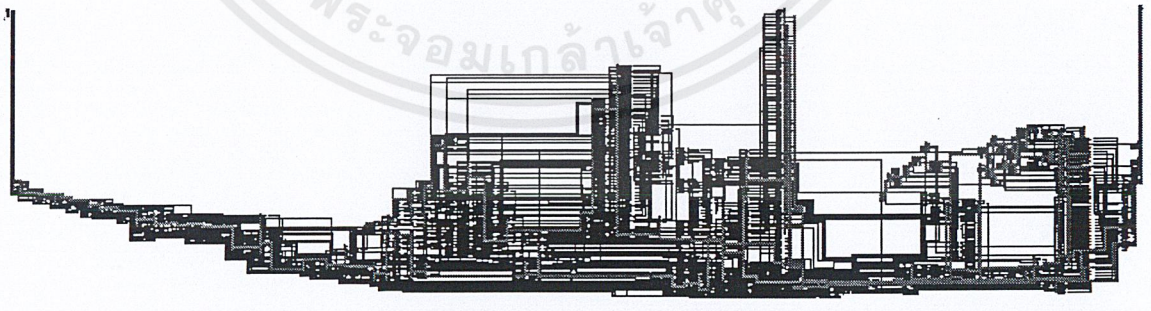


รูปที่ 7.13 แสดงรูปวงจรที่ได้จากการ Synthesis ของ datapath

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7.14 แสดงรูปวงจรที่ได้จากการ Synthesis ของ control unit



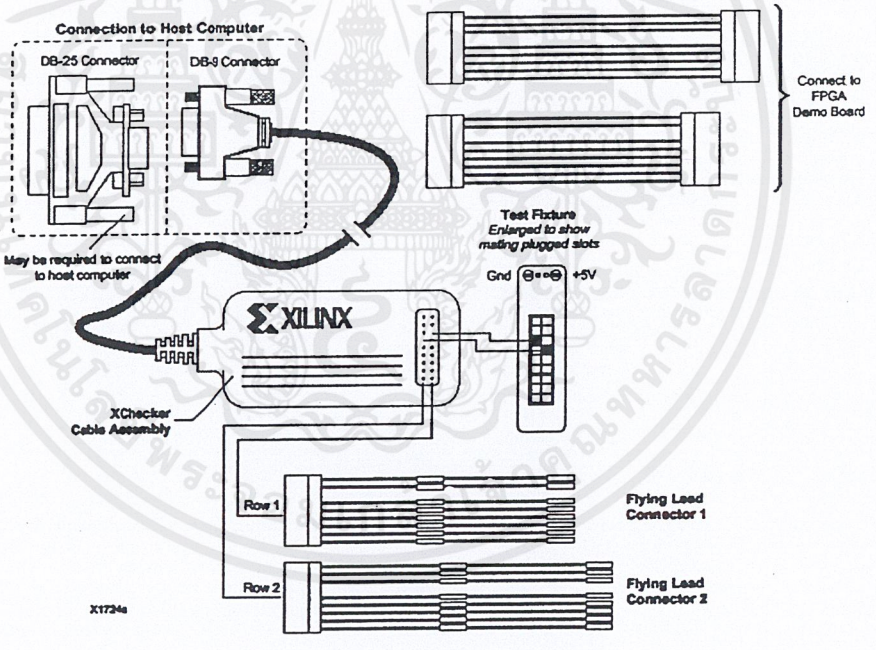
รูปที่ 7.15 แสดงรูปวงจรที่ได้จากการ Synthesis ของซีพียู

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.3 การทำ Place and Route และการบันทึกโปรแกรมลง FPGA

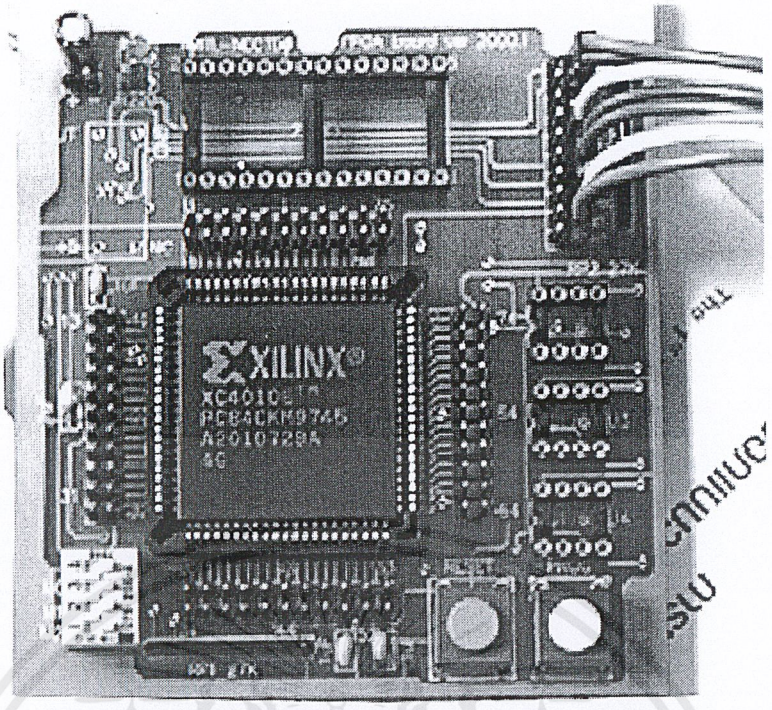
เมื่อทดสอบด้วยการ Simulation จนแน่ใจแล้วว่างานที่ Synthesis ออกมานั้นสามารถทำงานได้อย่างถูกต้องแล้วจึงนำงานที่ได้เข้าสู่ขั้นตอน Place and Route เพื่อทำการ map งานนั้นเข้ากับ FPGA ซึ่งในการทำ Place and Route นี้ก่อนที่จะสร้างไฟล์ .BIT เพื่อที่จะนำไปบันทึกลง FPGA นั้นควรจะสั่งให้โปรแกรม Xilinx สร้างไฟล์ VHDL ออกมาก่อน เพื่อที่จะได้นำไฟล์ .VHD นี้ไปทดสอบการทำงานด้วยว่าวงจรที่ได้จากการ Place and Route นี้สามารถทำงานได้ตรงตามที่ต้องการหรือไม่ เมื่อแน่ใจแล้วก็ทำการสั่งให้โปรแกรม Xilinx สร้างไฟล์นามสกุล .BIT เพื่อที่จะนำไปบันทึกลง FPGA ต่อไป

ในการบันทึกข้อมูลลงบน FPGA นั้นจะใช้สาย Xchecker ในการดาว์นโหลดข้อมูลจากเครื่อง PC ลงสู่ FPGA ซึ่งในการดาว์นโหลดนั้นจะต้องทำในขณะที่มีไฟเลี้ยงป้อนให้ตัว FPGA ด้วย

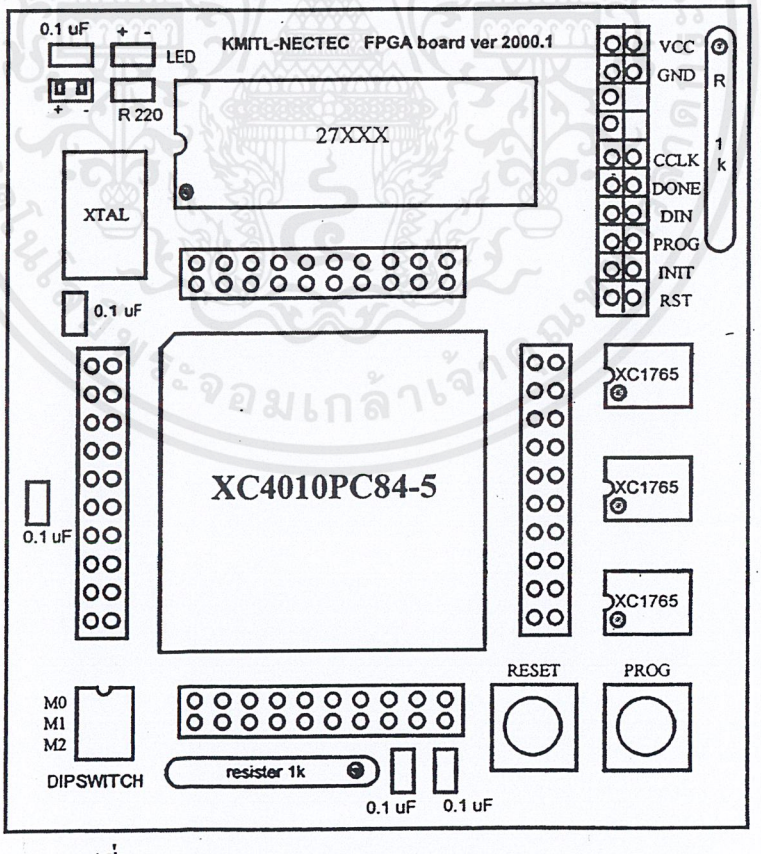


รูปที่ 7.16 แสดงรูปของสาย XChecker

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

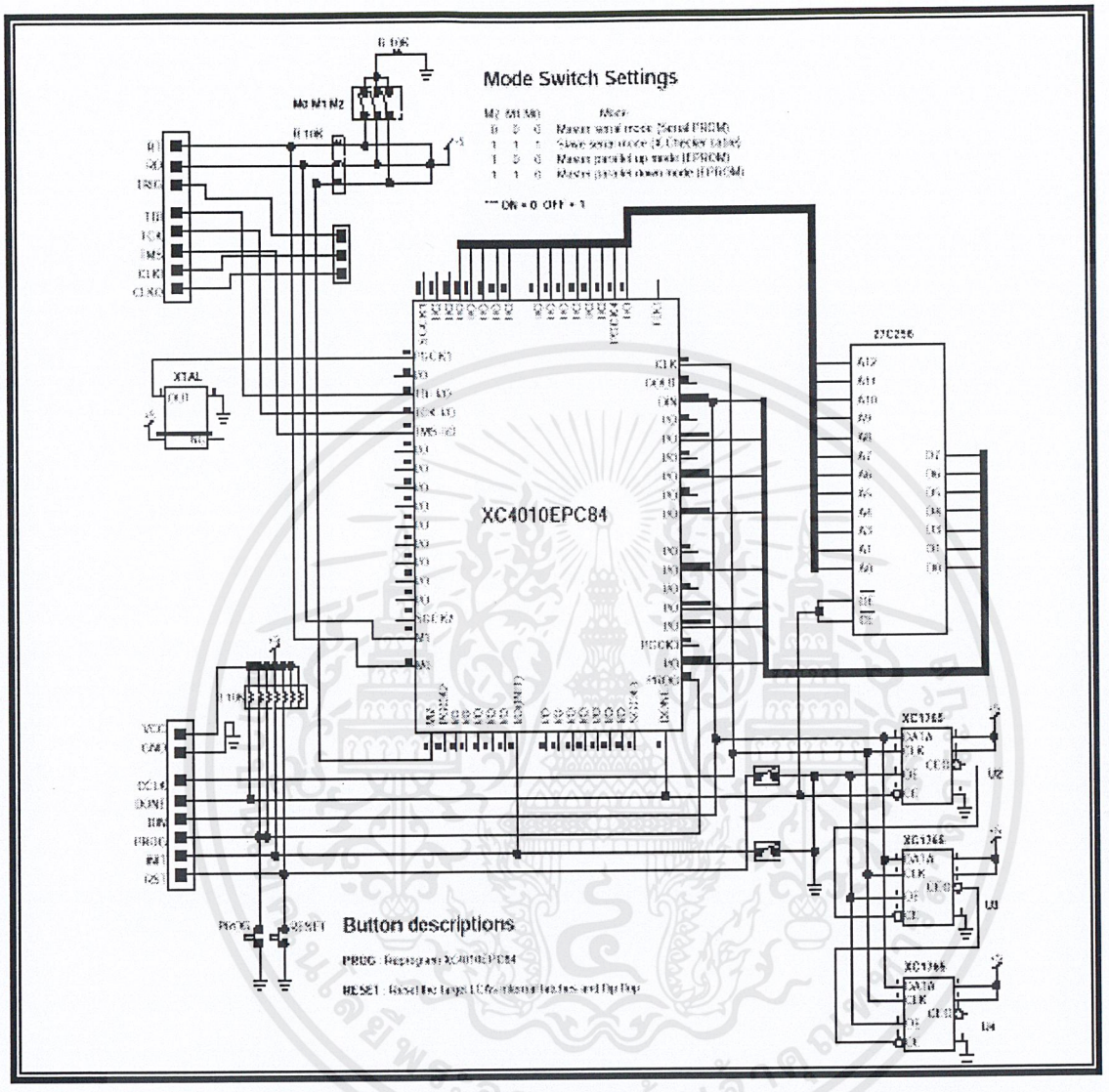


รูปที่ 7.17 แสดงรูป XILINX84 Demo board



รูปที่ 7.18 แสดง Lay out ของ XILINX84 Demo Board

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7.19 Schematic ของ XILINX84 Demo Borad

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.4 การทดสอบการทำงานของ FPGA

การทดสอบการทำงานของ FPGA จริงๆ นั้น สามารถทำได้โดยการเขียนโปรแกรมแล้วนำไปคอมไพล์และนำ hex file ที่ได้บันทึกลง EPROM และนำไปต่อกับวงจรทดสอบต่อไป

แต่ในที่นี้จะทดสอบการทำงานกับอุปกรณ์ภายนอกโดยการเขียน ROM ภายในขึ้นโดยจะทดสอบโดยให้ ซีพียูที่ออกแบบรับค่าจาก key pad โดยทดสอบการบวก การลบและการเลื่อนบิตไปทางซ้ายและทางขวา ซึ่งโปรแกรมทดสอบจะเขียนลงในภาคผนวกต่อไป

เนื่องจากในการทดสอบนั้นต้องการให้เห็นค่า output ที่ออกมา เราจึงใช้วิธีสร้าง clock ขึ้นภายใน FPGA ซึ่งของ Xilinx จะมี Block ที่มีชื่อว่า OSC4 ซึ่งทำหน้าที่ในการสร้างสัญญาณ Oscillator แบบ Square Wave ซึ่งมีความถี่ดังนี้คือ

1. 8 MHz
2. 500 kHz
3. 16 kHz
4. 490 Hz
5. 15 Hz

การใช้งานสามารถทำได้โดยการ Instatnt Component ที่ชื่อว่า OSC4 ใส่เข้าไปใน VHDL Code ที่ออกแบบ แล้วทำการ Port map สัญญาณโดย OSC4 Block มีรายละเอียด ดังนี้คือ

```
component OSC4
        Port ( F8M      : out std_logic; --- 8 MHz
               F500K    : out std_logic; --- 500 kHz
               F16K     : out std_logic; --- 16 kHz
               F490     : out std_logic; --- 490 Hz
               F15      : out std_logic); --- 15 Hz
End component;
```

ถ้าความถี่ของสัญญาณ clock ยังเร็วเกินไปก็สามารถสร้างวงจรหารขึ้นเพื่อลดความถี่ลงได้อีก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 8
สรุปและวิจารณ์

8.1 สรุปภาพรวมทั้งหมดของการออกแบบ

จากคุณสมบัติของซีพียูที่กำหนดคือ ประมวลผลข้อมูลแบบ 8 บิต อ้างอิงหน่วยความจำได้ 64 กิโลไบต์ และสามารถกระทำคำสั่งได้ 35 คำสั่ง จากการทดสอบการทำงานกับฮาร์ดแวร์พบว่าซีพียูที่สร้างขึ้นสามารถทำได้ตามที่ต้องการได้ผลเป็นที่น่าพอใจ

แม้ว่าผลการซิมูเลชั่นจะถูกต้องทุกคำสั่งก็ตาม แต่ก็ไม่สามารถยืนยันได้ว่าคำสั่งทั้ง 35 คำสั่งสามารถทำงานได้อย่างถูกต้อง 100% เพราะไม่สามารถเขียน test bench ให้ครอบคลุมทุกกรณี เนื่องจากเวลาในการทดสอบอันจำกัดและความสามารถของอุปกรณ์มีจำกัดระดับหนึ่ง

อย่างไรก็ตามโครงงานนี้ก็ถือว่าประสบความสำเร็จตามที่กำหนดไว้ คือสามารถสร้างซีพียูในระดับ FPGA ได้และสามารถทำงานได้ในระดับหนึ่ง ถึงแม้ว่าจะเกิดปัญหาต่างๆ แต่ก็ได้แก้ไขเป็นที่เรียบร้อยแล้ว ทำให้ผู้จัดทำได้เรียนรู้วิธีการสร้าง การใช้ Software Tools ต่างๆ และได้ฝึกการทำงานร่วมกับผู้อื่น

8.2 ผลการออกแบบ

1. Timing summary:

Constraints cover 293860 paths, 654 nets, and 4010 connections (100.0% coverage)

Design statistics:

Minimum period : 172.350ns

Maximum frequency : 5.802MHz

Maximum net delay : 43.640ns

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. *Device utilization summary:*

Number of External IOBs	29 out of 61	47%
Flops:	0	
Latches:	0	
Number of Global Buffer IOBs	1 out of 8	12%
Flops:	0	
Latches:	0	
Number of CLBs	323 out of 400	80%
Total CLB Flops:	129 out of 800	16%
4 input LUTs:	521 out of 800	65%
3 input LUTs:	213 out of 400	53%
Number of PRI-CLKs	1 out of 4	25%
Number of TBUFs	72 out of 880	8%
Total equivalent gate count for design:	6832	
Additional JTAG gate count for IOBs:	1488	

ใช้ IOB 29 ตัว จาก 61 ตัว ใช้ CLB ไป 129 ตัวจาก 800 ตัว ใช้เกทไป 6832 ตัว และความถี่สูงสุดคือ 5.802 MHz

8.3 ปัญหาและวิธีแก้ไข

ในการออกแบบวงจรใดๆ ก็ตาม โดยใช้ภาษา VHDL ต้องคำนึงถึงความสามารถของอุปกรณ์ที่จะนำมาทดสอบด้วย ซึ่งการทำโครงงานครั้งนี้ปัญหาที่พบปัญหาแรกคือความเร็วของ clock ที่ออกแบบในตอนแรกยังถือว่าช้าอยู่ ความเร็วของซีพียูที่ได้เป็น 2.8 MHz จึงแก้ไขส่วนของ control unit ขึ้นใหม่ทำให้ได้ความเร็ว clock เพิ่มขึ้น และปัญหาต่อมาคือตัว FPGA มีจำนวนเกทไม่เพียงพอต่อซีพียูที่ออกแบบ ทำให้ต้องเปลี่ยน chip และสุดท้ายการติดต่อกับอุปกรณ์ฮาร์ดแวร์ ซึ่งผู้ทำโครงงานใช้วิธีเขียน ROM ขึ้นภายใน FPGA แทนเนื่องจากสะดวกในการทดสอบการทำงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ก

VHDL Code

```
LIBRARY IEEE;   -- cpu_utilities
USE IEEE.std_logic_1164.ALL;

PACKAGE cpu_utilities IS
 SUBTYPE nibble IS std_logic_vector(3 DOWNTO 0);
 SUBTYPE byte IS std_logic_vector(7 DOWNTO 0);
FUNCTION add_cv(a,b:std_logic_vector;cin:std_logic) RETURN std_logic_vector;
 FUNCTION sub_cv(a,b:std_logic_vector;cin:std_logic) RETURN std_logic_vector;
 FUNCTION set_if_zero(a:std_logic_vector) RETURN std_logic;
 FUNCTION set_cv(ain,b:byte) RETURN byte;
 FUNCTION res_cv(ain,b:byte) RETURN byte;
 FUNCTION inc_byte(byte_in: byte) RETURN byte;
 FUNCTION dec_byte(byte_in: byte) RETURN byte;
END cpu_utilities;

 PACKAGE BODY cpu_utilities IS
  FUNCTION add_cv(a,b:std_logic_vector;cin:std_logic) RETURN std_logic_vector IS
   VARIABLE r,c:std_logic_vector(a'LEFT + 2 DOWNTO 0);
   VARIABLE a_sign,b_sign:std_logic;
  BEGIN
   a_sign:=a(a'LEFT);
   b_sign:=b(b'LEFT);
   r(0):=a(0) XOR b(0) XOR cin;
   c(0):=((a(0) XOR b(0)) AND cin) OR (a(0)AND b(0));
   FOR i IN 1 TO (a'LEFT) LOOP
    r(i):=a(i) XOR b(i) XOR c(i-1);
    c(i):=((a(i) XOR b(i))AND c(i-1)) OR (a(i) AND b(i));
   END LOOP;
   r(a'LEFT+1):=c(a'LEFT);
   IF a_sign=b_sign AND r(a'LEFT) /=a_sign
    THEN r(a'LEFT+2):='1';
   ELSE r(a'LEFT+2):='0'; END IF;
   RETURN r;
  END add_cv;
   FUNCTION sub_cv(a,b:std_logic_vector;cin:std_logic) RETURN std_logic_vector IS
   VARIABLE not_b:std_logic_vector(b'LEFT DOWNTO 0);
   VARIABLE not_c:std_logic;
   VARIABLE r:std_logic_vector(a'LEFT + 2 DOWNTO 0);
  BEGIN
   not_b:= NOT b;
   not_c:= NOT cin;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
 r:=add_cv(a,not_b,not_c);
 RETURN r;
END sub_cv;
FUNCTION set_if_zero(a:std_logic_vector) RETURN std_logic IS
 VARIABLE zero: std_logic;
 BEGIN
     zero :='1';
   FOR i IN a'range LOOP
   IF a(i)='1' THEN
     zero := '0';
   end if;
   end loop;
   RETURN zero;
 END set_if_zero;
 FUNCTION set_cv(ain,b:byte) RETURN byte IS
  VARIABLE a : byte;
  BEGIN
              a := ain;
              if b(0) = '1' then
                a(0) := '1';
              elsif b(1) = '1' then
                a(1) := '1';
              elsif b(2) = '1' then
                a(2) := '1';
              elsif b(3) = '1' then
                a(3) := '1';
              elsif b(4) = '1' then
                a(4) := '1';
              elsif b(5) = '1' then
                a(5) := '1';
              elsif b(6) = '1' then
                a(6) := '1';
              elsif b(7) = '1' then
                a(7) := '1';
              end if;
              RETURN a;
   END set_cv;
   FUNCTION res_cv(ain,b:byte) RETURN byte IS
    VARIABLE a : byte;
    BEGIN
              a := ain;
              if b(0) = '1' then
                a(0) := '0';
              end if;
              if b(1) = '1' then
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            a(1) := '0';
    end if;
            if  b(2) = '1' then
              a(2) := '0';
            end if;
            if b(3) = '1' then
              a(3) := '0';
    end if;
            if b(4) = '1' then
              a(4) := '0';
            end if;
            if b(5) = '1' then
              a(5) := '0';
            end if;
            if b(6) = '1' then
              a(6) := '0';
            end if;
            if b(7) = '1' then
              a(7) := '0';
            end if;
            RETURN a;
END res_cv;
FUNCTION inc_byte(byte_in:byte) RETURN byte IS
 VARIABLE result: byte;
  BEGIN
     result := byte_in;
            if byte_in(0)='0' then
              result(0):='1';
     else
              result(0):='0';
            if byte_in(1)='0' then
       result(1):='1';
     else
              result(1):='0';
            if byte_in(2)='0' then
              result(2):='1';
     else
 result(2):='0';
            if byte_in(3)='0' then
              result(3):='1';
     else
              result(3):='0';
     if byte_in(4)='0' then
       result(4):='1';
     else
              result(4):='0';
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            if byte_in(5)='0' then
              result(5):='1';
    else
            result(5):='0';
            if byte_in(6)='0' then
              result(6):='1';
    else
            result(6):='0';
            if byte_in(7)='0' then
      result(7):='1';
    else  result:="00000000";
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
   RETURN result;
END inc_byte;
FUNCTION dec_byte(byte_in:byte) RETURN byte IS
 VARIABLE result: byte;
 BEGIN
    result := byte_in;
            if byte_in(0)='1' then
              result(0):='0';
    else
            result(0):='0';
    if byte_in(1)='1' then
      result(1):='0';
    else
            result(1):='1';
            if byte_in(2)='1' then
              result(2):='0';
    else
            result(2):='1';
            if byte_in(3)='1' then
              result(3):='0';
     else
            result(3):='1';
    if byte_in(4)='1' then
      result(4):='0';
    else
            result(4):='1';
            if byte_in(5)='1' then
              result(5):='0';
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    else
            result(5):='1';
              if byte_in(6)='1' then
              result(6):='0';
    else
            result(6):='1';
            if byte_in(7)='1' then
      result(7):='0';
    else result:="11111111";
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
            end if;
  RETURN result;
 END dec_byte;
END cpu_utilities;

LIBRARY IEEE;          -- alu_operations
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;

PACKAGE alu_operations IS
 CONSTANT a_add_b :std_logic_vector(4 DOWNTO 0) :="00001";
 CONSTANT a_sub_b :std_logic_vector(4 DOWNTO 0) :="00010";
 CONSTANT incr_a :std_logic_vector(4 DOWNTO 0) :="00011";
 CONSTANT inc :std_logic_vector(4 DOWNTO 0) :="00100";
 CONSTANT decr_a :std_logic_vector(4 DOWNTO 0) :="00101";
 CONSTANT dec :std_logic_vector(4 DOWNTO 0) :="00111";
 CONSTANT a_adc_b :std_logic_vector(4 DOWNTO 0) :="01000";
 CONSTANT a_suc_b :std_logic_vector(4 DOWNTO 0) :="01001";
 CONSTANT a_or_cont :std_logic_vector(4 DOWNTO 0) :="01010";
 CONSTANT a_xor_cont :std_logic_vector(4 DOWNTO 0) :="01011";
 CONSTANT a_not :std_logic_vector(4 DOWNTO 0) :="01100";
 CONSTANT a_and_b :std_logic_vector(4 DOWNTO 0) :="01101";
 CONSTANT set_a : std_logic_vector(4 DOWNTO 0) :="01110";
 CONSTANT res_a : std_logic_vector(4 DOWNTO 0) :="01111";
 CONSTANT bit_a : std_logic_vector(4 DOWNTO 0) :="10000";
 CONSTANT cp_a : std_logic_vector(4 DOWNTO 0) :="10001";
 CONSTANT a_input : std_logic_vector(4 DOWNTO 0) :="10010";
END alu_operations;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
LIBRARY IEEE;  -- shu_operations
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;

PACKAGE shu_operations IS
 CONSTANT rrr :std_logic_vector(3 DOWNTO 0) :="0001";
 CONSTANT rrl :std_logic_vector(3 DOWNTO 0) :="0010";
 CONSTANT rrc :std_logic_vector(3 DOWNTO 0) :="0011";
 CONSTANT rlc :std_logic_vector(3 DOWNTO 0) :="0100";
 CONSTANT srs :std_logic_vector(3 DOWNTO 0) :="0101";
 CONSTANT sls :std_logic_vector(3 DOWNTO 0) :="0110";
 CONSTANT srr :std_logic_vector(3 DOWNTO 0) :="0111";
 CONSTANT slr :std_logic_vector(3 DOWNTO 0) :="1000";
 CONSTANT pass_a :std_logic_vector(3 DOWNTO 0) :="1001";
END shu_operations;

LIBRARY IEEE;  --accumulator
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY accumulator IS
 PORT (i8:IN byte;
     o8:OUT byte;
     load,zero,clk:IN std_logic);
END accumulator;

ARCHITECTURE behavioral OF accumulator IS
BEGIN
 PROCESS(clk,i8,zero,load)
 BEGIN
  IF (clk'EVENT AND clk='0') THEN
   IF load='1' THEN
              o8<=i8;
   END IF;
   IF zero='1' THEN
            o8<="00000000";
   END IF;
  END IF;
 END PROCESS;
END behavioral;

LIBRARY IEEE;  -- general register
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
ENTITY b_register IS
 PORT (i8:IN byte;
     o8:OUT byte;
     load,zero,clk:IN std_logic);
 END b_register;

ARCHITECTURE syn_behavioral OF b_register IS
BEGIN
 PROCESS(clk,i8,zero,load)
 BEGIN
 IF (clk'EVENT AND clk='0') THEN
              IF load='1' THEN
                o8<=i8;
              END IF;
              IF zero='1' THEN
                o8<="00000000";
      END IF;
 END IF;
 END PROCESS;
END syn_behavioral;

LIBRARY IEEE;            -- instruction register
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY instruction_register IS
 PORT (i8:IN byte;
     o8:OUT byte;
     load,clk:IN std_logic);
END instruction_register;

ARCHITECTURE behavioral OF instruction_register IS
BEGIN
 PROCESS(clk,load,i8)
 BEGIN
 IF (clk'EVENT AND clk='0') THEN
              IF load='1' THEN
        o8<=i8;
              END IF;
               END IF;
 END PROCESS;
END behavioral;

LIBRARY IEEE;    -- memory address register
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
ENTITY memory_address_register IS
 PORT (i8:IN byte;
      o16:OUT std_logic_vector(15 downto 0);
      load_page,load_offset,reset_pc,clk:IN std_logic);

END memory_address_register;

ARCHITECTURE syn_behavioral OF memory_address_register IS
BEGIN
mar_adding:PROCESS(clk,load_page,load_offset,i8,reset_pc)
 BEGIN
  IF (clk'EVENT AND clk='0') THEN
    IF (load_offset='1')and(load_page='0') THEN
                 o16(7 downto 0)<= i8;
               ELSIF (load_page='1')and(load_offset='0')  THEN
                 o16(15 DOWNTO 8)<= i8;
    END IF;
    IF reset_pc='1' THEN  o16<="0000000000000000";
    END IF;
   END IF;
 END PROCESS;
END syn_behavioral;

LIBRARY IEEE;   -- program counter
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY program_counter IS
 PORT (i16:IN std_logic_vector(15 downto 0);
      i162:IN std_logic_vector(15 downto 0);
      o16:BUFFER std_logic_vector(15 downto 0):="0000000000000000";
      increment,load_page_pc,s_pop,reset,interrupt,clk:IN std_logic);
END program_counter;

ARCHITECTURE syn_behavioral OF program_counter IS
BEGIN
 clocking:PROCESS(clk,reset,increment,i16,load_page_pc,s_pop,i162,interrupt)
 BEGIN
  IF (clk'EVENT AND clk='0') THEN
               IF reset='1' THEN
                 o16<="0000000000000000";
               ELSIF  (increment = '1') and (s_pop='1') THEN
                 o16<= i162 + 1;
               ELSIF increment='1' THEN
                 o16<=o16 + 1;
               ELSIF s_pop='1' THEN
        o16<=i162;
               ELSIF interrupt='1' THEN
                 o16<="0000000011100000";
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            ELSE
             IF load_page_pc='1' THEN
              o16 <= i16;
             END IF;
            END IF;
  END IF;
 END PROCESS;
END syn_behavioral;

LIBRARY IEEE;              -- Shifter unit
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;
use work1.shu_operations.all;

ENTITY shifter_unit IS
 PORT (alu_out:IN byte;
     shu_code:IN std_logic_vector(3 DOWNTO 0);
     in_flags:IN nibble;
     obus_side:OUT byte;
     out_flags:OUT nibble);
END shifter_unit;

ARCHITECTURE syn_behavioral OF shifter_unit IS
BEGIN
 coding:PROCESS(alu_out,shu_code,in_flags)
  VARIABLE t:std_logic_vector(7 DOWNTO 0);
  VARIABLE v,c,z,s:std_logic;
 BEGIN
  CASE shu_code IS
  when rrr =>
  t:=in_flags(2) & alu_out(7 DOWNTO 1) ;
  s:=t(7);
  z:=set_if_zero(t);
  c:=alu_out(0);
  v:=in_flags(3);
  when rrl =>
  t:=alu_out(6 DOWNTO 0) & in_flags(2) ;
  s:=t(7);
  z:=set_if_zero(t);
  c:=alu_out(7);
  v:=in_flags(3);
  when rrc =>
  t:=alu_out(0) & alu_out(7 DOWNTO 1) ;
  s:=t(7);
  z:=set_if_zero(t);
  c:=alu_out(0);
  v:=in_flags(3);
  when rlc =>
  t:=alu_out(6 DOWNTO 0) & alu_out(7) ;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
s:=t(7);
z:=set_if_zero(t);
c:=alu_out(7);
v:=in_flags(3);
when srs =>
t:='1' & alu_out(7 DOWNTO 1) ;
s:=t(7);
z:=set_if_zero(t);
c:=alu_out(0);
v:=in_flags(3);
when sls =>
t:=alu_out(6 DOWNTO 0) & '1' ;
s:=t(7);
z:=set_if_zero(t);
c:=alu_out(7);
v:=in_flags(3);
when srr =>
t:='0' & alu_out(7 DOWNTO 1) ;
s:=t(7);
z:=set_if_zero(t);
c:=alu_out(0);
v:=in_flags(3);
when slr =>
t:=alu_out(6 DOWNTO 0) & '0' ;
s:=t(7);
z:=set_if_zero(t);
c:=alu_out(7);
v:=in_flags(3);
when pass_a =>
t:=alu_out;
s:=in_flags(0);
z:=in_flags(1);
c:=in_flags(2);
v:=in_flags(3);
when others =>
END CASE ;
obus_side<=t;
out_flags<=v & c & z & s;
END PROCESS coding;
END syn_behavioral;

LIBRARY IEEE;      -- stack pointer
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY stack_pointer IS
PORT (i16:IN std_logic_vector(15 downto 0);
      o16:OUT std_logic_vector(15 downto 0);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    s_push,s_put,interrupt,reset_pc,clk:IN std_logic);
END stack_pointer;

ARCHITECTURE syn_behavioral OF stack_pointer IS
BEGIN
 clocking:PROCESS(clk,s_push,interrupt,i16,reset_pc,s_put)
 BEGIN
  IF (clk'EVENT AND clk='0') THEN
              IF (s_put='1' or interrupt='1') THEN
               o16<=i16;
              ELSIF s_push='1' THEN
               o16<=i16-1;
              END IF;
    IF   (reset_pc='1') THEN
              o16<="0000000000000000";
  end if;
    END IF;
 END PROCESS;
END syn_behavioral;

LIBRARY IEEE;          -- status register
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY status_register IS
 PORT (in_flags:IN std_logic_vector(3 downto 0);
     out_status:OUT std_logic_vector(3 downto 0);
     load_sr,reset_pc,clk:IN std_logic);
END status_register;

ARCHITECTURE syn_behavioral OF status_register IS
 SIGNAL status:nibble;
BEGIN
 clocking:PROCESS(clk,load_sr,in_flags,reset_pc)
 BEGIN
  IF (clk'EVENT AND clk='0') THEN
              IF load_sr='1' THEN
               status<=in_flags;
              END IF;
              IF reset_pc='1' THEN
                status<="0000";
              end if;
  END IF;
 END PROCESS;
 outputting:PROCESS(status)
 BEGIN
              out_status<=status;
 END PROCESS;
END syn_behavioral;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
LIBRARY IEEE;            -- arithmetic_logic_unit
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY arithmetic_logic_unit IS
 PORT (a_side,b_side:IN byte;
      code:IN std_logic_vector(4 DOWNTO 0);
      in_flags:IN nibble;
      z_out:OUT byte;
      out_flags:OUT nibble);
END arithmetic_logic_unit;

ARCHITECTURE syn_behavioral OF arithmetic_logic_unit IS
BEGIN
 coding:PROCESS(a_side,b_side,code,in_flags)
  VARIABLE t:std_logic_vector(9 DOWNTO 0);
  VARIABLE v,c,z,n:std_logic;
 BEGIN
  CASE code IS
   when incr_a =>
     t(7 DOWNTO 0):= inc_byte(a_side);
                c:= in_flags(2); v:=in_flags(3);
   when inc =>
     t(7 DOWNTO 0):= inc_byte(b_side);
                c:= in_flags(2); v:=in_flags(3);
   when decr_a =>
     t(7 DOWNTO 0):= dec_byte(a_side);
                c:= in_flags(2); v:=in_flags(3);
   when dec =>
     t(7 DOWNTO 0):= dec_byte(b_side);
                c:= in_flags(2); v:=in_flags(3);
   when a_add_b =>
                t:= add_cv(b_side,a_side,'0');
                c:= t(8); v:=t(9);
   when a_sub_b =>
                t:= sub_cv(a_side,b_side,'0');
                c:= t(8); v:=t(9);
   when a_adc_b =>
     t:= add_cv(b_side,a_side,in_flags(2));
                c:= t(8); v:=t(9);
   when a_suc_b =>
                t:= sub_cv(a_side,b_side,in_flags(2));
                c:= t(8); v:=t(9);
   when a_or_cont =>
     t:= "00" & (a_side OR b_side);
                c:= in_flags(2); v:=in_flags(3);
   when a_xor_cont =>
                t:= "00" & (a_side XOR b_side);
                c:= in_flags(2); v:=in_flags(3);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
when a_not =>
        t:="00" & (NOT a_side);
        c:=in_flags(2); v:=in_flags(3);
when a_and_b =>
        t:="00" & (a_side AND b_side);
        c:=in_flags(2); v:=in_flags(3);
when set_a =>
        t(7 downto 0) := set_cv(a_side,b_side);
        c:=in_flags(2); v:=in_flags(3);
when res_A =>
        t(7 downto 0):= res_cv(a_side,b_side);
        c:=in_flags(2); v:=in_flags(3);
when bit_a =>
        if b_side(0) = '1' then
         if a_side(0) = '1' then
         z := '0';
         else z:='1';
         end if;
        elsif b_side(1) = '1' then
         if  a_side(1) = '1'then
         z := '0';
        else z:='1';
         end if;
        elsif b_side(2) = '1' then
         if  a_side(2) = '1'then
         z := '0';else z:='1';
         end if;
        elsif b_side(3) = '1' then
         if  a_side(3) = '1'then
         z := '0';
        else z:='1';
         end if;
        elsif b_side(4) = '1' then
         if  a_side(4) = '1'then
         z := '0';
        else z:='1';
         end if;
        elsif b_side(5) = '1' then
         if  a_side(5) = '1'then
        else z:='1';
         end if;
        elsif b_side(6) = '1' then
         if  a_side(6) = '1'then
         z := '0';
        else z:='1';
         end if;
        elsif b_side(7) = '1' then
         if  a_side(7) = '1'then
         z := '0';
         else z:='1';
         end if;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            end if;
            c:=in_flags(2); v:=in_flags(3);
  when cp_a =>
            if (a_side = b_side)then
        z:='1';
      end if;
            c:=in_flags(2); v:=in_flags(3);
  when a_input =>
            t:= "00" & a_side;
      c:=in_flags(2); v:=in_flags(3);
            WHEN OTHERS =>null;
  END CASE;
  n:=t(7);
  z_out<=t(7 DOWNTO 0);
  out_flags<=v & c & z & n;
 END PROCESS coding;
END syn_behavioral;

LIBRARY  IEEE;          -- datapath
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY data_path IS
 PORT(databus:INOUT byte:="ZZZZZZZZ"; adbus:OUT std_logic_vector(15 DOWNTO 0);
    clk:IN std_logic;
            load_ac,load_ir,load_br,load_cr,
      load_dr,load_er,load_fr,load_gr,load_hr,
            inc_pc,reset_pc,s_push,s_pop,s_put,s_interrupt,
            load_page_mar,load_offset_mar,load_mar_on_pc,
            load_sr,
            brbus_on_dbus,crbus_on_dbus,drbus_on_dbus,erbus_on_dbus,
      frbus_on_dbus,grbus_on_dbus,hrbus_on_dbus,pc_on_adbus,
            shu_on_dbus,
            databus_on_dbus,dbus_on_databus:IN std_logic;
      shu_code:IN std_logic_vector(3 DOWNTO 0);
            alu_code:IN std_logic_vector(4 DOWNTO 0);
            ir_lines:OUT byte; status:OUT nibble);
 END data_path;

ARCHITECTURE structural_syn OF data_path IS

COMPONENT accumulator
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT instruction_register
 PORT (i8:IN byte;o8:OUT byte;load,clk:IN std_logic);
END COMPONENT;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
COMPONENT b_register
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT c_register
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT d_register
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT e_register
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT f_register
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT g_register
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT h_register
 PORT (i8:IN byte;o8:OUT byte;load,zero,clk:IN std_logic);
END COMPONENT;

COMPONENT memory_address_register
 PORT (i8:IN byte; o16:OUT std_logic_vector(15 DOWNTO 0);
     load_page,load_offset,reset_pc,clk:IN std_logic);
END COMPONENT;

COMPONENT program_counter
 PORT (i16:IN std_logic_vector(15 DOWNTO 0);
     i162:IN std_logic_vector(15 DOWNTO 0);
     o16:BUFFER std_logic_vector(15 DOWNTO 0);
     increment,load_page_pc,s_pop,reset,interrupt,clk:IN std_logic);
END COMPONENT;

COMPONENT stack_pointer
 PORT (i16:IN std_logic_vector(15 downto 0);
     o16:OUT std_logic_vector(15 downto 0);
     s_push,s_put,interrupt,reset_pc,clk:IN std_logic);
END COMPONENT;

COMPONENT status_register
  PORT (in_flags:IN nibble; out_status:OUT nibble;
            load_sr,reset_pc,clk:IN std_logic);
END COMPONENT;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
COMPONENT arithmetic_logic_unit
 PORT (a_side,b_side:IN byte; code:IN std_logic_vector(4 DOWNTO 0);
     in_flags:IN nibble;
          z_out:OUT byte; out_flags:OUT nibble);
END COMPONENT;

COMPONENT shifter_unit
 PORT (alu_out:IN byte; shu_code:IN std_logic_vector(3 DOWNTO 0);
          in_flags:IN nibble; obus_side:OUT byte; out_flags:OUT nibble);
END COMPONENT;

 SIGNAL ac_out,ir_out,alu_out,shu_out:byte;
 SIGNAL pc_out,mar_out,sp_out:std_logic_vector(15 DOWNTO 0);
 SIGNAL dbus,brbus,crbus,drbus,erbus,frbus,grbus,hrbus:byte;
 SIGNAL shu_flags,alu_flags,sr_out:nibble;

BEGIN
 databus<=dbus WHEN dbus_on_databus='1' ELSE "ZZZZZZZZ";
 dbus<=databus WHEN databus_on_dbus='1' ELSE "ZZZZZZZZ";
 dbus<=shu_out WHEN shu_on_dbus='1' ELSE "ZZZZZZZZ";
 r1:accumulator PORT MAP(dbus,ac_out,load_ac,reset_pc,clk);
 r2:instruction_register PORT MAP(dbus,ir_out,load_ir,clk);
   ir_lines<=ir_out;
 r3:b_register PORT MAP(dbus,brbus,load_br,reset_pc,clk);
  dbus<=brbus when brbus_on_dbus='1' else "ZZZZZZZZ";
 r4:c_register PORT MAP(dbus,crbus,load_cr,reset_pc,clk);
   dbus<=crbus when crbus_on_dbus='1' else "ZZZZZZZZ";
 r5:d_register PORT MAP(dbus,drbus,load_dr,reset_pc,clk);
   dbus<=drbus when drbus_on_dbus='1' else "ZZZZZZZZ";
 r6:e_register PORT MAP(dbus,erbus,load_er,reset_pc,clk);
   dbus<=erbus when erbus_on_dbus='1' else "ZZZZZZZZ";
 r7:f_register PORT MAP(dbus,frbus,load_fr,reset_pc,clk);
   dbus<=frbus when frbus_on_dbus='1' else "ZZZZZZZZ";
 r8:g_register PORT MAP(dbus,grbus,load_gr,reset_pc,clk);
   dbus<=grbus when grbus_on_dbus='1' else "ZZZZZZZZ";
 r9:h_register PORT MAP(dbus,hrbus,load_hr,reset_pc,clk);
   dbus<=hrbus when hrbus_on_dbus='1' else "ZZZZZZZZ";
 r10:memory_address_register
   PORT MAP(dbus,mar_out,load_page_mar,load_offset_mar,reset_pc,clk);
 r11:program_counter
   PORT MAP(mar_out,sp_out,pc_out,inc_pc,load_mar_on_pc,s_pop,reset_pc,s_interrupt,clk);
   adbus<=pc_out WHEN pc_on_adbus='1' ELSE "ZZZZZZZZZZZZZZZZ";
 r12:stack_pointer
    PORT MAP(pc_out,sp_out,s_push,s_put,s_interrupt,reset_pc,clk);
 r13:status_register PORT MAP(shu_flags,sr_out,load_sr,reset_pc,clk);
    status<=sr_out;
 r14:arithmetic_logic_unit PORT MAP(ac_out,dbus,alu_code,sr_out,alu_out,alu_flags);
 r15:shifter_unit
   PORT MAP(alu_out,shu_code,alu_flags,shu_out,shu_flags);
END structural_syn;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
library ieee;  -- control unit
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work1;
use work1.cpu_utilities.ALL;
use work1.alu_operations.ALL;
use work1.shu_operations.ALL;

entity CONTROL_UNIT is
 PORT ( reset,interrupt,ck : in std_logic ;
              status :in std_logic_vector(3 downto 0);
              ir_out :in std_logic_vector(7 downto 0);
              reset_pc,load_ir,load_sr,load_ac,load_br,load_cr,load_dr,load_er,
              load_fr,load_gr,load_hr,brbus_on_dbus,crbus_on_dbus,drbus_on_dbus,dbus_on_dataout,
              erbus_on_dbus,frbus_on_dbus,grbus_on_dbus,hrbus_on_dbus,s_push,s_pop,s_put,
              pc_on_adbus,databus_on_dbus,s_interrupt,read,datain_on_dbus,
              inc_pc,load_offset_mar,load_page_mar,load_mar_on_pc,shu_on_dbus :out std_logic;
              alu_code : out std_logic_vector(4 downto 0);
              shift_code : out std_logic_vector(3 downto 0) );
 end CONTROL_UNIT;

 architecture  DATAFLOW_SYN of CONTROL_UNIT is
  TYPE cpu_states1 IS (fetch_state,initial1);
  TYPE cpu_states2 IS (initial2,interrupt_state,do_one_byte,do_two_byte,do_others);
  SIGNAL present_state1,next_state1 : cpu_states1;
  SIGNAL  present_state2,next_state2 : cpu_states2;
  begin
  clocking:PROCESS (ck,next_state1,next_state2,reset)
    BEGIN
                if (reset = '1' ) then
                 present_state1 <= fetch_state;
                 present_state2 <= initial2;
                elsif ck 'EVENT and ck = '0' then
                 present_state1 <= next_state1;
                 present_state2 <= next_state2;
               end if;
  END PROCESS clocking;
  sequencing : PROCESS (present_state1,present_state2,ir_out,status,reset,interrupt)
    BEGIN
            reset_pc <= '0';load_ir <= '0';load_ac <= '0';load_br <= '0';dbus_on_dataout<='0';
            load_cr <= '0';load_dr <= '0';load_er <= '0';load_fr <= '0';load_gr <= '0';
            load_sr <= '0';load_hr <= '0';brbus_on_dbus <= '0';crbus_on_dbus <= '0';
            drbus_on_dbus <= '0';erbus_on_dbus <= '0';frbus_on_dbus <= '0';
            grbus_on_dbus <= '0';hrbus_on_dbus <= '0';s_push<='0';s_pop <= '0';
            s_put<='0';pc_on_adbus<= '0';databus_on_dbus <= '0';datain_on_dbus<='0';
            s_interrupt <= '0';read <= '0';
            inc_pc <= '0';load_mar_on_pc<='0';load_offset_mar <= '0';
            load_page_mar <= '0';
            alu_code <= "00000";
            shift_code <= "0000";
                    if reset = '1' then                                    --reset
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                reset_pc <= '1';
                next_state2 <= initial2;
                next_state1 <= fetch_state;
                end if;
CASE present_state1 IS
        WHEN  fetch_state =>                    --fetch
                read <='1' ;
        pc_on_adbus <='1' ;
          dbus_on_dataout <='0';
                datain_on_dbus <= '0';
                shu_on_dbus <= '0';
                databus_on_dbus <= '1';
                load_ir <= '1' ;
                inc_pc <= '1' ;
                next_state1 <= initial1;
                next_state2 <= do_one_byte;

        WHEN initial1 =>
        END CASE;
CASE present_state2 IS
        WHEN  initial2 =>                                   -- initial2
          if  (interrupt = '1') then
          next_state2 <= interrupt_state;
          end if;
        WHEN  interrupt_state =>               --interrupt
          s_interrupt <= '1';
          s_put <= '1';
          next_state1 <= fetch_state ;
          next_state2 <= initial2 ;
        WHEN  do_one_byte =>                   --do_one_byte
          CASE  ir_out (7 downto 6 ) IS
                WHEN  "00" =>
                        CASE  ir_out (5 downto 4) IS
                        when "00" =>                --load
                        read <='1' ;
                pc_on_adbus <='1' ;
                        dbus_on_dataout <='0';
                        datain_on_dbus <= '0';
                        shu_on_dbus <= '0';
                    databus_on_dbus <= '1';
                                case ir_out(3 downto 1) IS
                         when  "000" =>load_ac <= '1';
                         when  "001" =>load_br <= '1';
                         when  "010" =>load_cr <= '1';
                         when  "011" =>load_dr <= '1';
                         when  "100" =>load_er <= '1';
                         when  "101" =>load_fr <= '1';
                         when  "110" =>load_gr <= '1';
                         when  others =>load_hr <= '1';
                         end case;
                         inc_pc <= '1';
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                when "01" =>
                 case ir_out (3 downto 1) is
                  when "001" =>brbus_on_dbus <= '1';
                  when "010" =>crbus_on_dbus <= '1';
                  when "011" =>drbus_on_dbus <= '1';
                  when "100" =>erbus_on_dbus <= '1';
                  when "101" =>frbus_on_dbus <= '1';
                  when "110" =>grbus_on_dbus <= '1';
                  when "111" =>hrbus_on_dbus <= '1';
                  when others => null;
                  end case;
                  load_ac <= '1';
                when  "10" =>                          --push
                  s_push <= '1';
                when  others =>                        --pop
                  s_pop <= '1';
                end CASE;
                next_state1 <= fetch_state;
                next_state2 <= initial2;
WHEN "01" =>                                           --alu
     if ir_out (5 downto 4 ) = "11" then
                        if ir_out( 3 downto 1)= "010" then
                        alu_code <= a_not;
                        else
                        read <='1' ;
 pc_on_adbus <='1' ;
 dbus_on_dataout <='0';
                          datain_on_dbus <= '0';
                          shu_on_dbus <= '0';
                          databus_on_dbus <= '1';
                          inc_pc <= '1';
                                  case ir_out (3 downto 1) is
   when  "000" =>alu_code <= a_or_cont;
                when "001" =>alu_code <= a_xor_cont;
                when "010" =>alu_code <= a_not;
                when "011" =>alu_code <= a_and_b;
                when "100" =>alu_code <= set_a;
                when "101" =>alu_code <= res_a;
                when "110" =>alu_code <= bit_a;
                when others =>alu_code <= cp_a;
                end case;
               end if;
     else
                case ir_out (2 downto 0) is
                  when "001" =>brbus_on_dbus <= '1';
                  when "010" =>crbus_on_dbus <= '1';
                  when "011" =>drbus_on_dbus <= '1';
                  when "100" =>erbus_on_dbus <= '1';
                  when "101" =>frbus_on_dbus <= '1';
                  when "110" =>grbus_on_dbus <= '1';
                  when "111" =>hrbus_on_dbus <= '1';
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            when others =>null;
            end case;
            case ir_out(5 downto 3) IS
            when "000" => alu_code <= a_add_b;
when "001" => alu_code <= a_sub_b;
when "010" =>
              if (ir_out (2 downto 0)= "000" ) then
  alu_code <= decr_a;
              else alu_code <= dec;
              end if ;
when "011" =>
              if (ir_out (2 downto 0)= "000" ) then
   alu_code <= incr_a;
              else alu_code <= inc;
              end if ;
when "100"=> alu_code <= a_adc_b;
when "101"=> alu_code <= a_suc_b;
when others =>NULL;
end case;
          end if;
        next_state2 <= do_two_byte;
                    next_state1 <= initial1;
WHEN  "10" =>                       --shift
  alu_code <= a_input;
  next_state2 <= do_two_byte;
  next_state1 <= initial1;
WHEN  others =>
 if ir_out = "11111111" then
            inc_pc<='1';
            next_state2<=initial2;
            next_state1<=fetch_state;
 else
     if ir_out(5) = '0' then        --jump
                    read <='1' ;
pc_on_adbus <='1' ;
                     shu_on_dbus <='0';
            databus_on_dbus <= '1';
                      load_page_mar <= '0';
                load_offset_mar <= '1';
                      inc_pc <= '0';
                      next_state1 <= initial1;
                      next_state2 <= do_two_byte;
    else
          if ir_out (4 downto 3) = "01" then
                      case ir_out(2)is
                      when '0'=>                        --in
                      shu_on_dbus <= '0';
                      databus_on_dbus<='0';
                      dbus_on_dataout <='0';
                      datain_on_dbus <= '1';
                      load_ac <= '1';
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                next_state2 <= initial2;
                next_state1 <= fetch_state;
                when '1' =>                                --out
                databus_on_dbus <= '0';
                datain_on_dbus <= '0';
                shu_on_dbus <= '1';
                dbus_on_dataout <= '1';
                alu_code <= a_input;
                shift_code<= pass_a;
                next_state2 <= initial2;
                next_state1 <= fetch_state;
                when others => null;
                end case;
        elsif ir_out(2) = '0' then          --call
            read <='1' ;
      pc_on_adbus <='1' ;
            dbus_on_dataout <='0';
                                    datain_on_dbus <= '0';
                                    shu_on_dbus <= '0';
            databus_on_dbus <= '1';
                load_offset_mar <='1';
            inc_pc <= '1';
            next_state2 <= do_two_byte;
                                    next_state1<= initial1;
        elsif ir_out(2) = '1' then     --retrun
            s_pop <= '1';
            inc_pc <= '1';
                next_state1 <= fetch_state;
                next_state2 <= initial2;
            end if;
        end if;
        end if;
    END CASE;
WHEN do_two_byte =>                    --do_two_byte
    CASE ir_out(7 downto 6) IS
        WHEN "01" =>                                --alu
            shift_code <= pass_a;
            databus_on_dbus <= '0';
            shu_on_dbus <= '1';
            next_state2 <= initial2;
            next_state1 <= fetch_state;
                    if (ir_out (5 downto 3)= "010")or (ir_out (5 downto 3)= "011" )then
      case ir_out (2 downto 0) is
                        when "000" =>
            load_ac <='1';
            load_sr <= '1';
            when "001" =>
            load_br <='1';
            load_sr <= '1';
                        when "010" =>
            load_cr <='1';
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                load_sr <= '1';
                        when "011" =>
                load_dr <='1';
                load_sr <= '1';
                        when "100" =>
                load_er <='1';
                load_sr <= '1';
                        when "101" =>
                load_fr <='1';
                load_sr <= '1';
                        when "110" =>
                load_gr <='1';
                load_sr <= '1';
                        when others =>
                load_hr <='1';
                load_sr <= '1';
        end case ;
    else
    load_ac <= '1';
    load_sr <= '1';
end if;
WHEN "10" =>                        --shift
    case ir_out (4 downto 2) is
            when  "000" => shift_code <= rrr;
            when  "001" => shift_code <= rrl;
            when  "010" => shift_code <= rrc;
            when  "011" => shift_code <= rlc;
            when  "100" => shift_code <= srs;
            when  "101" => shift_code <= sls;
            when  "110" => shift_code <= srr;
            when  others => shift_code <= slr;
  end case;
     next_state2 <= do_others;
WHEN others =>
      if ir_out(5) = '0' then       --jump
                    read <='1' ;
          pc_on_adbus <='1' ;

                        dbus_on_dataout <='0';
                        datain_on_dbus <= '0';
                        shu_on_dbus <= '0';
          databus_on_dbus <= '1';
                        load_offset_mar <= '0';
              load_page_mar <= '1';
                        inc_pc <= '0';
            next_state2 <= do_others;
                    next_state1 <= initial1;
           else
        if ir_out (4 downto 2) = "000" then             --call
            read <='1' ;
    pc_on_adbus <='1' ;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                                dbus_on_dataout <='0';
                                datain_on_dbus <= '0';
                                shu_on_dbus <= '0';
                        databus_on_dbus <= '1';
                                load_offset_mar <= '0';
                        load_page_mar <='1';
                                s_put <= '1';
                        next_state2 <= do_others;
                                next_state1 <= initial1;
                            end if;
                    end if;
    END CASE;
WHEN do_others =>                       --do_others
    case ir_out(7 downto 6) IS
            when "10" =>            -- shift
                    databus_on_dbus <= '0';
                    shu_on_dbus <= '1';
                load_ac <='1';
                    load_sr<= '1';
                next_state2 <= initial2;
                next_state1 <= fetch_state;
    when "11" =>
                    if ir_out(5) <='0' then             -- jump
                        case ir_out (4 downto 2) IS
                        when "000" =>
                load_mar_on_pc <='1';
                    when "001" =>
                        if status(1) = '1' then
                    load_mar_on_pc <='1';
                        end if;
                    when "010" =>
                        if status(1) = '0' then
                    load_mar_on_pc <='1';
                        end if;
                        when "011" =>
                        if status(2) = '1' then
                     load_mar_on_pc <='1';
                        end if;
                    when "100" =>
                        if status(2) = '0' then
                load_mar_on_pc <='1';
                        end if;
                        when others=>NULL;
                    end case;
             next_state2 <= initial2;
             next_state1 <= fetch_state;
            elsif ir_out(3 downto 2)="00" then              --call
                     load_mar_on_pc <='1';
                next_state2 <= initial2;
              next_state1 <= fetch_state;
            end if;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                                    when others => NULL;
        end case;
      END CASE;
   end process sequencing;
 END dataflow_syn;

LIBRARY IEEE;                       -- central_processing_unit
USE IEEE.std_logic_1164.ALL;
LIBRARY work1;
USE work1.cpu_utilities.ALL;
USE work1.alu_operations.ALL;

ENTITY central_processing_unit IS
PORT   ( clk : IN std_logic;
               interrupt: IN std_logic;
               rd,wr : OUT std_logic;
               databus : INOUT std_logic_vector(7 downto 0);
               adbus : OUT std_logic_vector(15 downto 0);
               reset : IN std_logic );
END central_processing_unit ;

ARCHITECTURE dataflow OF central_processing_unit IS
COMPONENT data_path
PORT (  databus:INOUT byte; adbus:OUT std_logic_vector(15 DOWNTO 0);
     clk:IN std_logic;
               load_ac,load_ir,load_br,load_cr,
     load_dr,load_er,load_fr,load_gr,load_hr,
               inc_pc,reset_pc,s_push,s_pop,s_put,s_interrupt,
               load_page_mar,load_offset_mar,load_mar_on_pc,load_sr,
               brbus_on_dbus,crbus_on_dbus,drbus_on_dbus,erbus_on_dbus,
     frbus_on_dbus,grbus_on_dbus,hrbus_on_dbus,pc_on_adbus,
               shu_on_dbus,
               databus_on_dbus,dbus_on_databus:IN std_logic;
     shu_code:IN std_logic_vector(3 DOWNTO 0);
               alu_code:IN std_logic_vector(4 DOWNTO 0);
               ir_lines:OUT byte; status:OUT nibble);
 END COMPONENT;
 COMPONENT control_unit
 PORT(  reset,interrupt,ck : in std_logic;
               status :in std_logic_vector(3 downto 0);
               ir_out :in std_logic_vector(7 downto 0);
               reset_pc,load_ir,load_sr,load_ac,load_br,load_cr,load_dr,load_er,
               load_fr,load_gr,load_hr,brbus_on_dbus,crbus_on_dbus,drbus_on_dbus,
               erbus_on_dbus,frbus_on_dbus,grbus_on_dbus,hrbus_on_dbus,s_push,s_pop,s_put,
               pc_on_adbus,databus_on_dbus,dbus_on_databus,s_interrupt,read,
               write,inc_pc,load_offset_mar,load_page_mar,load_mar_on_pc,shu_on_dbus :out std_logic;
               alu_code : out std_logic_vector(4 downto 0);
               shift_code : out std_logic_vector(3 downto 0) );
 END COMPONENT;
  SIGNAL  load_ac,load_ir,load_br,load_cr,
     load_dr,load_er,load_fr,load_gr,load_hr,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            inc_pc,reset_pc,s_push,s_pop,s_put,s_interrupt,
            load_page_mar,load_offset_mar,load_mar_on_pc,
            load_sr,
            brbus_on_dbus,crbus_on_dbus,drbus_on_dbus,erbus_on_dbus,
    frbus_on_dbus,grbus_on_dbus,hrbus_on_dbus,pc_on_adbus,
            shu_on_dbus,
            databus_on_dbus,dbus_on_databus: std_logic;
 SIGNAL  alu_code : std_logic_vector(4 DOWNTO 0);
 SIGNAL  shu_code : std_logic_vector(3 DOWNTO 0);
 SIGNAL  ir_out : byte;
 SIGNAL  status  : nibble;
BEGIN
data : data_path PORT MAP
            ( databus, adbus,clk,
            load_ac,load_ir,load_br,load_cr,
    load_dr,load_er,load_fr,load_gr,load_hr,
            inc_pc,reset_pc,s_push,s_pop,s_put,s_interrupt,
            load_page_mar,load_offset_mar,load_mar_on_pc,load_sr,
            brbus_on_dbus,crbus_on_dbus,drbus_on_dbus,erbus_on_dbus,
    frbus_on_dbus,grbus_on_dbus,hrbus_on_dbus,pc_on_adbus,
    shu_on_dbus,
            databus_on_dbus,dbus_on_databus,
    shu_code,alu_code,ir_out,status);
ctrl : control_unit PORT MAP
   ( reset,interrupt,clk,
            status,      ir_out,
            reset_pc,load_ir,load_sr,load_ac,load_br,load_cr,load_dr,load_er,
            load_fr,load_gr,load_hr,brbus_on_dbus,crbus_on_dbus,drbus_on_dbus,
            erbus_on_dbus,frbus_on_dbus,grbus_on_dbus,hrbus_on_dbus,s_push,s_pop,s_put,
            pc_on_adbus,databus_on_dbus,dbus_on_databus,s_interrupt,rd,
            wr,inc_pc,load_offset_mar,load_page_mar,load_mar_on_pc,shu_on_dbus,
    alu_code,shu_code );
 END dataflow;

library IEEE;                          -- Memory
Use IEEE.std_logic_1164.all;

Entity InstMem is
port (rd      : in std_logic;
    addr      : in std_logic_vector(15 downto 0);
    data      : out std_logic_vector(7 downto 0));
End InstMem;

Architecture behavior of InstMem is
            type memory_array is array(0 to 404) of std_logic_vector(7 downto 0);
            signal mem_adbus          : memory_array;
            signal address_n          : integer;
            Function to_integer(a : in std_logic_vector(15 downto 0)) return integer is
                        variable sum : integer := 0;
                        variable tmp  : integer;
                        variable tmp2              : integer;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        begin
                for i in 15 downto 0 loop
                        tmp := 2**i;
                        if (a(i) = '0') then tmp2 := 0;
                        elsif (a(i) = '1') then tmp2 := 1; end if;
                        sum := sum + (tmp * tmp2);
                end loop;
                if sum < 0 then
                        sum:= 0 ;
                end if;
                return sum;
        end to_integer;
begin
        process(addr)-- convert addr(binary) to address_n(integer)
        begin
                address_n <= to_integer(addr);
        end process;
process(address_n)
begin
mem_adbus(0)<="11101000";--(a1),in
mem_adbus(1)<="01111100";--bit
mem_adbus(2)<="00010000";--10h
mem_adbus(3)<="11000100";--jz
mem_adbus(4)<="00000000";--a1
mem_adbus(5)<="00000000";--a1
mem_adbus(6)<="11101000";--in
mem_adbus(7)<="01111100";--bit
mem_adbus(8)<="00010000";--10h
mem_adbus(9)<="11010100";--jnz
mem_adbus(10)<="00000000";--a1
mem_adbus(11)<="00000000";--a1
mem_adbus(12)<="11111111";--in
mem_adbus(13)<="01111110";--cp
mem_adbus(14)<="00000000";--00h
mem_adbus(15)<="11010100";--jnz
mem_adbus(16)<="00011010";--b1
mem_adbus(17)<="00000000";--b1
mem_adbus(18)<="00000010";--ld b
mem_adbus(19)<="00000000";--00h
mem_adbus(20)<="00000000";--ld a
mem_adbus(21)<="00000000";--00h
mem_adbus(22)<="11101100";--out
mem_adbus(23)<="11000000";--jp
mem_adbus(24)<="11000101";--a2
mem_adbus(25)<="00000000";--a2
mem_adbus(26)<="11101000";--(b1),in
mem_adbus(27)<="01111100";--cp
mem_adbus(28)<="00000001";--01h
mem_adbus(29)<="11001000";--jnz
mem_adbus(30)<="00101000";--b2
mem_adbus(31)<="00000000";--b2
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(32)<="00000010";--ld b
mem_adbus(33)<="00000001";--01h
mem_adbus(34)<="00000000";--ld a
mem_adbus(35)<="00000001";--01h
mem_adbus(36)<="11101100";--out
mem_adbus(37)<="11000000";--jp
mem_adbus(38)<="11000101";--a2
mem_adbus(39)<="00000000";--a2
mem_adbus(40)<="11101000";--(b2),in
mem_adbus(41)<="01111100";--cp
mem_adbus(42)<="00000010";--02h
mem_adbus(43)<="11001000";--jnz
mem_adbus(44)<="00110110";--b3
mem_adbus(45)<="00000000";--b3
mem_adbus(46)<="00000010";--ld b
mem_adbus(47)<="00000010";--02h
mem_adbus(48)<="00000000";--ld a
mem_adbus(49)<="00000010";--02h
mem_adbus(50)<="11101100";--out
mem_adbus(51)<="11000000";--jp
mem_adbus(52)<="11000101";--a2
mem_adbus(53)<="00000000";--a2
mem_adbus(54)<="11101000";--(b3),in
mem_adbus(55)<="01111100";--cp
mem_adbus(56)<="00000011";--03h
mem_adbus(57)<="11001000";--jnz
mem_adbus(58)<="01000100";--b4
mem_adbus(59)<="00000000";--b4
mem_adbus(60)<="00000010";--ld b
mem_adbus(61)<="00000011";--03h
mem_adbus(62)<="00000000";--ld a
mem_adbus(63)<="00000011";--03h
mem_adbus(64)<="11101100";--out
mem_adbus(65)<="11000000";--jp
mem_adbus(66)<="11000101";--a2
mem_adbus(67)<="00000000";--a2
mem_adbus(68)<="11101000";--(b4),in
mem_adbus(69)<="01111100";--cp
mem_adbus(70)<="00000100";--04h
mem_adbus(71)<="11001000";--jnz
mem_adbus(72)<="01010010";--b5
mem_adbus(73)<="00000000";--b5
mem_adbus(74)<="00000010";--ld b
mem_adbus(75)<="00000100";--04h
mem_adbus(76)<="00000000";--ld a
mem_adbus(77)<="00000100";--04h
mem_adbus(78)<="11101100";--out
mem_adbus(79)<="11000000";--jp
mem_adbus(80)<="11000101";--a2
mem_adbus(81)<="00000000";--a2
mem_adbus(82)<="11101000";--(b5),in
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(83)<="01111100";--cp
mem_adbus(84)<="00000101";--05h
mem_adbus(85)<="11001000";--jnz
mem_adbus(86)<="01100010";--b6
mem_adbus(87)<="00000000";--b6
mem_adbus(88)<="00000010";--ld b
mem_adbus(89)<="00000101";--05h
mem_adbus(90)<="00000000";--ld a
mem_adbus(91)<="00000101";--05h
mem_adbus(92)<="11101100";--out
mem_adbus(93)<="11000000";--jp
mem_adbus(94)<="11000101";--a2
mem_adbus(95)<="00000000";--a2
mem_adbus(96)<="11101000";--(b6),in
mem_adbus(97)<="01111100";--cp
mem_adbus(98)<="00000110";--06h
mem_adbus(99)<="11001000";--jnz
mem_adbus(100)<="01101110";--b7
mem_adbus(101)<="00000000";--b7
mem_adbus(102)<="00000010";--ld b
mem_adbus(103)<="00000110";--06h
mem_adbus(104)<="00000000";--ld a
mem_adbus(105)<="00000110";--06h
mem_adbus(106)<="11101100";--out
mem_adbus(107)<="11000000";--jp
mem_adbus(108)<="11000101";--a2
mem_adbus(109)<="00000000";--a2
mem_adbus(110)<="11101000";--(b7),in
mem_adbus(111)<="01111100";--cp
mem_adbus(112)<="00000111";--07h
mem_adbus(113)<="11001000";--jnz
mem_adbus(114)<="01111100";--b8
mem_adbus(115)<="00000000";--b8
mem_adbus(116)<="00000010";--ld b
mem_adbus(117)<="00000111";--07h
mem_adbus(118)<="00000000";--ld a
mem_adbus(119)<="00000111";--07h
mem_adbus(120)<="11101100";--out
mem_adbus(121)<="11000000";--jp
mem_adbus(122)<="11000101";--a2
mem_adbus(123)<="00000000";--a2
mem_adbus(124)<="11101000";--(b8),in
mem_adbus(125)<="01111100";--cp
mem_adbus(126)<="00001010";--08h
mem_adbus(127)<="11001000";--jnz
mem_adbus(128)<="10001010";--b9
mem_adbus(129)<="00000000";--b9
mem_adbus(130)<="00000010";--ld b
mem_adbus(131)<="00001010";--08h
mem_adbus(132)<="00000000";--ld a
mem_adbus(133)<="00001010";--08h
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(134)<="11101100";--out
mem_adbus(135)<="11000000";--jp
mem_adbus(136)<="11000101";--a2
mem_adbus(137)<="00000000";--a2
mem_adbus(138)<="11101000";--(b9),in
mem_adbus(139)<="01111100";--cp
mem_adbus(140)<="00001011";--09h
mem_adbus(141)<="11001000";--jnz
mem_adbus(142)<="10011000";--b10
mem_adbus(143)<="00000000";--b10
mem_adbus(144)<="00000010";--ld b
mem_adbus(145)<="00001011";--09h
mem_adbus(146)<="00000000";--ld a
mem_adbus(147)<="00001011";--09h
mem_adbus(148)<="11101100";--out
mem_adbus(149)<="11000000";--jp
mem_adbus(150)<="11000101";--a2
mem_adbus(151)<="00000000";--a2
mem_adbus(152)<="11101000";--(b10),in
mem_adbus(153)<="01111100";--cp
mem_adbus(154)<="00001100";--0Ah
mem_adbus(155)<="11001000";--jnz
mem_adbus(156)<="10100100";--b11
mem_adbus(157)<="00000000";--b11
mem_adbus(158)<="00010010";--ld a,b
mem_adbus(159)<="01000010";--add c
mem_adbus(160)<="11101100";--out
mem_adbus(161)<="11000000";--jp
mem_adbus(162)<="00000000";--a1
mem_adbus(163)<="00000000";--a1
mem_adbus(164)<="11101000";--(b11),in
mem_adbus(165)<="01111100";--cp
mem_adbus(166)<="00001101";--0Bh
mem_adbus(167)<="11001000";--jnz
mem_adbus(168)<="10110000";--b12
mem_adbus(169)<="00000000";--b12
mem_adbus(170)<="00010010";--ld a,b
mem_adbus(171)<="01101010";--sub c
mem_adbus(172)<="11101100";--out
mem_adbus(173)<="11000000";--jp
mem_adbus(174)<="00000000";--a1
mem_adbus(175)<="00000000";--a1
mem_adbus(176)<="11101000";--(b12),in
mem_adbus(177)<="01111100";--cp
mem_adbus(178)<="00001110";--0Ch
mem_adbus(179)<="11001000";--jnz
mem_adbus(180)<="10111001";--b13
mem_adbus(181)<="00000000";--b13
mem_adbus(182)<="11000000";--jp
mem_adbus(183)<="01100000";--d1
mem_adbus(184)<="00000001";--d1
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(185)<="11101000";--(b13),in
mem_adbus(186)<="01111100";--cp
mem_adbus(187)<="00001111";--0Dh
mem_adbus(188)<="11001000";--jnz
mem_adbus(189)<="11000010";--b14
mem_adbus(190)<="00000000";--b14
mem_adbus(191)<="11000000";--jp
mem_adbus(192)<="01110001";--d2
mem_adbus(193)<="00000001";--d2
mem_adbus(194)<="11000000";--(b14),jp
mem_adbus(195)<="00000000";--a1
mem_adbus(196)<="00000000";--a1
mem_adbus(197)<="11101000";--(a2),in
mem_adbus(198)<="01111100";--bit
mem_adbus(199)<="00010000";--10h
mem_adbus(200)<="11000100";--jz
mem_adbus(201)<="11000101";--a2
mem_adbus(202)<="00000000";--a2
mem_adbus(203)<="11101000";--in
mem_adbus(204)<="01111100";--bit
mem_adbus(205)<="00010000";--10h
mem_adbus(206)<="11010100";--jnz
mem_adbus(207)<="11000101";--a2
mem_adbus(208)<="00000000";--a2
mem_adbus(209)<="11101000";--in
mem_adbus(210)<="01111110";--cp
mem_adbus(211)<="00000000";--00h
mem_adbus(212)<="11010100";--jnz
mem_adbus(213)<="11011111";--c1
mem_adbus(214)<="00000000";--c1
mem_adbus(215)<="00000100";--ld c
mem_adbus(216)<="00000000";--00h
mem_adbus(217)<="00000000";--ld a
mem_adbus(218)<="00000000";--00h
mem_adbus(219)<="11101100";--out
mem_adbus(220)<="11000000";--jp
mem_adbus(221)<="00000000";--a1
mem_adbus(222)<="00000000";--a1
mem_adbus(223)<="11101000";--(c1),in
mem_adbus(224)<="01111100";--cp
mem_adbus(225)<="00000001";--01h
mem_adbus(226)<="11001000";--jnz
mem_adbus(227)<="11101101";--c2
mem_adbus(228)<="00000000";--c2
mem_adbus(229)<="00000100";--ld c
mem_adbus(230)<="00000001";--01h
mem_adbus(231)<="00000000";--ld a
mem_adbus(232)<="00000001";--01h
mem_adbus(233)<="11101100";--out
mem_adbus(234)<="11000000";--jp
mem_adbus(235)<="00000000";--a1
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(236)<="00000000";--a1
mem_adbus(237)<="11101000";--(c2),in
mem_adbus(238)<="01111100";--cp
mem_adbus(239)<="00000010";--02h
mem_adbus(240)<="11001000";--jnz
mem_adbus(241)<="11111011";--c3
mem_adbus(242)<="00000000";--c3
mem_adbus(243)<="00000100";--ld c
mem_adbus(244)<="00000010";--02h
mem_adbus(245)<="00000000";--ld a
mem_adbus(246)<="00000010";--02h
mem_adbus(247)<="11101100";--out
mem_adbus(248)<="11000000";--jp
mem_adbus(249)<="00000000";--a1
mem_adbus(250)<="00000000";--a1
mem_adbus(251)<="11101000";--(c3),in
mem_adbus(252)<="01111100";--cp
mem_adbus(253)<="00000011";--03h
mem_adbus(254)<="11001000";--jnz
mem_adbus(255)<="00001001";--c4
mem_adbus(256)<="00000001";--c4
mem_adbus(257)<="00000100";--ld c
mem_adbus(258)<="00000011";--03h
mem_adbus(259)<="00000000";--ld a
mem_adbus(260)<="00000011";--03h
mem_adbus(261)<="11101100";--out
mem_adbus(262)<="11000000";--jp
mem_adbus(263)<="00000000";--a1
mem_adbus(264)<="00000000";--a1
mem_adbus(265)<="11101000";--(c4),in
mem_adbus(266)<="01111100";--cp
mem_adbus(267)<="00000100";--04h
mem_adbus(268)<="11001000";--jnz
mem_adbus(269)<="00010111";--c5
mem_adbus(270)<="00000001";--c5
mem_adbus(271)<="00000100";--ld c
mem_adbus(272)<="00000100";--04h
mem_adbus(273)<="00000000";--ld a
mem_adbus(274)<="00000100";--04h
mem_adbus(275)<="11101100";--out
mem_adbus(276)<="11000000";--jp
mem_adbus(277)<="00000000";--a1
mem_adbus(278)<="00000000";--a1
mem_adbus(279)<="11101000";--(c5),in
mem_adbus(280)<="01111100";--cp
mem_adbus(281)<="00000101";--05h
mem_adbus(282)<="11001000";--jnz
mem_adbus(283)<="00100101";--c6
mem_adbus(284)<="00000001";--c6
mem_adbus(285)<="00000010";--ld c
mem_adbus(286)<="00000101";--05h
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(287)<="00000000";--ld a
mem_adbus(288)<="00000101";--05h
mem_adbus(289)<="11101100";--out
mem_adbus(290)<="11000000";--jp
mem_adbus(291)<="00000000";--a1
mem_adbus(292)<="00000000";--a1
mem_adbus(293)<="11101000";--(c6),in
mem_adbus(294)<="01111100";--cp
mem_adbus(295)<="00000110";--06h
mem_adbus(296)<="11001000";--jnz
mem_adbus(297)<="00110011";--c7
mem_adbus(298)<="00000001";--c7
mem_adbus(299)<="00000100";--ld c
mem_adbus(300)<="00000110";--06h
mem_adbus(301)<="00000000";--ld a
mem_adbus(302)<="00000110";--06h
mem_adbus(303)<="11101100";--out
mem_adbus(304)<="11000000";--jp
mem_adbus(305)<="00000000";--a1
mem_adbus(306)<="00000000";--a1
mem_adbus(307)<="11101000";--(c7),in
mem_adbus(308)<="01111100";--cp
mem_adbus(309)<="00000111";--07h
mem_adbus(310)<="11001000";--jnz
mem_adbus(311)<="01000001";--c8
mem_adbus(312)<="00000001";--c8
mem_adbus(313)<="00000100";--ld c
mem_adbus(314)<="00000111";--07h
mem_adbus(315)<="00000000";--ld a
mem_adbus(316)<="00000111";--07h
mem_adbus(317)<="11101100";--out
mem_adbus(318)<="11000000";--jp
mem_adbus(319)<="00000000";--a1
mem_adbus(320)<="00000000";--a1
mem_adbus(321)<="11101000";--(c8),in
mem_adbus(322)<="01111100";--cp
mem_adbus(323)<="00001010";--08h
mem_adbus(324)<="11001000";--jnz
mem_adbus(325)<="01001111";--c9
mem_adbus(326)<="00000001";--c9
mem_adbus(327)<="00000100";--ld c
mem_adbus(328)<="00001010";--08h
mem_adbus(329)<="00000000";--ld a
mem_adbus(330)<="00001010";--08h
mem_adbus(331)<="11101100";--out
mem_adbus(332)<="11000000";--jp
mem_adbus(333)<="00000000";--a1
mem_adbus(334)<="00000000";--a1
mem_adbus(335)<="11101000";--(c9),in
mem_adbus(336)<="01111100";--cp
mem_adbus(337)<="00001011";--09h
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(338)<="11001000";--jnz
mem_adbus(339)<="01011101";--c10
mem_adbus(340)<="00000001";--c10
mem_adbus(341)<="00000100";--ld c
mem_adbus(342)<="00001011";--09h
mem_adbus(343)<="00000000";--ld a
mem_adbus(344)<="00001011";--09h
mem_adbus(345)<="11101100";--out
mem_adbus(346)<="11000000";--jp
mem_adbus(347)<="00000000";--a1
mem_adbus(348)<="00000000";--a1
mem_adbus(349)<="11000000";--(c10),jp
mem_adbus(350)<="11000101";--a2
mem_adbus(351)<="00000000";--a2
mem_adbus(352)<="00000000";--(d1),ld a
mem_adbus(353)<="00000001";--01h
mem_adbus(354)<="11101100";--out
mem_adbus(355)<="11100000";--(d12),call
mem_adbus(356)<="10000010";--e1
mem_adbus(357)<="00000001";--e1
mem_adbus(358)<="10001100";--rlc
mem_adbus(359)<="11101100";--out
mem_adbus(360)<="11101000";--in
mem_adbus(361)<="01111100";--bit
mem_adbus(362)<="00010000";--10h
mem_adbus(363)<="11000100";--jz
mem_adbus(364)<="01100011";--d12
mem_adbus(365)<="00000001";--d12
mem_adbus(366)<="11000000";--jp
mem_adbus(367)<="00000000";--a1
mem_adbus(368)<="00000000";--a1
mem_adbus(369)<="00000000";--(d2)ld a
mem_adbus(370)<="10000000";--80h
mem_adbus(371)<="11101100";--out
mem_adbus(372)<="11100000";--(d22),call
mem_adbus(373)<="10000010";--e1
mem_adbus(374)<="00000001";--e1
mem_adbus(375)<="10001000";--rrc
mem_adbus(376)<="11101100";--out
mem_adbus(377)<="11101000";--in
mem_adbus(378)<="01111100";--bit
mem_adbus(379)<="00010000";--10h
mem_adbus(380)<="11000100";--jz
mem_adbus(381)<="01110100";--d22
mem_adbus(382)<="00000001";--d22
mem_adbus(383)<="11000000";--jp
mem_adbus(384)<="00000000";--a1
mem_adbus(385)<="00000000";--a1
mem_adbus(386)<="00000100";--(e11),ld c
mem_adbus(387)<="11111111";--ffh
mem_adbus(388)<="00000010";--ld b
```

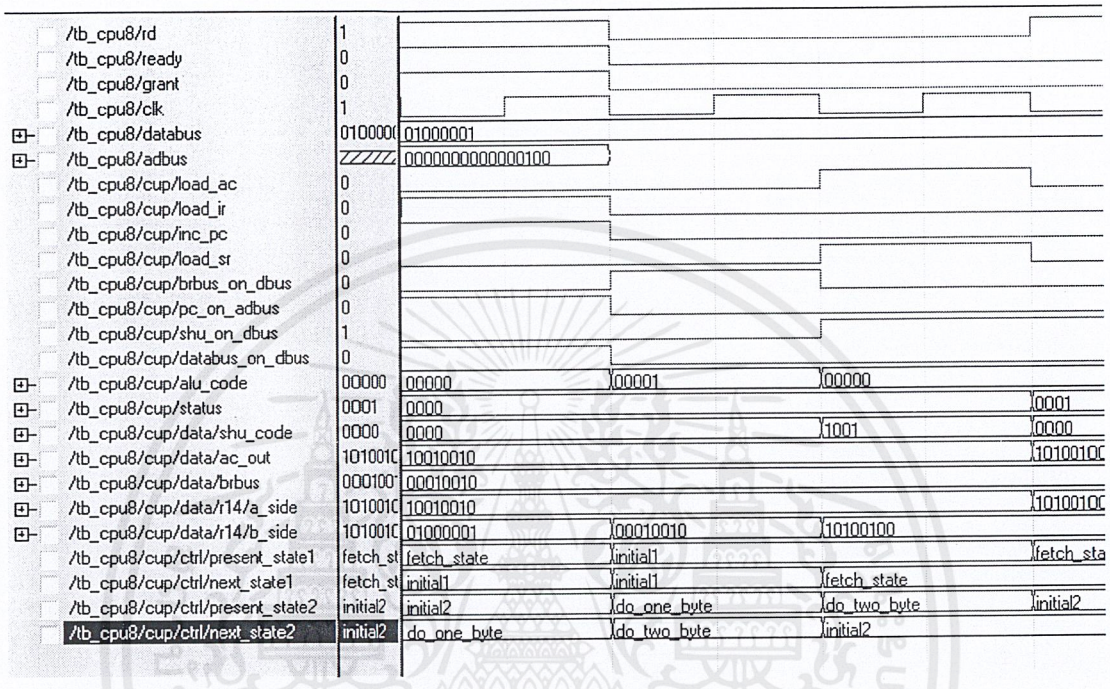
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
mem_adbus(389)<="11111111";--ffh
mem_adbus(390)<="01011001";--(e12)dec b
mem_adbus(391)<="11100110";--jc
mem_adbus(392)<="10010100";--e14
mem_adbus(393)<="00000001";--e14
mem_adbus(394)<="11000000";--jp
mem_adbus(395)<="10000110";--e12
mem_adbus(396)<="00000001";--e12
mem_adbus(397)<="01011010";--(e13)dec c
mem_adbus(398)<="11001100";--jc
mem_adbus(399)<="10000110";--e12
mem_adbus(400)<="00000001";--e12
mem_adbus(401)<="11000000";--jp
mem_adbus(402)<="10001101";--e13
mem_adbus(403)<="00000001";--e13
mem_adbus(404)<="11100100";--(e14)ret
        if address_n < 0 then
        data <="ZZZZZZZZ";
        else
        data <= mem_adbus(address_n);
        end if;
        end process;
end behavior;
```

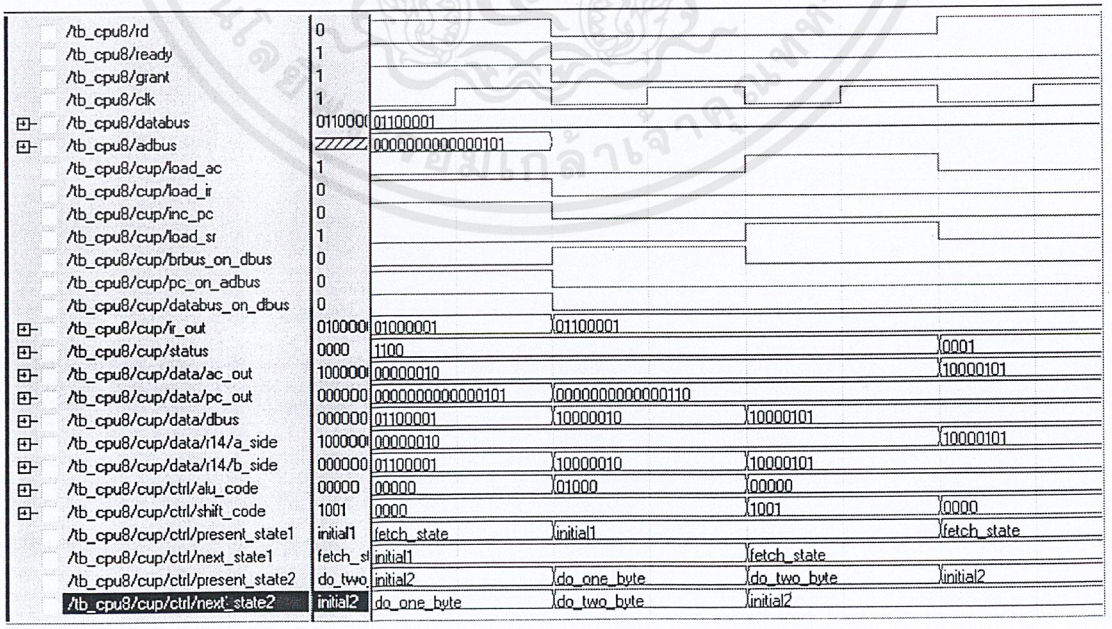
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ข

Wave form ที่ได้จากการ Simulation

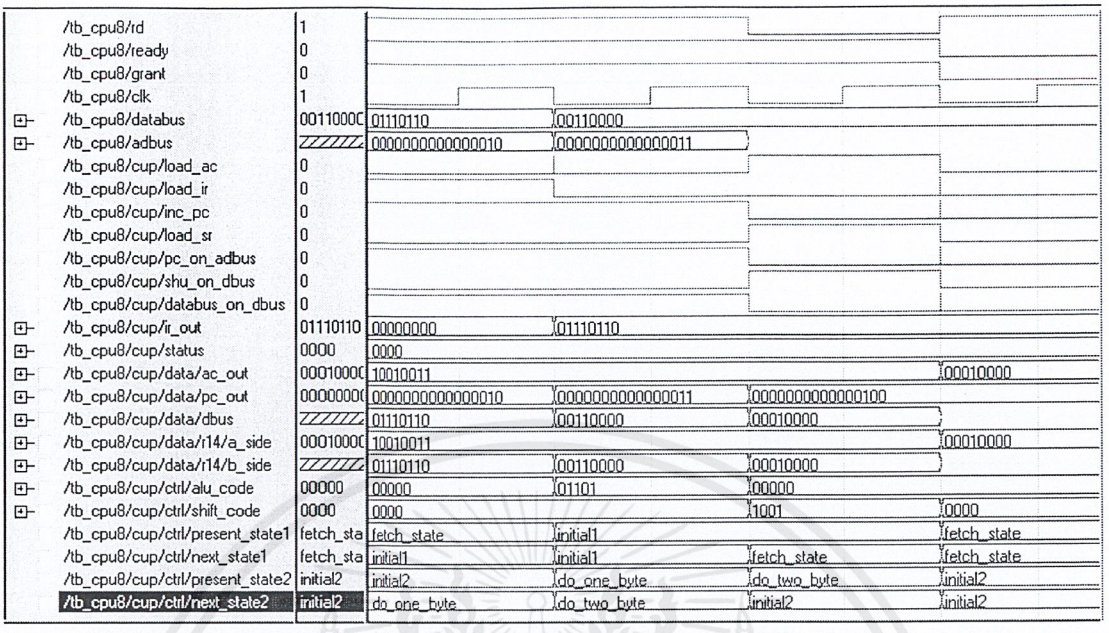


รูปที่ 1 แสดง wave form ที่ได้จากคำสั่ง add

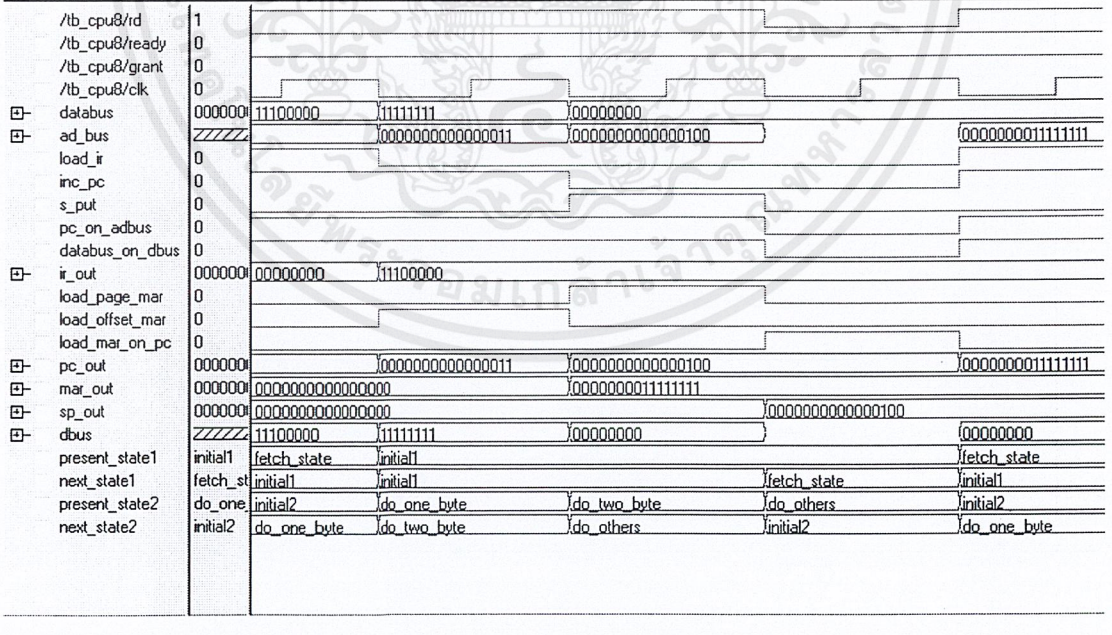


รูปที่ 2 แสดง wave form ที่ได้จากคำสั่ง adc

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

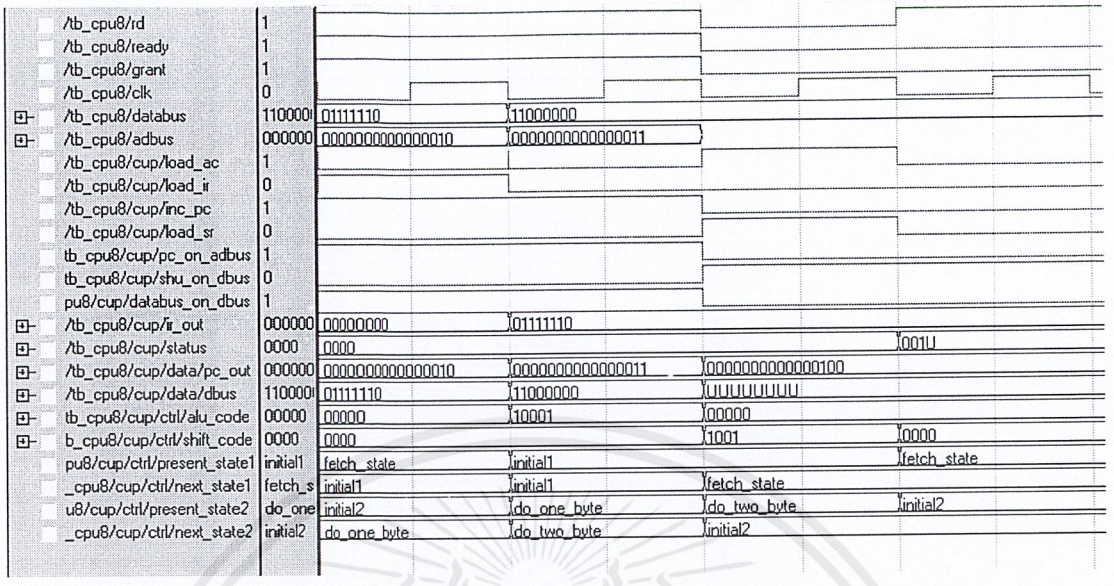


รูปที่ 3 แสดง wave form ที่ได้จากคำสั่ง and

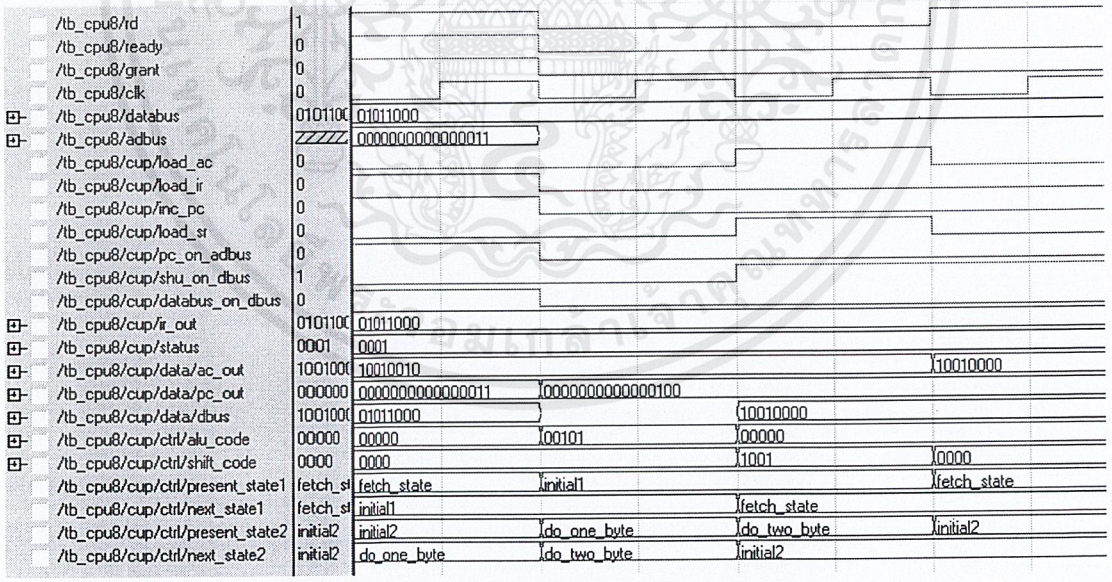


รูปที่ 4 แสดง wave form ที่ได้จากคำสั่ง call

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

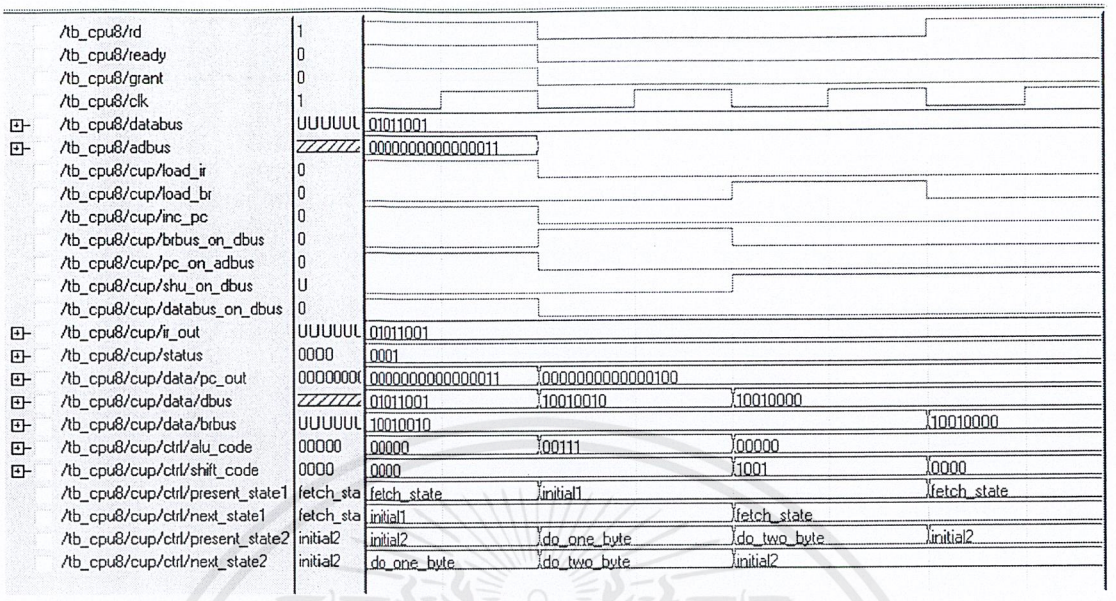


รูปที่ 5 แสดง wave form ที่ได้จากคำสั่ง cp

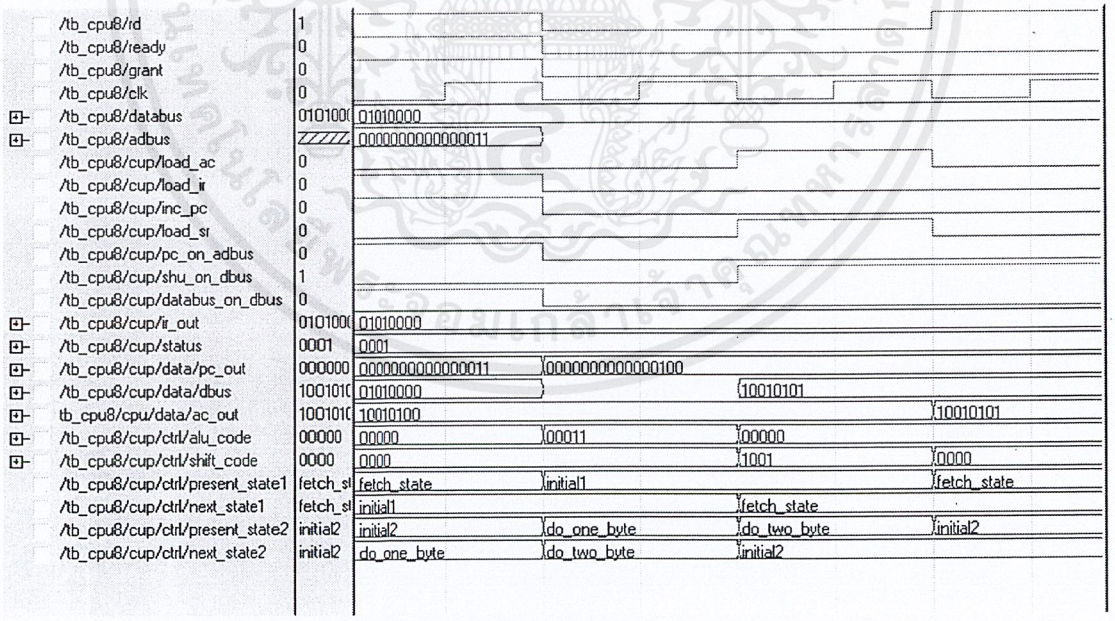


รูปที่ 6 แสดง wave form ที่ได้จากคำสั่ง deca

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

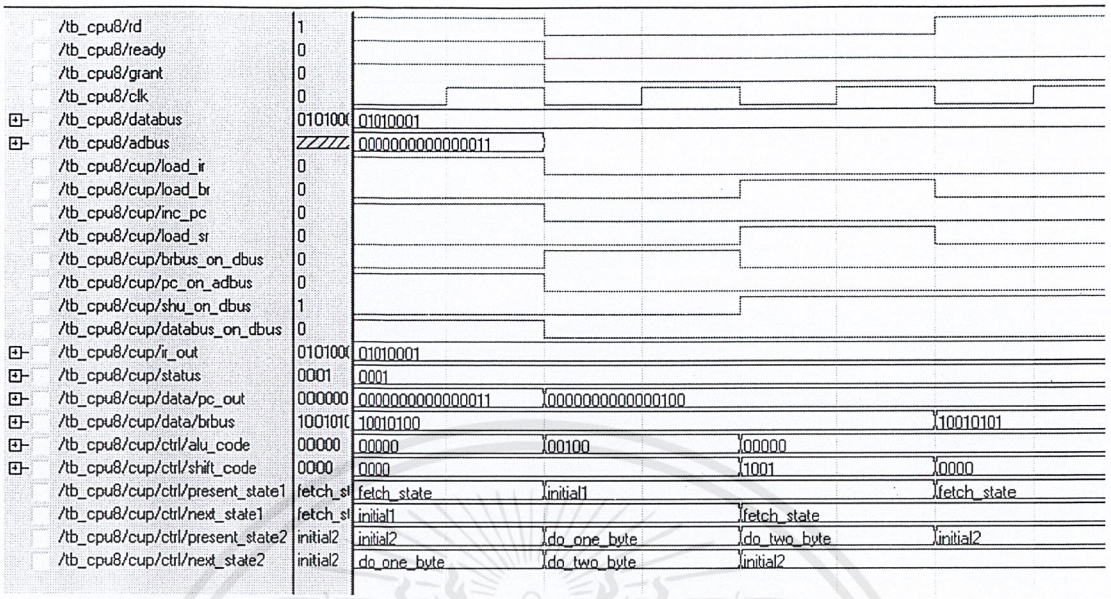


รูปที่ 7 แสดง wave form ที่ได้จากคำสั่ง decr

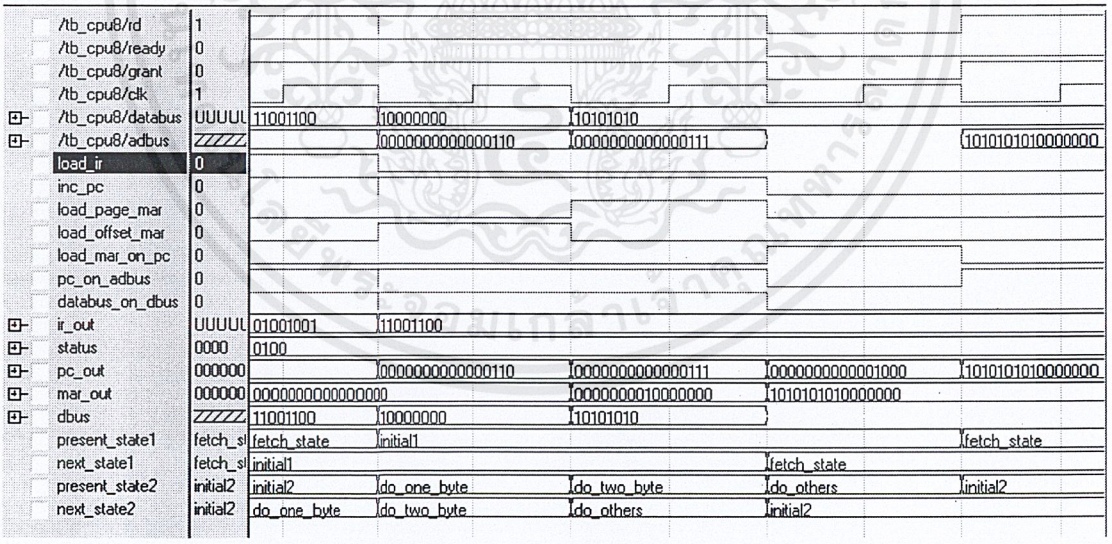


รูปที่ 8 แสดง wave form ที่ได้จากคำสั่ง inca

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

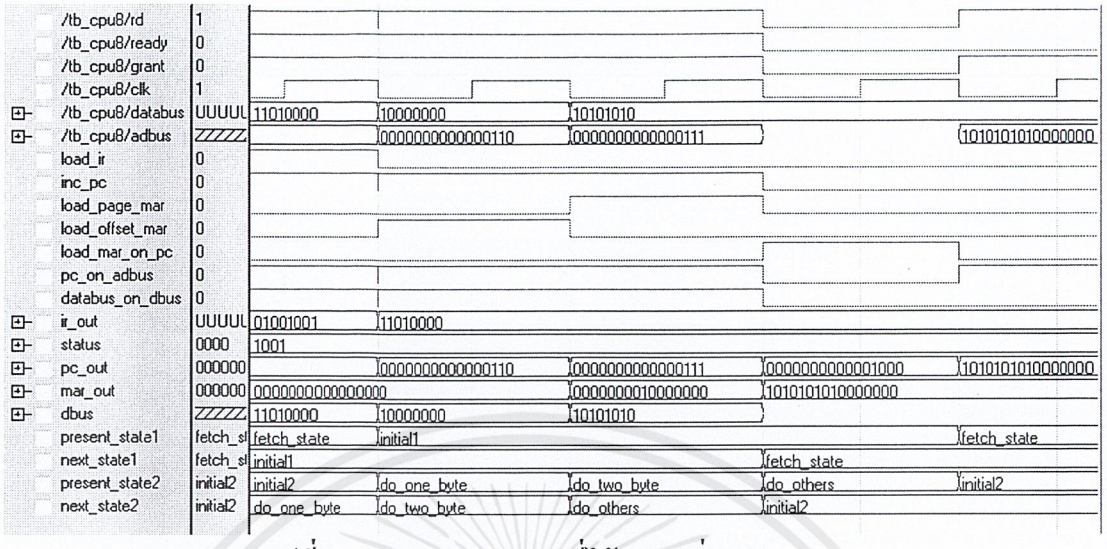


รูปที่ 9 แสดง wave form ที่ได้จากคำสั่ง incr

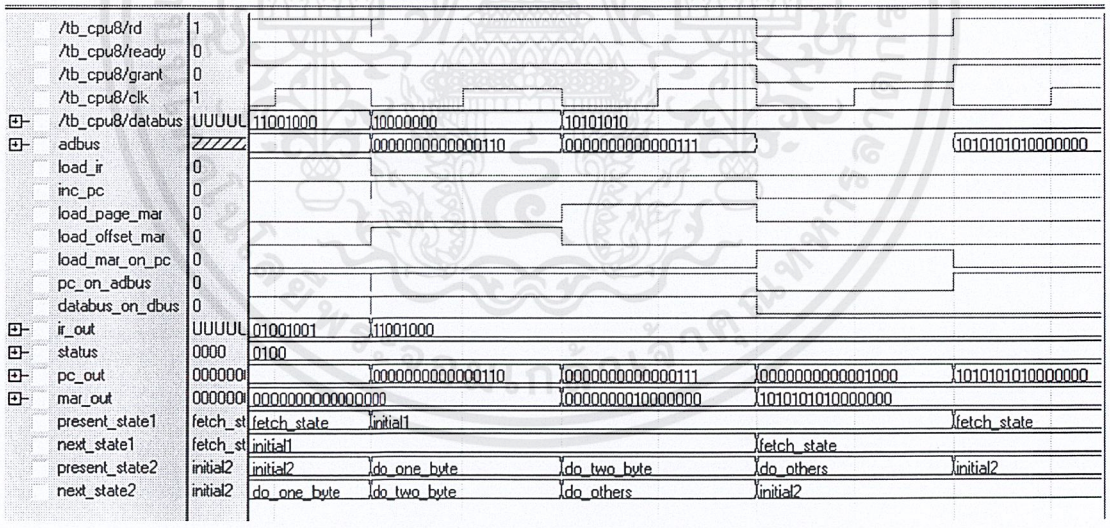


รูปที่ 10 แสดง wave form ที่ได้จากคำสั่ง jc

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

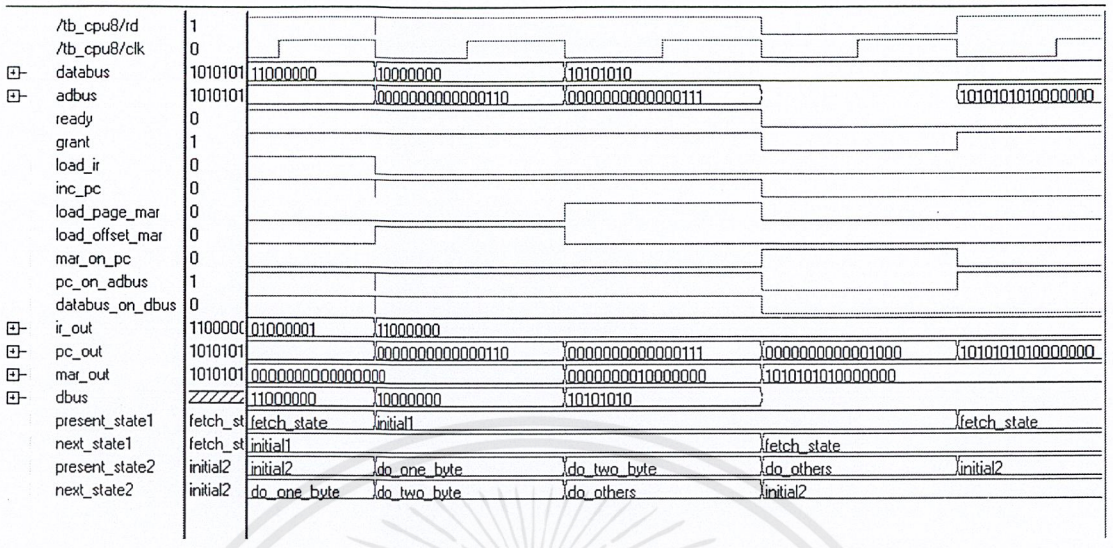


รูปที่ 11 แสดง wave form ที่ได้จากคำสั่ง jnc

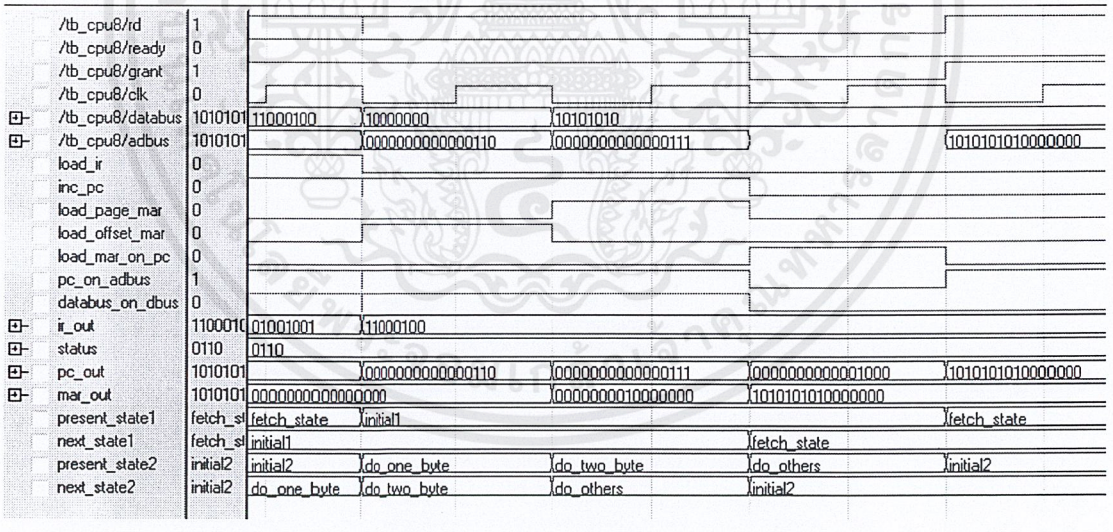


รูปที่ 12 แสดง wave form ที่ได้จากคำสั่ง jnz

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

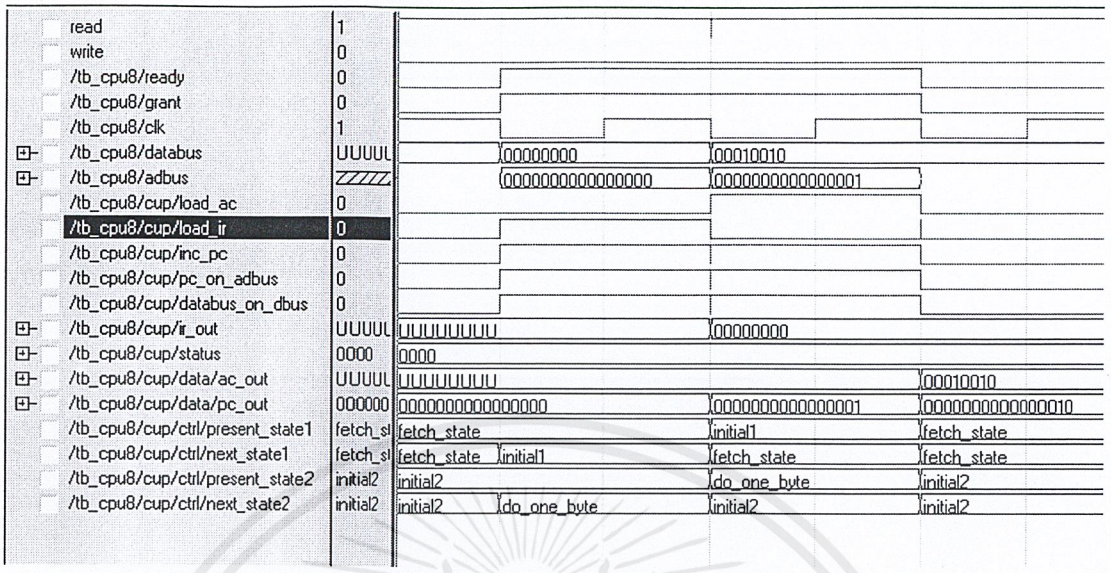


รูปที่ 13 แสดง wave form ที่ได้จากคำสั่ง jp

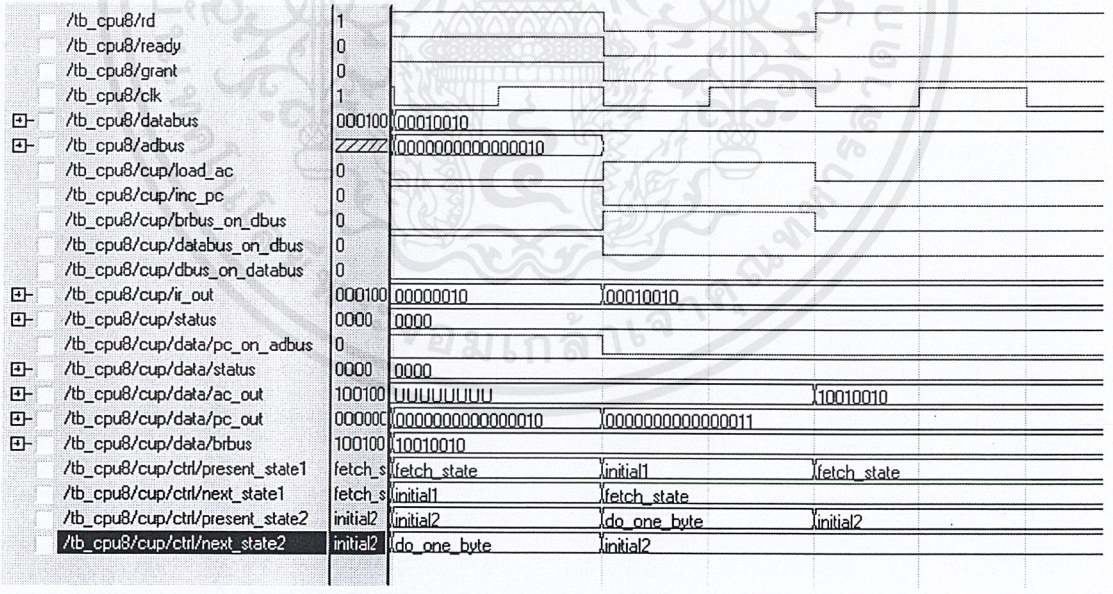


รูปที่ 14 แสดง wave form ที่ได้จากคำสั่ง jz

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

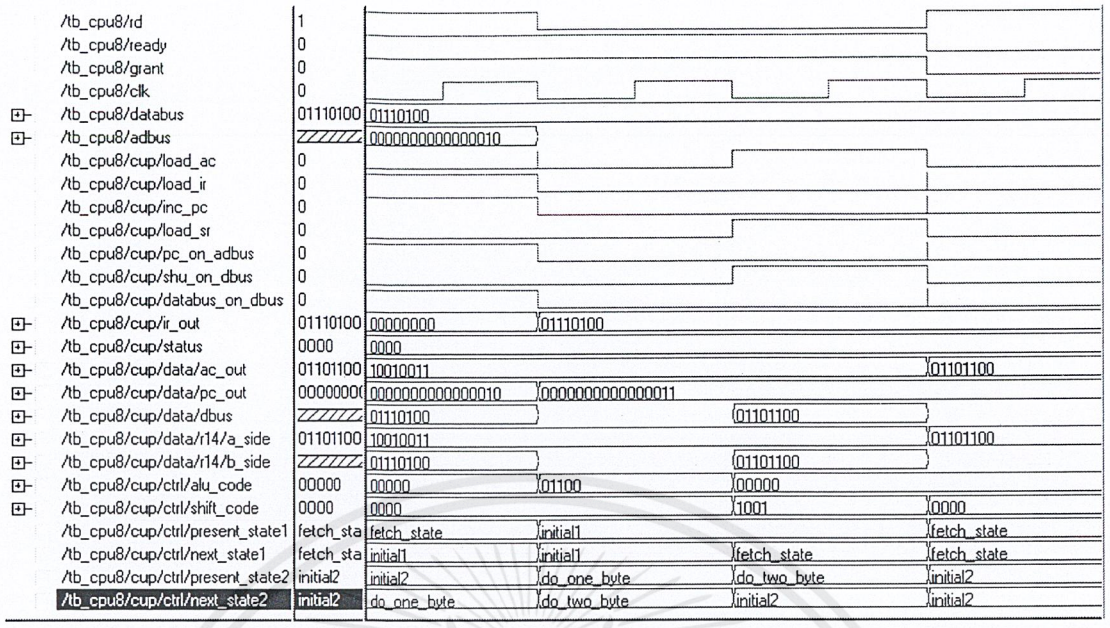


รูปที่ 15 แสดง wave form ที่ได้จากคำสั่ง ld

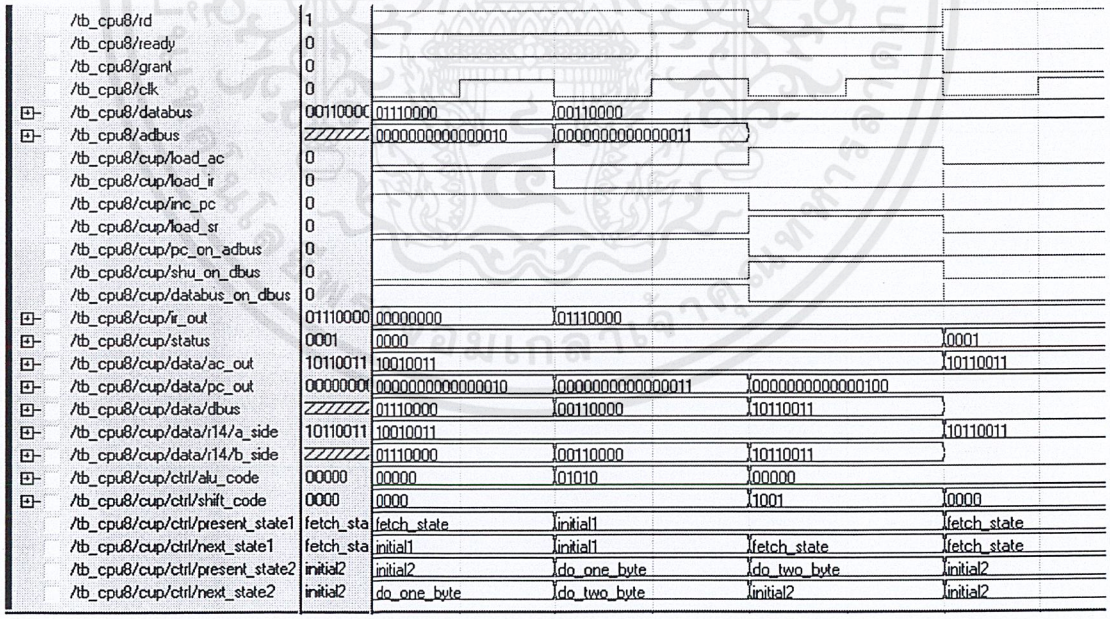


รูปที่ 16 แสดง wave form ที่ได้จากคำสั่ง ldr

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

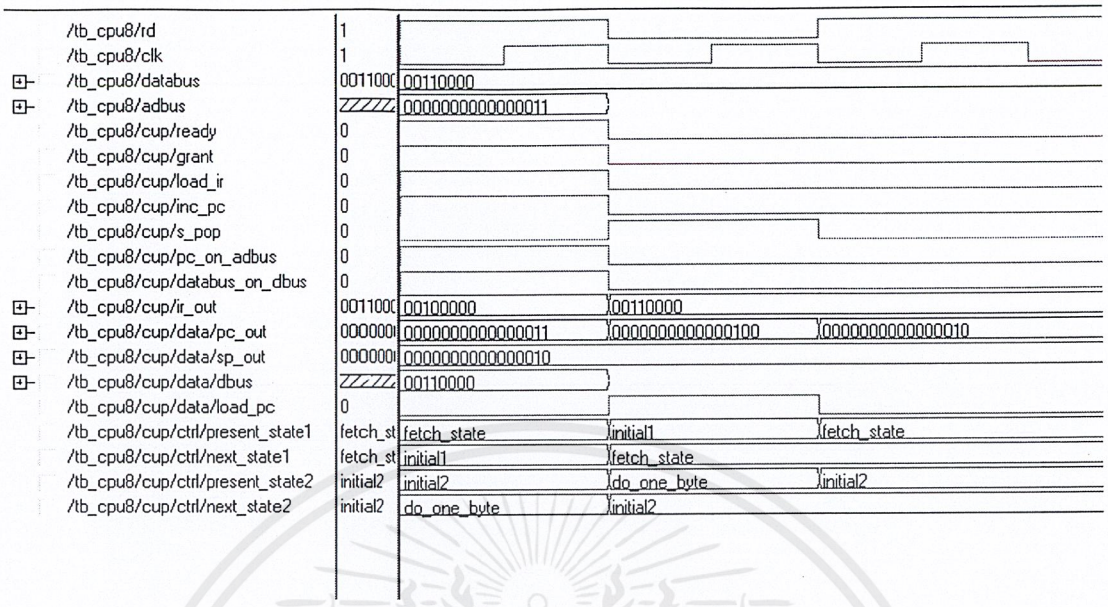


รูปที่ 17 แสดง wave form ที่ได้จากคำสั่ง not

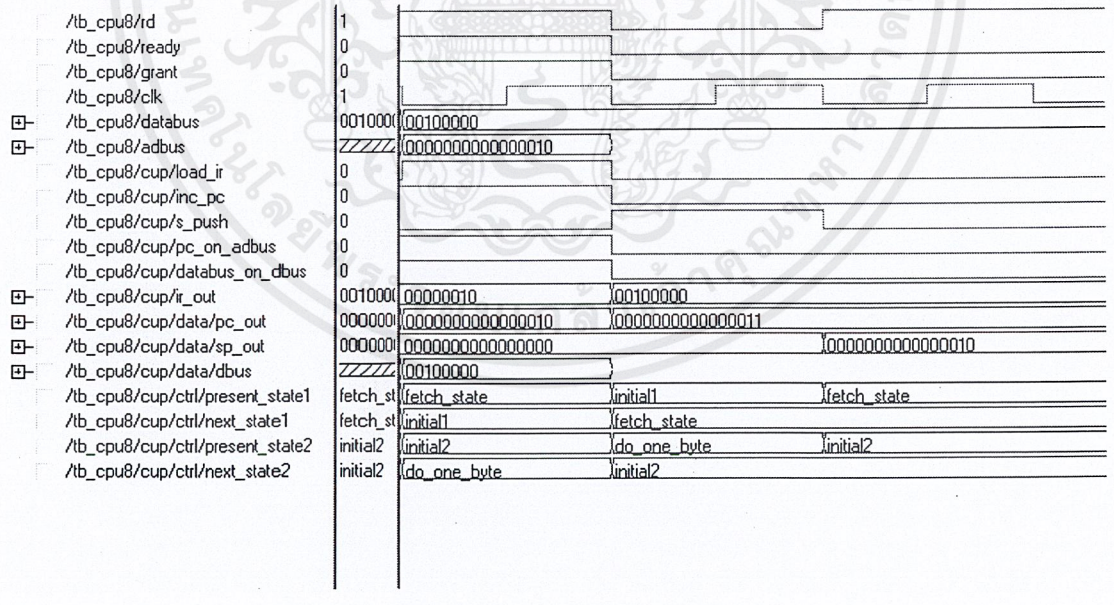


รูปที่ 18 แสดง wave form ที่ได้จากคำสั่ง or

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

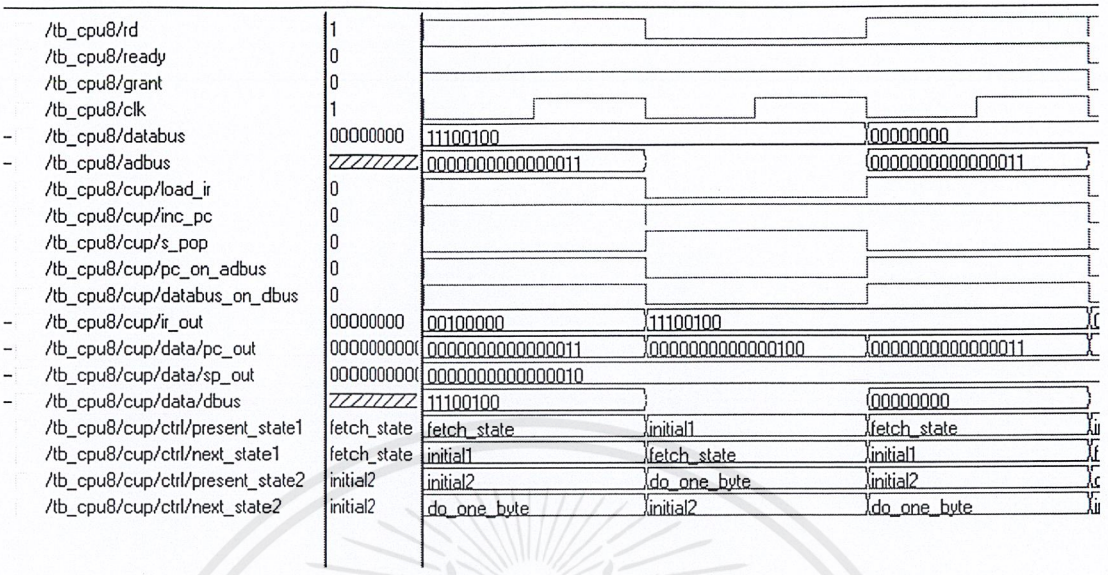


รูปที่ 19 แสดง wave form ที่ได้จากคำสั่ง pop

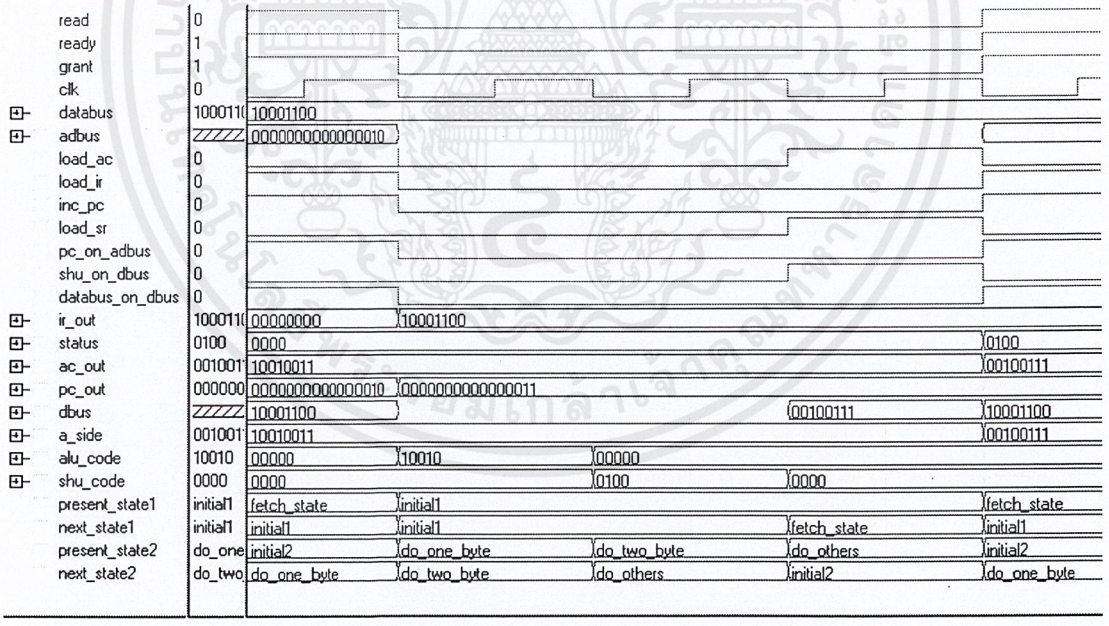


รูปที่ 20 แสดง wave form ที่ได้จากคำสั่ง push

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

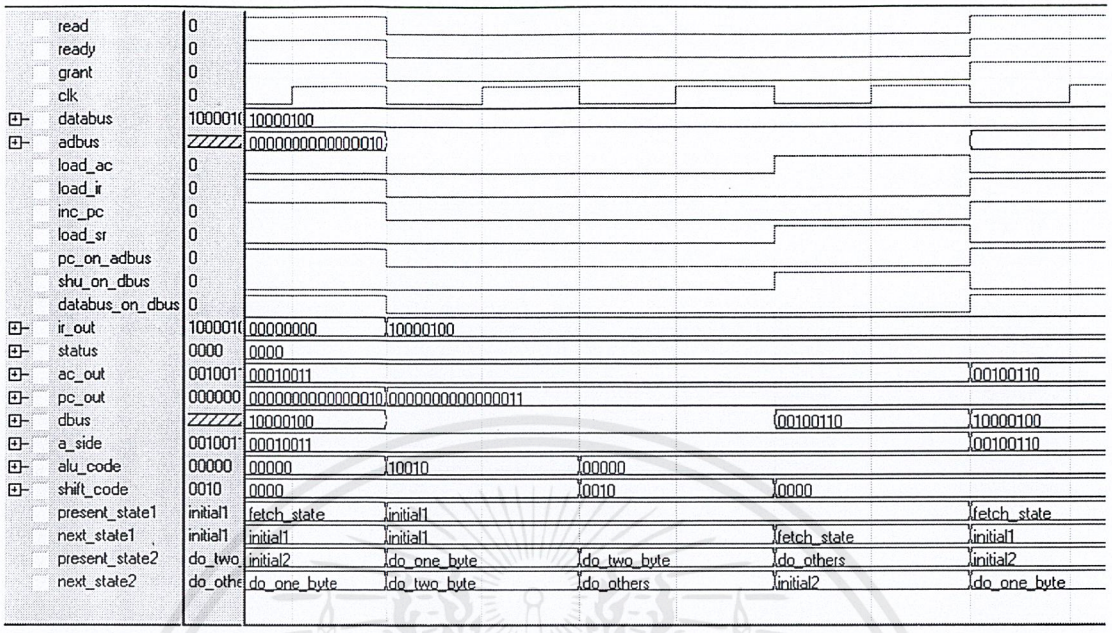


รูปที่ 21 แสดง wave form ที่ได้จากคำสั่ง ret

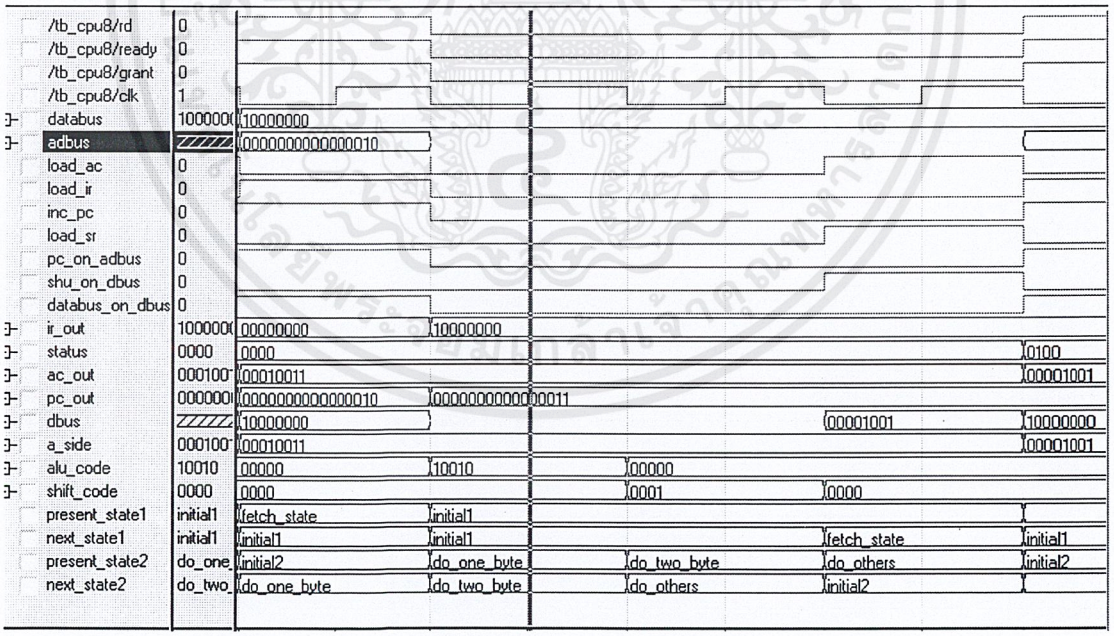


รูปที่ 22 แสดง wave form ที่ได้จากคำสั่ง rlc

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

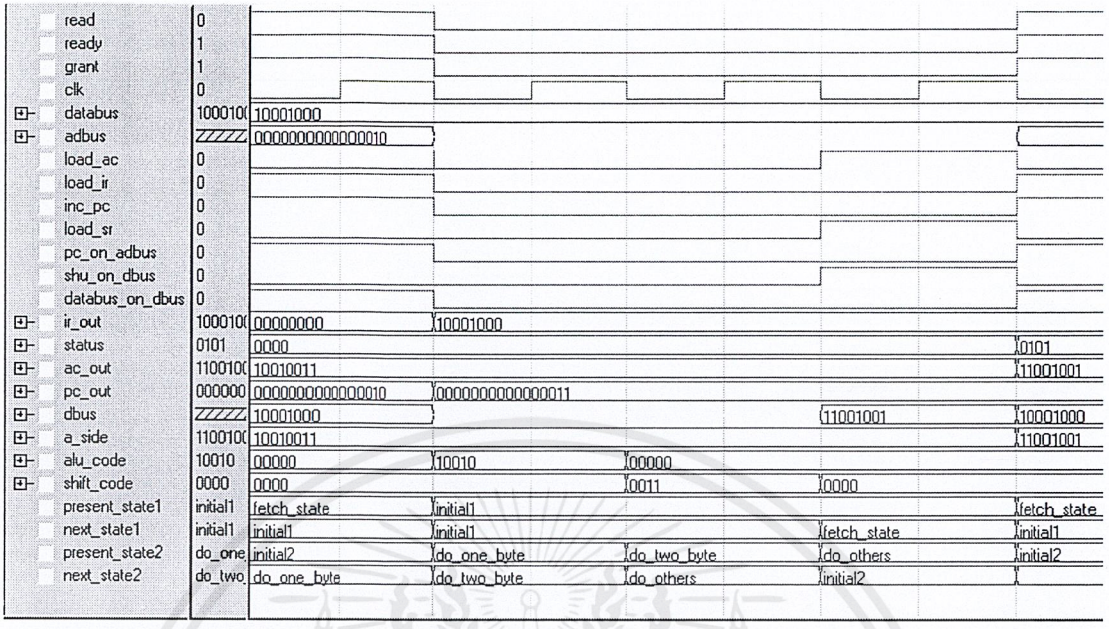


รูปที่ 23 แสดง wave form ที่ได้จากคำสั่ง rol

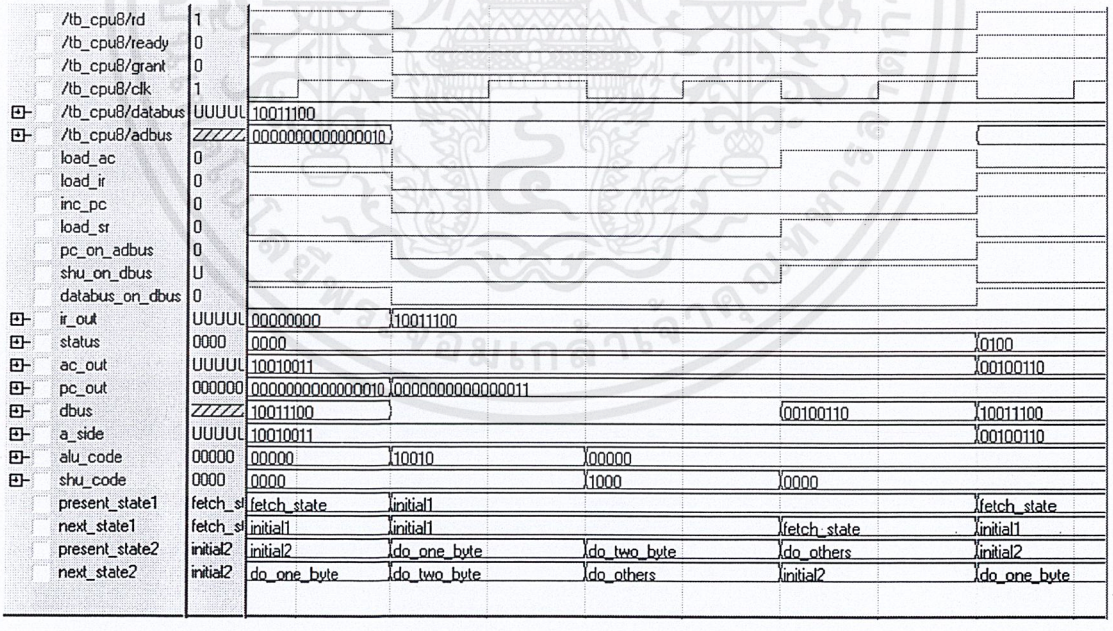


รูปที่ 24 แสดง wave form ที่ได้จากคำสั่ง ror

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

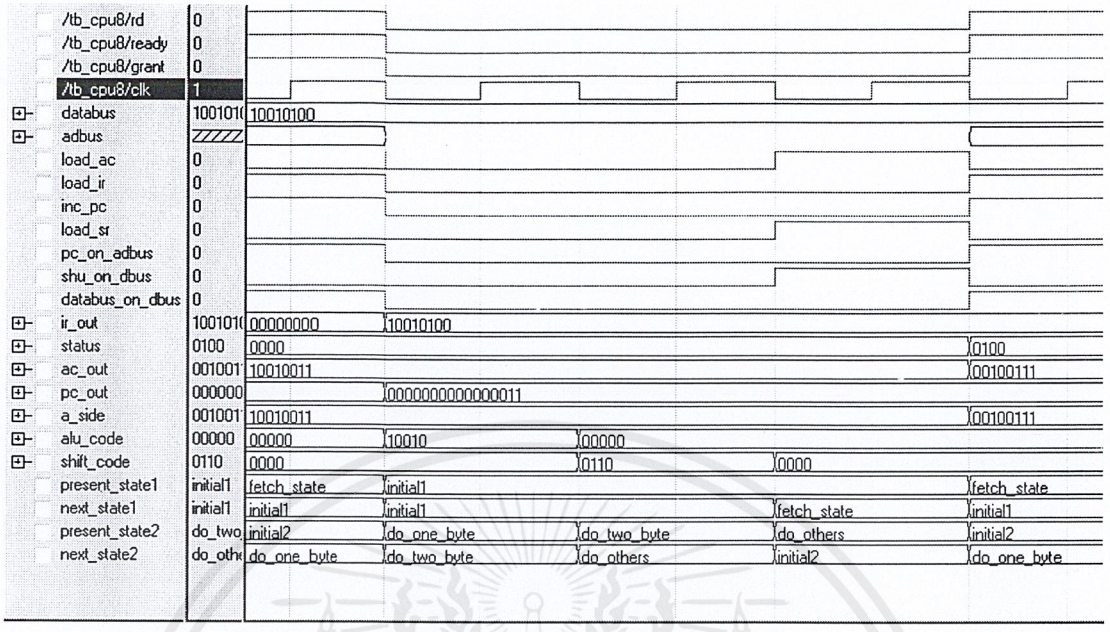


รูปที่ 25 แสดง wave form ที่ได้จากคำสั่ง rrc

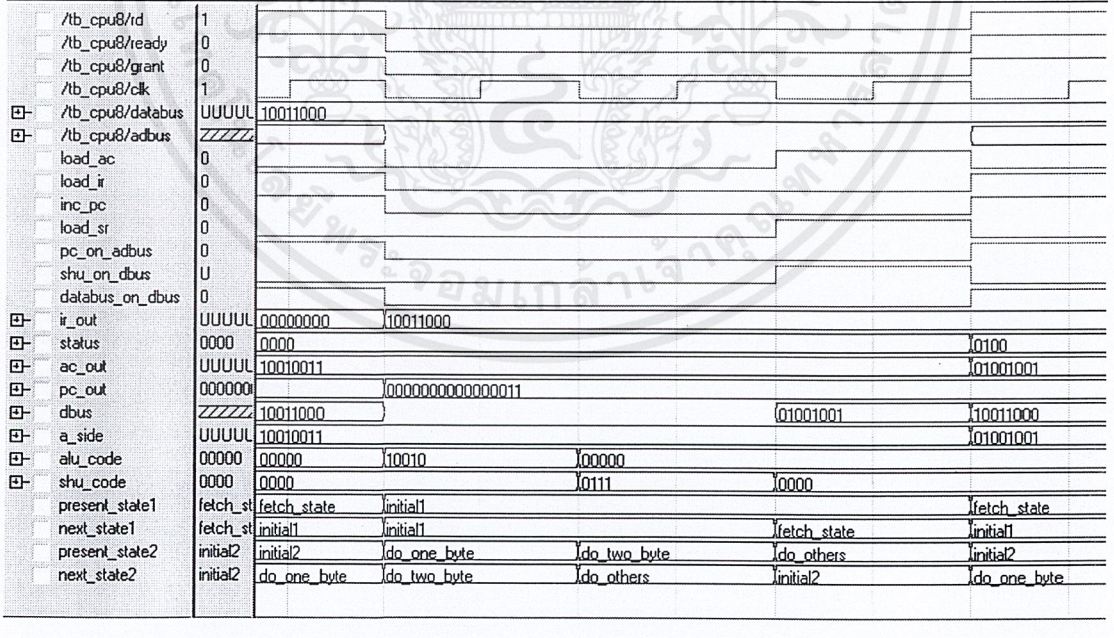


รูปที่ 26 แสดง wave form ที่ได้จากคำสั่ง slr

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

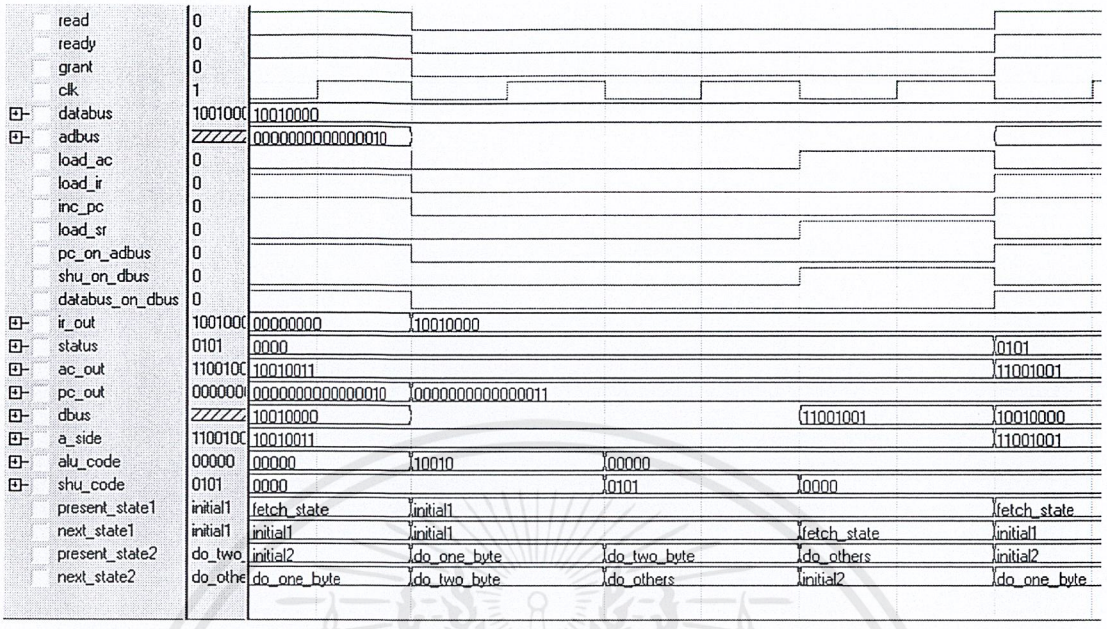


รูปที่ 27 แสดง wave form ที่ได้จากคำสั่ง sls

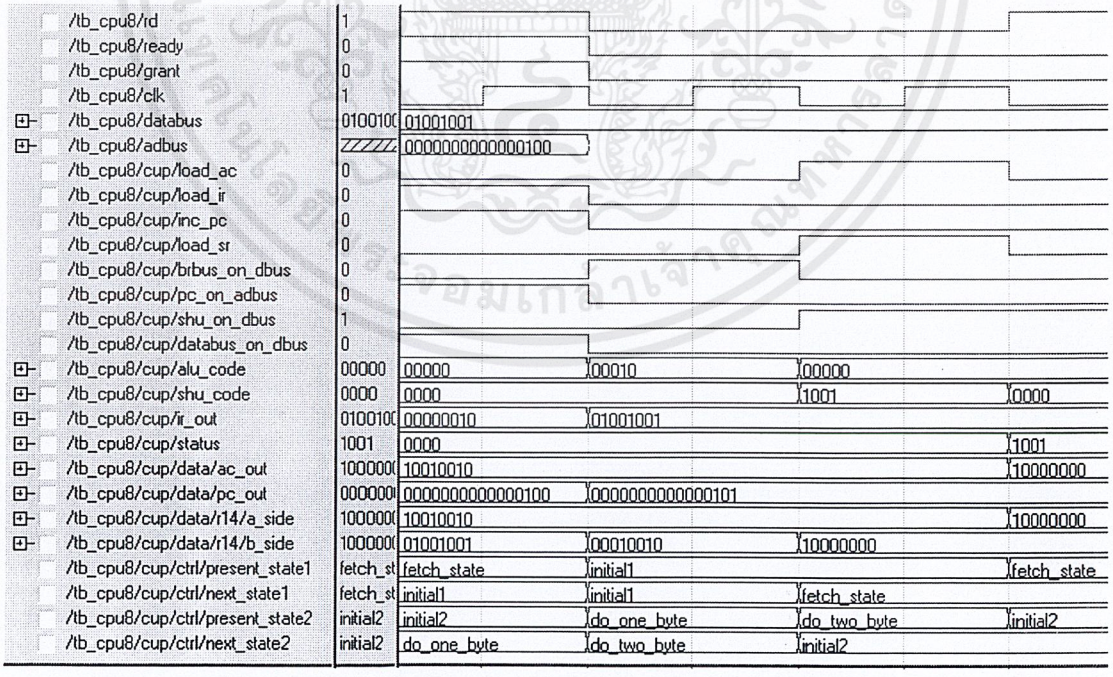


รูปที่ 28 แสดง wave form ที่ได้จากคำสั่ง srr

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

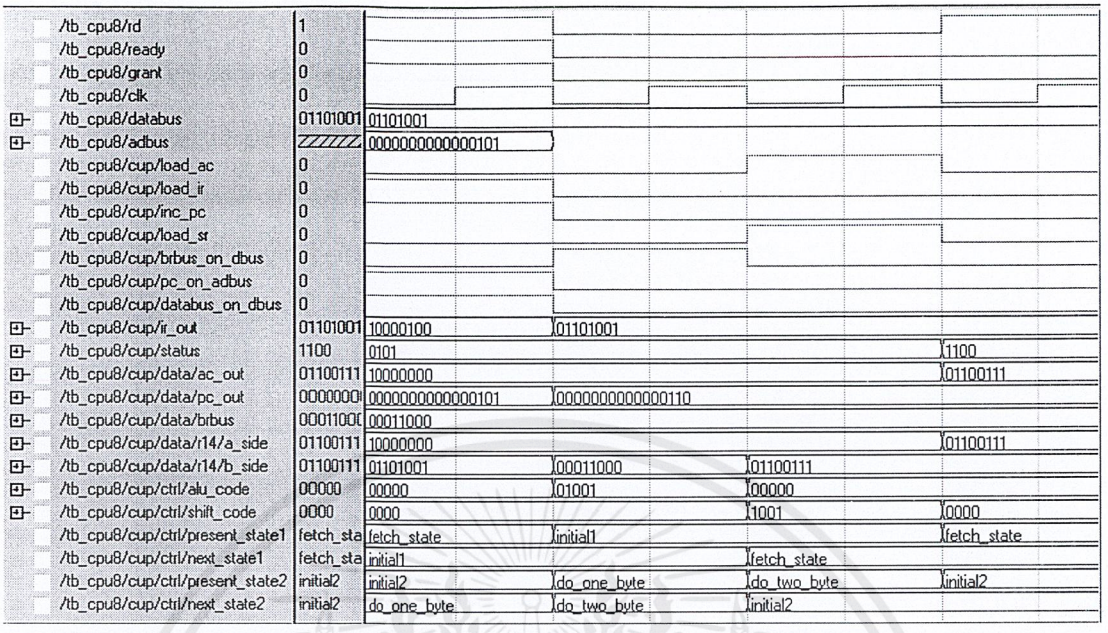


รูปที่ 29 แสดง wave form ที่ได้จากคำสั่ง srs

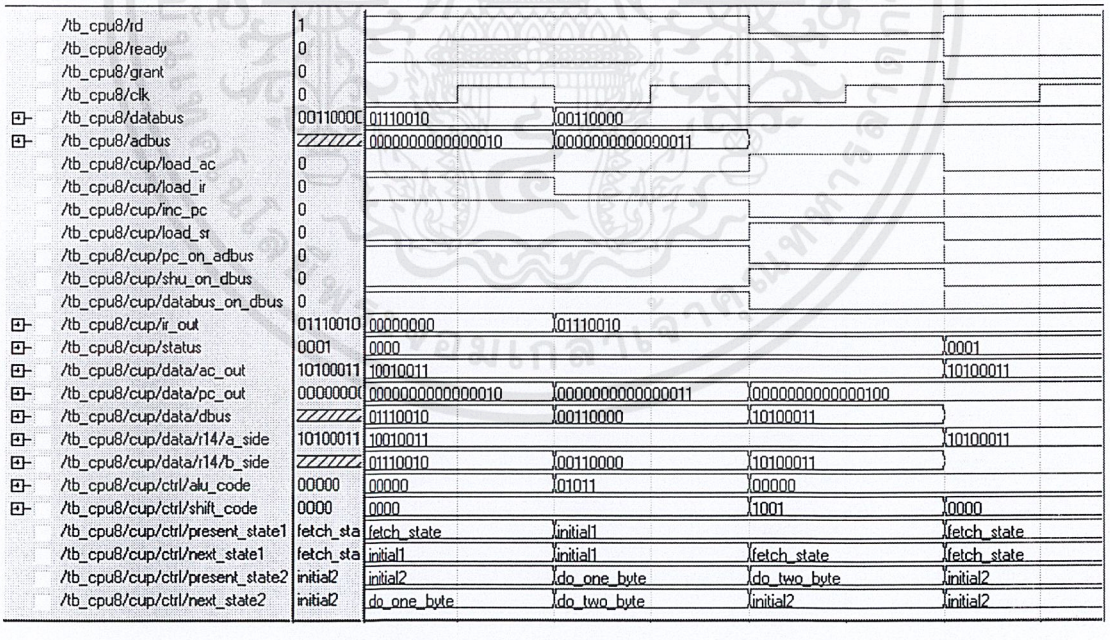


รูปที่ 30 แสดง wave form ที่ได้จากคำสั่ง sub

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 31 แสดง wave form ที่ได้จากคำสั่ง suc



รูปที่ 32 แสดง wave form ที่ได้จากคำสั่ง xor

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หนังสืออ้างอิง

1. ปัญญยศ ไชยกาพ, ปิยะ มารัตน์ : "การออกแบบไมโครคอนโทรลเลอร์ 8031 โดยใช้ภาษา VHDL" , ปริญญานิพนธ์ ปีการศึกษา 2541, ภาควิชาวิศวกรรมคอมพิวเตอร์, คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
2. วิบูลย์ ชื่นแขก: " ไมโครโปรเซสเซอร์ ", สำนักพิมพ์สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ
3. Esperan: "The VHDL Reference Guide",2nd ed.,Esperan Ltd.,1994-95
4. Noothong Watcharakon, "VHDL APPLICATION WORKGROUP",IC Design Group. Thai Microelectronics Center,2000
5. Noothong Watcharakon, "FPGA Design Workshop", IC design Group.TMEC,NECTEC,2001.
6. Zainalabedin Navabi, "VHDL: Analysis and Modeling of Digital System",2nd ed.,McGraw-Hill Inc., 1997

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้