การออกแบบไมโครโปรเซสเซอร์ 8 บิต ที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ โดยใช้เอฟพีจีเอ

นางสาวปัญจภา เรืองสินทรัพย์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2544
ISBN 974-03-0894-5

DESIGN OF 8-BIT SCALABLE-DELAY-INSENSITIVE MICROPROCESSOR USING FPGA

Miss Phunjapa Ruangsinsup

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2001
ISBN 974-03-0894-5

หัวข้อวิทยานิพนธ์	การออกแบบไมโครโปรเซสเซอร์ 8 บิต ที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้โดยใช้เอฟพีจีเอ
โดย	นางสาวปัญจภา เรืองสินทรัพย์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	อาจารย์ ดร.อาทิตย์ ทองทักษ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

…… ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สาธิต วงศ์ประทีป)

…….. อาจารย์ที่ปรึกษา
(อาจารย์ ดร.อาทิตย์ ทองทักษ์)

…… กรรมการ
(อาจารย์ ดร.ฐิต ศิริบูรณ์)

…… กรรมการ
(รองศาสตราจารย์ สมศักดิ์ มิตะถา)

ปัญจภา เรืองสินทรัพย์ : การออกแบบไมโครโปรเซสเซอร์ 8 บิต ที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้โดยใช้เอฟพีจีเอ. (DESIGN OF 8-BIT SCALABLE-DELAY-INSENSITIVE MICROPROCESSOR USING FPGA) อ.ที่ปรึกษา: อ.ดร.อาทิตย์ ทองทักษ์, 110 หน้า. ISBN 974-03-0894-5.

วิทยานิพนธ์นี้เสนอการออกแบบไมโครโปรเซสเซอร์ 8 บิต ที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ เพื่อเป็นแนวทางเริ่มต้นสำหรับการสร้างวงจรแบบอสมวารขึ้นมาใช้งาน และตรวจสอบความถูกต้องของวงจรก่อนที่จะนำไปผลิตเป็นชิพ งานวิจัยนี้ได้เสนอแนวทางการออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ โดยแบ่งการออกแบบออกเป็นสองส่วนคือ การออกแบบส่วนวงจรรางคู่โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ และการออกแบบส่วนวงจรตอบรับ โดยใช้การวิเคราะห์ฟังก์ชันภายใน และค่าความหน่วงประมาณของลุคอัพเทเบิ้ลในวงจรรางคู่ที่สังเคราะห์แล้ว

งานวิจัยนี้ได้ออกแบบและสร้างไมโครโปรเซสเซอร์แบบอสมวารบนเอฟพีจีเอเบอร์ XCV200Epq240-6 โดยได้ทำการออกแบบแต่ละวงจรย่อยด้วยภาษาวีเอชดีแอล แล้วนำไปสังเคราะห์ และสร้างเป็นโมดูลย่อยก่อน จากนั้นจึงนำทุกโมดูลมารวมกัน และสร้างเป็นไมโครโปรเซสเซอร์บนบนบอร์ดทดสอบวงจร จากการทดลองบนบอร์ดทดสอบ โดยหาค่ารากที่สองของอินพุตซึ่งเป็นค่าของสวิตช์ และแสดงผลลัพธ์ที่ได้บนตัวแสดงผลแบบ 7-Segment พบว่าไมโครโปรเซสเซอร์แบบอสมวารที่ออกแบบสามารถทำงานได้จริง

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต.....
สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา.....
ปีการศึกษา 2544 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม. .....

4270416221 : MAJOR COMPUTER ENGINEERING

KEY WORD: ASYNCHRONOUS CIRCUIT DESIGN / SCALABLE-DELAY-INSENSITIVE / FPGA / VHDL / MICROPROCESSOR

PHUNJAPA RUANGSINSUP : DESIGN OF 8-BIT SCALABLE-DELAY-INSENSITIVE MICROPROCESSOR USING FPGA. THESIS ADVISOR: ARTHIT THONGTAK,Ph.D., 110 pp. ISBN 974-03-0894-5.

This thesis proposes a design of 8-bit scalable-delay-insensitive microprocessor on FPGA, which can be the beginning step for asynchronous circuit implementation and verification before the fabrication. This research presents a design method of scalable-delay-insensitive model for asynchronous combinational circuit on FPGA. The designed circuit is divided into two parts: dual-rail circuit using Reduced-Ordered-Binary Decision Diagram (ROBDD) implementation and acknowledgement circuit using analysis function and estimated delay of lookup table in synthesized dual-rail circuit.

This research also presents a design and implementation of asynchronous microprocessor using FPGA no. XCV200Epq240-6. We use VHDL for designing the microprocessor by dividing to each circuit part as a module, and synthesize and implement individually. After that, we combine all of them and implement to be an asynchronous microprocessor on the test board. For the test board experiment of the square root computation, which the inputs applied from dip-switch's value and the output result show on 7-segment display, it shows that our designed asynchronous microprocessor is workable.

Department Computer Engineering Student's signature .....

Field of study Computer Engineering Advisor's signature .....

Academic year 2001 Co-advisor's signature .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดีด้วยความกรุณาอย่างดียิ่งของอาจารย์ ดร. อาทิตย์ ทองทักษ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งได้ให้คำแนะนำ และข้อคิดเห็นต่างๆ ในการทำวิจัยด้วยดีมาโดยตลอด

ขอขอบคุณ นายกวี วัฒนะวิรุณ ที่ให้คำแนะนำเกี่ยวกับทฤษฎีเบื้องต้นต่างๆ

ขอขอบคุณ นายพหล ศิริเหลืองทอง ที่ให้ความช่วยเหลือ ให้คำปรึกษาเกี่ยวกับการออกแบบ และการทดสอบการทำงาน

ขอขอบคุณห้องปฏิบัติการ Digital System Engineering Laboratory และห้องปฏิบัติการ Digital System Research Laboratory ที่เอื้อเฟื้อสถานที่ และอุปกรณ์ในการทำวิจัย

ท้ายที่สุดนี้ ผู้วิจัยขอกราบขอบพระคุณบิดา มารดา ที่สนับสนุน ห่วงใย และให้กำลังใจแก่ผู้วิจัยเสมอมา

ปัญจภา เรืองสินทรัพย์

22 มีนาคม 2545

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญภาพ

สารบัญภาพ (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

การสร้างวงจรแบบสมวาร (Synchronous Circuit) ให้ทำงานเร็วขึ้นโดยใช้สัญญาณนาฬิกาที่มีความถี่สูงควบคุมการทำงาน จะทำให้วงจรที่ได้มีการใช้พลังงานสูง และความหน่วงของสายสัญญาณทำให้เกิดการแกว่งของสัญญาณนาฬิกา (Clock Skew) [1] เป็นผลให้สัญญาณนาฬิกาที่กระจายไปยังส่วนต่างๆ ของวงจรเกิดขึ้นไม่พร้อมกัน จากปัญหาดังกล่าวจึงทำให้เกิดแนวความคิดในการสร้างวงจรแบบอสมวาร (Asynchronous Circuit) โดยใช้สัญญาณร้องขอ (Request Signal) และสัญญาณตอบรับ (Acknowledge Signal) ควบคุมการทำงาน ทำให้วงจรที่ได้มีความเร็วเป็นแบบเฉลี่ย (Average Case) และใช้พลังงานต่ำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เทคโนโลยีที่ก้าวหน้าในปัจจุบัน ทำให้สามารถสร้างวงจรดิจิตอลขึ้นมาใช้งานได้ง่าย และรวดเร็วด้วยการออกแบบวงจรที่ต้องการ แล้วนำไปสร้างเป็นวงจรลงบนอุปกรณ์ตรรกะแบบสั่งการได้ (Programmable Logic Device : PLD) ซึ่งแบ่งออกเป็นสามประเภทคือ เอสพีแอลดี (SPLD : Simple Programmable Logic Device), ซีพีแอลดี (CPLD : Complex Programmable Logic Device) และเอฟพีจีเอ (FPGA : Field Programmable Gate Array) เอสพีแอลดีเป็นอุปกรณ์ตรรกะแบบสั่งการได้ที่มีขนาดเล็ก บางครั้งอาจเรียกว่า พีเอแอล (PAL : Programmable Array Logic, Vantis) หรือจีเอแอล (GAL : Generic Array Logic, Lattice) โดยมีโครงสร้างภายในเป็นแบบแมโครเซลล์ (Macrocell) ซึ่งใช้สร้างเป็นตรรกะเชิงผสม (Combinational Logic) คือเกตแอนด์ (AND Gate),เกตออร์ (OR Gate) หรือเกตผกผัน (Inverter Gate) อุปกรณ์นี้จึงเหมาะสำหรับการนำมาสร้างเป็นวงจรขนาดเล็กที่มีความซับซ้อนน้อย เช่นวงจรถอดรหัสตำแหน่ง (Address Decoder)

ซีพีแอลดีมีโครงสร้างเป็นแบบแมโครเซลล์เช่นเดียวกับเอสพีแอลดี แต่ว่ามีขนาดใหญ่กว่าทำให้สามารถสร้างวงจรที่มีขนาดใหญ่ และซับซ้อนมากขึ้นได้ โดยภายในซีพีแอลดีจะประกอบด้วยลอจิกบล็อค (Logic Block) ซึ่งมีแมโครเซลล์จำนวน 4 - 16 แมโครเซลล์อยู่ภายในและเชื่อมต่อลอจิกบล็อคทุกบล็อคเข้าด้วยกันด้วยสวิตซ์เมตริกซ์ (Switch Matrix) โดยซีพีแอลดีที่มีใช้งานในปัจจุบันมีขนาดใหญ่สุดอยู่ที่ 512 แมโครเซลล์ หรือเท่ากับ 12,500 เกตเท่านั้น [2]

เอฟพีจีเอเป็นอุปกรณ์ตรรกะแบบสั่งการได้ที่มีโครงสร้างภายในแบ่งออกเป็นสามส่วนหลักได้แก่ ลอจิกบล็อค, ส่วนเชื่อมต่อภายใน (Interconnect Wiring) และไอโอบล็อก (I/O Block) ใน

ลอจิกบล็อคแต่ละตัวจะประกอบด้วยลุคอัพเทเบิ้ล (Lookup Table : LUT), ฟลิปฟล็อป (Flip-Flop) และอุปกรณ์รวมส่งสัญญาณ (Multiplexor) โดยเอฟพีจีเอที่มีใช้งานในปัจจุบันมีขนาดใหญ่สุดอยู่ที่ 104,832 ลอจิกบล็อค หรือเท่ากับ 8 ล้านเกต [2] จึงเหมาะสำหรับการนำมาสร้างเป็นวงจรขนาดใหญ่ และมีความซับซ้อนมาก เช่น ไมโครคอนโทรลเลอร์ หรือไมโครโปรเซสเซอร์

ดังนั้นนักออกแบบจึงนิยมนำเอฟพีจีเอมาใช้ในการสร้างวงจรต้นแบบอย่างรวดเร็ว (Rapid Prototyping) เพื่อนำมาใช้งานจริง และตรวจสอบความถูกต้องของวงจรก่อนที่จะนำไปผ่านกระบวนการผลิต (Fabrication) ออกมาเป็นชิพ (Chip) ด้วยการใช้ภาษาอธิบายฮาร์ดแวร์ (Hardware Description Language) เช่น ภาษาวีเอชดีแอล (VHDL) [3] เขียนอธิบายการทำงานของวงจรที่ต้องการ แล้วนำไปสร้างเป็นวงจรบนเอฟพีจีเอ และเนื่องจากซอฟต์แวร์ที่ช่วยด้านการออกแบบมีประสิทธิภาพมาก จึงสามารถออกแบบ แก้ไข และตรวจสอบความถูกต้องของวงจรด้วยการจำลองการทำงาน (Simulation) โดยทำงานทั้งหมดอยู่บนเครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer : PC) เมื่อวงจรมีความถูกต้องแล้วจึงโปรแกรมลงเอฟพีจีเอ เพื่อทดสอบการทำงานจริง และหากต้องการปรับปรุงแก้ไขวงจรเพิ่มเติมก็กลับไปทำตามขั้นตอนข้างต้นซ้ำใหม่ได้ไม่จำกัดจำนวนครั้ง ทำให้สามารถพัฒนาวงจรดิจิตอลที่ต้องการได้ง่าย รวดเร็ว และเสียค่าใช้จ่ายน้อย จากข้อดีดังกล่าวจึงได้เกิดแนวความคิดในการออกแบบวงจรแบบอสมวารบนเอฟพีจีเอ เพื่อตรวจสอบ และแก้ไขวงจรให้มีความถูกต้องก่อนนำไปผลิตเป็นชิพ ซึ่งจะทำให้สามารถลดค่าใช้จ่ายในด้านการตรวจสอบความถูกต้อง และค่าใช้จ่ายในการผลิตวงจรที่สูญเปล่าไป เนื่องจากผลิตวงจรที่ยังมีข้อผิดพลาดอยู่ รวมทั้งสามารถนำวงจรแบบอสมวารที่ออกแบบบนเอฟพีจีเอมาใช้งานจริงได้ด้วย

เนื่องจากวงจรแบบอสมวารไม่มีสัญญาณนาฬิกาควบคุมการทำงาน วงจรจึงตอบสนองต่อการเปลี่ยนระดับสัญญาณที่เกิดขึ้นทุกครั้ง การออกแบบวงจรแบบอสมวารจึงต้องมีการกำหนดแบบจำลองการทำงานสิ่งแวดล้อม (Environment Operation Model) [4] ซึ่งเป็นการกำหนดให้สิ่งแวดล้อมหรือวงจรทำหน้าที่ตรวจสอบการสิ้นสุดการเปลี่ยนระดับสัญญาณ เพื่อให้วงจรสามารถรับส่งข้อมูลได้ถูกต้อง รวมทั้งต้องมีการกำหนดแบบจำลองความหน่วง ซึ่งเป็นข้อกำหนดของความหน่วงที่ใช้ในการออกแบบ โดยในกลุ่มแบบจำลองความหน่วงชนิดมีขอบเขต (Bounded Delay Model) [1] และแบบจำลองความหน่วงแบบที่ไม่ขึ้นต่ออัตราเร็ว (Speed Independent : SI) [1] สามารถกำหนดค่าความหน่วงเกตและค่าความหน่วงของสายสัญญาณได้ การออกแบบวงจรให้มีการทำงานถูกต้องจึงทำได้ง่าย แต่การสร้างวงจรให้มีค่าความหน่วงเกตและสายตามที่กำหนดบางครั้งไม่สามารถทำได้ รวมทั้งวงจรจะทนต่อความแปรปรวนความหน่วงได้ต่ำ

ในกลุ่มแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วง (Delay Insensitive : DI) [1,5], แบบจำลองความหน่วงจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดเสมือน (Quasi-Delay-Insensitive : QDI) [4,6] และแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ (Scalable-Delay-Insensitive : SDI) [7] จะไม่กำหนดค่าความหน่วงของเกตและสาย การสร้างวงจรจึงทำได้ง่าย โดยกลุ่มผู้วิจัยในมหาวิทยาลัยโตเกียวได้พัฒนาไมโครโปรเซสเซอร์ TITAC-2 [7] โดยใช้แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งวงจรที่ออกแบบโดยใช้แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้จะมีขนาดเล็ก และสามารถทนต่อความแปรปรวนได้ตามสภาพความจริง คือไม่สูงเกินไปเช่นวงจรที่ไม่ไวต่อความหน่วงชนิดเสมือน และไม่ต่ำเกินไปเช่นวงจรที่ออกแบบโดยใช้กลุ่มแบบจำลองความหน่วงชนิดมีขอบเขต หรือแบบจำลองความหน่วงที่ไม่ขึ้นต่ออัตราเร็ว

การออกแบบไมโครโปรเซสเซอร์แบบอสมวาร จะมีหน่วยคำนวณและประมวลผลตรรกะ (Arithmetic Logic Unit : ALU) เป็นวงจรเชิงผสมที่สำคัญที่ต้องออกแบบตามข้อกำหนดของแบบจำลองความหน่วงที่ใช้ โดยในการออกแบบได้แบ่งวงจรออกเป็นสองส่วนคือ ส่วนวงจรที่ทำหน้าที่ประมวลตามฟังก์ชันตรรกะ เรียกว่า ส่วนวงจรรางคู่ (Dual-Rail Circuit) และส่วนวงจร ที่ทำหน้าที่ตรวจสอบการเปลี่ยนแปลงระดับสัญญาณภายในทั้งวงจร เรียกว่า ส่วนวงจรตอบรับ (Acknowledgement Circuit) สำหรับขั้นตอนการสร้างวงจรบนเอฟพีจีเอจะมีการแปลงวงจรที่ออกแบบให้เป็นวงจรย่อยที่สามารถนำลงแต่ละลุคอัพเทเบิ้ล เรียกว่า การทำดีคอมโพซิชั่น (Decomposition) หรือการแมปปิ้งวงจร (Technology Mapping) วงจรที่ผ่านการแปลงแล้วจะถูกนำไปกำหนดลงในแต่ละลุคอัพเทเบิ้ล และเชื่อมต่อสายสัญญาณเข้าด้วยกัน เรียกว่า การเพลสและเราต์ (Place and Route) ซึ่งเมื่อมีการเปลี่ยนแปลงแก้ไขวงจรที่ออกแบบ วงจรจะต้องผ่านขั้นตอนทั้งหมดใหม่ทุกครั้ง ดังนั้นในขั้นตอนการสร้างวงจรเชิงผสมแบบอสมวารบนเอฟพีจีเอ อาจทำให้สัญญาณภายในวงจรรางคู่บางเส้นมีค่าความหน่วงที่เปลี่ยนไป และส่งผลให้การตรวจสอบการเปลี่ยนระดับสัญญาณภายในของส่วนวงจรตอบรับผิดพลาดได้ จึงเป็นสิ่งที่ผู้ออกแบบจะต้องคำนึงถึง และหาแนวทางการออกแบบที่จะสามารถแก้ไขปัญหาดังกล่าว

งานวิจัยนี้จึงได้เสนอวิธีการออกแบบวงจรแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ให้สามารถสังเคราะห์ (Synthesis) และสร้างเป็นวงจรบนเอฟพีจีเอ โดยวงจรที่ออกแบบจะเป็นไมโครโปรเซสเซอร์ขนาด 8 บิต ที่สามารถนำมาใช้กับงานควบคุมขนาดเล็ก ที่ไม่ต้องการความเร็วสูง และมีหน่วยคำนวณ และประมวลผลตรรกะ ที่ออกแบบโดยใช้แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้

1.2 วัตถุประสงค์ของการวิจัย

1. เพื่อศึกษาความเป็นไปได้ในการสร้างวงจรแบบอสมวารโดยใช้เอฟพีจีเอ
2. เพื่อหาวิธีการออกแบบ และสร้างวงจรแบบอสมวารโดยใช้เอฟพีจีเอ
3. เพื่อเป็นแนวทางเริ่มต้นสำหรับการสร้างวงจรแบบอสมวารขึ้นมาใช้งาน และตรวจสอบความถูกต้องของวงจรก่อนที่จะนำไปผลิตเป็นชิพจริงโดยใช้เอฟพีจีเอ

1.3 ขอบเขตของการวิจัย

1. สร้างวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้โดยใช้โครงสร้างของเอฟพีจีเอ
2. ออกแบบไมโครโปรเซสเซอร์ 8 บิต ซึ่งภายในมีวงจรคำนวณและประมวลผลตรรกะที่มีแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ให้สามารถสังเคราะห์และสร้างเป็นวงจรลงเอฟพีจีเอได้ โดยมีชุดคำสั่งและโครงสร้างของไมโครโปรเซสเซอร์ตามที่กำหนดขึ้นเอง

1.4 ประโยชน์ที่ได้รับ

1. ได้แนวทางการออกแบบและสร้างวงจรอสมวารโดยใช้เอฟพีจีเอ
2. สามารถตรวจสอบความถูกต้องของวงจรอสมวารก่อนที่จะนำไปผลิตเป็นชิพจริง ทำให้สามารถลดค่าใช้จ่ายที่เกิดจากความผิดพลาดของวงจรลงได้
3. มีความรู้ และความเข้าใจถึงการออกแบบวงจรอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้
4. ได้ไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิตที่นำมาใช้งานจริงได้ในระดับหนึ่ง

1.5 ขั้นตอนดำเนินการวิจัย

1. ศึกษาการออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ รวมทั้งศึกษาวิธีการออกแบบไมโครโปรเซสเซอร์แบบอสมวาร
2. เสนอวิธีการออกแบบเอฟพีจีเอสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้
3. ออกแบบชุดคำสั่ง และโครงสร้างของไมโครโปรเซสเซอร์ 8 บิต ที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้

4. ออกแบบไมโครโปรเซสเซอร์แบบอสมวาร และตรวจสอบความถูกต้องของวงจรด้วยการจำลองการทำงาน
5. โปรแกรมไมโครโปรเซสเซอร์ที่ออกแบบลงเอฟพีจีเอ และทดสอบการทำงานกับบอร์ดทดสอบ
6. สรุปผลการวิจัย และจัดทำวิทยานิพนธ์

1.6 ลำดับขั้นตอนในการเสนอผลการวิจัย

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 6 บทดังนี้ บทที่ 1 เป็นบทนำซึ่งกล่าวถึงที่มาและความสำคัญของปัญหา รวมทั้งวัตถุประสงค์ของงานวิจัย บทที่ 2 สรุปแนวคิดและทฤษฎีที่เกี่ยวข้อง บทที่ 3 เสนอวิธีการออกแบบเอฟพีจีเอสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งนำไปใช้ในการออกแบบวงจรคำนวณและประมวลผลตรรกะ บทที่ 4 อธิบายสถาปัตยกรรมและชุดคำสั่งของไมโครโปรเซสเซอร์ รวมทั้งเสนอการออกแบบและสร้างไมโครโปรเซสเซอร์โดยใช้เอฟพีจีเอ บทที่ 5 เสนอผลการทดสอบ รวมทั้งประสิทธิภาพของไมโครโปรเซสเซอร์ที่ออกแบบ โดยเปรียบเทียบกับไมโครโปรเซสเซอร์แบบสมวารที่มีชุดคำสั่งและสถาปัตยกรรมเดียวกัน และบทที่ 6 เป็นบทที่สรุปผลการวิจัยและข้อเสนอแนะ

1.7 ผลงานที่ตีพิมพ์จากงานวิจัย

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์เป็นบทความทางวิชาการ ในหัวข้อ “การออกแบบเอฟพีจีเอสำหรับวงจรตอบรับในวงจรเชิงผสมแบบอสมวาร โดยการวิเคราะห์ฟังก์ชันและค่าความหน่วงประมาณของลุคอัพเทเบิ้ล” โดยปัญจภา เรืองสินทรัพย์ และอาทิตย์ ทองทักษ์ ในงานประชุมวิชาการ “The fifth National Computer Science and Engineering Conference (NCSEC’2001)” ซึ่งจัดโดยภาควิชาวิทยาศาสตร์คอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่ ณ โรงแรมโลตัสปางสวนแก้ว จังหวัดเชียงใหม่ ในระหว่างวันที่ 7-9 พฤศจิกายน 2544

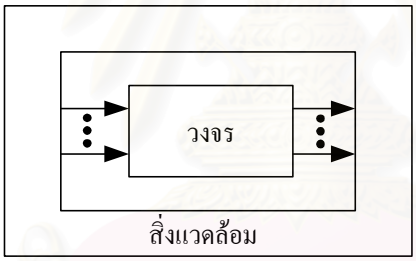
ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์เป็นบทความทางวิชาการ ในหัวข้อ “การออกแบบวงจรเชิงผสมแบบอสมวารโดยใช้โครงสร้างของเอฟพีจีเอ” โดยปัญจภา เรืองสินทรัพย์ และอาทิตย์ ทองทักษ์ ในงานประชุมวิชาการ “การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 24 (24th Electrical Engineering Conference: EECON24)” ซึ่งจัดโดยคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ณ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในระหว่างวันที่ 22-23 พฤศจิกายน 2544

บทที่ 2
แนวคิดและทฤษฎีที่เกี่ยวข้อง

บทนี้จะอธิบายถึงทฤษฎีต่างๆ ที่เกี่ยวข้องกับงานวิจัยซึ่งได้แก่ แบบจำลองการทำงานสิ่งแวดล้อม, แบบจำลองความหน่วง, อุปกรณ์ชนิดซี และการออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ

2.1 แบบจำลองการทำงานสิ่งแวดล้อม (Environment Operation Model)

ระบบอสมวารประกอบด้วยวงจร และสิ่งแวดล้อมที่ทำหน้าที่รับส่งอินพุตเอาต์พุตกับวงจร ดังแสดงในรูปที่ 2.1 โดยใช้แบบจำลองการทำงานสิ่งแวดล้อมเป็นตัวกำหนดว่าจะให้สิ่งแวดล้อมหรือวงจรทำหน้าที่ตรวจสอบการสิ้นสุดของการเปลี่ยนระดับสัญญาณในวงจร ด้วยการพิจารณาความหน่วงของสิ่งแวดล้อมเทียบกับความหน่วงของวงจร ซึ่งสามารถแบ่งได้เป็นสองสภาวะคือ สภาวะแวดล้อมมูลฐาน และสภาวะแวดล้อมรับเข้าส่งออก



รูปที่ 2.1 ระบบอสมวาร

2.1.1 สภาวะแวดล้อมมูลฐาน (Fundamental Mode Environment : FM Mode)

เมื่อความหน่วงของสิ่งแวดล้อมมีค่ามากกว่าความหน่วงของวงจร สามารถใช้ค่าความหน่วงการทำงานของสิ่งแวดล้อมรับประกันการสิ้นสุดของการเปลี่ยนระดับสัญญาณในวงจรได้ การออกแบบจึงกำหนดให้สิ่งแวดล้อมทำหน้าที่ตรวจสอบการสิ้นสุดของการเปลี่ยนระดับสัญญาณของวงจร โดยสิ่งแวดล้อมจะรับเอาต์พุตมาจากวงจร และส่งอินพุตชุดใหม่ไปให้วงจร ก็ต่อเมื่อทุกการเปลี่ยนระดับสัญญาณภายในวงจรเสร็จสิ้นแล้ว

2.1.2 สภาวะแวดล้อมรับเข้าส่งออก (Input-Output Mode Operation : IO Mode)

เมื่อความหน่วงของสิ่งแวดล้อมมีค่าน้อยกว่าความหน่วงของวงจร การออกแบบจะกำหนดให้วงจรสามารถตรวจสอบการสิ้นสุดการเปลี่ยนระดับสัญญาณภายในของวงจรได้เอง

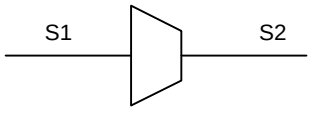
และจะให้เอาต์พุตออกมาก็ต่อเมื่อทุกการเปลี่ยนระดับสัญญาณภายในวงจรเสร็จสิ้นแล้ว สภาวะแวดล้อมนี้เป็นที่นิยมใช้ในการออกแบบ เนื่องจากเมื่อมีความแปรปรวนความหน่วงเกิดขึ้นในระบบ พบว่าวงจรที่ออกแบบในสภาวะแวดล้อมรับเข้าส่งออกสามารถประกันความถูกต้องในการทำงานได้ดีกว่าวงจรที่ออกแบบในสภาวะแวดล้อมมูลฐาน

2.2 **แบบจำลองความหน่วง** (Delay Model)

การออกแบบวงจรนอกเหนือจากการกำหนดแบบจำลองการทำงานสิ่งแวดล้อมแล้ว ยังต้องกำหนดแบบจำลองความหน่วง ซึ่งเป็นการกำหนดลักษณะความหน่วงของเกตและสายสัญญาณภายในวงจรที่ออกแบบ สามารถแบ่งได้เป็นสองกลุ่มหลัก คือ กลุ่มแบบจำลองความหน่วงที่มีขอบเขต ซึ่งประกอบด้วยแบบจำลองความหน่วงชนิดมีขอบเขตเพียงแบบเดียว และกลุ่มแบบจำลองความหน่วงที่ไม่มีขอบเขต ซึ่งประกอบด้วย แบบจำลองความหน่วงที่ไม่ขึ้นต่ออัตราเร็ว, แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วง, แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดเสมือน และแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ในที่นี้จะขออธิบายเฉพาะแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วง, แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดเสมือน และแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งเกี่ยวข้องกับงานวิจัยนี้เท่านั้น

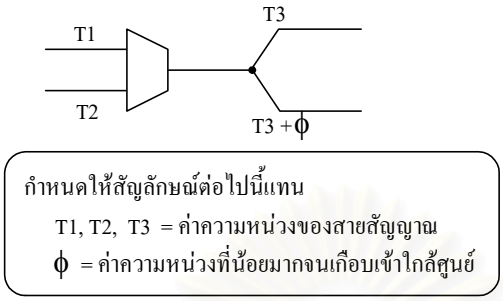
2.2.1 **แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วง** (Delay-Insensitive : DI)

แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วง [5] เป็นแบบจำลองความหน่วงที่ไม่กำหนดค่าความหน่วงเกตและค่าความหน่วงสายในการสร้างวงจรในระดับเลย์เอาต์ (Layout Circuit Implementation) แต่ทราบว่ามีค่าอยู่ในขอบเขตหนึ่งที่ไม่ใช่อนันต์ ดังนั้นเมื่อกำหนดให้วงจรมีการเปลี่ยนระดับสัญญาณ S1 เกิดก่อนการเปลี่ยนระดับสัญญาณ S2 การออกแบบวงจรจะต้องออกแบบให้สัญญาณ S1 เป็นอินพุตของเส้นทางส่งผ่านสัญญาณ (Signal Propagation Path) ของสัญญาณ S2 เท่านั้น ดังในรูปที่ 2.2 การออกแบบวงจรจึงสามารถใช้ได้เพียงเกตผกผันและอุปกรณ์ชนิดซี (C-Element) เท่านั้น



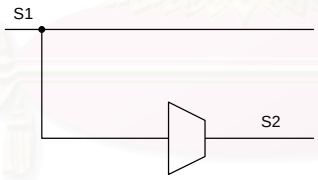
รูปที่ 2.2 การออกแบบวงจรที่ไม่ไวต่อความหน่วง เมื่อกำหนดให้การเปลี่ยนระดับสัญญาณ S1 เกิดก่อนการเปลี่ยนระดับสัญญาณ S2

2.2.2 แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดเสมือน (Quasi-Delay-Insensitive : QDI)



รูปที่ 2.3 แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดเสมือน

แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดเสมือน [4,6] เป็นแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงที่กำหนดให้ค่าความหน่วงในกิ่งของสาย (Fork Wire) ทุกกิ่งมีค่าเท่ากันดังรูปที่ 2.3 การออกแบบวงจรโดยใช้แบบจำลองความหน่วงนี้มีความซับซ้อนน้อยกว่าการใช้แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วง เนื่องจากไม่ต้องคำนึงถึงลำดับการเปลี่ยนระดับสัญญาณที่ทุกกิ่งของสาย โดยเมื่อกำหนดให้วงจรมีการเปลี่ยนระดับสัญญาณ S1 เกิดก่อนการเปลี่ยนระดับสัญญาณ S2 การออกแบบวงจรสามารถใช้กิ่งของสายสัญญาณ S1 กิ่งใดกิ่งหนึ่งเป็นอินพุตของเส้นทางส่งผ่านสัญญาณของสัญญาณ S2 ได้ดังรูปที่ 2.4



รูปที่ 2.4 การออกแบบวงจรที่ไม่ไวต่อความหน่วงชนิดเสมือน เมื่อกำหนดให้การเปลี่ยนระดับสัญญาณ S1 เกิดก่อนการเปลี่ยนระดับสัญญาณ S2

อย่างไรก็ตามการสร้างวงจรโดยใช้แบบจำลองดังกล่าวนั้นไม่สามารถทำได้ เนื่องจากในความเป็นจริงแล้วแต่ละกิ่งของสายสัญญาณมีค่าความหน่วงไม่เท่ากัน เมื่อทำการสร้างวงจรตามที่ออกแบบ และกิ่งของสายสัญญาณ S1 ที่เลือกมีความหน่วงน้อยกว่ากิ่งอื่นๆ อาจทำให้สัญญาณ S2 เกิดการเปลี่ยนแปลงก่อนสัญญาณ S1 ได้ ดังนั้นหากต้องการสร้างวงจรให้ใช้งานได้จริง ในการออกแบบจะต้องเลือกกิ่งของสายสัญญาณ S1 ทุกกิ่งเป็นอินพุตของเส้นทางส่งผ่านระดับสัญญาณ S2 ซึ่งจะทำให้วงจรที่สร้างได้มีขนาดใหญ่มาก

2.2.3 แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ (Scalable-Delay-Insensitive : SDI)

แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ [7] เป็นแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงที่มีการวิเคราะห์ความแปรปรวนความหน่วงในการออกแบบ โดยการกำหนดอัตราส่วนความแปรปรวนความหน่วงสูงสุด (Maximum Delay Variation Ratio : K) รวมทั้งประมาณค่าความหน่วงจริงสัมพัทธ์ (Actual Relative Delay : D_a) และค่าความหน่วงประมาณสัมพัทธ์ (Estimated Relative Delay : D_e) ระหว่างเส้นทางส่งผ่านสัญญาณสองเส้นทางในวงจร โดยมีอัตราส่วนความหน่วงสัมพัทธ์ (Relative Delay Ratio : R) ซึ่งเป็นค่าความคลาดเคลื่อนของการประมาณได้ไม่เกินอัตราส่วนความแปรปรวนความหน่วงสูงสุด

เมื่อต้องการให้วงจรมีการเปลี่ยนระดับสัญญาณ S1 เกิดก่อนการเปลี่ยนระดับสัญญาณ S2 สามารถทำได้โดยการออกแบบให้เส้นทางส่งผ่านสัญญาณ S1 และ S2 มีอินพุตเป็นกิ่งของสายสัญญาณ S และออกแบบให้วงจรมีค่าความหน่วงของการเปลี่ยนระดับสัญญาณถูกต้องภายใต้ความคลาดเคลื่อนของการประมาณค่าความหน่วงซึ่งไม่เกิน K เท่าดังในรูปที่ 2.5 โดยเมื่อกำหนดให้

D_e1, D_e2 = ค่าความหน่วงประมาณของเส้นทางส่งผ่านสัญญาณ S1 และ S2 ตามลำดับ

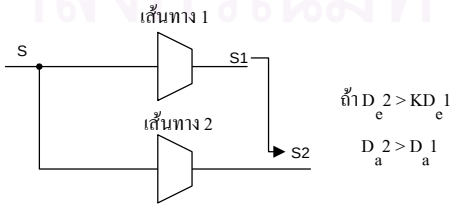
D_a1, D_a2 = ค่าความหน่วงจริงของเส้นทางส่งผ่านสัญญาณ S1 และ S2 ตามลำดับ

$D_a = D_a1 / D_a2$

$D_e = D_e1 / D_e2$

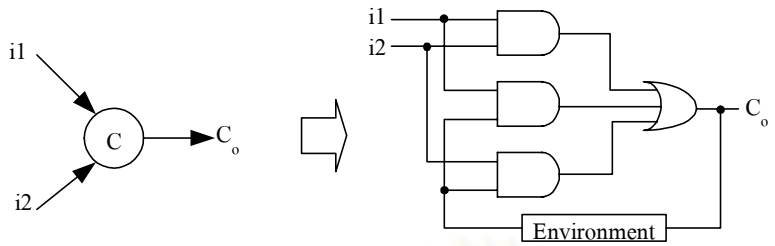
$R = D_a / D_e$ โดยที่ $1/K < R < K$

จะได้ว่า วงจรจะมีการเปลี่ยนระดับสัญญาณ S1 เกิดก่อนการเปลี่ยนระดับสัญญาณ S2 ถ้าหากวงจรมีค่าความหน่วงประมาณของการเปลี่ยนระดับสัญญาณ S2 มากกว่าค่าความหน่วงประมาณที่คลาดเคลื่อนสูงสุดของการเปลี่ยนระดับสัญญาณ S1 หรือ $D_e2 > KD_e1$ นั่นเอง



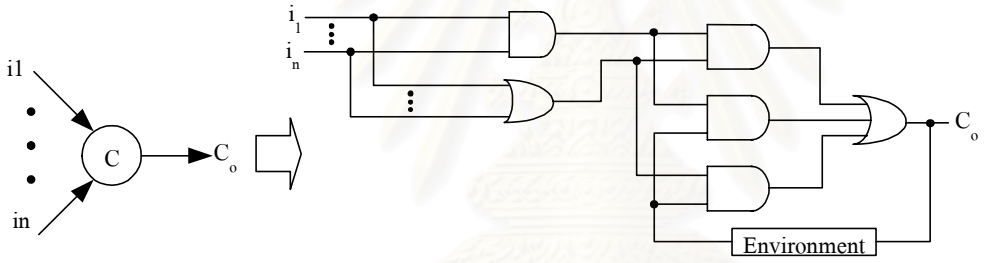
รูปที่ 2.5 การออกแบบวงจรที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ เมื่อกำหนดให้การเปลี่ยนระดับสัญญาณ S1 เกิดก่อนการเปลี่ยนระดับสัญญาณ S2

2.3 **อุปกรณ์ชนิดซี** (C-element)



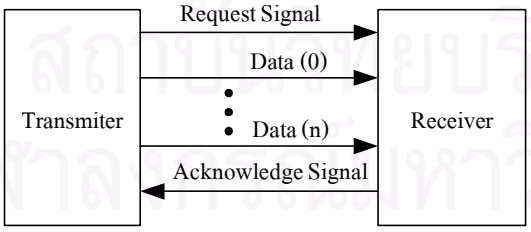
รูปที่ 2.6 การออกแบบอุปกรณ์ชนิดซีขนาดสองอินพุต

อุปกรณ์ชนิดซี [8] คืออุปกรณ์ที่ให้เอาต์พุตมีค่าเป็นหนึ่ง เมื่ออินพุตของอุปกรณ์ทุกอินพุตมีค่าเป็นหนึ่ง และให้เอาต์พุตมีค่าเป็นศูนย์ เมื่ออินพุตทุกอินพุตมีค่าเป็นศูนย์แล้วเท่านั้น ไม่เช่นนั้นจะคงค่าเอาต์พุตเดิมไว้ การออกแบบอุปกรณ์ชนิดซีขนาดสองอินพุต และขนาดอินพุตมากกว่าสองอินพุตแสดงดังรูปที่ 2.6และรูปที่ 2.7 ตามลำดับ



รูปที่ 2.7 การออกแบบอุปกรณ์ชนิดซีขนาดอินพุตมากกว่าสองอินพุตขึ้นไป

2.4 **รหัสรางคู่** (Dual-rail Code)

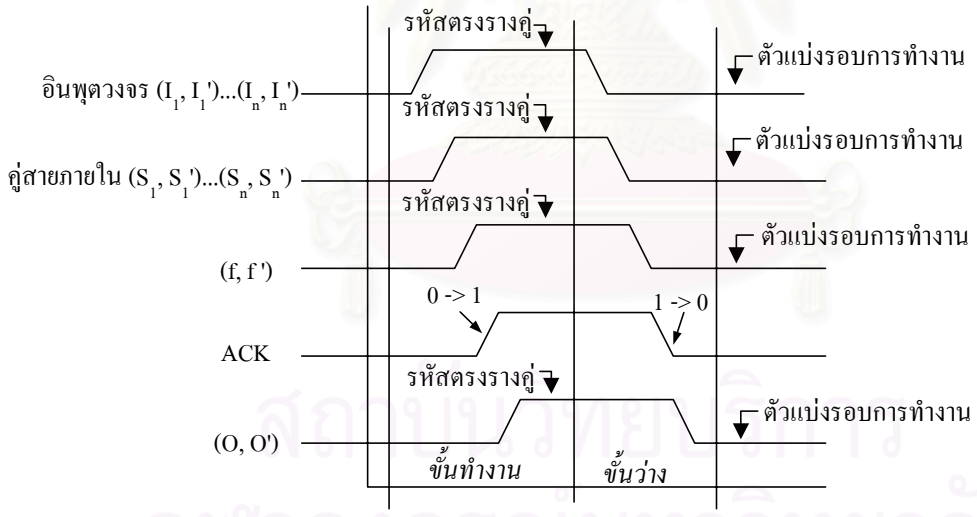


รูปที่ 2.8 การรับส่งข้อมูลด้วยวิธีข้อมูลรวมชุด

แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ จะไม่มีการกำหนดความหน่วงของเกตและสาย เมื่อวงจรรับส่งข้อมูลด้วยวิธีข้อมูลรวมชุด (Bundle Data) [1] ดังรูปที่ 2.8 และความหน่วงของสายสัญญาณร้องขอมีค่าน้อยกว่าความหน่วงของสายสัญญาณข้อมูล

ฝ่ายรับอาจจะได้รับสัญญาณร้องขอก่อนที่สัญญาณข้อมูลชุดใหม่จะไปถึง ทำให้ได้รับข้อมูลที่ผิดพลาด ดังนั้นข้อมูลจึงต้องสามารถบอกการมาถึงให้กับฝ่ายรับได้โดยไม่ต้องใช้สายสัญญาณร้องขอ แต่การรับส่งข้อมูลด้วยสายสัญญาณหนึ่งเส้นต่อข้อมูลขนาดหนึ่งบิต ทำให้ไม่สามารถระบุได้ว่าระดับสัญญาณที่คงที่ในช่วงเวลาหนึ่งแทนข้อมูลกี่ชุด จึงต้องเข้ารหัสข้อมูลด้วยรหัสรางคู่

รหัสรางคู่ [6] เป็นรหัสที่มีคุณสมบัติประสานจังหวะในตัว (Self Synchronizing) คือ สามารถใช้ได้ทั้งการรับส่งค่าระดับสัญญาณ และแบ่งรอบการทำงานของวงจร โดยจะแทนสายสัญญาณข้อมูลขนาดหนึ่งบิตที่มีค่าตรรกะศูนย์ และค่าตรรกะหนึ่งด้วยสายสัญญาณคู่ (X, X') ที่มีค่าเป็น (0,1) และ (1,0) ตามลำดับ เรียกว่ารหัสตรงรางคู่ (2-rail Codeword) และแบ่งรอบการทำงานของวงจรโดยให้สายสัญญาณคู่ (X,X') มีค่าเป็น (0,0) เรียกว่าตัวแบ่งรอบการทำงาน (Spacer) ทำให้วงจรมีการทำงานเป็นแบบรางคู่สองขั้นชนิดกลับสู่ศูนย์ (2-rail 2-phase return-to-zero operation) คือมีการทำงานในขั้นทำงาน (Working Phase) และขั้นว่าง (Idle Phase) สลับกัน ซึ่งวงจรเชิงผสมแบบอสมวารที่กำหนดแบบจำลองการทำงานสิ่งแวดล้อมเป็นแบบรับเข้าส่งออก และมีการทำงานเป็นแบบรางคู่สองขั้นชนิดกลับสู่ศูนย์ จะมีลักษณะการเปลี่ยนระดับสัญญาณดังในรูปที่ 2.9 โดยมีลักษณะการทำงานดังนี้คือ



รูปที่ 2.9 ลักษณะการเปลี่ยนระดับสัญญาณของวงจรเชิงผสมแบบอสมวาร ในการทำงานรางคู่แบบสองขั้นชนิดกลับสู่ศูนย์

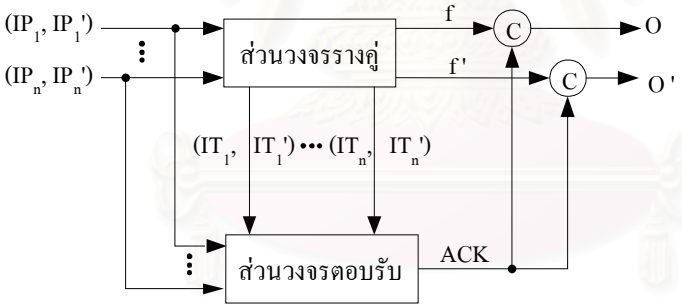
<u>ขั้นทำงาน</u> ส่วนวงจรรางคู่จะได้รับอินพุตของวงจร (I_1,I_1')...(I_n,I_n') เป็นรหัสตรงรางคู่ ทำให้คู่สายภายใน (S_1,S_1')...(S_n,S_n') บางเส้นเกิดการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1 และได้เอาต์พุตของวงจรรางคู่ (f_1,f_1')...(f_n,f_n') เป็นรหัสตรงรางคู่ เมื่อส่วนวงจรตอบรับตรวจสอบพบว่าการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1 ภายในส่วนวงจรรางคู่สิ้นสุดแล้วก็จะให้สัญญาณแสดงความ

บริบูรณ์ (Completion Signal : ACK) เปลี่ยนระดับสัญญาณจาก 0 เป็น 1 ส่งผลให้เอาต์พุตของวงจร $(O_1,O_1')\ldots(O_n,O_n')$ มีค่าเป็นรหัสตรงรางคู่

ขั้นว่าง ส่วนวงจรรางคู่จะได้รับอินพุตของวงจรจากสิ่งแวดล้อมเป็นตัวแบ่งรอบการทำงาน ทำให้คู่สายภายในที่มีค่าระดับสัญญาณเป็น 1 เกิดการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 และได้เอาต์พุตของวงจรรางคู่เป็นตัวแบ่งรอบการทำงาน เมื่อส่วนวงจรตอบรับตรวจสอบพบว่า สายภายในส่วนวงจรรางคู่ที่มีค่าระดับสัญญาณเป็น 1 ในขั้นทำงานเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 เสร็จสิ้นแล้ว สัญญาณแสดงความบริบูรณ์จะเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ส่งผลให้เอาต์พุตของวงจรมีค่าเป็นตัวแบ่งรอบการทำงาน

2.5 การออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ

เมื่อกำหนดแบบจำลองการทำงานสิ่งแวดล้อมเป็นแบบรับเข้าส่งออก วงจรเชิงผสมแบบอสมวารสามารถแบ่งได้เป็นวงจรย่อยสองส่วนคือ ส่วนวงจรรางคู่ ที่ทำการประมวลผลตามฟังก์ชันตรรกะ และส่วนวงจรตอบรับ ที่ทำหน้าที่ตรวจสอบการสิ้นสุดการเปลี่ยนระดับสัญญาณในวงจรดังรูปที่ 2.10



รูปที่ 2.10 โครงสร้างวงจรเชิงผสมแบบอสมวาร เมื่อกำหนดแบบจำลองการทำงานสิ่งแวดล้อมเป็นแบบรับเข้าส่งออก

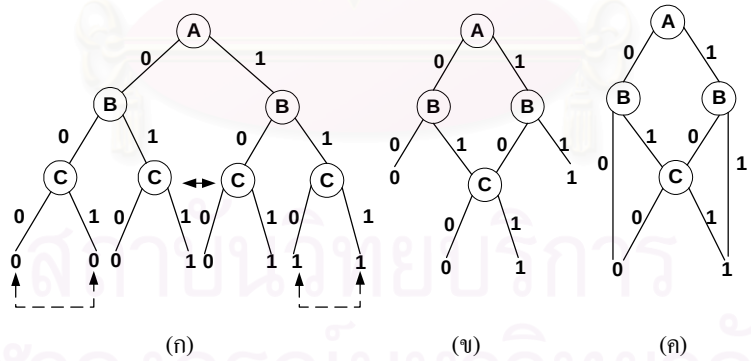
การออกแบบวงจรที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้และมีการทำงานเป็นแบบรางคู่สองขั้นชนิดกลับสู่ศูนย์ จะเริ่มจากการออกแบบส่วนวงจรรางคู่โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ จากนั้นทำการวิเคราะห์ลักษณะการส่งผ่านระดับสัญญาณภายในส่วนวงจรรางคู่ แล้วจึงทำการออกแบบส่วนวงจรตอบรับโดยการเลือกกลุ่มสายที่สามารถครอบคลุมทุกการเปลี่ยนระดับสัญญาณภายในส่วนวงจรรางคู่ทั้งในขั้นทำงานและขั้นว่าง และใช้เกตออร์ตรวจสอบการสิ้นสุดการเปลี่ยนระดับสัญญาณภายในวงจร จากนั้นจึงทำการรวมเอาต์พุตของทั้งสองส่วนเป็นเอาต์พุตของวงจรโดยใช้อุปกรณ์ชนิดซี

2.5.1 แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ (Reduced-Ordered-Binary Decision Diagram : ROBDD)

แผนภาพตัดสินใจแบบทวิภาค (Binary Decision Diagram) [9] เป็นแผนภาพที่ใช้อธิบายการทำงานของฟังก์ชันตรรกะ เพื่อช่วยในการออกแบบและสังเคราะห์วงจรที่มีขนาดใหญ่ โดยแผนภาพตัดสินใจแบบทวิภาคที่มีการกำหนดลำดับตัวแปรเรียกว่า แผนภาพตัดสินใจแบบทวิภาคชนิดมีอันดับ (Ordered-BDD : OBDD) และแผนภาพตัดสินใจแบบทวิภาคชนิดมีอันดับที่สามารถลดขนาดของแผนภาพลงได้จะเรียกว่า แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ (Reduced-Ordered-BDD : ROBDD)

รูปที่ 2.11 แสดงตัวอย่างการสร้างแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับโดยจากตารางค่าความจริง (Truth Table) ทำการสร้างแผนภาพตัดสินใจโดยแทนตัวแปรอินพุตด้วยบัพ (Node) แล้วแทนค่าตรรกะศูนย์และค่าตรรกะหนึ่งของตัวแปรนั้นๆ ด้วยกิ่งด้านซ้ายและด้านขวาตามลำดับดังรูปที่ 2.11(ก) จากนั้นจึงทำการลดขนาดของแผนภาพโดยการนำบัพที่มีกิ่งทางด้านซ้ายและด้านขวาเหมือนกันออกดังแสดงในรูปที่ 2.11(ข) และ 2.11(ค) จากแผนภาพจะเห็นได้ว่า ในอินพุตแต่ละแบบจะมีเส้นเชื่อมจากบัพบนสุดไปจนถึงบัพล่างสุดเพียงหนึ่งเส้นเชื่อมเท่านั้นที่ได้เอาต์พุตเป็นค่าตรรกะศูนย์หรือค่าตรรกะหนึ่ง และสามารถเขียนฟังก์ชันของตัวแปรอินพุตได้ในรูปผลรวมของผลคูณ (Sum of Product)

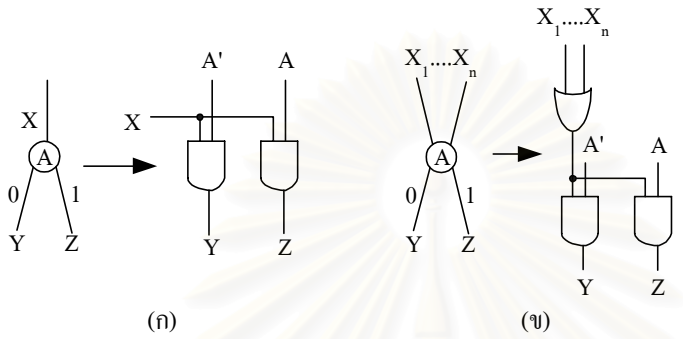
I/P			O/P
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



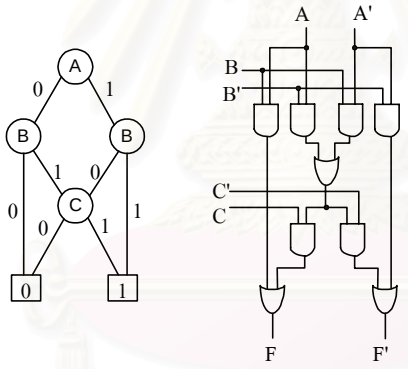
รูปที่ 2.11 การสร้างแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ สำหรับฟังก์ชัน F= AB + (AB'+A'B)C และ F'= A'B' + (AB'+A'B)C'

2.5.2 การออกแบบส่วนวงจรรางคู่

เมื่อทำการสร้างแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับแล้ว จึงทำการแปลงแผนภาพตัดสินใจให้เป็นส่วนวงจรรางคู่ โดยแทนแต่ละกิ่งของบัพด้วยเกตแอนด์ ดังรูปที่ 2.12(ก) และรวมอินพุตของบัพทุกอินพุตเข้าด้วยกันด้วยเกตออร์ดังรูปที่ 2.12(ข)



รูปที่ 2.12 การแปลงแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับเป็นวงจรรางคู่



รูปที่ 2.13 ตัวอย่างการออกแบบส่วนวงจรรางคู่สำหรับฟังก์ชัน F= AB + (AB'+A'B)C และ F'= A'B' + (AB'+A'B)C โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ

รูปที่ 2.13 แสดงตัวอย่างการออกแบบส่วนวงจรรางคู่ของฟังก์ชัน F= AB + (AB'+A'B)C และ F'= A'B' + (AB'+A'B)C' โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ จากรูปจะเห็นได้ว่า ส่วนวงจรรางคู่จะประกอบด้วยเกตแอนด์ซึ่งมีการจัดเรียงกันเป็นเส้นทางเทียบเท่ากับเส้นเชื่อมในแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ เรียกว่า เส้นทางต่อเชื่อมเกตแอนด์ และเกตออร์ซึ่งใช้รวมเส้นทางต่อเชื่อมเกตแอนด์ที่มีมากกว่าหนึ่ง โดยมีการเปลี่ยนระดับสัญญาณของเกตออร์ และเส้นทางต่อเชื่อมเกตแอนด์เป็นดังนี้คือ

ขั้นทำงาน จะมีเส้นทางต่อเชื่อมเกตแอนด์เพียงหนึ่งเส้นทางที่สายทุกเส้นในเส้นทางมีการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1 และทำให้เอาต์พุตของวงจรรางคู่มีค่าเป็นรหัสตรงรางคู่ สำหรับในเกตออร์ที่เอาต์พุตมีการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1 เกตออร์นั้นจะมีอินพุตซึ่งเป็นเส้นทางต่อเชื่อมเกตแอนด์เพียงหนึ่งอินพุตที่สายทุกเส้นในเส้นทางมีการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1

ขั้นว่าง สายสัญญาณที่มีค่าระดับสัญญาณ 1 ในขั้นทำงาน จะมีการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 และส่งผลให้เอาต์พุตของวงจรรางคู่มีค่าเป็นตัวแบ่งรอบการทำงาน

2.5.3 การออกแบบส่วนวงจรตอบรับ

การออกแบบส่วนวงจรตอบรับ [10] เริ่มจากการเลือกกลุ่มสายภายในส่วนวงจรรางคู่ที่สามารถครอบคลุมทุกการเปลี่ยนระดับสัญญาณทั้งในขั้นทำงานและขั้นว่าง แล้วจึงใช้เกตออร์รวมสายสัญญาณทั้งหมดที่เลือก โดยจะต้องสร้างสัญญาณแสดงความบริบูรณ์ให้มีค่าความหน่วงการเปลี่ยนระดับสัญญาณเป็น K เท่าของค่าความหน่วงการเปลี่ยนระดับสัญญาณสุดท้ายภายในส่วนวงจรรางคู่ เพื่อให้วงจรเชิงผสมมีความทนต่อความแปรปรวนความหน่วงที่อัตราส่วนความแปรปรวนความหน่วงสูงสุด (K)

เมื่อพิจารณาโครงสร้างของส่วนวงจรรางคู่ พบว่าในแต่ละรอบการทำงาน เกตออร์จะมีอินพุตเพียงหนึ่งอินพุตที่มีการเปลี่ยนระดับจาก 0 เป็น 1 และเอาต์พุตของเกตแอนด์จะเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ทันทีที่อินพุตของเกตแอนด์เส้นใดเส้นหนึ่งเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ดังนั้นค่าความหน่วงของสายอินพุตที่ต่ำสุดจึงมีผลต่อการเปลี่ยนระดับสัญญาณที่เอาต์พุตเกต การหาค่าความหน่วงการเปลี่ยนระดับสัญญาณที่จุดเอาต์พุตของเกต และที่จุดปลายของสายจึงเป็นดังสมการที่ 2.1 และ 2.2 ตามลำดับ

กำหนดให้

G = ค่าความหน่วงเกต

W = ค่าความหน่วงสายที่เชื่อมระหว่างจุดสองจุด

$\tau(x)$ = ค่าความหน่วงการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ที่จุด x

$$\tau(\text{เอาต์พุตเกต}) = \tau_{\text{ต่ำสุด}}(\text{สายอินพุต}) + G \qquad (2.1)$$

$$\tau(\text{ปลายสาย}) = \tau(\text{เอาต์พุตเกต}) + W \qquad (2.2)$$

เมื่อพิจารณาแต่ละเกตออร์ ในเกตออร์ที่มีค่าเอาต์พุตเป็น 1 จะมีเส้นทางต่อเชื่อมเกตแอนด์ที่เป็นอินพุตเพียงหนึ่งเส้นทางที่สายทุกเส้นในเส้นทางมีค่าระดับสัญญาณ 1 และสามารถใช้สายหลักของการต่อเชื่อมเกตแอนด์ ซึ่งเป็นสายที่มีค่าความหน่วงการเปลี่ยนระดับสัญญาณจาก 1

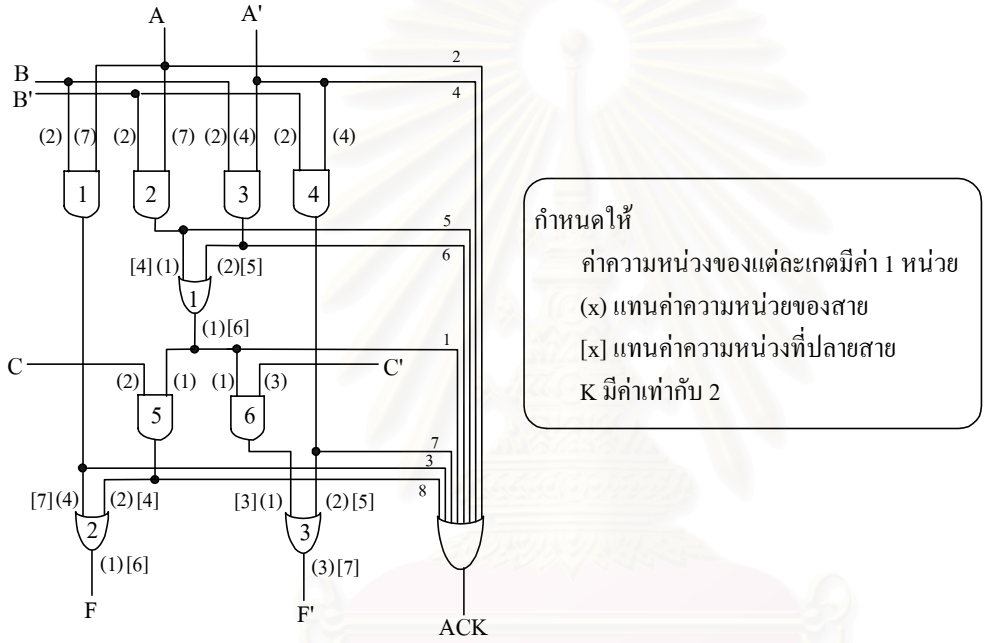
เป็น 0 สูงสุดรับประกันการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของสายสัญญาณทุกสายในเส้นทางได้ และหากค่าความหน่วงการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของสายเอาต์พุตของเกตออร์มีค่าสูงกว่าค่าความหน่วงการเปลี่ยนระดับสัญญาณของสายหลักของการต่อเชื่อมเกตแอนด์ สายเอาต์พุตของเกตออร์จะสามารถรับประกันการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของสายสัญญาณทุกสายในเส้นทางต่อเชื่อมเกตแอนด์ที่มีค่าระดับสัญญาณเป็น 1 ได้เช่นกัน

แต่ในเส้นทางต่อเชื่อมเกตแอนด์ที่มีค่าระดับสัญญาณเป็น 0 พบว่าอาจมีสายบางเส้นในเส้นทางที่มีค่าระดับสัญญาณเป็น 1 และหากสายหลักของการต่อเชื่อมเกตแอนด์ รวมทั้งสายเอาต์พุตของเกตออร์ที่พิจารณาเลือกไว้ในตอนแรกทุกสายมีค่าระดับสัญญาณเป็น 0 กลุ่มสายเหล่านี้จะไม่สามารถรับประกันการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของสายที่มีค่าระดับสัญญาณเป็น 1 ในเส้นทางนั้นได้ ดังนั้นการเลือกสายสำหรับแต่ละเกตออร์จึงเลือกสายหลักของการต่อเชื่อมเกตออร์ ซึ่งเป็นสายที่มีค่าความหน่วงการเปลี่ยนระดับสัญญาณสูงสุดระหว่าง สายหลักของการต่อเชื่อมเกตแอนด์ที่มีค่าความหน่วงต่ำสุด กับสายเอาต์พุตของเกตออร์ ร่วมกับสายในเส้นทางต่อเชื่อมเกตแอนด์ที่มีค่าความหน่วงมากกว่า

ในกรณีที่เอาต์พุตของเกตออร์มีค่าเป็น 0 พบว่าอาจมีสายบางเส้นในเส้นทางต่อเชื่อมเกตแอนด์ที่มีค่าระดับสัญญาณเป็น 1 และหากกลุ่มสายที่เลือกไว้ดังข้างต้นมีค่าระดับสัญญาณเป็น 0 ทุกสาย จะไม่สามารถรับประกันการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของสายเหล่านั้นได้ ดังนั้นเมื่อพิจารณารวมทุกเกตออร์ในวงจรรางคู่ การเลือกสายเพื่อให้วงจรสามารถทนความแปรปรวนความหน่วงที่อัตราส่วนความแปรปรวนความหน่วงสูงสุด จึงเลือกสายหลักของวงจร ซึ่งเป็นสายหลักของการต่อเชื่อมเกตออร์ที่มีค่าความหน่วงต่ำสุด ร่วมกับสายในเส้นทางต่อเชื่อมเกตแอนด์ทุกเส้นทางของวงจรที่มีค่าความหน่วงการเปลี่ยนระดับสัญญาณมากกว่าค่าความหน่วงของสายหลักของวงจรหาร์ด้วยอัตราส่วนความแปรปรวนสูงสุด ตัวอย่างการเปรียบเทียบค่าความหน่วงในการเลือกสายจากส่วนวงจรรางคู่แสดงดังรูปที่ 2.14

จากรูปเมื่อกำหนดให้อัตราส่วนความแปรปรวนความหน่วงสูงสุด (K) มีค่าเท่ากับ 2 และค่าความหน่วงของเกตมีค่า 1 หน่วย จะได้ว่าค่าความหน่วงที่จุดเอาต์พุตของเกตแอนด์ตัวที่ 2 จะมีค่าเท่ากับ 3 หน่วย ซึ่งเป็นค่าความหน่วงของสายอินพุตของเกตแอนด์ที่มีค่าต่ำสุดรวมกับค่าความหน่วงเกต และค่าความหน่วงที่ปลายสายเอาต์พุตของเกตแอนด์ตัวที่ 2 จะมีค่าเท่ากับ 4 หน่วย ซึ่งเป็นค่าความหน่วงที่จุดเอาต์พุตรวมกับค่าความหน่วงสายเอาต์พุตนั่นเอง จากหลักการเปรียบเทียบค่าความหน่วงดังกล่าว ค่าความหน่วงที่ปลายสายเอาต์พุตของเกตแอนด์ตัวที่ 3 จึงมีค่าเท่ากับ 5 หน่วย ดังนั้นมื่อพิจารณาที่เกตออร์ตัวที่ 1 จะได้ว่าสายหลักของเส้นทางต่อเชื่อมเกต

แอนด์ที่ต่อเป็นอินพุตของเกตออร์ คือสายอินพุต A กับสายเอาต์พุตของเกตแอนด์ตัวที่ 3 ซึ่งมีค่าความหน่วงที่ปลายสายเท่ากับ 7 และ 5 หน่วยตามลำดับ และเมื่อพิจารณาร่วมกับสายเอาต์พุตของเกตออร์ตัวที่ 1 ซึ่งมีค่าความหน่วงที่ปลายสายเท่ากับ 6 หน่วย (4+1+1) จึงเลือกสายเอาต์พุตของเกตออร์ (สายที่ 1 ที่ถูกเลือก) เป็นสายหลักของการต่อเชื่อมเกตออร์สำหรับเกตออร์ตัวที่ 1 และเลือกสายอินพุต A (สายที่ 2 ที่ถูกเลือก) เพิ่มอีกสาย เนื่องจากเป็นสายในเส้นทางต่อเชื่อมเกตแอนด์ที่มีค่าความหน่วงการเปลี่ยนระดับสัญญาณมากกว่าค่าความหน่วงของสายหลักของการต่อเชื่อมเกตออร์



รูปที่ 2.14 การเปรียบเทียบค่าความหน่วงในการเลือกสายจากส่วนวงจรรางคู่

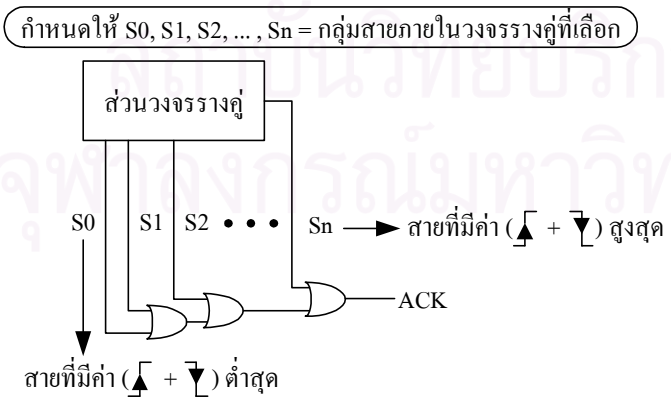
เมื่อพิจารณาที่เกตออร์ตัวที่ 2 และเกตออร์ตัวที่ 3 จะได้สายหลักของการต่อเชื่อมเกตออร์ คือสายเอาต์พุตของเกตแอนด์ตัวที่ 1 (สายที่ 3 ที่ถูกเลือก) และสายเอาต์พุตของเกตออร์ตัวที่ 3 ตามลำดับ โดยที่ทั้งสองสายมีค่าความหน่วงที่ปลายสายเท่ากับ 7 หน่วยเท่ากัน แต่สายเอาต์พุตของเกตออร์ตัวที่ 3 จะไม่ถูกเลือก เนื่องจากเป็นสายเอาต์พุตของวงจรรางคู่ เมื่อนำค่าความหน่วงของสายหลักของการต่อเชื่อมเกตออร์ทั้งหมดมาพิจารณาจะได้สายหลักของวงจรคือ สายเอาต์พุตของเกตออร์ตัวที่ 1 ซึ่งมีค่าความหน่วงที่ปลายสายเท่ากับ 6 หน่วย และเลือกสายเพิ่มเติมได้เป็นสายอินพุต A' (สายที่ 4 ที่ถูกเลือก), สายเอาต์พุตของเกตแอนด์ตัวที่ 2 (สายที่ 5 ที่ถูกเลือก), สายเอาต์พุตของเกตแอนด์ตัวที่ 3 (สายที่ 6 ที่ถูกเลือก), สายเอาต์พุตของเกตแอนด์ตัวที่ 4 (สายที่ 7 ที่ถูกเลือก) และสายเอาต์พุตของเกตแอนด์ตัวที่ 5 (สายที่ 8 ที่ถูกเลือก) โดยสายทั้งหมดเป็นสายใน

เส้นทางต่อเชื่อมเกตแอนด์ที่มีค่าความหน่วงที่ปลายสายมากกว่าค่าความหน่วงของสายหลักของวงจรหารด้วย K นั่นเอง

สรุปขั้นตอนการเลือกสายคือ

1. ไม่เลือกคู่สายเอาต์พุตของวงจรรางคู่ เนื่องจากคู่สายดังกล่าวจะต่อเป็นอินพุตของอุปกรณ์ชนิดซี ซึ่งสามารถตรวจสอบการสิ้นสุดการเปลี่ยนระดับสัญญาณได้
2. พิจารณาที่แต่ละเกตออร์ตั้งแต่อินพุตของเส้นทางต่อเชื่อมเกตแอนด์ที่เป็นอินพุตของเกตออร์ไปจนถึงเอาต์พุตของเกตออร์
   a. หา*สายหลักของการต่อเชื่อมเกตแอนด์* ซึ่งเป็นสายที่มี $\downarrow_{\text{สูงสุด}}$ ในแต่ละเส้นทางต่อเชื่อมเกตแอนด์
   b. ทำการเลือกสายหลักของการต่อเชื่อมเกตออร์ ซึ่งเป็นสายที่มี $\downarrow_{\text{สูงสุด}}$ ระหว่างสายหลักของการต่อเชื่อมเกตแอนด์ที่มี $\downarrow_{\text{ต่ำสุด}}$ กับสายเอาต์พุตของเกตออร์
   c. ทำการเลือกสายในเส้นทางต่อเชื่อมเกตแอนด์ที่มี $\downarrow > \downarrow$ สายหลักของการต่อเชื่อมเกตออร์
3. พิจารณารวมทั้งวงจร
   a. หา*สายหลักของวงจร* ซึ่งเป็นสายหลักของการต่อเชื่อมเกตออร์ที่มี $\downarrow_{\text{ต่ำสุด}}$
   b. <u>เลือกสายในเส้นทางต่อเชื่อมเกตแอนด์</u>ที่มี $\downarrow > \downarrow$ สายหลักของวงจร / K
4. ถ้าเลือกสายเอาต์พุตของเกตออร์แล้วให้นำสายอินพุตของเกตออร์ที่ถูกเลือกไว้ออก

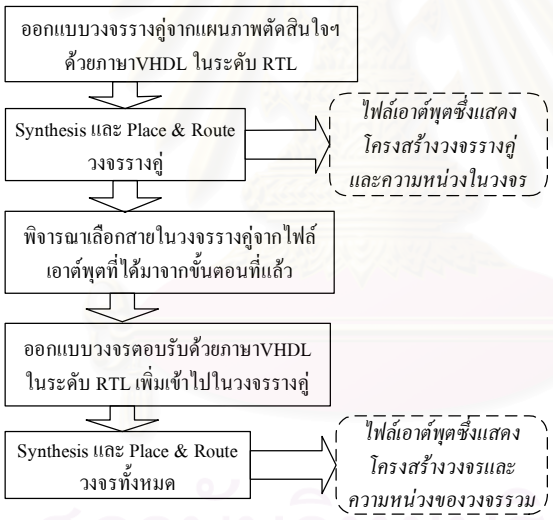
จากนั้นจึงนำกลุ่มสายที่เลือกมาเรียงต่อกันจากสายที่มีค่าความหน่วงต่ำสุดไปสูงสุด โดยจะต้องทำการประมาณค่าความหน่วงที่เหมาะสมของแต่ละเกตออร์ และทำการจัดเรียงเกตออร์ดังรูปที่ 2.15



รูปที่ 2.15 การจัดเรียงเกตออร์ในส่วนวงจรตอบรับ

บทที่ 3
การออกแบบเอฟพีจีเอสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้

การออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอของบริษัท Xilinx เบอร์ XCV200Epq240-6 เริ่มจากการใช้ภาษาวีเอชดีแอล เขียนอธิบายการทำงานของส่วนวงจรรางคู่ แล้วนำไปสังเคราะห์ และสร้างเป็นส่วนวงจรรางคู่ที่สามารถนำไปโปรแกรมลงบนเอฟพีจีเอ หลังจากนั้นจึงทำการออกแบบส่วนวงจรตอบรับเพิ่มเติมเข้าไป แล้วนำไปสังเคราะห์ และสร้างเป็นวงจรเชิงผสมแบบอสมวารที่สามารถนำไปโปรแกรมลงบนเอฟพีจีเออีกครั้งดังรูปที่ 3.1 ซึ่งเอฟพีจีเอและซอฟต์แวร์ที่ช่วยในด้านการออกแบบที่มีอยู่ในปัจจุบันนั้นยังไม่สนับสนุนการสร้างวงจรแบบอสมวาร ทำให้มีข้อจำกัดมากมายในการออกแบบ ในบทนี้จึงนำเสนอแนวทางการออกแบบส่วนวงจรรางคู่บนเอฟพีจีเอ และวิธีการออกแบบส่วนวงจรตอบรับ สำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ

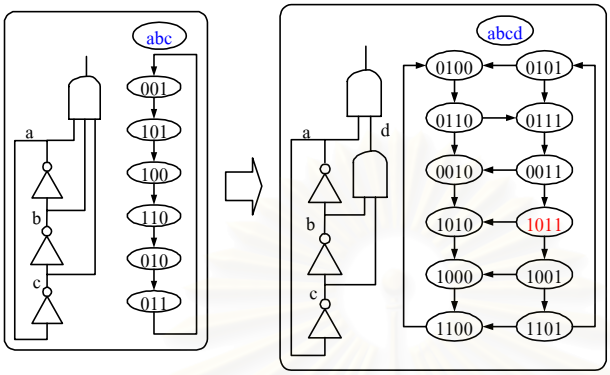


รูปที่ 3.1 ขั้นตอนการออกแบบเอฟพีจีเอสำหรับวงจรเชิงผสมแบบอสมวาร

3.1 การออกแบบส่วนวงจรรางคู่บนเอฟพีจีเอ

เนื่องจากโครงสร้างภายในของเอฟพีจีเอประกอบด้วยลุคอัพเทเบิ้ลขนาด 4 อินพุต จึงต้องมีการแปลงวงจรที่ออกแบบด้วยภาษาอธิบายฮาร์ดแวร์ให้อยู่ในรูปของกลุ่มฟังก์ชันที่สามารถนำไปกำหนดให้แต่ละลุคอัพเทเบิ้ลที่อยู่ภายในเอฟพีจีเอ เรียกว่า การดีคอมโพซิชั่น จากการศึกษาพบว่าวงจรแบบอสมวารที่ถูกดีคอมโพซิชั่นแล้วอาจเกิดการทำงานที่ผิดพลาดได้ [11] รูปที่ 3.2 แสดงตัวอย่างการดีคอมโพซิชั่นวงจร Ring Oscillator ซึ่งเป็นวงจรแบบอสมวารที่มีสัญญาณอินพุต 3

สัญญาณ และเอาต์พุตของวงจรจะมีค่าระดับสัญญาณเป็น 0 เสมอ แต่เมื่อทำการดีคอมโพซิชั่นแล้ว วงจรที่ได้จะมีสัญญาณอินพุต 4 สัญญาณ และจะมีค่าระดับสัญญาณเอาต์พุตเป็น 1 เมื่อสัญญาณอินพุตของวงจร A และ D มีค่าเป็น 1 ทั้งคู่ ซึ่งเป็นการทำงานที่ผิดพลาดไปจากวงจรเดิม



รูปที่ 3.2 การดีคอมโพซิชั่นวงจร Ring Oscillator

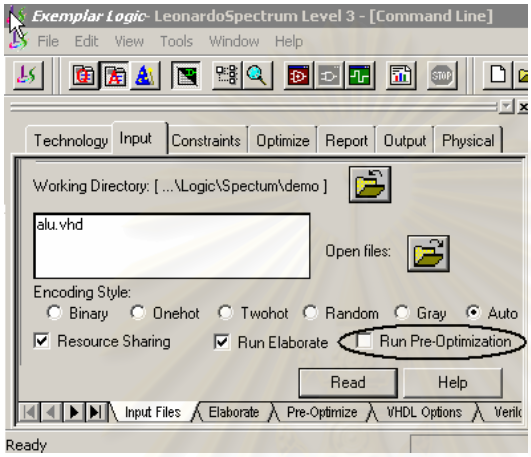
งานวิจัยนี้จึงได้นำเสนอแนวทางการออกแบบส่วนวงจรรางคู่บนเอฟพีจีเอที่สามารถป้องกันไม่ให้เกิดปัญหาดังกล่าว โดยเมื่อทำการสร้างแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับจากตารางค่าความจริง และแปลงแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับให้เป็นส่วนวงจรรางคู่แล้ว ในการออกแบบวงจรด้วยภาษาวีเอชดีแอล จะต้องเขียนอธิบายการทำงานของวงจรรางคู่ในระดับอาร์ทีแอล (RTL : Register Transfer Level) [3] ให้แต่ละกลุ่มวงจรย่อยเชื่อมต่อกันด้วยเกตแอนด์, เกตออร์ หรือเกตแอนด์กับเกตออร์ที่ต่อกันอยู่ในรูปของผลรวมของผลคูณ (Sum of Product) โดยแต่ละกลุ่มวงจรย่อยมีสัญญาณอินพุตได้ไม่เกิน 4 อินพุตดังรูปที่ 3.3

```
Entity temp is
    Port(a,a_l,b,b_l ,c,c_l: in std_logic; F:out std_logic);
End temp;
Architecture rtl of temp is
     Signal T1,T2 : std_logic;
Begin
    T1 = ((a and b_l) or (a_l and b)) and c_l; -- (5 i/p)
    T2 = ((a_l and b_l) or (a and b)) and c;    -- (5 i/p)
    F = T1 or T2;  (2 i/p)
End rtl;
------------------------เปลี่ยนเป็น-------------------------
Architecture rtl of temp is (แบบที่ถูกต้อง)
    Signal or1,or2 : std_logic;
Begin
    or1 = (a and b_l) or (a_l and b);  -- (4 i/p)
    or2 = (a_l and b_l) or (a and b);  -- (4 i/p)
    F = (or1 and c_l) or (or2 and c);  -- (4 i/p)
End rtl;
```

รูปที่ 3.3 การออกแบบวงจรรางคู่ด้วยภาษาวีเอชดีแอลในระดับ RTL

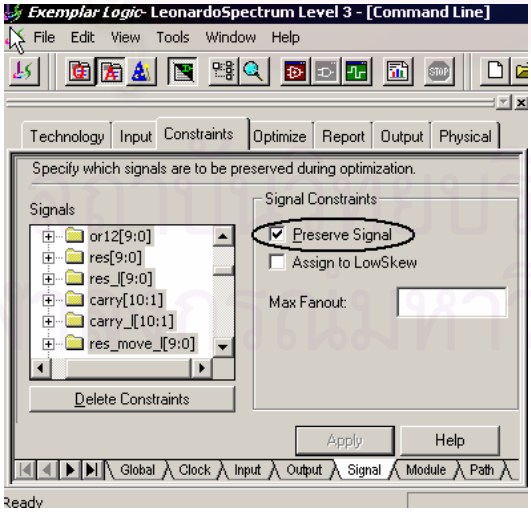
การสังเคราะห์วงจรด้วยซอฟต์แวร์ที่ชื่อว่า Leonardo Spectrum [12] ต้องทำการกำหนดตัวเลือก (Option) เพื่อทำให้สัญญาณ และโครงสร้างของวงจรรางคู่ที่ได้ออกแบบยังคงเหมือนเดิม ทำให้การพิจารณาเลือกสายสัญญาณเพื่อสร้างส่วนวงจรตอบรับสามารถทำได้ง่ายดังนี้

1. ไม่ทำ Run Pre-Optimization โดยกำหนดที่ Input Option ดังรูปที่ 3.4 เพื่อไม่ให้เกิดการแปลงวงจรย่อยหรือเปลี่ยนชื่อสัญญาณไปจากเดิม



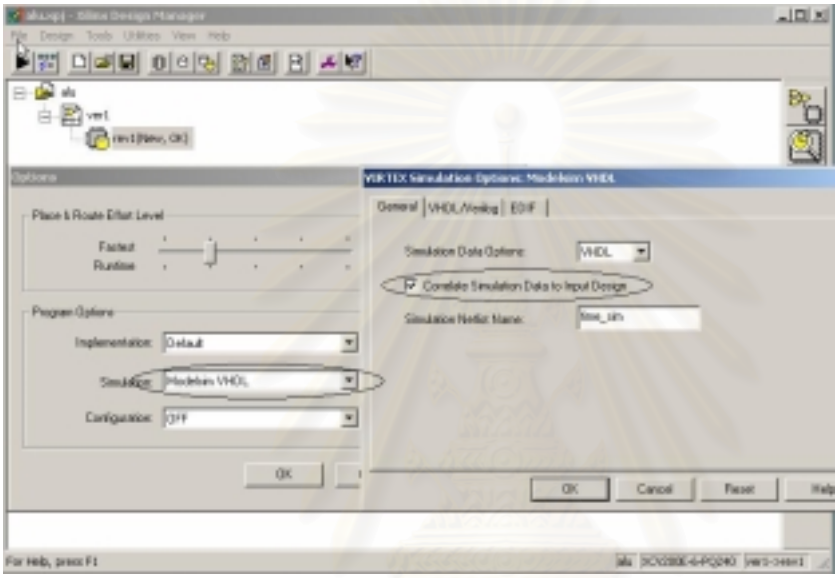
รูปที่ 3.4 การกำหนดตัวเลือกเพื่อไม่ทำ Run Pre-Optimization

2. ทำการ Preserve Signal ชื่อสัญญาณทุกเส้นที่เราได้กำหนดไว้ในวงจร เพื่อเป็นการป้องกันไม่ให้เกิดการลดรูปวงจรที่เราออกแบบ ซึ่งอาจทำให้วงจรเกิดการทำงานที่ผิดพลาดได้ โดยกำหนดที่ Constraint Option ดังรูปที่ 3.5



รูปที่ 3.5 การกำหนดตัวเลือกเพื่อทำการ Preserve Signal

การนำวงจรที่สังเคราะห์แล้วไปสร้างเป็นวงจรแบบอสมวารที่สามารถนำไปโปรแกรมลงเอฟพีจีเอได้ เป็นการเพลสและเราต์ (Place and Route) วงจรด้วยซอฟต์แวร์ที่ชื่อว่า Xilinx Foundation 3.1i [13] โดยจะต้องกำหนดตัวเลือก Simulation Option ใน Program Option ให้เป็น Modelsim VHDL และกำหนดใน Simulation Option : Modelsim VHDL ให้ทำ Correlate Simulation Data to Input Design เพื่อให้ซอฟต์แวร์ทำการสร้างไฟล์เอาต์พุตเป็นไฟล์สกุล .VHD และให้ชื่อของสัญญาณหรือวงจรย่อยยังคงเดิมดังรูปที่ 3.6



รูปที่ 3.6 การกำหนดตัวเลือกในซอฟต์แวร์ Xilinx Foundation เพื่อสร้างวงจรรางคู่

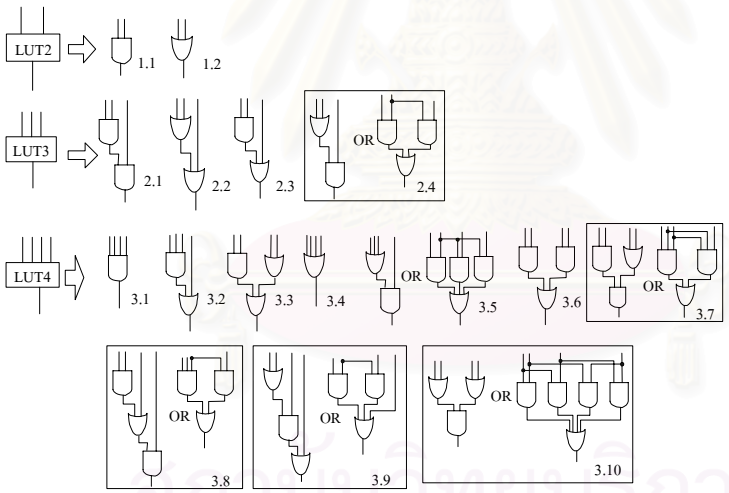
ไฟล์เอาต์พุตที่ได้จะแสดงให้เห็นโครงสร้างและชื่อสัญญาณภายในวงจรหลังจากที่ทำการเพลสและเราต์เสร็จสิ้นแล้ว ทำให้ง่ายต่อการพิจารณาวงจร และเลือกสายเพื่อสร้างส่วนวงจรตอบรับต่อไป รูปที่ 3.7 เป็นตัวอย่างโครงสร้างภายในของส่วนวงจรรางคู่ที่แสดงไว้ในไฟล์เอาต์พุตสกุล .VHD ที่ได้มา

```
IX1318 : X_LUT4
  generic map
    (INIT => X"EAC0")
  port map (
    ADR0 => a,
    ADR1 => a_l,
    ADR2 => b,
    ADR3 => b_l,
    O => or11);

IX1319 : X_LUT4
  generic map
   (INIT => X"EAC0")
  port map (
     ADR0 => a,
     ADR1 => a_l,
     ADR2 => b,
     ADR3 => b_l,
     O => or12);

IX1320 : X_LUT4
  generic map
  (INIT => X"EAC0")
  port map (
     ADR0 => or11,
     ADR1 => or12,
     ADR2 => c,
     ADR3 => c_l,
     O => F);
```

รูปที่ 3.7 ตัวอย่างโครงสร้างภายในส่วนวงจรรางคู่ที่แสดงไว้ใน ไฟล์เอาต์พุตสกุล.VHD ที่ได้หลังการสร้างวงจร

3.2 การออกแบบส่วนวงจรตอบรับสำหรับส่วนวงจรรางคู่บนเอฟพีจีเอ

เนื่องจากโครงสร้างภายในของส่วนวงจรรางคู่ที่ออกแบบบนเอฟพีจีเอจะอยู่ในรูปของการเชื่อมต่อกันระหว่างลุคอัพเทเบิ้ล และการออกแบบวงจรดิจิตอลบนเอฟพีจีเอนั้นไม่สามารถกำหนดค่าความหน่วงของเกตและสายในวงจรได้ ดังนั้นการออกแบบส่วนวงจรตอบรับสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ จึงไม่สามารถกำหนดอัตราส่วนความแปรปรวนความหน่วงสูงสุด (K) รวมทั้งใช้วิธีการเลือกสายสัญญาณภายในส่วนวงจรรางคู่ และกำหนดค่าความหน่วงของแต่ละเกตออร์ที่นำมาเรียงต่อกันเพื่อสร้างสัญญาณตอบรับดังที่กล่าวไว้ในหัวข้อที่ 2.5.3 ได้ งานวิจัยนี้จึงนำเสนอแนวทางการออกแบบส่วนวงจรตอบรับสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ โดยใช้การวิเคราะห์ฟังก์ชันภายใน และความหน่วงประมาณของแต่ละลุคอัพเทเบิ้ล เพื่อเลือกกลุ่มสายสัญญาณที่สามารถรับประกันการสิ้นสุดของการเปลี่ยนระดับสัญญาณจาก 1เป็น 0 ของทุกสายสัญญาณในวงจร และสร้างสัญญาณแสดงความบริบูรณ์ โดยการรวมสายสัญญาณที่เลือกทั้งหมดด้วยเกตออร์



รูปที่ 3.8 กลุ่มวงจรย่อยของส่วนวงจรรางคู่ที่อยู่ภายใน LUT2, LUT3 และ LUT4

จากรูปที่ 3.8 แต่ละลุคอัพเทเบิ้ลสามารถแทนกลุ่มวงจรรางคู่ที่มีสัญญาณอินพุตจำนวน 2 สัญญาณ, 3 สัญญาณ หรือ 4 สัญญาณ ซึ่งเรียกว่า LUT2, LUT3 และ LUT4 ตามลำดับ โดยกลุ่มวงจรย่อยที่อยู่ภายในแต่ละลุคอัพเทเบิ้ลสามารถเชื่อมต่อกันได้ด้วยเกตแอนด์, เกตออร์ หรือเกตแอนด์กับเกตออร์ที่เชื่อมต่อกันอยู่ในรูปของผลรวมของผลคูณ ซึ่งจะเห็นได้ว่า หากสายสัญญาณภายในส่วนวงจรรางคู่ที่ต้องการเลือกเป็นสายที่เชื่อมต่อกันอยู่ภายในลุคอัพเทเบิ้ล จะไม่สามารถ

ดึงสายดังกล่าวออกมาได้ การเลือกสายภายในส่วนวงจรรางคู่ที่ออกแบบบนเอฟพีจี-เอ จึงใช้การพิจารณาทุกลุคอัพเทเบิ้ล โดยแบ่งการพิจารณาออกเป็นสามแบบดังนี้

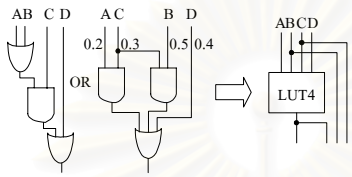
1. <u>เมื่อภายในลุคอัพเทเบิ้ลแทนวงจรย่อยที่เชื่อมต่อกันด้วยเกตแอนด์เพียงอย่างเดียว</u> สายสัญญาณอินพุตของเกตแอนด์เส้นที่มีค่าความหน่วงการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 สูงสุดสามารถรับประกันการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของสายสัญญาณอินพุตเส้นอื่นๆได้ นั่นคือเมื่อภายในลุคอัพเทเบิ้ลแทนวงจรที่ 1.1, 2.1 หรือ 3.1 จะเลือกสายสัญญาณอินพุตของลุคอัพเทเบิ้ลที่มีค่าความหน่วงประมาณการเปลี่ยนระดับสัญญาณสูงสุด

2. <u>เมื่อภายในลุคอัพเทเบิ้ลแทนวงจรย่อยที่เชื่อมต่อกันด้วยเกตออร์เพียงอย่างเดียว</u> โครงสร้างของวงจรรางคู่ที่ออกแบบโดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ ในขั้นทำงานจะมีอินพุตของเกตออร์เพียงอินพุตเดียวที่มีการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1 และจะเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 เมื่อเป็นการทำงานขั้นว่าง จึงกล่าวได้ว่าเอาต์พุตของเกตออร์สามารถรับประกันการเปลี่ยนระดับสัญญาณของสายอินพุตได้ ดังนั้นหากภายในลุคอัพเทเบิ้ลแทนวงจรที่ 1.2, 2.2 และ 3.4 จะเลือกสายเอาต์พุตของลุคอัพเทเบิ้ล

3. <u>เมื่อภายในลุคอัพเทเบิ้ลแทนวงจรย่อยที่มีเกตแอนด์กับเกตออร์เชื่อมต่อกันอยู่ในรูปผลรวมของผลคูณ</u> พบว่าในขั้นทำงานจะมีเส้นทางต่อเชื่อมเกตแอนด์ที่เป็นอินพุตของเกตออร์เพียงเส้นทางเดียวที่สายทุกเส้นมีค่าระดับสัญญาณ 1 จึงสามารถใช้สายเอาต์พุตของเกตออร์รับประกันการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของสายทุกเส้นในเส้นทางนั้นๆ ได้ แต่ในเส้นทางต่อเชื่อมเกตแอนด์ที่มีค่าระดับสัญญาณ 0 อาจจะมีสายบางเส้นที่มีค่าระดับสัญญาณเป็น 1 ซึ่งเอาต์พุตของเกตออร์ไม่สามารถรับประกันได้ จึงต้องใช้สายอินพุตของเกตแอนด์ที่มีค่าความหน่วงสูงสุดรับประกันการเปลี่ยนระดับสัญญาณด้วย นั่นคือถ้าภายในลุคอัพเทเบิ้ลแทนวงจรที่ 2.3, 3.2, 3.3, 3.5, 3.6, 3.7, 3.8, 3.9 และ 3.10 จะทำการเลือกสายเอาต์พุตของลุคอัพเทเบิ้ลร่วมกับสายอินพุตของแต่ละฟังก์ชันแอนด์ในลุคอัพเทเบิ้ลที่มีความหน่วงสูงสุด

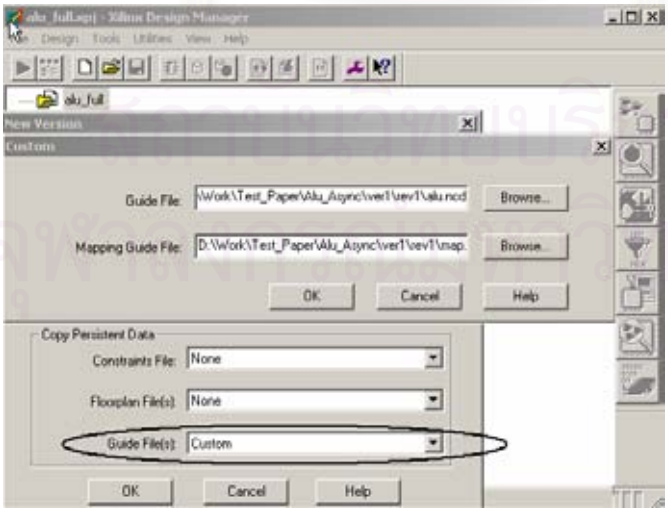
โดยค่าความหน่วงประมาณของสายสัญญาณในวงจรรางคู่จะแสดงไว้ในไฟล์เอาต์พุตสกุล .SDF ซึ่งเป็นไฟล์ที่ซอฟต์แวร์ Xilinx Foundation สร้างขึ้นพร้อมกับไฟล์สกุล .VHD และเมื่อเลือกกลุ่มสายที่เหมาะสมแล้ว จึงรวมสายที่เลือกทั้งหมดเข้าด้วยกันด้วยเกตออร์ จากนั้นใช้ภาษาวีเอชดีแอลเขียนอธิบายการทำงานส่วนวงจรตอบรับเพิ่มเติมเข้าไปในส่วนวงจรรางคู่ที่เคย

ออกแบบไว้ แล้วนำไปสังเคราะห์วงจร และกำหนดตัวเลือกต่างๆ เช่นเดียวกับการสังเคราะห์วงจรรางคู่ สำหรับตัวอย่างการเลือกสายสัญญาณภายในส่วนวงจรรางคู่ เพื่อสร้างส่วนวงจรตอบรับสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอแสดงดังรูปที่ 3.9 จากรูปจะเห็นว่าลักษณะของวงจรภายในลุคอัพเทเบิ้ลจะตรงกับวงจรที่ 3.9 ในรูปที่ 3.8 จึงทำการเลือกเอาต์พุตของลุคอัพเทเบิ้ล ร่วมกับอินพุตของแต่ละฟังก์ชันแอนด์ที่มีค่าความหน่วงสูงสุด นั่นคือสายอินพุต B, สายอินพุต C และสายเอาต์พุตของลุคอัพเทเบิ้ล



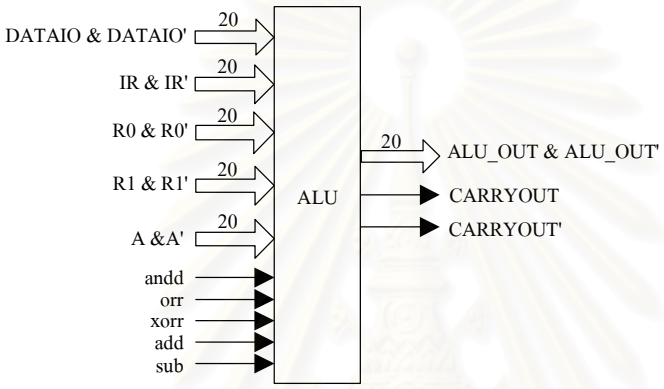
รูปที่ 3.9 ตัวอย่างการเลือกสายสัญญาณเพื่อสร้างส่วนวงจรตอบรับ

เมื่อรวมส่วนวงจรรางคู่ และส่วนวงจรตอบรับเข้าด้วยกันแล้ว ขั้นตอนการสร้างวงจรเชิงผสมแบบอสมวารให้สามารถนำไปโปรแกรมลงเอฟพีจีเอ จะเป็นการเพลสและเราต์วงจรใหม่ทั้งหมด ซึ่งอาจทำให้โครงสร้างและค่าความหน่วงประมาณภายในส่วนวงจรรางคู่เปลี่ยนไป ส่งผลให้กลุ่มสายสัญญาณที่เลือกไว้ไม่สามารถครอบคลุมการเปลี่ยนระดับสัญญาณของทุกสายสัญญาณในวงจรรางคู่ได้ ดังนั้นนอกเหนือจากการกำหนดตัวเลือกเช่นเดียวกับการสร้างวงจรรางคู่แล้ว จะต้องกำหนดรูปแบบของการเพลสและเราต์วงจรรางคู่ให้กับซอฟต์แวร์ด้วย โดยการกำหนด Guide File และ Mapping File ด้วยไฟล์สกุล .NCD ที่ได้จากการสร้างส่วนวงจรรางคู่ดังรูปที่ 3.10



รูปที่ 3.10 การกำหนดGuide File และ Mapping File

เมื่อใช้วิธีการออกแบบส่วนวงจรรางคู่ และส่วนวงจรตอบรับ สำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอที่นำเสนอในงานวิจัยนี้ ทดลองออกแบบหน่วยคำนวณ และประมวลผลตรรกะ (Arithmetic and Logic Unit : ALU) ที่สามารถทำคำสั่ง AND, OR, XOR, ADD และ SUB โดยมีอินพุตซึ่งเป็นสัญญาณรางคู่ขนาด 8 บิต จำนวน 5 อินพุตดังรูปที่ 3.11 ผลการออกแบบพบว่า ส่วนวงจรรางคู่ใช้จำนวนลุคอัพเทเบิ้ลภายในเอฟพีจีเอไปทั้งสิ้น 81 SLICEs ในการออกแบบส่วนวงจรตอบรับ ได้เลือกสายภายในวงจรรางคู่เป็นจำนวน 103 สาย โดยวงจรรวมใช้เนื้อที่ภายในเอฟพีจีเอไปทั้งสิ้น 112 SLICEs

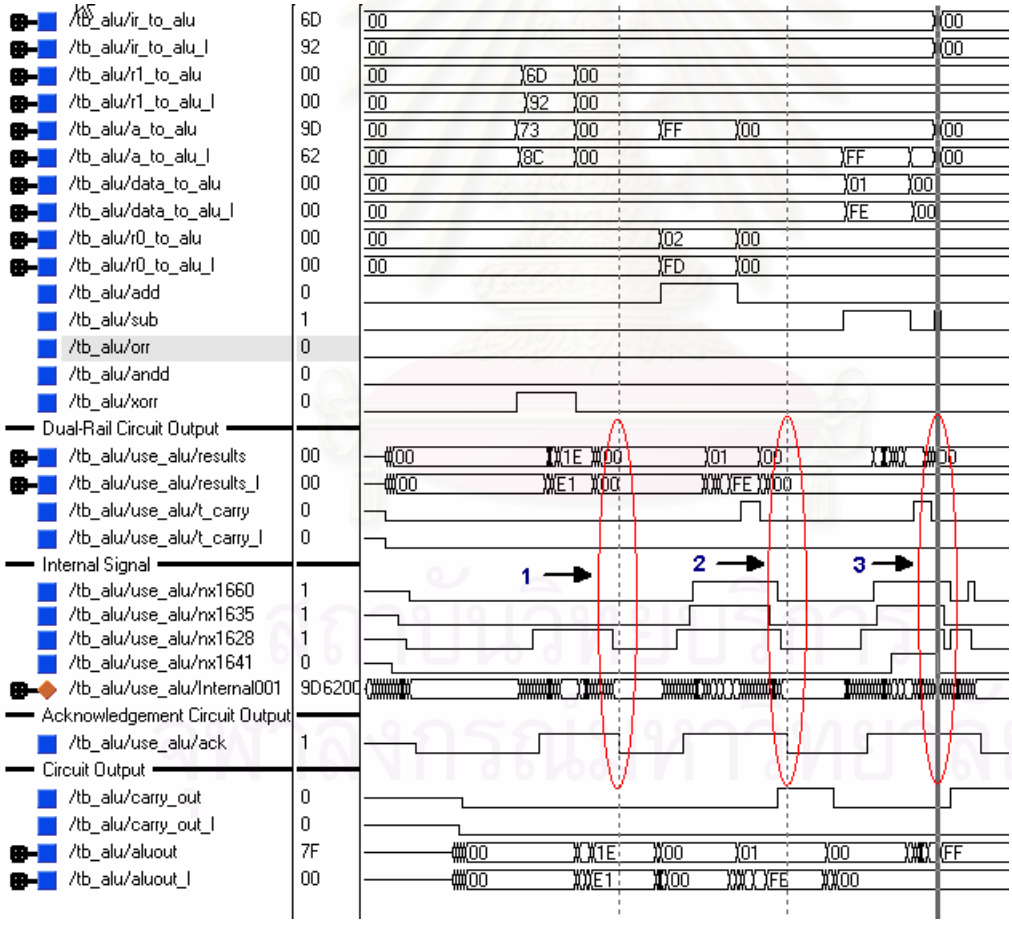


รูปที่ 3.11 ตัวอย่างวงจรเชิงผสมแบบอสมวารที่ออกแบบบนเอฟพีจีเอ

จากการวิเคราะห์อัตราส่วนระหว่างค่าความหน่วงสูงสุดของเส้นทางส่งผ่านสัญญาณจากอินพุตของวงจรไปยังเอาต์พุตของส่วนวงจรตอบรับกับค่าความหน่วงสูงสุดของเส้นทางส่งผ่านสัญญาณจากอินพุตของวงจรไปยังเอาต์พุตของส่วนวงจรรางคู่ เพื่อหาค่าอัตราส่วนความแปรปรวนความหน่วงสูงสุดของวงจร พบว่าหน่วยคำนวณ และประมวลผลตรรกะที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งออกแบบบนเอฟพีจีเอด้วยวิธีที่นำเสนอ มีอัตราส่วนความแปรปรวนความหน่วงสูงสุดเท่ากับ 1.15

การทดสอบความทนทานต่อความแปรปรวนของวงจรที่ออกแบบ และตรวจสอบว่าสัญญาณแสดงความบริบูรณ์ที่สร้างขึ้นสามารถรับประกันการเปลี่ยนระดับสัญญาณภายในส่วนวงจรรางคู่ได้ครอบคลุมหรือไม่ งานวิจัยนี้ได้ทำการทดลองสองรูปแบบ คือ การทดลองให้ความแปรปรวนความหน่วงในวงจรมีค่าเท่ากับอัตราส่วนความแปรปรวนความหน่วงสูงสุด โดยสิ่งแวดล้อมจะป้อนสัญญาณอินพุตให้กับวงจรทันทีที่สัญญาณเอาต์พุตของวงจรเปลี่ยนจาก 1 เป็น 0 และการทดลองให้ความแปรปรวนความหน่วงในวงจรมีค่าน้อยกว่าอัตราส่วนความแปรปรวนความหน่วงสูงสุด โดยสิ่งแวดล้อมป้อนสัญญาณอินพุตให้กับวงจรหลังจากที่เอาต์พุตของวงจรเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ไปเป็นเวลานานมากแล้ว

รูปที่ 3.12 แสดงผลการจำลองการทำงานหน่วยคำนวณ และประมวลผลตรรกะด้วยซอฟต์แวร์ Modelsim SE Plus [14] โดยผลการจำลองการทำงานที่ 1 เป็นการทดลองป้อนสัญญาณอินพุตให้กับวงจรทันทีที่สัญญาณเอาต์พุตของวงจรเปลี่ยนจาก 1 เป็น 0 และผลการจำลองการทำงานที่ 2 เป็นการทดลองป้อนสัญญาณอินพุตให้กับวงจรหลังจากที่เอาต์พุตของวงจรเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ไปเป็นเวลานานแล้ว จากผลการทดลองแสดงให้เห็นว่าสัญญาณ ack ซึ่งเป็นสัญญาณแสดงความบริบูรณ์มีการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 หลังจากที่สัญญาณภายในวงจรรางคู่มีการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 และสิ้นสุดที่ 0 หมดแล้วทั้งสองกรณี นั่นคือส่วนวงจรตอบรับสามารถรับประกันการสิ้นสุดการเปลี่ยนระดับสัญญาณของทุกสัญญาณในวงจรรางคู่ และวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งออกแบบบนเอฟพีจีเอด้วยวิธีที่นำเสนอสามารถทนต่อความแปรปรวนความหน่วงได้



รูปที่ 3.12 ผลการจำลองการทำงานวงจร ALU ที่ออกแบบ

สำหรับผลการจำลองการทำงานที่ 3 เป็นการทดลองให้ความแปรปรวนความหน่วงในวงจรมีค่ามากกว่าอัตราส่วนความแปรปรวนความหน่วงสูงสุด เพื่อพิสูจน์ว่าจำเป็นต้องมีส่วนวงจรตอบรับหรือไม่ โดยการป้อนสัญญาณอินพุตให้กับวงจรทันทีที่เอาต์พุตของส่วนวงจรรางคู่เปลี่ยนจาก 1 เป็น 0 จะเห็นได้ว่ามีสายสัญญาณภายในส่วนวงจรรางคู่บางสายที่ยังมีค่าระดับสัญญาณเป็น 1 ทำให้เมื่อได้รับอินพุตชุดใหม่เข้ามา วงจรจึงเกิดการทำงานที่ผิดพลาด นั่นคือ จำเป็นต้องมีส่วนวงจรตอบรับในวงจรเชิงผสมแบบอสมวาร เพื่อทำหน้าที่ตรวจสอบการสิ้นสุดของการเปลี่ยนระดับสัญญาณภายในวงจร

สรุป

งานวิจัยนี้ได้นำเสนอแนวทางการออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ โดยแบ่งการออกแบบออกเป็นสองส่วนคือ การออกแบบส่วนวงจรรางคู่ และการออกแบบส่วนวงจรตอบรับ ในการออกแบบส่วนวงจรรางคู่จะเริ่มจากการสร้างแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ แล้วแปลงให้เป็นส่วนวงจรรางคู่ จากนั้นใช้ภาษาวีเอชดีแอลเขียนอธิบายการทำงานของวงจรในระดับอาร์ทีแอล โดยให้แต่ละกลุ่มวงจรย่อยมีสัญญาณอินพุตได้ไม่เกิน 4 อินพุต และเชื่อมต่อกันด้วยเกตแอนด์, เกตออร์ หรือเกตแอนด์กับเกตออร์ที่ต่อกันอยู่ในรูปของผลรวมของผลคูณ แล้วนำวงจรที่ออกแบบได้ไปสังเคราะห์ด้วยซอฟต์แวร์ Leonardo Spectrum โดยต้องกำหนดตัวเลือกไม่ทำ Pre-Optimization และทำ Preserve Signal ทุกชื่อสัญญาณในวงจร เพื่อให้วงจรมีโครงสร้างตามที่ออกแบบ ทำให้วงจรที่ได้หลังจากการทำดีคอมโพซิชั่นไม่มีการทำงานที่ผิดพลาด จากนั้นจึงนำวงจรที่สังเคราะห์แล้วไปสร้างเป็นวงจรรางคู่บนเอฟพีจีเอด้วยซอฟต์แวร์ Xilinx Foundation โดยกำหนดตัวเลือกให้มี Simulation Option เป็น Modelsim VHDL และทำ Correlate Simulation Data to Input Design เพื่อสร้างไฟล์เอาต์พุตที่แสดงให้เห็นโครงสร้าง และค่าความหน่วงประมาณของแต่ละลุค-อัพเทเบิ้ลภายในส่วนวงจรรางคู่ รวมทั้งเป็นการกำหนดให้สัญญาณภายในวงจรยังคงมีชื่อสัญญาณดังเดิม เพื่อให้ง่ายต่อการสร้างส่วนวงจรตอบรับ

ในการสร้างส่วนวงจรตอบรับสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ จะเริ่มจากการนำไฟล์เอาต์พุตที่ได้จากการสร้างส่วนวงจรรางคู่มาวิเคราะห์ฟังก์ชันภายใน และค่าความหน่วงประมาณของแต่ละลุคอัพเทเบิ้ล แล้วเลือกกลุ่มสายสัญญาณภายในส่วนวงจรรางคู่ที่สามารถครอบคลุมการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของทุกสัญญาณในวงจร โดยแบ่งการเลือกสายได้เป็นสามแบบคือ

1. หากภายในลุคอัพเทเบิ้ลเชื่อมต่อกันด้วยเกตแอนด์เพียงอย่างเดียว จะเลือกสายอินพุตของลุคอัพเทเบิ้ลที่มีค่าความหน่วงสูงสุด
2. หากภายในลุคอัพเทเบิ้ลมีเพียงเกตออร์ จะเลือกสายเอาต์พุตของลุคอัพเทเบิ้ล
3. หากฟังก์ชันภายในเป็นการต่อกันของเกตแอนด์กับเกตออร์ ซึ่งอยู่ในรูปของผลรวมของผลคูณ จะเลือกสายเอาต์พุตของลุคอัพเทเบิ้ล ร่วมกับสายอินพุตของแต่ละฟังก์ชันแอนด์ที่มีค่าความหน่วงสูงสุด

จากนั้นจึงใช้เกตออร์รวมกลุ่มสายสัญญาณที่เลือก เพื่อสร้างสัญญาณแสดงความบริบูรณ์ และเขียนอธิบายการทำงานดังกล่าวด้วยภาษาวีเอชดีแอล แล้วนำไปสังเคราะห์และสร้างเป็นวงจรเชิงผสมแบบอสมวารบนเอฟพีจีเอ โดยนอกเหนือจากการกำหนดตัวเลือกเช่นเดียวกับการออกแบบส่วนวงจรรางคู่แล้ว ในการสร้างวงจรจะต้องกำหนด Guide File และ Mapping File ซึ่งเป็นรูปแบบการเพลสและเราต์ส่วนวงจรรางคู่ที่เคยออกแบบไว้ให้กับซอฟต์แวร์ด้วย เพราะการสร้างวงจรทั้งหมดอีกครั้ง อาจทำให้ส่วนวงจรรางคู่ที่ถูกเพลสและเราต์ใหม่มีโครงสร้าง และค่าความหน่วงเปลี่ยนไป ส่งผลให้กลุ่มสายสัญญาณที่เลือกไว้ไม่สามารถครอบคลุมทุกการเปลี่ยนระดับสัญญาณในวงจร

บทที่ 4
การออกแบบไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิต ที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ

จากการทดลองออกแบบวงจรเชิงผสมแบบอสมวารที่มีแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้โดยใช้เอฟพีจีเอ พบว่าวงจรสามารถทำงานได้ถูกต้อง งานวิจัยนี้จึงทำการทดลองออกแบบวงจรแบบอสมวารที่มีขนาดใหญ่มากขึ้น และสามารถนำมาใช้งานได้จริง โดยการออกแบบไมโครโปรเซสเซอร์แบบอสมวารบนเอฟพีจีเอ ซึ่งมีดาต้าพาท (Data Path) ขนาด 8 บิต และมีหน่วยคำนวณและประมวลผลตรรกะที่มีแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ โดยมีชุดคำสั่ง และสถาปัตยกรรมที่กำหนดขึ้นเอง

4.1 ชุดคำสั่งและรหัสดำเนินการ

ไมโครโปรเซสเซอร์ที่ออกแบบนั้นมีชุดคำสั่ง (Instruction Set) ที่ใช้งานแบ่งออกได้เป็นสามกลุ่มคือ กลุ่มคำสั่ง Data Transfer Operation, กลุ่มคำสั่ง Program and Machine Control Operation และกลุ่มคำสั่ง Arithmetic and Logic Operation โดยกลุ่มคำสั่ง Data Transfer Operation ประกอบด้วยคำสั่ง LD (Load) และ ST (Store), กลุ่มคำสั่ง Program and Machine Control Operation ประกอบด้วยคำสั่ง JZ (Jump Zero), JNZ (Jump not Zero), JC (Jump Carry), JNC (Jump not Carry), JMP (Jump), CALL และ RET (Return) และกลุ่มคำสั่ง Arithmetic and Logic Operation ประกอบด้วยคำสั่ง ADD, SUB, AND, OR, XOR, SL (Shift Left) และ SR (Shift Right) รวมทั้งสิ้น 16 คำสั่ง โดยมีรูปแบบของคำสั่ง รวมทั้งความหมายของสัญลักษณ์ดังตารางที่ 4.1 - ตารางที่ 4.4 ซึ่งทุกคำสั่งจะมีความยาวของรหัสดำเนินการ (Op code) เท่ากันคือ 16 บิต และกำหนดได้ดังรูปที่ 4.1

ตารางที่ 4.1 ชุดคำสั่งในกลุ่ม Data Transfer Operation

LD	A, #K	A <- K
LD	A, Reg	A <- Reg
LD	A, @K	A <- Mem[K]
LD	A, @Reg	A <- Mem[Reg]
ST	A, Reg	Reg <- A
ST	A, @K	Mem[K] <- A
ST	A, @Reg	Mem[Reg] <- A

ตารางที่ 4.2 ชุดคำสั่งในกลุ่ม Program and Machine Control Operation

JZ	Address	PC <- Address if Z = 1
JNZ	Address	PC <- Address S if Z = 0
JC	Address	PC <- Address if C = 1
JNC	Address	PC <- Address if C = 0
JMP	Address	PC <- Address
CALL	Address	SP <- PC; PC <- Address
RET		PC <- SP

ตารางที่ 4.3 ชุดคำสั่งในกลุ่ม Arithmetic and Logic Operation

AND	A, #K	A <- A AND K
AND	A, Reg	A <- A AND Reg
AND	A, @K	A <- A AND Mem[K]
AND	A, @Reg	A <- A AND Mem[Reg]
OR	A, #K	A <- A OR K
OR	A, Reg	A <- A OR Reg
OR	A, @K	A <- A OR Mem[K]
OR	A, @Reg	A <- A OR Mem[Reg]
XOR	A, #K	A <- A XOR K
XOR	A, Reg	A <- A XOR Reg
XOR	A, @K	A <- A XOR Mem[K]
XOR	A, @Reg	A <- A XOR Mem[Reg]
SUB	A, #K	A <- A – K
SUB	A, Reg	A <- A – Reg
SUB	A, @K	A <- A - Mem[K]
SUB	A, @Reg	A <- A - Mem[Reg]
ADD	A, #K	A <- A + K
ADD	A, Reg	A <- A + Reg
ADD	A, @K	A <- A + Mem[K]
ADD	A, @Reg	A <- A + Mem[Reg]
SL	A	A <- A[MSB-1..0] & ‘0’
SR	A	A <- ‘0’ & A[MSB..1]

ตารางที่ 4.4 ความหมายของสัญลักษณ์ต่างๆ

สัญลักษณ์	ความหมาย
K	ค่าคงที่
Reg	รีจิสเตอร์ทั่วไป (R0, R1)
Address	ตำแหน่งอ้างอิงในหน่วยความจำสำหรับโปรแกรม
#	การอ้างอิงค่าคงที่
@	การอ้างอิงค่าในหน่วยความจำสำหรับข้อมูล

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

000	0	Md	Op	LD
	1	Md	Op	ST
00100		0	0000000000	SL
		1	0000000000	SR
0	011	md	Op	AND
	100	md	Op	OR
	101	md	Op	XOR
	110	md	Op	SUB
	111	md	Op	ADD
1000		00	K	JZ
		01	K	JNZ
		10	K	JC
		11	K	JNC
1001		00	0000000000	JMP
		01	0000000000	CALL
		10	0000000000	RET

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

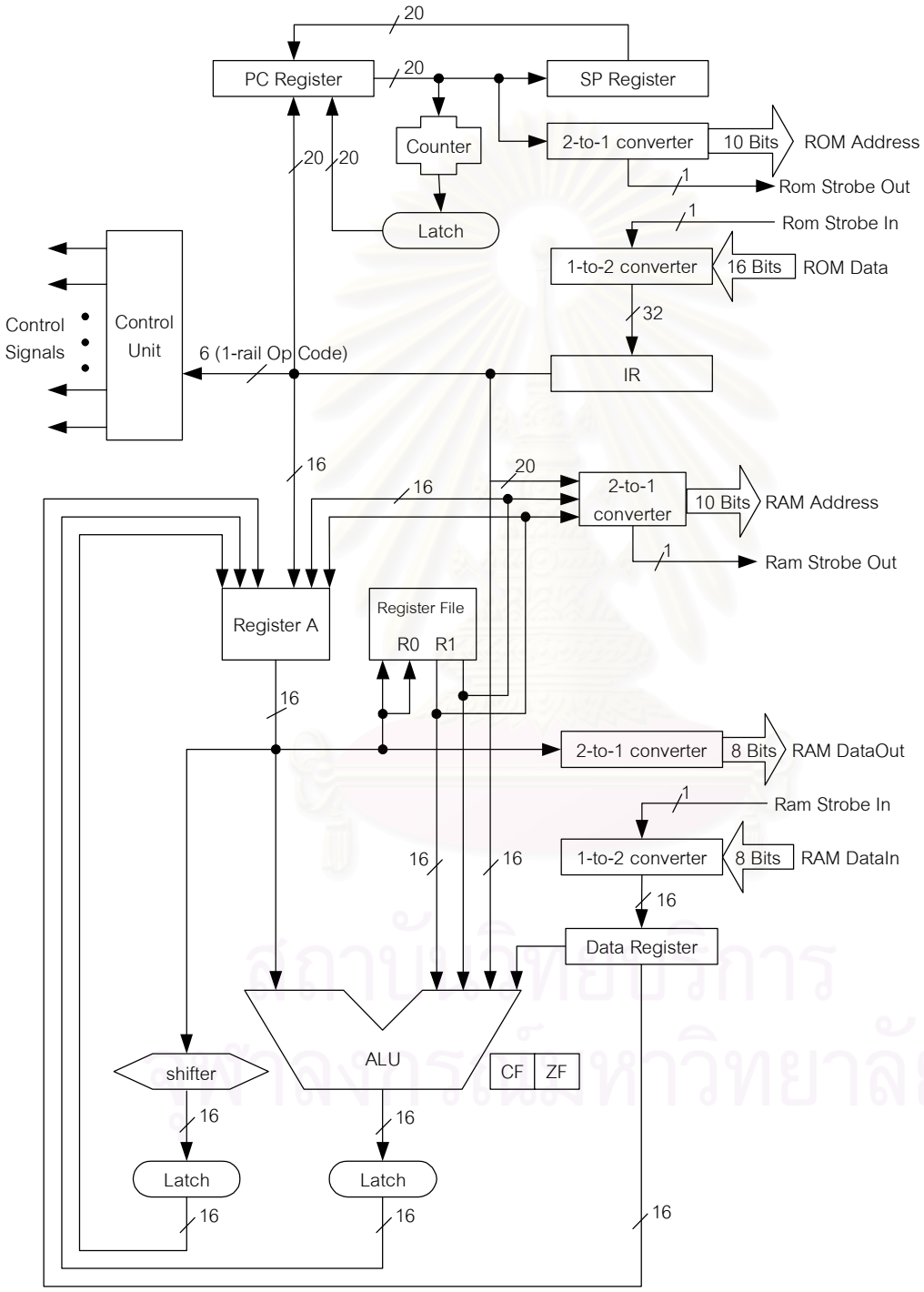
md	Op			
00	00	D		Immediate Mode
01	00000000	R1	R0	Register Mode
10	K			Direct Addressing
11	00000000	R1	R0	Indirect Addressing
				D = Binary 8 bits
				K = Binary 10 Bits

รูปที่ 4.1 รหัสดำเนินการของชุดคำสั่ง

4.2 สถาปัตยกรรมของไมโครโปรเซสเซอร์ 8 บิตแบบอสมวาร

สถาปัตยกรรมของไมโครโปรเซสเซอร์ที่ทดลองออกแบบบนเอฟพีจีเอแสดงดังรูปที่ 4.2 โดยมีรีจิสเตอร์ที่ใช้ในการประมวลผลประกอบด้วยรีจิสเตอร์ตัวสะสม (Accumulator) ขนาด 8 บิต เรียกว่ารีจิสเตอร์ A และรีจิสเตอร์ทั่วไป (General Register) ขนาด 8 บิตจำนวนสองตัวเรียกว่า รีจิสเตอร์ R0 และ R1 สำหรับรีจิสเตอร์เฉพาะซึ่งอยู่ภายในไมโครโปรเซสเซอร์ประกอบด้วยตัวนับระบุตำแหน่งคำสั่ง (Program Counter) เรียกว่า รีจิสเตอร์ PC, รีจิสเตอร์ที่ใช้เก็บค่า PC เมื่อมีการ

ทำคำสั่ง CALL เรียกว่า รีจิสเตอร์ SP และรีจิสเตอร์ที่ใช้เก็บรหัสดำเนินการของคำสั่งที่อ่านเข้ามาเรียกว่า รีจิสเตอร์ IR โดยมีสัญญาณแสดงสถานะ (Status Flag) สองสัญญาณคือ สัญญาณทดค่า (Carry Flag) และสัญญาณค่าศูนย์ (Zero Flag)



รูปที่ 4.2 สถาปัตยกรรมของไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิต

ในส่วนของวงจรที่ทำหน้าที่ประมวลผลจะประกอบด้วยวงจรเลื่อนข้อมูล (Shfiter) ซึ่งจะทำการเลื่อนค่าของรีจิสเตอร์ A ไปทางซ้ายหรือขวา 1 บิต กับหน่วยคำนวณและประมวลผลตรรกะที่มีแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งทำงานแบบ Accumulator Based คือ ทำการคำนวณระหว่างรีจิสเตอร์ A กับค่าของตัวถูกดำเนินการ (Operand) แล้วเก็บค่าผลลัพธ์ที่ได้ลงรีจิสเตอร์ A โดยสามารถอ้างอิงค่าของตัวถูกดำเนินการได้สี่แบบคือ แบบค่าคงที่ (Immediate Mode), แบบรีจิสเตอร์ (Register Mode), แบบอ้างอิงหน่วยความจำโดยตรง (Direct Memory Addressing Mode) และแบบอ้างอิงหน่วยความจำโดยอ้อมด้วยรีจิสเตอร์ (Indirect Memory Addressing with Register Indexing Mode)

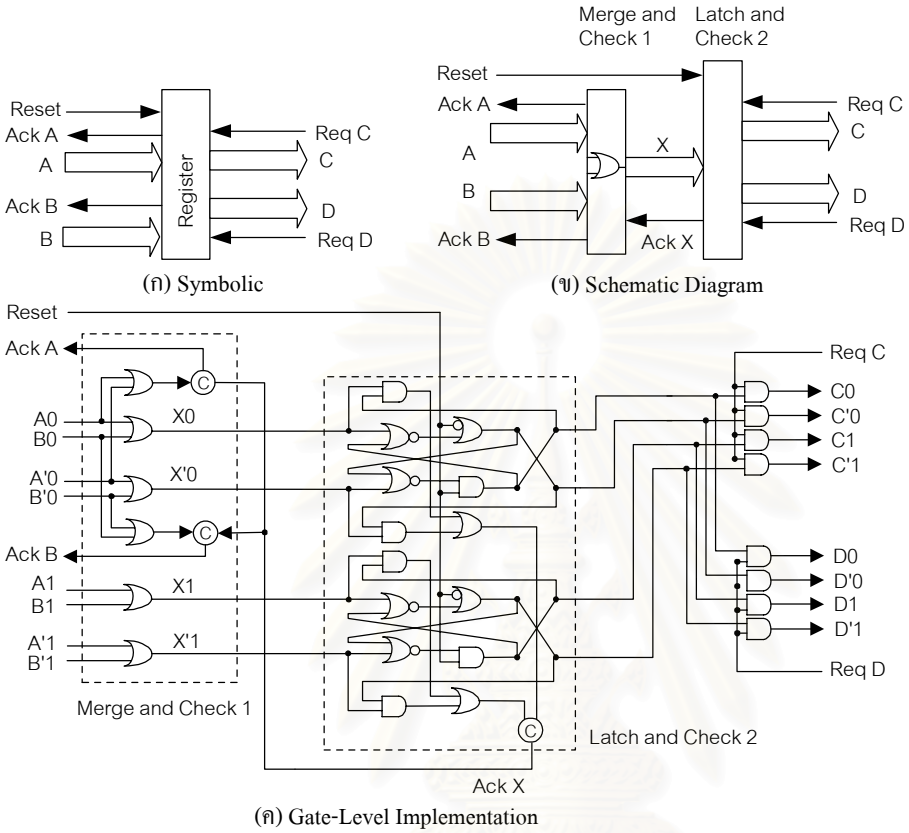
เนื่องจากการคำนวณและประมวลผลเป็นการทำงานแบบ Accumulator Based ซึ่งจะเขียนผลลัพธ์ที่ได้ไปยังรีจิสเตอร์ A หากค่าในรีจิสเตอร์ A มีการเปลี่ยนแปลง ก็จะทำให้เกิดการประมวลผลใหม่อีกครั้งทันที และทำให้เกิดการทำงานวนซ้ำไปเรื่อยๆ จึงต้องมีรีจิสเตอร์ซึ่งเรียกว่า Latch ทำหน้าที่เก็บผลลัพธ์ของการประมวลผลไว้ชั่วคราวก่อน โดยรีจิสเตอร์ Latch นี้จะถูกนำไปใช้ในการเก็บตัวดำเนินการที่อ่านมาจากหน่วยความจำไว้ชั่วคราว เพื่อใช้ในการประมวลผลต่อไปด้วยเช่นกัน

ไมโครโปรเซสเซอร์ที่ออกแบบนี้สามารถอ้างอิงหน่วยความจำสำหรับโปรแกรม (Programmed Memory) ซึ่งใช้เก็บรหัสดำเนินการขนาด 16 บิต ได้ 1 กิโลตำแหน่ง (1K * 16 bits) และหน่วยความจำสำหรับข้อมูล (Data Memory) ที่มีขนาด 8 บิต ได้ 1 กิโลตำแหน่ง (1K * 8 bits) เนื่องจากการออกแบบวงจรแบบอสมวาร จะใช้สายสัญญาณสองเส้นแทนสัญญาณข้อมูลขนาด 1 บิต การอ้างอิงหน่วยความจำที่มีอยู่ในปัจจุบัน จึงต้องมีวงจรที่แปลงสายสัญญาณ 2 เส้นให้เป็น 1 เส้นเพื่อส่งออกไปยังหน่วยความจำ และวงจรที่แปลงสายสัญญาณที่รับมาจากหน่วยความจำจาก 1 เส้นให้เป็น 2 เส้น

4.3 การออกแบบรีจิสเตอร์แบบอสมวาร

งานวิจัยนี้ใช้โครงสร้างของรีจิสเตอร์ที่ใช้ในไมโครโปรเซสเซอร์ TITAC1 [6] ในการออกแบบรีจิสเตอร์แบบอสมวาร โดยเพิ่มส่วนของการตั้งค่าข้อมูลใหม่ (Data Reset) เพื่อให้สายสัญญาณคู่ภายในรีจิสเตอร์ของข้อมูลแต่ละบิตมีค่าระดับสัญญาณเริ่มต้นเป็น (0,1) เมื่อสัญญาณรีเซต (Reset Signal) มีค่าระดับสัญญาณเป็น 0 ดังรูปที่ 4.3 โดยในการอ่านค่าข้อมูลจากรีจิสเตอร์ ส่วนควบคุมจะเปลี่ยนค่าระดับสัญญาณร้องขอจาก 0 เป็น 1 เพื่อเป็นสัญญาณอนุมัติ (Enable Signal) ให้รีจิสเตอร์ส่งค่ารหัสตรงรางคู่ของข้อมูลออกไปยังวงจรเอาต์พุตที่ต้องการ และทำการเปลี่ยนค่าระดับสัญญาณร้องขอจาก 1 เป็น 0 เพื่อทำให้สายสัญญาณคู่ของ

ข้อมูลที่ส่งไปยังวงจรเอาต์พุตมีค่าระดับสัญญาณกลับเป็นตัวแบ่งรอบการทำงาน หรือเป็นการหยุดอ่านค่าข้อมูลจากรีจิสเตอร์นั่นเอง



รูปที่ 4.3 โครงสร้างของรีจิสเตอร์แบบอสมวารแบบหลายช่องทางขนาด 2 บิต

ในการเขียนค่าข้อมูลลงรีจิสเตอร์ เมื่อวงจรอินพุตส่งค่าข้อมูลเป็นรหัสตรงรางคู่เข้ามา วงจรแลตช์ (Latch) ซึ่งมีโครงสร้างของวงจรแบบเอส-อาร์ฟลิปฟล็อป (S-R flip-flop) จะเก็บค่ารหัสตรงรางคู่ที่ได้รับมาไว้ และจะทำให้สัญญาณตอบรับมีการเปลี่ยนแปลงระดับสัญญาณจาก 0 เป็น 1 เพื่อบอกส่วนควบคุมว่าได้ทำการเก็บข้อมูลไว้ครบทุกบิตแล้ว จากนั้นวงจรอินพุตจะส่งค่าข้อมูลเป็นตัวแบ่งรอบการทำงานเข้ามาอีกครั้ง และทำให้สัญญาณตอบรับมีการเปลี่ยนแปลงระดับสัญญาณจาก 1 เป็น 0 เป็นการเสร็จสิ้นการเขียนค่าข้อมูลลงรีจิสเตอร์ โดยรีจิสเตอร์จะเก็บค่ารหัสตรงรางคู่ของข้อมูลอินพุตเดิมไว้ จนกว่าวงจรอินพุตจะส่งรหัสตรงรางคู่ค่าใหม่เข้ามา สำหรับการสร้างสัญญาณตอบรับกลับไปยังส่วนควบคุมนั้น รีจิสเตอร์จะทำการตรวจสอบสองขั้นตอนคือ การตรวจสอบว่าวงจรใดเป็นวงจรที่ส่งข้อมูลอินพุตเข้ามา และการตรวจสอบว่าวงจรแลตช์เก็บข้อมูลครบทุกบิตแล้วหรือยัง แล้วจึงนำผลที่ได้จากการตรวจสอบทั้งสองขั้นตอนมาเชื่อมต่อกันด้วยอุปกรณ์ซนิดซี เพื่อสร้างเป็นสัญญาณตอบรับ

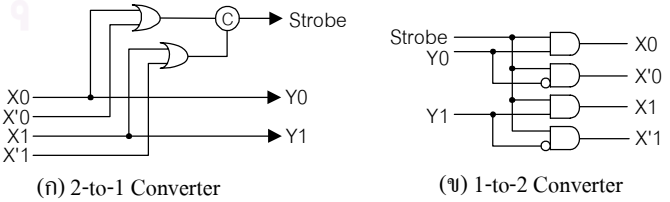
4.4 การออกแบบส่วนประมวลผล

ภายในไมโครโปรเซสเซอร์ประกอบด้วยวงจรประมวลผลสองวงจรคือ วงจรคำนวณและประมวลผลตรรกะ กับวงจรเลื่อนข้อมูล โดยวงจรคำนวณและประมวลผลตรรกะจะทำการบวก (ADD), ลบ (SUB), แอนด์ (AND), ออร์ (OR) หรือเอ็กซ์คิวซีฟออร์ (XOR) ค่าข้อมูลของอินพุตทั้งสองอินพุตเข้าด้วยกัน แล้วส่งผลลัพธ์ที่ได้ออกไปยังวงจรแลตช์ การออกแบบวงจรจะแบ่งออกเป็นสองส่วนคือ การออกแบบส่วนวงจรรางคู่ และการออกแบบส่วนวงจรตอบรับ โดยจะใช้วิธีการออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอซึ่งนำเสนอไว้ในบทที่ 3

วงจรเลื่อนข้อมูล เป็นวงจรที่จะทำการเลื่อนข้อมูลในรีจิสเตอร์ A ไปทางซ้ายหรือทางขวาจำนวน 1 บิต ซึ่งจากหลักการออกแบบวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงที่นำเสนอไว้ในบทที่ 3 พบว่าโครงสร้างวงจรรางคู่ภายในแต่ละลุคอัพเทเบิ้ลของวงจรเลื่อนข้อมูลจะมีลักษณะตรงกับวงจรที่ 3.6 ของรูปที่3.8 สายสัญญาณภายในที่จะถูกเลือกเพื่อสร้างส่วนวงจรตอบรับ จึงเป็นสายอินพุตของวงจรเลื่อนข้อมูล กับสายเอาต์พุตของวงจรรางคู่ ซึ่งสามารถนำออกจากการเลือกได้ ดังนั้นในการออกแบบวงจรเลื่อนข้อมูลนี้จึงไม่มีการสร้างส่วนวงจรตอบรับ และใช้การจัดเก็บค่าลงรีจิสเตอร์ Latch เป็นการตรวจสอบการสิ้นสุดของการประมวลผลแทน

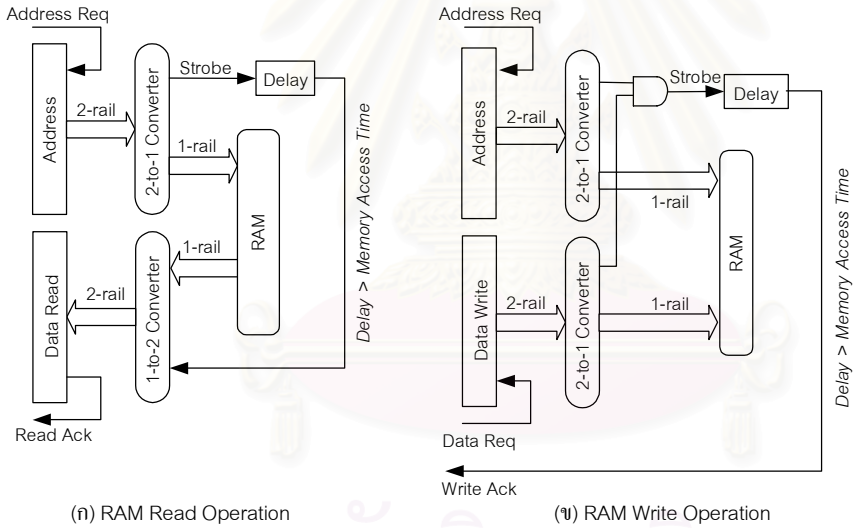
4.5 การออกแบบส่วนวงจรที่ติดต่อกับหน่วยความจำ

ภายในไมโครโปรเซสเซอร์จะใช้สายสัญญาณ 2 เส้นแทนข้อมูลขนาด 1 บิต ดังนั้นการติดต่อกับอุปกรณ์ภายนอก เช่นการอ้างอิงหน่วยความจำ จึงต้องมีวงจรที่ทำหน้าที่แปลงสายสัญญาณ 1 เส้นให้เป็น 2 เส้น เรียกว่าวงจร 1-to-2 converter เพื่อใช้ในการแปลงสายสัญญาณข้อมูลที่อ่านมาจากหน่วยความจำ (Data Read) โดยมีโครงสร้างของวงจรแสดงดังรูปที่ 4.4(ก) และวงจรที่ทำหน้าที่แปลงสายสัญญาณ 2 เส้นให้เป็น 1 เส้น เรียกว่าวงจร 2-to-1 converter ซึ่งมีโครงสร้างของวงจรดังรูปที่ 4.4(ข) เพื่อใช้ในการแปลงสายสัญญาณตำแหน่งอ้างอิง (Address) กับสายสัญญาณข้อมูลที่จะเขียนไปยังหน่วยความจำ (Data Write)



รูปที่ 4.4 การออกแบบวงจร 2-to-1 Converter และวงจร 1-to-2 Converter

การอ่านข้อมูลจากหน่วยความจำ ไมโครโปรเซสเซอร์จะมีการต่อประสานกับหน่วยความจำ (Memory Interface) แสดงดังรูปที่ 4.5(ก) โดยส่วนควบคุมจะส่งสัญญาณร้องขอไปยังรีจิสเตอร์ เพื่อให้ส่งค่าตำแหน่งอ้างอิงมายังวงจร 2-to-1 converter ซึ่งจะทำการแปลงค่าตำแหน่งอ้างอิงจากสายสัญญาณคู่ (X,X') ให้เป็นสายสัญญาณเส้นเดียว แล้วส่งออกไปยังหน่วยความจำ โดยสัญญาณ Strobe ซึ่งใช้บอกว่าทำการแปลงเสร็จสิ้นแล้วจะถูกส่งออกไปยังวงจรสร้างความหน่วง เพื่อให้มีค่าความหน่วงมากกว่าค่าความหน่วงในการอ้างอิงของหน่วยความจำ (Memory Access Time) และส่งต่อไปยังวงจร 1-to-2 converter เพื่อเป็นสัญญาณร้องขอให้ทำการแปลงค่าข้อมูลที่อ่านมาจากหน่วยความจำ ซึ่งเป็นแบบสายสัญญาณเส้นเดียวให้กลายเป็นข้อมูลแบบสายสัญญาณ 2 เส้น เมื่อรีจิสเตอร์ปลายทางได้รับค่าข้อมูลที่อ่านมาเรียบร้อยแล้ว ก็จะส่งสัญญาณตอบรับกลับไปยังส่วนควบคุม และเข้าสู่การทำงานในขั้นว่าง จนกระทั่งสัญญาณตอบรับที่ส่งไปยังส่วนควบคุมมีการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 จึงเสร็จสิ้นกระบวนการอ่านข้อมูลจากหน่วยความจำ



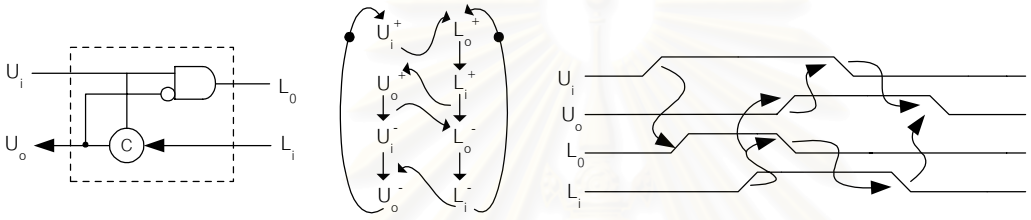
รูปที่ 4.5 การต่อประสานกับหน่วยความจำ

การเขียนค่าข้อมูลลงหน่วยความจำ ไมโครโปรเซสเซอร์จะมีการต่อประสานกับหน่วยความจำแสดงดังรูปที่ 4.5(ข) โดย วงจร 2-to-1 converter จะทำการแปลงค่าตำแหน่งอ้างอิงและข้อมูลที่ต้องการจะเขียนให้เป็นสายสัญญาณเส้นเดียว และใช้สัญญาณ Strobe เป็นสัญญาณเปิดทางให้เขียน (Write Enable: WE) ส่งไปยังหน่วยความจำ จากนั้นจึงนำสัญญาณ Strobe ซึ่งมีค่าความหน่วงมากกว่าค่าความหน่วงในการอ้างอิงของหน่วยความจำแล้ว มาสร้างเป็นสัญญาณตอบรับส่งกลับไปยังส่วนควบคุม เพื่อบอกว่าเขียนข้อมูลลงหน่วยความจำแล้ว และเข้าสู่การทำงานในขั้นว่าง จนกระทั่งส่วนควบคุมได้รับสัญญาณตอบรับที่มีการเปลี่ยนระดับสัญญาณจาก

1 เป็น 0 จึงเป็นการเสร็จสิ้นกระบวนการเขียนข้อมูลลงหน่วยความจำ โดยไมโครโปรเซสเซอร์ TITAC1 ใช้วิธีการต่อประสานกับหน่วยความจำในการอ่านและเขียนข้อมูลดังรูปที่ 4.5 เช่นกัน

4.6 การออกแบบส่วนวงจรควบคุม

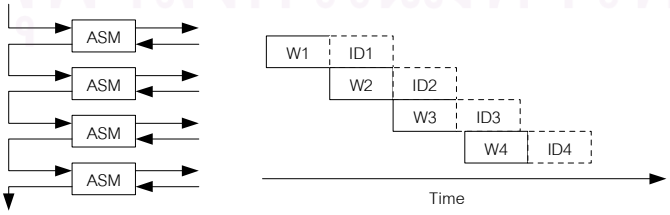
การทำงานของวงจรภายในไมโครโปรเซสเซอร์เป็นแบบ 2-rail 2-phase Return to Zero คือมีการทำงานในขั้นทำงาน และขั้นว่างสลับกันไปเรื่อยๆ จึงใช้ Autosweeping Module [7] ซึ่งมีโครงสร้างของวงจร รวมทั้งรูปแบบการเปลี่ยนระดับสัญญาณร้องขอและสัญญาณตอบรับดังรูปที่ 4.6 ในการควบคุม และสร้างสัญญาณร้องขอกับสัญญาณตอบรับ



(ก) Implementation (ข) STG Specification (ค) Signal Transition Diagram

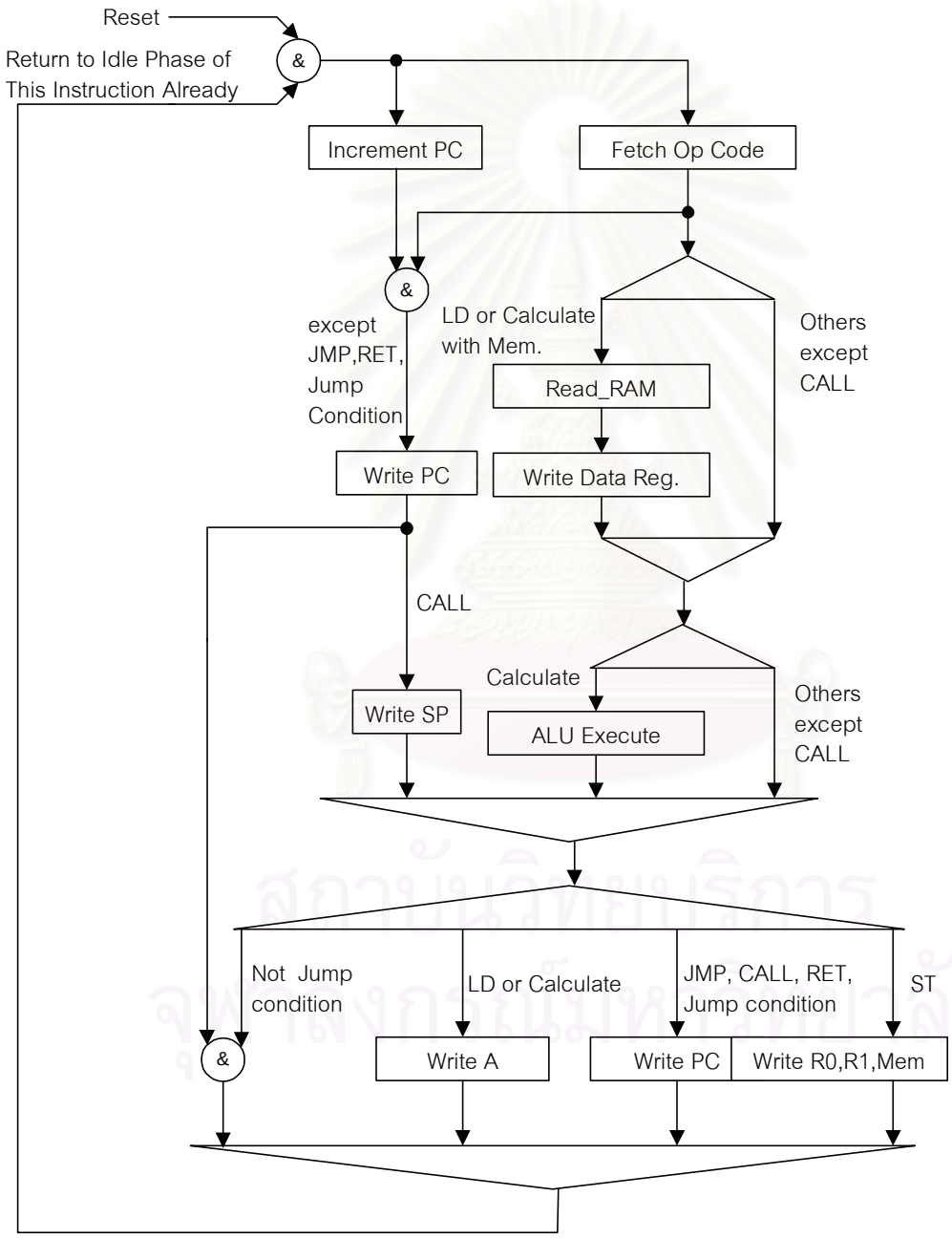
รูปที่ 4.6 Autosweeping Module : ASM

จากรูปเมื่อสัญญาณร้องขอของการทำงานก่อนหน้า (U_i) มีการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1 ASM จะเปลี่ยนระดับสัญญาณร้องขอของการทำงานถัดไป (L_o) จาก 0 เป็น 1 เพื่อเข้าสู่ขั้นทำงาน หลังจากที่การทำงานถัดไปเสร็จสิ้นแล้ว จึงส่งสัญญาณตอบรับ (L_i) กลับมายัง ASM ซึ่งจะส่งผลให้สัญญาณตอบรับของการทำงานก่อนหน้า (U_o) เกิดการเปลี่ยนระดับสัญญาณจาก 0 เป็น 1 พร้อมกับที่สัญญาณร้องขอของการทำงานถัดไป เกิดการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 เพื่อเข้าสู่ขั้นว่างได้เลย นั่นคือ เมื่อเสร็จสิ้นการทำงานในขั้นทำงานแล้ว สัญญาณร้องขอของการทำงานถัดไปจะเกิดการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ได้เองทันที ดังนั้นเมื่อนำวงจร ASM มาใช้ในการออกแบบวงจรควบคุม (Controller) จะทำให้สามารถทำงานในขั้นว่างของการทำงานก่อนหน้า ขนานไปพร้อมๆ กับทำงานในขั้นทำงานของการทำงานถัดไปดังรูปที่ 4.7

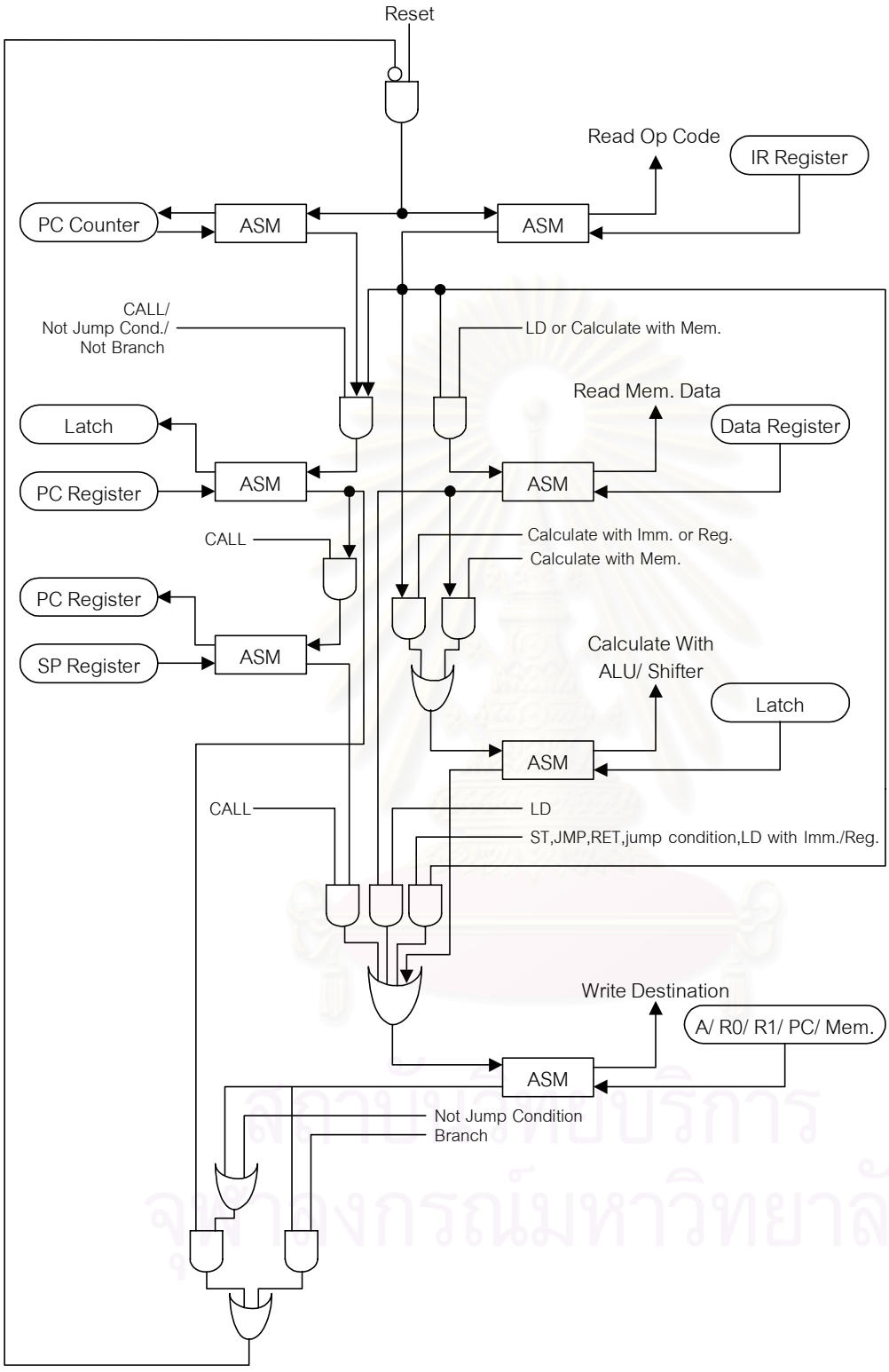


รูปที่ 4.7 การทำงานขนานกันระหว่างการทำงานในขั้นทำงานและการทำงานในขั้นว่าง

ในการออกแบบส่วนวงจรควบคุม จะเริ่มจากการออกแบบโครงสร้างการทำงานของแต่ละคำสั่งในไมโครโปรเซสเซอร์ซึ่งจะแสดงไว้ในภาคผนวก ก แล้วจึงลดรูปเพื่อให้เห็นลำดับขั้นตอนการถอดรหัสคำสั่ง ซึ่งทำให้ง่ายต่อการออกแบบส่วนถอดรหัสคำสั่ง (Instruction Decoder) โดยสามารถแสดงลำดับการถอดรหัสคำสั่งและสร้างสัญญาณควบคุมได้ดังรูปที่ 4.8 และสร้างวงจรควบคุมด้วย ASM ได้ดังรูปที่ 4.9



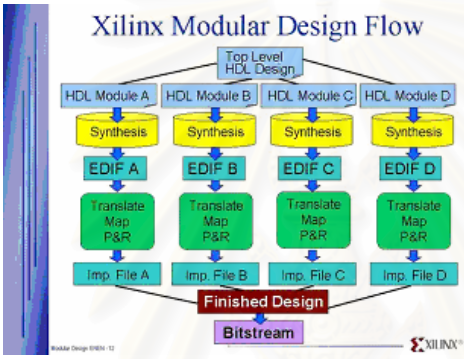
รูปที่ 4.8 ลำดับการถอดรหัสคำสั่ง และสร้างสัญญาณควบคุม



รูปที่ 4.9 การสร้างวงจรควบคุมด้วย ASM

4.7 การสังเคราะห์ และสร้างไมโครโปรเซสเซอร์แบบอสมวารบนเอฟพีจีเอ

เมื่อมีการแก้ไขวงจรที่ออกแบบด้วยภาษาวีเอชดีแอล จะต้องทำการสังเคราะห์ และสร้างเป็นวงจรบนเอฟพีจีเอใหม่ทุกครั้ง ซึ่งอาจจะทำให้ส่วนวงจรเดิมที่ไม่ได้แก้ไข เกิดการเพลสและเราต์สายไม่เหมือนเดิมได้ จึงต้องหาแนวทางที่จะทำให้สามารถสังเคราะห์ และสร้างแต่ละวงจรย่อยในไมโครโปรเซสเซอร์ได้ โดยไม่ทำให้เกิดการเปลี่ยนแปลงในวงจรส่วนอื่น งานวิจัยนี้จึงได้ใช้เครื่องมือที่เรียกว่า Modular Design Tool [15] ซึ่งเป็นเครื่องมือเสริมของบริษัท Xilinx ที่ทำให้สามารถสังเคราะห์ และสร้างแต่ละวงจรย่อยแยกกันก่อน แล้วจึงนำมาต่อกันเป็นวงจรรวมอีกครั้งดังแสดงในรูปที่ 4.10



รูปที่ 4.10 Modular Design Flow

ดังนั้นหลังจากที่ทำการออกแบบวงจรแต่ละส่วนด้วยภาษาวีเอชดีแอลแล้ว จึงนำแต่ละส่วนมาสังเคราะห์ด้วยซอฟต์แวร์ Leonardo Spectrum โดยกำหนดตัวเลือกให้ไม่ทำการ Insert I/O Ports ดังรูปที่ 4.11 เพื่อให้ซอฟต์แวร์ทำการสังเคราะห์แต่ละวงจรย่อยออกมาเป็นโมดูล (Module) เพื่อที่จะนำไปเชื่อมต่อกันในภายหลัง สำหรับการสังเคราะห์วงจรคำนวณและประมวลผลนั้นจะต้องกำหนดตัวเลือกตามที่กล่าวไว้ในบทที่ 3 ด้วย และทำการสังเคราะห์วงจรสองครั้งคือ สังเคราะห์วงจรที่มีเฉพาะส่วนวงจรรางคู่ และสังเคราะห์วงจรที่มีส่วนวงจรตอบรับรวมอยู่ด้วยแล้ว

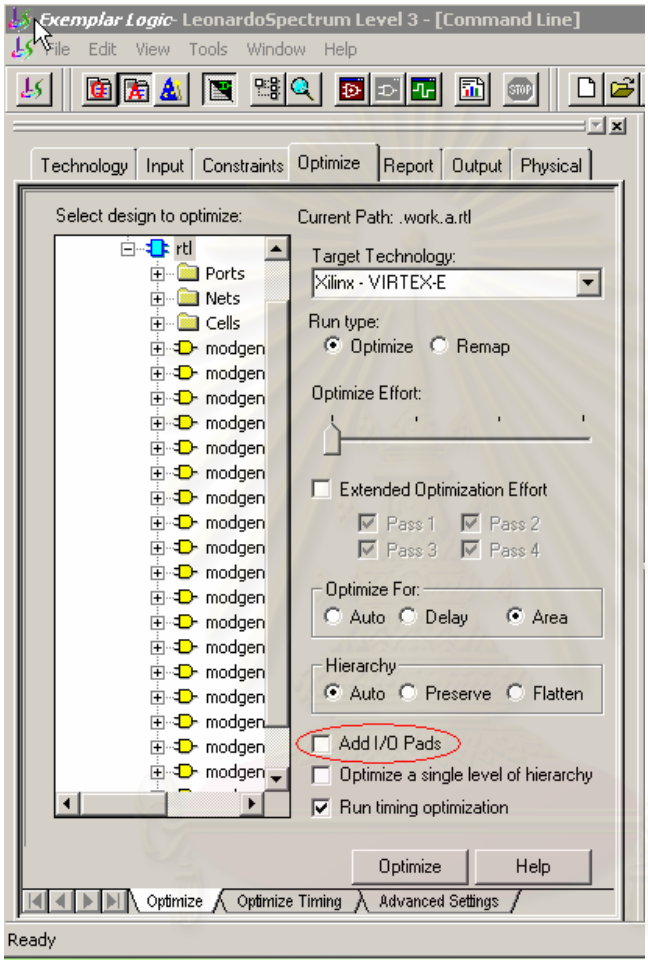
หลังจากนั้นจึงนำแต่ละโมดูลมาทำการสร้างเป็นวงจรด้วยซอฟต์แวร์ Xilinx Foundation โดยมีขั้นตอนหลักสามขั้นตอนคือ Initial Budgeting, Active Module Implementation และ Final Assembly โดยมีรายละเอียดการทำงานดังนี้

1. Initial Budgeting เป็นการกำหนดขนาด, อินพุต, เอาต์พุต และตำแหน่งในการเพลสแต่ละโมดูลดังนี้
   a. สร้างไฟล์ .NGD ของโมดูลบนสุด โดยที่ยังไม่มีข้อมูลของโมดูลย่อย ด้วยคำสั่ง

ngdbuild –modular initial <Top-level Module Name>.edf

b. กำหนดค่า Constraints ซึ่งเป็นการกำหนดอินพุตและเอาต์พุตของวงจรรวม นั่นก็คือการกำหนดอินพุตและเอาต์พุตของไมโครโปรเซสเซอร์ โดยใช้คำสั่ง

constraints_editor <Top-level Module Name>.ngd



รูปที่ 4.11 การกำหนดตัวเลือกในการสังเคราะห์ส่วนวงจรย่อย

c. กำหนดขนาด, อินพุต, เอาต์พุต และตำแหน่งของแต่ละโมดูล ด้วยเครื่องมือชื่อ Floorplanner ซึ่งมีอยู่ในซอฟต์แวร์ Xilinx Foundation ด้วยคำสั่ง

floorplanner <Top-level Module Name>.ngd

2. Active Module Implementation เป็นการสร้างวงจรของแต่ละโมดูลย่อย โดยมีขั้นตอนคือ

a. สำเนาไฟล์สกุล .EDF ของแต่ละโมดูล ไปยังไดเร็กทอรี่ (Directory) ที่แยกกัน

b. สำเนาไฟล์สกุล .UCF ของโมดูลบนสุดที่สร้างไว้ในขั้นตอน Initial Budgeting ไปยังทุกๆ ไดเร็กทอรี่ แล้วทำขั้นตอน c – f เพื่อสร้างแต่ละโมดูลย่อย

c. สร้างไฟล์สกุล .NGD ของโมดูลย่อยโดยใช้คำสั่ง

ngdbuild –modular module –active <Module Name> <Top-level Path Name>\<Top-level Module Name>.ngd

d. ทำการเพลสโมดูลย่อยโดยใช้คำสั่ง

map <Top-level Module Name>.ngd

e. ทำการเราต์สายภายในโมดูลย่อยโดยใช้คำสั่ง

par –w <Top-level Module Name>.ncd <Other Name>.ncd

f. pimcreate –ncd <Other Name>.ncd <Pim Path Name>

กรณีที่ทำการสร้างวงจรคำนวณและประมวลผลนั้น จะเป็นการสร้างเฉพาะส่วนวงจรรางคู่ลงไปก่อน โดยการสำเนาไฟล์สกุล .EDF ที่เป็นของส่วนวงจรรางคู่ลงไปที่ไดเร็กทอรี่ของโมดูลนี้แล้วใช้คำสั่งดังข้างต้น

3. Final Assembly เป็นการรวมทุกโมดูลย่อย แล้วสร้างเป็นวงจรรวม โดยใช้คำสั่งดังนี้

a. ngdbuild –p <FPGA Part Number> -modular assemble –pimpath <Pim Path Name> -use_pim <Module Name> <Top-level Module Name>.ngo

b. map <Top-level Module Name>.ngd

c. par –w <Top-level Module Name>.ncd <Circuit Name>.ncd

จากนั้นจึงทำการสร้างไฟล์เอาต์พุตสกุล .VHD และ .SDF เพื่อใช้ในการพิจารณา และเลือกสายสัญญาณเพื่อสร้างส่วนวงจรตอบรับสำหรับวงจรคำนวณ และประมวลผล โดยใช้คำสั่งดังนี้

1. ngdanno –o <Top-level Module Name>.nga <Circuit Name>.ncd <Top-level Module Name>.ngm
2. ngd2vhdl –w <Top-level Module Name>.nga
3. bitgen <Circuit Name>.ncd –l –w –f bitgen.ut

หลังจากที่ทำการเลือกสายสัญญาณที่เหมาะสม และออกแบบส่วนวงจรตอบรับเพิ่มเติมเข้าไปในส่วนวงจรรางคู่ของหน่วยคำนวณ และประมวลผลตรรกะด้วยภาษาวีเอชดีแอลแล้ว จึงนำวงจรรวมไปสังเคราะห์ แล้วนำกลับมาสร้างเป็นโมดูลอีกครั้ง ซึ่งครั้งนี้จะทำเฉพาะขั้นตอน Active Module Implementation และ Final Assembly โดยเริ่มจากลบไฟล์ในไดเร็กทอรี่ให้เหลือเพียงไฟล์ .UCF และ <Top-level Module Name>.NCD แล้วเปลี่ยนชื่อไฟล์สกุล .NGD ที่มีอยู่ให้เป็น

ชื่อใหม่ จากนั้นสำเนาไฟล์สกุล .EDF ของวงจรรวมที่ได้ทับไฟล์เดิมที่มีอยู่ แล้วจึงทำขั้นตอน Active Module Implementation โดยใช้คำสั่งต่างๆ ดังนี้

1. ngdbuild –modular module –active <Module Name> <Top-level Path Name>\<Top-level Module Name>.ngd
2. map <Top-level Module Name>.ngd
3. par -gf <**ไฟล์สกุล** .NCD **เดิม**>.ncd -gm exact -w -ol 2 <Top-level Module Name>.ncd <Other Name>.ncd
4. pimcreate –ncd <Other Name>.ncd <Pim Path Name>

หลังจากนั้นจึงทำขั้นตอน Final Assembly และสร้างไฟล์ .VHD และ .SDF เพื่อใช้ในการตรวจสอบความถูกต้องของวงจรด้วยการจำลองการทำงานอีกครั้ง สำหรับรายละเอียดการออกแบบแต่ละวงจรย่อยด้วยภาษาวีเอชดีแอลจะแสดงไว้ในภาคผนวก ข และตัวอย่างการสร้างวงจรลงเอฟพีจีเอ โดยใช้ Modular Design Tool สามารถดูได้จากในเวปไซด์ (Web Site) ของ Xilinx [16]

สรุป

งานวิจัยนี้ได้ทำการทดลองสร้างไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิตที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ โดยไมโครโปรเซสเซอร์นี้มีชุดคำสั่งให้ใช้งานทั้งสิ้น 16 คำสั่ง แบ่งออกได้เป็น 3 กลุ่มคือ กลุ่มคำสั่ง Arithmetic and Logic Operation, กลุ่มคำสั่ง Data Transfer Operation และกลุ่มคำสั่ง Program and Machine Control Operation โดยทุกคำสั่งจะมีรหัสดำเนินการยาวเท่ากันหมดคือ 16 บิต สำหรับรีจิสเตอร์แบบอสมวารที่มีให้ใช้งานประกอบด้วย รีจิสเตอร์A, R0 และ R1 และมีสัญญาณบอกสถานะคือ สัญญาณทดค่า กับสัญญาณค่าศูนย์ โดยมีส่วนวงจรที่ทำหน้าที่ประมวลผลคือ วงจรเลื่อนข้อมูล กับวงจรคำนวณและประมวลผลตรรกะ ซึ่งการทำงานเป็นแบบ Accumulator-Based ในการอ้างอิงตัวดำเนินการสามารถทำได้ 4 แบบคือ แบบค่าคงที่, แบบรีจิสเตอร์, แบบอ้างอิงหน่วยความจำโดยตรง และแบบอ้างอิงหน่วยความจำโดยอ้อม

เนื่องจากภายในไมโครโปรเซสเซอร์มีการเข้ารหัสข้อมูลด้วยรหัสรางคู่ ซึ่งใช้สายสัญญาณ 2 เส้นแทนข้อมูลขนาด 1 บิต ในการอ้างอิงหน่วยความจำสำหรับโปรแกรมขนาด 1K * 16 bits และหน่วยความจำสำหรับข้อมูลขนาด 1K * 8 bits จึงต้องมีส่วนที่ทำหน้าที่แปลงสายสัญญาณ 2 เส้นให้เป็น 1 เส้น กับส่วนที่ทำหน้าที่แปลงสายสัญญาณ 1 เส้นให้เป็น 2 เส้น ซึ่งเรียกว่า 2-to-1 Converter และ 1-to-2 Converter ตามลำดับ

เมื่อมีการแก้ไขบางส่วนในวงจร แล้วทำการสังเคราะห์ และสร้างวงจรใหม่ อาจทำให้ส่วนวงจรเดิมที่ไม่ได้แก้ไขมีโครงสร้าง และความหน่วงในวงจรเปลี่ยนไปได้ งานวิจัยนี้จึงทำการสังเคราะห์ และสร้างไมโครโปรเซสเซอร์แบบอสมวารบนเอฟพีจีเอ โดยเริ่มจากการออกแบบแต่ละวงจรย่อยด้วยภาษาวีเอชดีแอล แล้วทำการสังเคราห์และสร้างแต่ละวงจรย่อยให้เป็นโมดูลก่อน จากนั้นจึงนำทุกโมดูลมาต่อกัน แล้วนำไปสร้างเป็นไมโครโปรเซสเซอร์แบบอสมวารบนเอฟพีจีเออีกครั้ง โดยใช้เครื่องมือที่ชื่อว่า Modular Design Tool

บทที่ 5
การทดสอบ

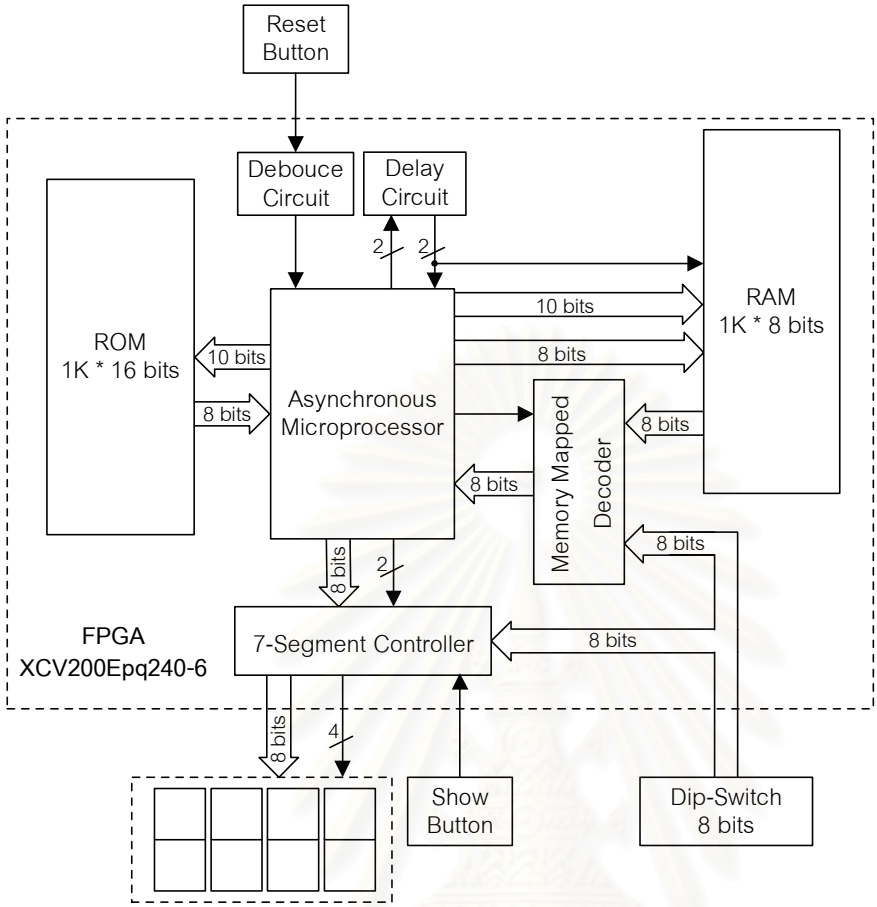
หลังจากที่ทำการออกแบบวงจรทั้งหมดแล้ว จะต้องนำวงจรที่ได้มาจำลองการทำงาน เพื่อตรวจสอบความถูกต้องของไมโครโปรเซสเซอร์ที่ออกแบบ และทำการแก้ไขจนกระทั่งมีความถูกต้องสมบูรณ์เสียก่อน จึงจะทำการโปรแกรมวงจรลงบนเอฟพีจีเอ เพื่อนำไปทดสอบการทำงานจริง โดยในบทนี้จะอธิบายถึงการสร้างตัวแปลโปรแกรม (Compiler) ให้เป็นรหัสดำเนินการ, โครงสร้างของวงจรรวมทั้งหมดที่อยู่ภายในเอฟพีจีเอ XCV200Epq240-6, การสร้างหน่วยความจำสำหรับโปรแกรมและหน่วยความจำสำหรับข้อมูลบนเอฟพีจีเอ เพื่อนำมาใช้ทำงานร่วมกับไมโครโปรเซสเซอร์ที่ออกแบบ, ผลการทดสอบการทำงานโดยการจำลองการทำงาน และการใช้งานจริง รวมทั้งผลการเปรียบเทียบประสิทธิภาพด้านความเร็วในการประมวลผลของไมโครโปรเซสเซอร์แบบอสมวารกับไมโครโปรเซสเซอร์แบบสมวารที่มีโครงสร้าง และชุดคำสั่งให้ใช้งานเหมือนกัน

5.1 การสร้างตัวแปลโปรแกรม

การแปลโปรแกรม (Compile) ที่เขียนขึ้นเพื่อทดสอบการทำงานของไมโครโปรเซสเซอร์ให้เป็นรหัสดำเนินการนั้น จะใช้โปรแกรมที่ชื่อว่า C32 ซึ่งเป็นแอสเซมเบลอร์ (Assembler) ที่เปิดโอกาสให้ผู้วิจัยสามารถสร้างตารางรหัสช่วยจำ (Mnemonic code table) สำหรับไมโครโปรเซสเซอร์ที่ออกแบบได้เองดังแสดงในภาคผนวก ค และจะทำการแปลโปรแกรมที่เขียนขึ้นให้เป็นรหัสดำเนินการ โดยใช้รหัสช่วยจำที่อยู่ในตารางที่สร้างขึ้นนั่นเอง

5.2 โครงสร้างของวงจรรวมภายในเอฟพีจีเอ XCV200Epq240-6

การนำไมโครโปรเซสเซอร์ที่ออกแบบมาตรวจสอบความถูกต้อง จะมีการติดต่อกับอุปกรณ์หรือวงจรภายนอกต่างๆ โดยในการทดสอบนี้จะมีการจำลองวงจรบางส่วนไว้ภายในเอฟพีจีเอซึ่งได้แก่ วงจรสร้างค่าความหน่วง (Delay Circuit), หน่วยความจำสำหรับโปรแกรม, หน่วยความจำสำหรับข้อมูล, วงจรกันผลการเด้ง (Debounce Circuit) ของปุ่มรีเซต และวงจรควบคุมการแสดงผลของ 7-Segment ส่วนอุปกรณ์ภายนอกที่ต่ออยู่กับเอฟพีจีเอจริงๆ ได้แก่ 7-Segment, สวิตช์ขนาด 8 บิต, ปุ่มรีเซต และปุ่มเลือกแสดงค่าของ 7-Segment ดังรูปที่ 5.1



รูปที่ 5.1 โครงสร้างของวงจรรวมที่ออกแบบอยู่ภายในเอฟพีจีXCV200Epq240-6

5.3 การสร้างหน่วยความจำบนเอฟพีจีเอ

งานวิจัยนี้ได้ทำการสร้างหน่วยความจำสำหรับโปรแกรมขนาด 1K * 16 bits และหน่วยความจำสำหรับข้อมูลขนาด 1K * 8 bits ไว้ภายในเอฟพีจีเอด้วย โดยใช้เครื่องมือที่ชื่อว่า Core Generator System ซึ่งมีมาพร้อมกับซอฟต์แวร์ Xilinx โดยในการสร้างหน่วยความจำขึ้นภายในเอฟพีจีเอนั้น จะใช้โครงสร้างแบบ Single Port Block Memory ซึ่งเป็นการใช้พื้นที่ภายในเอฟพีจีเอที่เรียกว่า BLOCKRAMs ที่มีไว้เพื่อสร้างเป็นหน่วยความจำโดยฉพาะ สำหรับเอฟพีจีเอเบอร์ XCV200Epq240-6 นั้นมี BLOCKRAMs ให้ใช้งานทั้งสิ้น 28 BLOCKRAMs โดยถูกใช้ในการสร้างหน่วยความจำสำหรับโปรแกรมไปเป็นจำนวน 4 BLOCKRAMs และถูกใช้ในการสร้างหน่วยความจำสำหรับข้อมูลไปเป็นจำนวน 2 BLOCKRAMs

ในการทดสอบความถูกต้องด้วยการจำลองการทำงาน จะต้องทำการแก้ไขรหัสดำเนินการที่อยู่ในหน่วยความจำสำหรับโปรแกรมให้เป็นรหัสดำเนินการของชุดโปรแกรมที่ต้องการทดสอบด้วยการแก้ไขในไฟล์ .VHD ที่ซอฟต์แวร์ Xilinx Foundation สร้างขึ้นมาหลังจากที่ทำการสร้าง

วงจรรวมทั้งหมดแล้ว โดยจะทำการค้นหาตำแหน่งที่ต้องการ แล้วแทนค่าในตำแหน่งนั้นด้วยค่ารหัสดำเนินการที่อยู่ในไฟล์ .BIN ที่แอสเซมเบลอร์ C32 สร้างขึ้น ก่อนที่จะนำไปจำลองการทำงาน

สำหรับการทดสอบการทำงานจริงบนบอร์ดทดสอบนั้น การแก้ไขค่าในหน่วยความจำสำหรับโปรแกรม ให้เป็นรหัสดำเนินการของชุดโปรแกรมที่ต้องการทดสอบ สามารถทำได้ด้วยการแก้ไขในไฟล์ .EDF ที่ Core Generator System สร้างขึ้นหลังจากที่ทำการสร้างหน่วยความจำสำหรับโปรแกรมแล้ว โดยจะทำการค้นหาตำแหน่งที่ต้องการ แล้วแทนค่าในตำแหน่งนั้นด้วยค่ารหัสดำเนินการที่อยู่ในไฟล์ .BIN ที่แอสเซมเบลอร์ C32 สร้างขึ้น ก่อนที่จะนำไปสังเคราะห์, สร้างเป็นวงจรรวม และโปรแกรมวงจรรวมลงเอฟพีจีเอ

5.4 การทดสอบการทำงาน

การทดสอบการทำงานของไมโครโปรเซสเซอร์ที่ออกแบบนี้ จะแบ่งออกเป็นสองส่วนหลัก คือการทดสอบโดยการจำลองการทำงาน กับการทดสอบการทำงานจริงกับบอร์ดทดสอบ ซึ่งเมื่อได้ผลการจำลองการทำงานที่ถูกต้องแล้ว จึงจะโปรแกรมวงจรลงเอฟพีจีเอ แล้วนำไปทดสอบการทำงานจริงกับบอร์ดทดสอบ

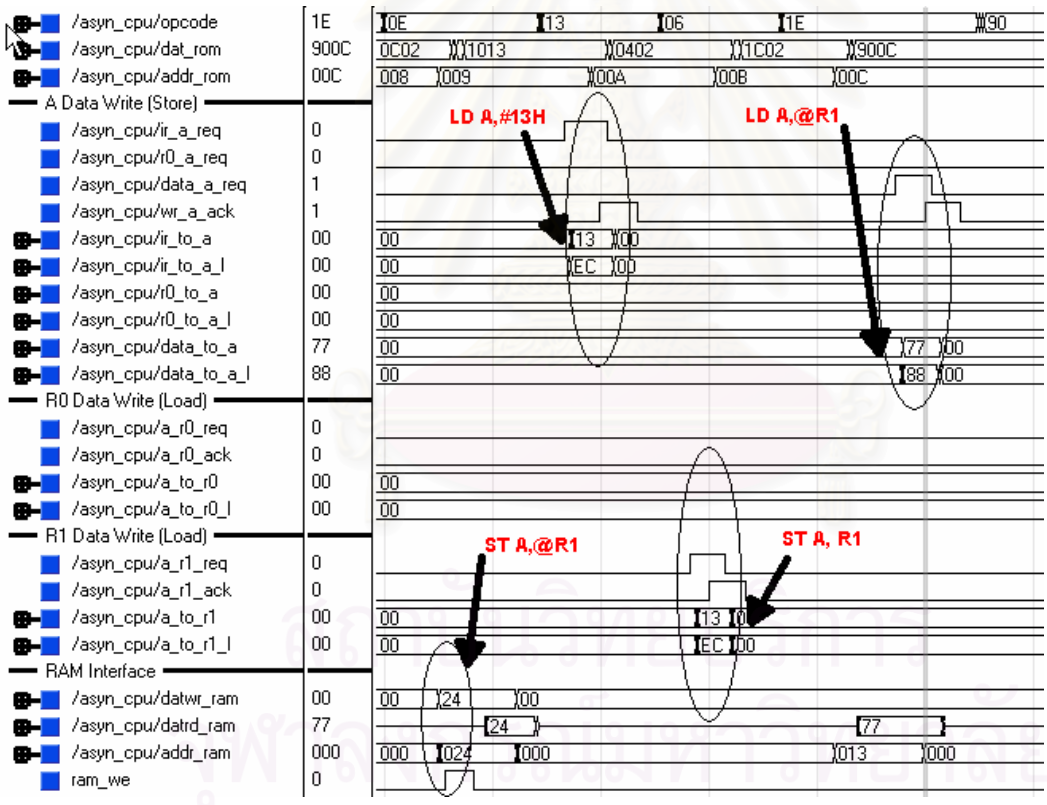
5.4.1 การทดสอบโดยการจำลองการทำงาน

การจำลองการทำงาน จะแบ่งออกเป็นสี่ขั้นตอนคือ การจำลองการทำงานในกลุ่มคำสั่ง Data Transfer Operation, การจำลองการทำงานในกลุ่มคำสั่ง Arithmetic and Logic Operation, การจำลองการทำงานในกลุ่มคำสั่ง Program and Machine Control Operation และการจำลองการทำงานรวมทุกคำสั่ง โดยมีรายละเอียดดังนี้

1. <u>การจำลองการทำงานของกลุ่มคำสั่ง Data Transfer Operation</u> เป็นการตรวจสอบความถูกต้องในการทำคำสั่ง LD และ ST ทุกรูปแบบการใช้งาน โดยมีโปรแกรมทดสอบดังตารางที่ 5.1 และมีผลการจำลองการทำงานดังรูปที่ 5.2

ตารางที่ 5.1 โปรแกรมทดสอบกลุ่มคำสั่ง Data Transfer Operation

```
CPU     "CPU.TBL"
HOF     "BIN8"
WDLN    2
ORG     0000H
LD              A,#77H          ; A = 77H
ST              A,R0            ; R0 = 77H
LD              A,#24H          ; A = 24H
ST              A,R1            ; R1 = 24H
ST              A,@R0           ; M(77H) = 24H
LD              A,R0            ; A = 77H
ST              A,@13H          ; M(13H) = 77H
LD              A,@77H          ; A = 24H
ST              A,@R1           ; M(24H) = 24H
LD              A,#13H          ; A = 13H
ST              A,R1            ; R1 = 13H
LD              A,@R1           ; A = 77H
HALT:
JMP             HALT
END
```

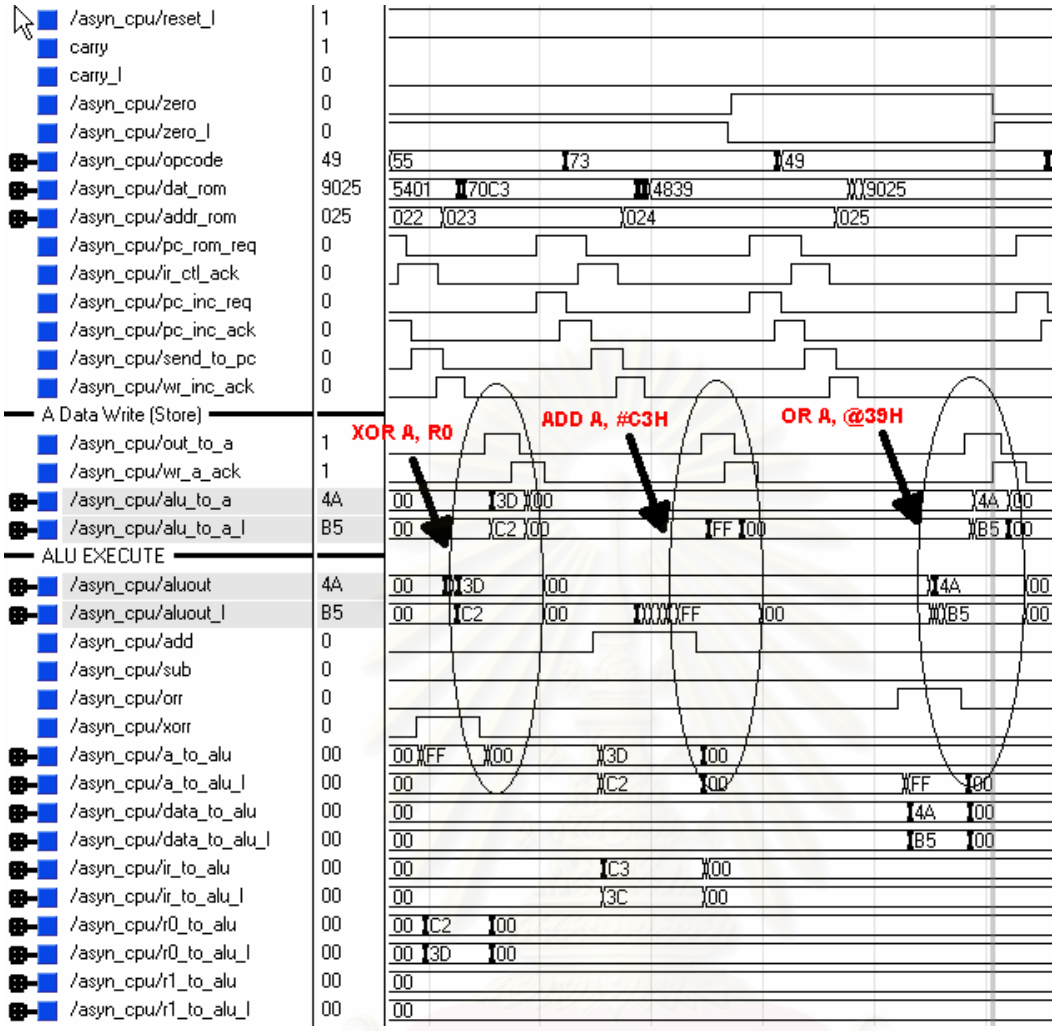


รูปที่ 5.2 ผลการจำลองการทำงานกลุ่มคำสั่ง Data Transfer Operation

2. <u>การจำลองการทำงานของกลุ่มคำสั่ง Arithmetic and Logic Operation</u> เป็นการตรวจสอบความถูกต้องในการทำคำสั่ง AND, OR, XOR, ADD, SUB, SL และ SR ทุกรูปแบบการใช้งาน โดยใช้โปรแกรมทดสอบดังตารางที่ 5.2 และได้ผลการจำลองการทำงานแสดงดังรูปที่ 5.3

ตารางที่ 5.2 โปรแกรมทดสอบกลุ่มคำสั่ง Arithmetic and Logic Operation

```
CPU     "CPU.TBL"
HOF     "BIN8";  "INT8"
WDLN    2

ORG     0000H
ADD             A,#39H          ; A = 39H
ST              A,R0            ; R0 = 39H
AND             A,#13H          ; A       = 11H
ADD             A,R0            ; A = 4AH
ST              A,@R0           ; M(39H) = 4AH
LD              A,#0C8H         ; A = 0C8H
ADD             A,@R0           ; A = 12H, CARRY = 1
ST              A,R1            ; R1 = 12H
OR              A,R0            ; A = 3BH
ST              A,@R1           ; M(12H) = 3BH
OR              A,#0B7H         ; A = 0BFH
ADD             A,@12H          ; A = 0FAH, CARRY = 0
ST              A,@77H          ; M(77H) = FAH
AND             A,@R0           ; A       = 4AH
OR              A,@R1           ; A = 7BH
ST              A,R1            ; R1 = 7BH
AND             A,R0            ; A = 39H
OR              A,@77H          ; A = 0FBH
SUB             A,R1            ; A = 80H, CARRY = 0
AND             A,@39H          ; A = 00H, ZERO = 1
XOR             A,R0            ; A = 39H, ZERO = 0
SUB             A,#77H          ; A = 0C2H, CARRY = 1
ST              A,R0            ; R0 = 0C2H
ST              A,@R1           ; M(7BH) = C2H
XOR             A,@39H          ; A = 88H
SR              A               ; A = 44H
ST              A,@R0           ; M(C2H) = 44H
XOR             A,#0EDH         ; A = A9H
SUB             A,@39H          ; A = 5FH, CARRY = 0
XOR             A,@R1           ; A = 9DH
SUB             A,@R0           ; A = 59H, CARRY = 0
SL              A               ; A = B2H
LD              A,#00H          ; A = 00H
SUB             A,#01H          ; A = FFH, CARRY = 1
XOR             A,R0            ; A = 3D;
ADD             A,#0C3H         ; A = 00H, CARRY = 1
OR              A,@39H          ; A = 4AH
HALT:
JMP             HALT
END
```

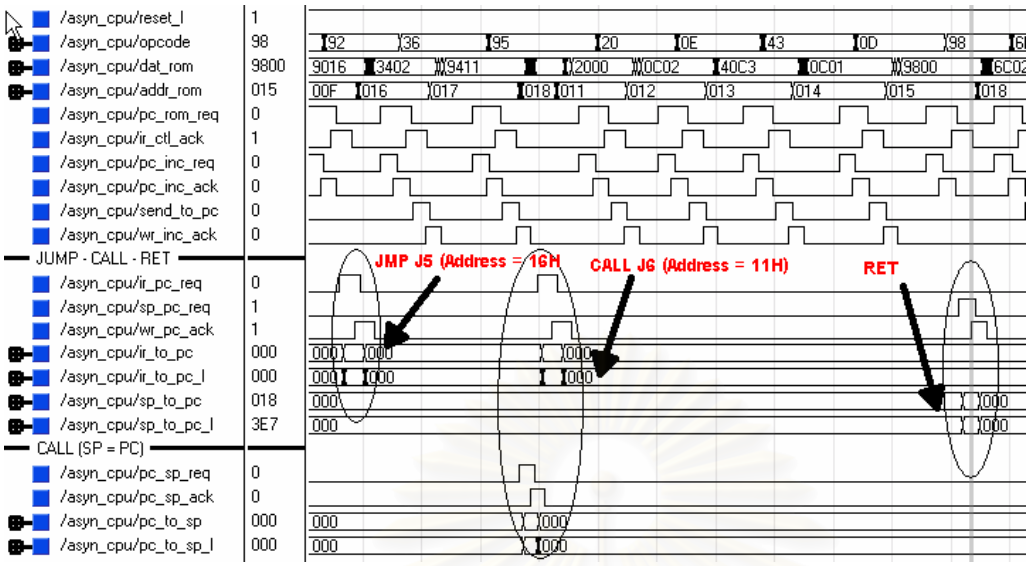


รูปที่ 5.3 ผลการจำลองการทำงานกลุ่มคำสั่ง Arithmetic and Logic Operation

3. <u>การจำลองการทำงานของกลุ่มคำสั่ง Program and Machine Control Operation</u>
เป็นการตรวจสอบความถูกต้องในการทำคำสั่ง JMP, CALL, RET, JC, JNC, JZ และ JNC สำหรับกลุ่มคำสั่งที่กระโดดแบบมีเงื่อนไข (Jump with Condition) นั้นจะทดสอบทั้งในกรณีที่เกิดการกระโดดเพราะสัญญาณสถานะตรงตามเงื่อนไข กับไม่กระโดดไปยังตำแหน่งที่ระบุ เนื่องจากไม่ตรงตามเงื่อนไข โดยมีโปรแกรมทดสอบดังตารางที่ 5.3 และมีผลการจำลองการทำงานแสดงดังรูปที่ 5.4

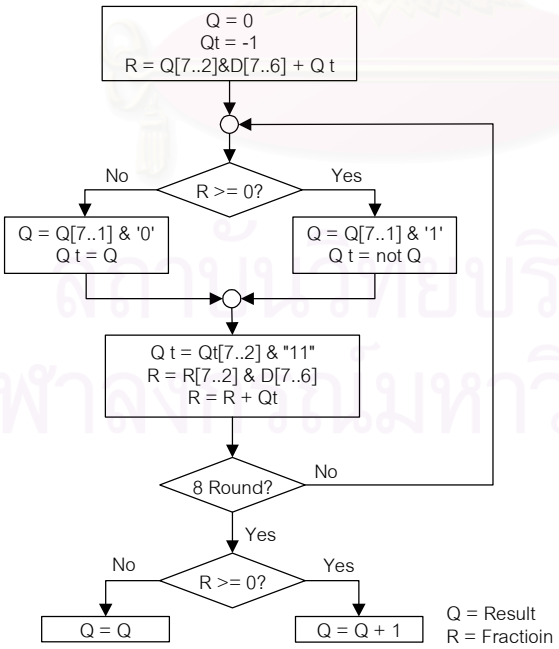
ตารางที่ 5.3 โปรแกรมทดสอบกลุ่มคำสั่ง Program and Machine Control Operation

```
CPU     "CPU.TBL"
HOF     "BIN8"
WDLN    2

ORG     0000H
LD              A,#77H          ; A = 77H
ADD             A,#0A3H         ; A = 1AH, CARRY = 1
JC              J1              ;
J2:
SUB             A,#5BH          ; A = A5H, CARRY = 1
JNC             HALT            ;
ST              A,R0            ; R0 = A5H
SR              A               ; A = 52H
ADD             A,R0            ; A = F7H, CARRY = 0
JNC             J4              ;
J1:
SR              A               ; A = 0DH,
SUB             A,#0DH          ; A = 0, CARRY = 0
JZ              J2              ;
J4:
ST              A,R1            ; R1 = F7H
XOR             A,#29H          ; A = DEH
SUB             A,R0            ; A = 39H, CARRY = 1
JMP             J5              ;
HALT:
JMP             HALT            ;
J6:
SL              A               ; A = 62H
ST              A,@R1           ; M(F7H) = 62H
OR              A,#0C3H         ; A = E3H
ST              A,@R0           ; M(77H) = E3H
RET                             ;
J5:
AND             A,R1            ; A = 31H
CALL             J6
SUB             A,@R1           ; A = 81H, CARRY = 0
JC              HALT            ;
ST              A,@34H          ; M(34)   = 81H
SL              A               ; A = 02H
JNZ             HALT
END
```

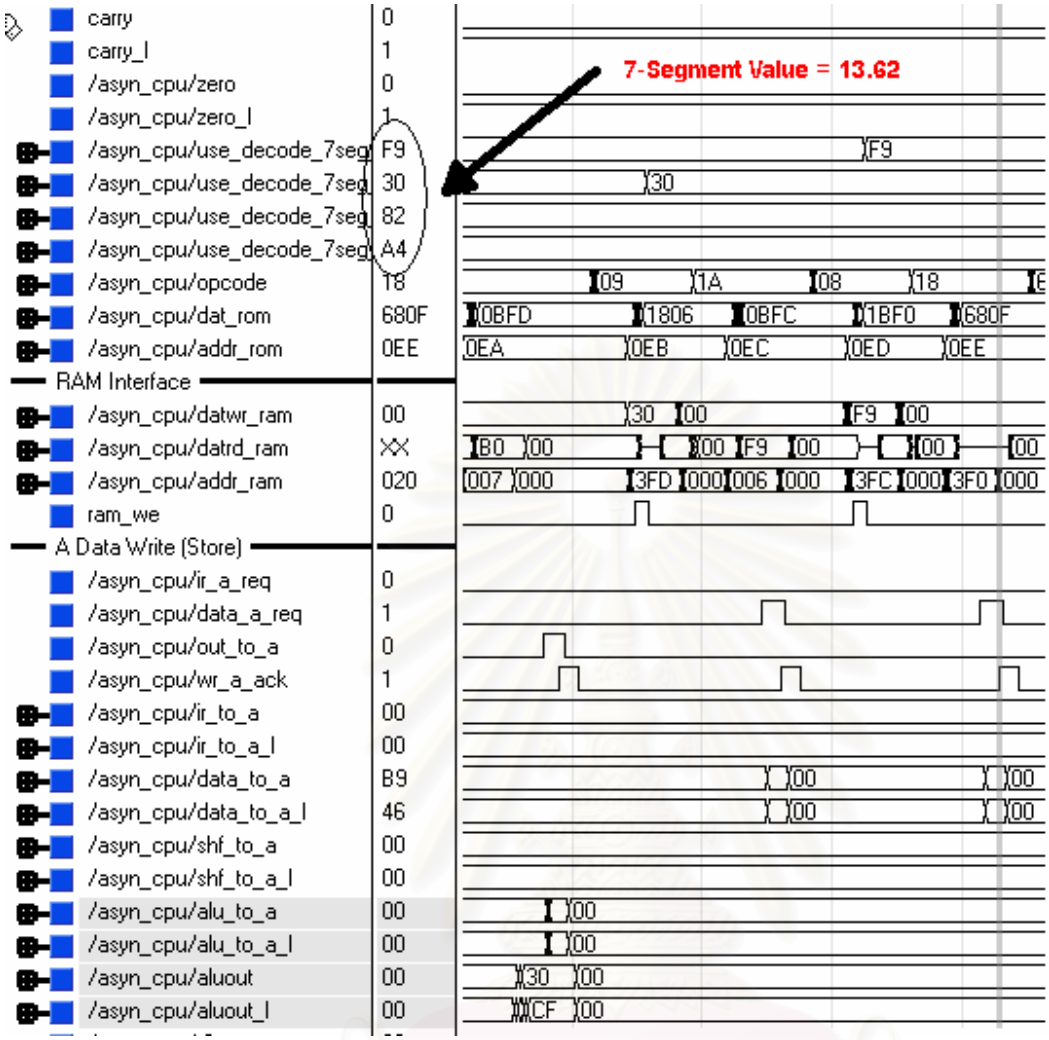


รูปที่ 5.4 ผลการจำลองการทำงานกลุ่มคำสั่ง Program and Machine Control Operation

4. <u>การจำลองการทำงานรวมทุกคำสั่ง</u> เป็นการเขียนโปรแกรมหาค่ารากที่สองด้วยวิธี Non-Restoring Algorithm [17] ดังรูปที่ 5.5 ซึ่งจะใช้ทุกคำสั่ง และแสดงรายละเอียดของโปรแกรมไว้ในภาคผนวก ง จากการทดลองหาค่ารากที่สองของค่า 0xB9 ซึ่งได้ผลลัพธ์เป็นค่า 13.62 และมีค่าเป็น F9, 30, 82, A4 เมื่อส่งไปแสดงผลยัง 7-Segment มีผลการจำลองการทำงานแสดงดังรูปที่ 5.6



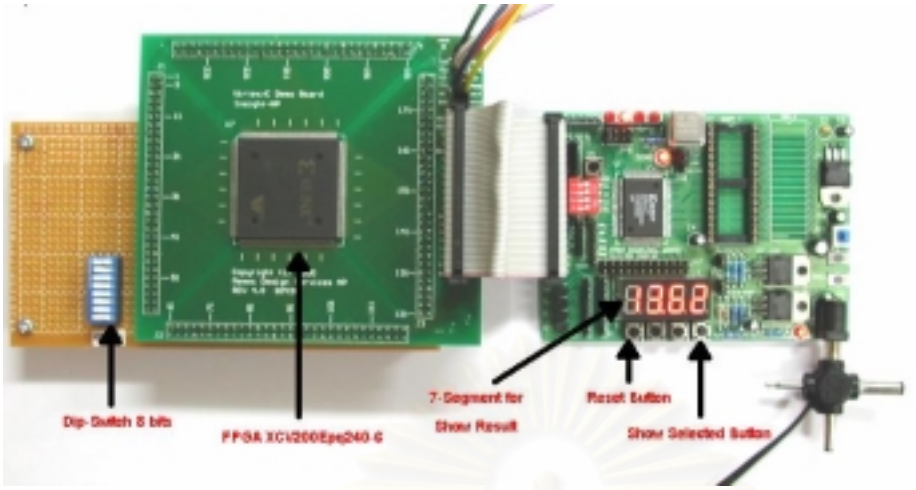
รูปที่ 5.5 หลักการหาค่ารากที่สองด้วยวิธี Non-Restoring Algorithm



รูปที่ 5.6 ผลการจำลองการหาค่ารากที่สองของ 0xB9

5.4.2 การทดสอบการทำงานจริงกับบอร์ดทดสอบ

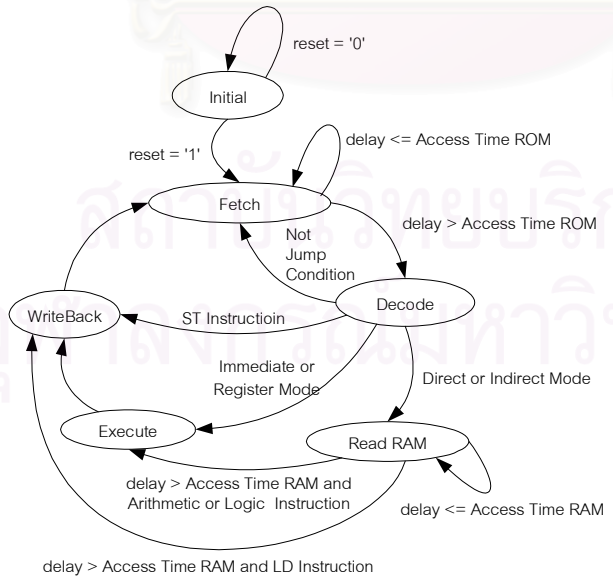
หลังจากที่การทดสอบการทำงานด้วยการจำลองการทำงาน แสดงให้เห็นว่าไมโครโปรเซสเซอร์ และวงจรรอบข้างทั้งหมดมีการทำงานที่ถูกต้องแล้ว จึงทำการโปรแกรมวงจรลงบนเอฟพีจีเอ แล้วนำมาทดสอบการทำงานกับอุปกรณ์ภายนอก ซึ่งจากการทดสอบพบว่าไมโครโปรเซสเซอร์ที่ออกแบบสามารถหาค่ารากที่สองได้ถูกต้องทุกค่า โดยรูปที่ 5.7 เป็นการทดสอบการทำงานจริงของไมโครโปรเซสเซอร์ เพื่อหาค่ารากที่สองของ 0xB9 ซึ่งได้ผลลัพธ์เป็นเลขฐานสิบเท่ากับ 13.62



รูปที่ 5.7 การทำงานจริงกับบอร์ดทดสอบ เพื่อหาค่ารากที่สองของ 0xB9

5.5 การเปรียบเทียบประสิทธิภาพกับไมโครโปรเซสเซอร์แบบสมวาร

ผลการออกแบบไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิตที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอเบอร์ XCV200Epq240-6 พบว่าใช้เนื้อที่ภายในเอฟพีจีเอไปทั้งสิ้น 689 SLICEs หรือประมาณ 106,685 เกต สำหรับการเปรียบเทียบประสิทธิภาพทางด้านความเร็วที่ใช้ในการประมวลผลแต่ละคำสั่งของไมโครโปรเซสเซอร์แบบอสมวารที่ออกแบบ กับไมโครโปรเซสเซอร์แบบสมวารที่มีสถาปัตยกรรม และชุดคำสั่งให้ใช้งานเหมือนกัน ซึ่งมีแผนภาพสถานะ (State Diagram) การทำงานดังรูปที่ 5.8 ได้ผลดังตารางที่ 5.4



รูปที่ 5.8 แผนภาพสถานะการทำงาน ของไมโครโปรเซสเซอร์แบบสมวารที่นำมาเปรียบเทียบ

ตารางที่ 5.4 ผลการเปรียบเทียบความเร็วที่ใช้ในการประมวลผลแต่ละคำสั่ง

กับไมโครโปรเซสเซอร์แบบสมวาร

Memory Frequency		100 MHz				50 MHz				16 MHz			
Type		Async.		Sync.		Async.		Sync.		Async.		Sync.	
Code	Mode	W	T	Cyc.	T	W	T	Cyc.	T	W	T	Cyc.	T
LD	imm.	37	57	3	60	37	67	4	80	37	99	6	120
	reg.	37	57	3	60	37	67	4	80	37	99	6	120
	direct	63	93	4	80	63	113	6	120	63	197	10	200
	indirect	63	93	4	80	63	113	6	120	63	197	10	200
ST	reg.	37	57	3	60	37	67	4	80	37	109	6	120
	direct	41	71	3	60	41	91	5	100	41	175	9	180
	indirect	41	71	3	60	41	91	5	100	41	175	9	180
SL		48	68	4	80	48	78	5	100	48	110	7	140
SR		47	67	4	80	47	77	5	100	47	109	7	140
AND	imm.	60	80	4	80	60	90	5	100	60	122	7	140
	reg.	61	81	4	80	61	91	5	100	61	123	7	140
	direct	96	126	5	100	96	146	7	140	96	230	11	220
	indirect	97	127	5	100	96	146	7	140	96	230	11	220
OR	imm.	66	86	4	80	66	96	5	100	66	128	7	140
	reg.	65	85	4	80	65	95	5	100	65	127	7	140
	direct	93	123	5	100	93	143	7	140	93	227	11	220
	indirect	93	123	5	100	93	143	7	140	93	227	11	220
XOR	imm.	67	87	4	80	67	97	5	100	67	129	7	140
	reg.	64	84	4	80	64	94	5	100	64	126	7	140
	direct	93	123	5	100	93	143	7	140	93	227	11	220
	indirect	94	124	5	100	94	144	7	140	94	228	11	220
SUB	imm.	76	96	4	80	76	106	5	100	76	138	7	140
	reg.	85	105	4	80	85	115	5	100	85	147	7	140
	direct	114	144	5	100	114	164	7	140	114	248	11	220
	indirect	113	143	5	100	113	163	7	140	113	247	11	220
ADD	imm.	73	93	4	80	73	103	5	100	73	135	7	140
	reg.	93	113	4	80	93	123	5	100	93	155	7	140
	direct	108	138	5	100	108	158	7	140	108	242	11	220
	indirect	109	139	5	100	109	159	7	140	109	243	11	220
JZ,JNZ JC,JNC		39	59	3	40	39	69	4	60	39	111	6	100
JZ,JNZ JC,JNC	Not Jump	34	54	2	60	34	64	3	80	34	106	5	120
JMP		39	59	3	60	39	69	4	80	39	101	6	120
CALL		70	90	3	60	70	100	4	80	70	132	6	120
RET		39	59	3	60	39	69	4	80	39	101	6	120

กำหนดให้

W = เวลาที่ใช้ในการประมวลผลเมื่อพิจารณาเฉพาะการทำงานภายในไมโครโปรเซสเซอร์

T = เวลาที่ใช้ในการประมวลผลทั้งหมด (รวมเวลาที่ใช้ในการอ้างอิงกับหน่วยความจำ)

Cyc = จำนวนสัญญาณนาฬิกาที่ใช้

จากตารางผลการเปรียบเทียบประสิทธิภาพด้านความเร็วในการประมวลผลจะเห็นได้ว่า ไมโครโปรเซสเซอร์แบบอสมวารนั้น ในแต่ละคำสั่ง และแต่ละรูปแบบการอ้างอิงตัวดำเนินการ จะใช้เวลาในการประมวลเท่ากันหมด เมื่อพิจารณาเฉพาะเวลาที่ใช้ในการประมวลผลภายใน (W) ไมโครโปรเซสเซอร์ และเวลาที่ใช้ในการประมวลผลทั้งหมด (T) ของไมโครโปรเซสเซอร์แบบอสม-วาร ที่มีค่ามากขึ้นเมื่อใช้หน่วยความจำที่มีความเร็วต่ำลง พบว่าค่าความหน่วงในการประมวลผลทั้งหมดที่เพิ่มขึ้น เป็นค่าความหน่วงที่เกิดจากการรอหน่วยความจำ ซึ่งจะมีค่าความผิดพลาดในการรอไม่นานเกินไปมาก แต่สำหรับไมโครโปรเซสเซอร์แบบสมวาร เวลาที่ใช้ในการรอหน่วยความจำนั้นจะมีค่าความผิดพลาดในการรอนานกว่า เนื่องจากถึงแม้จะสามารถอ้างอิงหน่วยความจำได้แล้ว แต่ก็ยังต้องรอให้มีสัญญาณนาฬิกาเข้ามาก่อน ถึงจะเริ่มทำงานได้

เนื่องจากโครงสร้างภายในของเอฟพีจีเอ รวมทั้งซอฟต์แวร์ที่ช่วยในการออกแบบนั้นยังไม่สนับสนุนการออกแบบวงจรแบบอสมวาร ทำให้ต้องมีข้อกำหนดหลายอย่างในการสังเคราะห์ และสร้างวงจรบนเอฟพีจีเอ ส่งผลให้วงจรที่ได้มีขนาดที่ใหญ่ รวมทั้งมีความหน่วงในวงจรมากกว่าวงจรที่ให้ซอฟต์แวร์เป็นผู้สังเคราะห์ และสร้างเองโดยอัตโนมัติ ดังนั้นเมื่อเปรียบเทียบความเร็วในการประมวลผลแต่ละคำสั่งระหว่างไมโครโปรเซสเซอร์แบบอสมวารกับไมโครโปรเซสเซอร์แบบสม-วาร จะเห็นได้ว่าไมโครโปรเซสเซอร์แบบอสมวารจะมีความเร็วในการประมวลผลแต่ละคำสั่งช้ากว่า โดยเฉพาะอย่างยิ่งเมื่อเป็นคำสั่งในกลุ่ม Arithmetic and Logic Operation แต่เมื่อความเร็วของไมโครโปรเซสเซอร์แบบสมวารมีค่ามากกว่าความเร็วของหน่วยความจำแล้ว ไมโครโปรเซสเซอร์แบบอสมวารจะสามารถประมวลผลคำสั่งส่วนใหญ่ได้เร็วกว่าไมโครโปรเซสเซอร์แบบสมวาร

สรุป

การทดสอบความถูกต้องของไมโครโปรเซสเซอร์แบบอสมวารที่ออกแบบ โดยใช้หน่วยความจำสำหรับโปรแกรมขนาด 1K * 16 bits และหน่วยความจำสำหรับข้อมูลขนาด 1K * 8 bits ที่ถูกสร้างไว้ภายในเอฟพีจีเอนั้นแบ่งออกเป็นสองส่วนหลักคือ การทดสอบความถูกต้องโดยการจำลองการทำงาน และการทดสอบการทำงานจริง โดยเมื่อได้ผลการจำลองการทำงานที่ถูกต้องแล้วจึงทำการทดสอบการทำงานจริงกับบอร์ดทดสอบอีกครั้ง จากผลการทดลองหาค่ารากที่สองโดยใช้ไมโครโปรเซสเซอร์ที่ออกแบบ พบว่า ไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิตที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ที่อยู่ภายในเอฟพีจีเอนี้สามารถทำงานได้ถูกต้อง

เมื่อเปรียบเทียบประสิทธิภาพกับไมโครโปรเซสเซอร์แบบสมวาร ที่มีโครงสร้าง และชุดคำสั่งเหมือนกัน พบว่า เวลาที่ใช้ในการประมวลแต่ละคำสั่งของไมโครโปรเซสเซอร์แบบอสมวารส่วนใหญ่จะใช้เวลามากกว่า โดยเฉพาะอย่างยิ่งเมื่อเป็นคำสั่งที่อยู่ในกลุ่ม Arithmetic and Logic

Operation แต่เมื่อใช้หน่วยความจำที่มีความเร็วต่ำกว่าความเร็วของไมโครโปรเซสเซอร์แบบสม-วารมากขึ้น ไมโครโปรเซสเซอร์แบบอสมวารที่ออกแบบนั้นจะสามารถประมวลผลคำสั่งส่วนใหญ่ได้เร็วกว่าไมโครโปรเซสเซอร์แบบสมวาร

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

งานวิจัยนี้มีวัตถุประสงค์คือ นำเสนอแนวทางการออกแบบวงจรแบบอสมวารให้สามารถสังเคราะห์ และสร้างเป็นวงจรบนเอฟพีจีเอ เพื่อเป็นแนวทางเริ่มต้นในการสร้างวงจรแบบอสมวารขึ้นมาใช้งาน และตรวจสอบความถูกต้องของวงจรก่อนที่จะนำไปผลิตเป็นชิพจริง โดยวงจรที่ทดลองออกแบบในงานวิจัยนี้คือ ไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิต ซึ่งภายในประกอบด้วยหน่วยคำนวณ และประมวลผลตรรกะ ที่ใช้แบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้

จากการศึกษาวิธีการออกแบบวงจรแบบอสมวารพบว่า ข้อจำกัดที่สำคัญที่มีผลต่อการออกแบบวงจรแบบอสมวารบนเอฟพีจีเอ ซึ่งผู้ออกแบบจะต้องคำนึงถึงคือ

1. การกำหนดค่าความหน่วงประมาณให้กับเกต และสายในวงจรไม่สามารถทำได้
2. เนื่องจากวงจรแบบอสมวารไม่ใช้สัญญาณนาฬิกาควบคุมการทำงาน ดังนั้นการดีคอมโพซิชั่น ซึ่งเป็นขั้นตอนหนึ่งในการทำให้เกิดเป็นวงจรบนเอฟพีจีเอ อาจทำให้วงจรแบบอสมวารที่ได้มีการทำงานที่ผิดพลาดไปจากวงจรเดิมได้
3. ทุกครั้งที่มีการแก้ไขวงจร แล้วนำไปสร้างเป็นวงจรบนเอฟพีจีเอ ซอฟต์แวร์ที่ช่วยในด้านการออกแบบ จะทำการเพลสและเราต์สายในวงจรทั้งหมดใหม่ทุกครั้ง ซึ่งอาจทำให้ส่วนวงจรเดิมที่ไม่ได้แก้ไข มีโครงสร้าง และการทำงานที่เปลี่ยนไปได้

ดังนั้นจึงต้องหาแนวทางการออกแบบวงจรแบบอสมวารที่จะทำให้สามารถสังเคราะห์และสร้างเป็นวงจรบนเอฟพีจีเอ เพื่อนำมาใช้งานได้จริง โดยงานวิจัยนี้เริ่มจากการศึกษาแนวทางการออกแบบวงจรเชิงผสมแบบอสมวารที่มีแบบจำลองความหน่วงที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ เพื่อนำไปใช้ในการออกแบบหน่วยคำนวณ และประมวลผลตรรกะ ซึ่งจะแบ่งการออกแบบออกเป็นสองส่วนคือ การออกแบบส่วนวงจรรางคู่ที่ทำงานตามฟังก์ชันตรรกะ และการออกแบบส่วนวงจรตอบรับที่ทำหน้าที่ตรวจสอบการสิ้นสุดของการเปลี่ยนระดับสัญญาณของทั้งวงจร

แนวทางในการออกแบบส่วนวงจรรางคู่บนเอฟพีจีเอที่งานวิจัยนี้ได้นำเสนอ เริ่มจากการสร้างแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ แล้วแปลงให้เป็นส่วนวงจรรางคู่ และใช้ภาษาวีเอชดีแอลเขียนอธิบายการทำงานของวงจรในระดับอาร์ทีแอล โดยให้แต่ละกลุ่มวงจร

ย่อยมีสัญญาณอินพุตได้ไม่เกิน 4 อินพุต และเชื่อมต่อกันด้วยเกตแอนด์เพียงอย่างเดียว, เกตออร์เพียงอย่างเดียว หรือเกตแอนด์กับเกตออร์ที่ต่อกันอยู่ในรูปของผลรวมของผลคูณ จากนั้นนำวงจรที่ออกแบบได้ไปสังเคราะห์ด้วยซอฟต์แวร์ Leonardo Spectrum โดยต้องกำหนดตัวเลือกไม่ทำ Pre-Optimization และทำ Preserve Signal ทุกชื่อสัญญาณในวงจร เพื่อให้วงจรมีโครงสร้างตามที่ออกแบบ ทำให้วงจรที่ได้หลังจากการทำดี-คอมโพซิชั่นไม่มีการทำงานที่ผิดพลาด จากนั้นจึงนำวงจรที่สังเคราะห์แล้วไปสร้างเป็นวงจรรางคู่บนเอฟพีจีเอด้วยซอฟต์แวร์ Xilinx Foundation โดยกำหนดตัวเลือกให้มี Simulation Option เป็น Modelsim VHDL และทำ Correlate Simulation Data to Input Design เพื่อสร้างไฟล์เอาต์พุตที่แสดงให้เห็นโครงสร้าง และค่าความหน่วงประมาณของแต่ละลุคอัพเทเบิ้ลภายในส่วนวงจรรางคู่ รวมทั้งเป็นการกำหนดให้สัญญาณภายในวงจรยังคงมีชื่อสัญญาณดังเดิม เพื่อให้ง่ายต่อการสร้างส่วนวงจรตอบรับ

ในการสร้างส่วนวงจรตอบรับสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้บนเอฟพีจีเอ งานวิจัยนี้ได้นำเสนอแนวทางการออกแบบโดยเริ่มจากการนำไฟล์เอาต์พุตที่ได้จากการสร้างส่วนวงจรรางคู่ มาวิเคราะห์ฟังก์ชันภายใน และค่าความหน่วงประมาณของแต่ละลุคอัพเทเบิ้ล แล้วเลือกกลุ่มสายสัญญาณภายในส่วนวงจรรางคู่ที่สามารถครอบคลุมการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 ของทุกสัญญาณในวงจร โดยแบ่งการเลือกสายได้เป็นสามแบบคือ

1. หากภายในลุคอัพเทเบิ้ลเชื่อมต่อกันด้วยเกตแอนด์เพียงอย่างเดียว จะเลือกสายอินพุตของลุคอัพเทเบิ้ลที่มีค่าความหน่วงสูงสุด
2. หากภายในลุคอัพเทเบิ้ลมีเพียงเกตออร์ จะเลือกสายเอาต์พุตของลุคอัพเทเบิ้ล
3. หากฟังก์ชันภายในเป็นการต่อกันของเกตแอนด์และเกตออร์ ซึ่งอยู่ในรูปของผลรวมของผลคูณ จะเลือกสายเอาต์พุตของลุคอัพเทเบิ้ล ร่วมกับสายอินพุตของแต่ละฟังก์ชันแอนด์ที่มีค่าความหน่วงสูงสุด

จากนั้นจึงใช้เกตออร์รวมกลุ่มสายสัญญาณที่เลือก เพื่อสร้างสัญญาณแสดงความบริบูรณ์ และเขียนอธิบายการทำงานดังกล่าวด้วยภาษาวีเอชดีแอล แล้วนำไปสังเคราะห์และสร้างเป็นวงจรเชิงผสมแบบอสมวารบนเอฟพีจีเอ โดยนอกเหนือจากการกำหนดตัวเลือกต่างๆ เช่นเดียวกับการออกแบบส่วนวงจรรางคู่แล้ว ในขั้นตอนการสร้างวงจรบนเอฟพีจีเอจะต้องกำหนด Guide File และ Mapping File ซึ่งเป็นรูปแบบการเพลสและเราต์ส่วนวงจรรางคู่ที่เคยออกแบบไว้ให้กับซอฟต์แวร์ เพราะการทำให้เกิดเป็นวงจรทั้งหมดอีกครั้ง อาจทำให้ส่วนวงจรรางคู่ที่ถูกเพลสและเราต์ใหม่มีโครงสร้าง และค่าความหน่วงเปลี่ยนไป ส่งผลให้กลุ่มสายสัญญาณที่เลือกไว้ไม่สามารถครอบคลุมทุกการเปลี่ยนระดับสัญญาณในวงจร

จากการทดลองออกแบบหน่วยคำนวณ และประมวลผลตรรกะที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ด้วยวิธีที่นำเสนอ และทดสอบการทำงานโดยทดลองสองรูปแบบคือ เมื่อความแปรปรวนความหน่วงในวงจรมีค่ามากกว่าอัตราส่วนความแปรปรวนความหน่วงสูงสุด และเมื่อความแปรปรวนความหน่วงในวงจรมีค่าเท่ากับอัตราส่วนความแปรปรวนความหน่วงสูงสุด พบว่าวงจรที่ออกแบบมีความทนทานต่อความแปรปรวนในระดับที่ไม่เกินขอบเขตของอัตราส่วนความแปรปรวนความหน่วงสูงสุดของวงจร และสามารถทำงานได้ถูกต้อง

การวิจัยในขั้นต่อมาจึงเป็นทดลองออกแบบวงจรที่มีขนาดใหญ่ขึ้น โดยทดลองสร้างไมโครโปรเซสเซอร์แบบอสมวารขนาด 8 บิตบนเอฟพีจีเอเบอร์ XCV200Epq240-6 ซึ่งสามารถอ้างอิงหน่วยความจำสำหรับโปรแกรมขนาด 16 บิตได้ 1 กิโลตำแหน่ง และอ้างอิงหน่วยความจำสำหรับข้อมูลขนาด 8 บิตได้ 1 กิโลตำแหน่ง โดยมีสถาปัตยกรรมภายในประกอบหน่วยคำนวณและประมวลผลตรรกะที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งมีการทำงานเป็นแบบ Accumulator Based และมีรีจิสเตอร์ให้ใช้งานสามตัวได้แก่ รีจิสเตอร์ A, R0 และ R1 โดยมีชุดคำสั่งให้ใช้งานทั้งสิ้น 16 คำสั่ง และสามารถอ้างอิงตัวดำเนินการได้สี่แบบคือ แบบค่าคงที่, แบบรีจิสเตอร์, แบบอ้างอิงหน่วยความจำโดยตรง และแบบอ้างอิงหน่วยความจำโดยอ้อมด้วยรีจิสเตอร์

ทุกครั้งที่มีการแก้ไขวงจรที่ออกแบบด้วยภาษาวีเอชดีแอล ซอฟต์แวร์ที่ใช้ช่วยในสร้างวงจรบนเอฟพีจีเอจะทำการเพลส และเราต์สายในวงจรทั้งหมดใหม่ทุกครั้ง และเนื่องจากการออกแบบหน่วยคำนวณ และประมวลผลตรรกะจะต้องทำการส่วนวงจรรางคู่ก่อน จากนั้นจึงทำการออกแบบส่วนวงจรตอบรับเพิ่มเติมเข้าไป แล้วนำไปสร้างเป็นวงจรเชิงผสมแบบอสมวารอีกครั้ง จึงอาจทำให้ส่วนวงจรอื่นๆ ภายในไมโครโปรเซสเซอร์ที่เคยออกแบบไว้มีโครงสร้างที่เปลี่ยนไป และเกิดการทำงานที่ผิดพลาดได้ งานวิจัยนี้จึงได้นำเสนอแนวทางการสร้างไมโครโปรเซสเซอร์บนเอฟพีจีเอโดยใช้เครื่องมือช่วยที่เรียกว่า Modular Design Tool ซึ่งจะทำให้สามารถที่จะออกเป็นแต่ละส่วนย่อยภายในไมโครโปรเซสเซอร์ด้วยภาษาวีเอชดีแอล แล้วนำไปสังเคราะห์ และสร้างเป็นโมดูลต่างๆ ก่อน จากนั้นจึงต่อทุกโมดูลเข้าด้วยกันเป็นไมโครโปรเซสเซอร์อีกครั้ง โดยไม่ทำให้เกิดการเปลี่ยนแปลงในวงจรส่วนเดิมที่มีอยู่

จากผลการจำลองการทำงาน รวมทั้งนำไปทดสอบการทำงานจริงบนบอร์ดทดสอบ โดยการหาค่ารากที่สองของอินพุตที่รับมาจากสวิตซ์บนบอร์ด และแสดงค่าผลลัพธ์บน 7-Segment พบว่าไมโครโปรเซสเซอร์แบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ ซึ่งออกแบบบน

เอฟพีจีเอนี้สามารถทำงานได้ถูกต้อง และใช้พื้นที่ภายในเอฟพีจีเอของบริษัท Xilinx เบอร์ XCV200Epq240-6 ไปทั้งสิ้น 29% หรือประมาณ 106,685 เกต

เมื่อทำการเปรียบเทียบประสิทธิภาพของไมโครโปรเซสเซอร์แบบอสมวารที่ออกแบบกับไมโครโปรเซสเซอร์แบบสมวารที่มีชุดคำสั่ง และสถาปัตยกรรมเดียวกัน พบว่าไมโครโปรเซสเซอร์แบบอสมวารจะมีความเร็วในการประมวลผลแต่ละคำสั่งช้ากว่า โดยเฉพาะอย่างยิ่งเมื่อเป็นคำสั่งในกลุ่ม Arithmetic and Logic Operation ซึ่งมีสาเหตุมาจากโครงสร้างของเอฟพีจีเอ รวมทั้งซอฟต์แวร์ที่ช่วยในการออกแบบนั้นยังไม่สนับสนุนการออกแบบวงจรแบบอสมวาร ทำให้ต้องมีข้อกำหนดหลายอย่างในการสังเคราะห์ และสร้างวงจรบนเอฟพีจีเอ ส่งผลให้วงจรที่ได้มีขนาดที่ใหญ่ รวมทั้งมีความหน่วงในวงจรมากกว่าวงจรที่ให้ซอฟต์แวร์เป็นผู้สังเคราะห์ และสร้างวงจรเองโดยอัตโนมัติ อย่างไรก็ตามเมื่อความเร็วของไมโครโปรเซสเซอร์แบบสมวารมีค่ามากกว่าความเร็วของหน่วยความจำมากๆ แล้ว มีแนวโน้มว่าไมโครโปรเซสเซอร์แบบอสมวารจะสามารถประมวลผลคำสั่งส่วนใหญ่ได้เร็วกว่าไมโครโปรเซสเซอร์แบบสมวาร

ผลการวิจัยแสดงให้เห็นว่า การสร้างวงจรแบบอสมวารบนเอฟพีจีเอสามารถทำได้จริง จึงเป็นแนวทางเริ่มต้นสำหรับการออกแบบ และพัฒนาวงจรแบบอสมวารโดยใช้เอฟพีจีเอ รวมทั้งตรวจสอบความถูกต้องของวงจรก่อนที่จะนำไปผลิตเป็นชิพจริง ซึ่งจะทำให้สามารถลดค่าใช้จ่ายในการตรวจสอบความถูกต้อง และค่าใช้จ่ายในการผลิตชิพที่สูญเปล่า เนื่องจากยังมีข้อผิดพลาดในวงจรลงได้ เนื่องจากการทำงานส่วนใหญ่เป็นการทำงานอยู่บนเครื่องคอมพิวเตอร์ส่วนบุคคล และเอฟพีจีเอก็เป็นอุปกรณ์ที่สามารถนำมาโปรแกรมให้เป็นวงจรต่างๆ ได้หลายครั้งนั่นเอง

6.2 ข้อเสนอแนะ

วิธีการออกแบบวงจรแบบอสมวารที่นำเสนอนั้น จะเป็นการออกแบบวงจรในระดับเกตโดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ แล้วใช้ภาษาวีเอชดีแอลเขียนอธิบายการทำงาน ซึ่งจะไม่มีการใช้ฟลิปฟล็อป รวมทั้งอุปกรณ์รวมส่งสัญญาณที่อยู่ภายในแต่ละลอจิกบล็อคเลย จึงเป็นการใช้พื้นที่ภายในเอฟพีจีเอไปอย่างไม่ได้ประโยชน์สูงสุด ซึ่งวิธีการที่นำเสนอนี้น่าจะเหมาะที่จะนำไปใช้ในการออกแบบบนซีพีแอลดีซึ่งมีโครงสร้างภายในเป็นแบบแมโครเซลล์มากกว่า แต่ก็จะสามารถออกแบบ และสร้างวงจรแบบอสมวารที่มีขนาดเล็กเท่านั้น เพราะซีพีแอลดีที่ใหญ่ที่สุดมีขนาดเพียง 12,500 เกตเท่านั้น ซึ่งไม่เพียงพอต่อการนำมาสร้างเป็นไมโครโปรเซสเซอร์แบบอสมวาร เนื่องจากผลการทดลองออกแบบไมโครโปรเซสเซอร์แบบอสมวาร 8 บิต ที่มีชุดคำสั่งเพียง 16 คำสั่ง ซึ่งเป็นไมโครโปรเซสเซอร์ที่เล็กมาก ก็มีขนาด 106,685 เกตแล้ว

จากการทดลองออกแบบไมโครโปรเซสเซอร์แบบอสมวารที่มีการทำงานเป็นแบบตามลำดับ (Sequential) พบว่า สิ่งแวดล้อมมีค่าความหน่วงมากกว่าค่าความหน่วงภายในวงจรมาก จึงทำให้ค่าระดับสัญญาณภายในส่วนวงจรเชิงผสมแบบอสมวาร หรือหน่วยคำนวณ และประมวลผลตรรกะ มีการเปลี่ยนระดับสัญญาณจาก 1 เป็น 0 เสร็จสิ้นหมดก่อนที่จะได้รับอินพุตชุดใหม่เข้ามาอย่างแน่นอน นั่นคือ หากเป็นการออกแบบไมโครโปรเซสเซอร์แบบอสมวารที่มีการทำงานเป็นแบบตามลำดับ และสิ่งแวดล้อมมีค่าความหน่วงมากกว่าค่าความหน่วงในวงจรแล้ว ไม่จำเป็นต้องมีส่วนวงจรตอบรับในวงจรเชิงผสมแบบอสมวาร และแนวทางการพัฒนาต่อไปที่ควรจะทำก็คือ การออกแบบไมโครโปรเซสเซอร์แบบอสมวารที่มีการทำงานเป็นแบบกระจายการทำงาน (Pipeline) ให้สามารถสังเคราะห์ และสร้างเป็นวงจรบนเอฟพีจีเอ เพื่อนำมาใช้งานได้จริง และทำการเปรียบเทียบประสิทธิภาพกับไมโครโปรเซสเซอร์แบบสมวารที่มีการทำงานเป็นแบบกระจายการทำงานเหมือนกัน เพื่อศึกษาว่าไมโครโปรเซสเซอร์แบบอสมวารมีความเร็วในการทำงานสูงหรือต่ำกว่าอย่างไร

เนื่องจากวงจรแบบอสมวารไม่มีการใช้สัญญาณนาฬิกาควบคุมการทำงาน ดังนั้นวงจรจึงเกิดการทำงานก็ต่อเมื่อสายสัญญาณในวงจรมีการเปลี่ยนแปลง ดังนั้นพลังงานที่ใช้ภายในวงจรจึงมีค่าต่ำว่าวงจรแบบสมวาร อย่างไรก็ตามงานวิจัยนี้ไม่ได้มีการทดลองวัดค่าพลังงานที่ใช้ เนื่องจากโครงสร้างภายในของเอฟพีจีเอจะมีการใช้พลังงานอยู่ตลอดเวลา รวมทั้งขาดเครื่องมือที่เหมาะสม ดังนั้นแนวทางการพัฒนาที่ควรจะทำอีกทางหนึ่งคือ หาแนวทางการวัดค่าพลังงานที่ใช้ภายในวงจรแบบอสมวารซึ่งออกแบบบนเอฟพีจีเอ แล้วนำมาเปรียบเทียบกับพลังงานที่ใช้ภายในวงจรแบบสมวารซึ่งออกแบบบนเอฟพีจีเอเช่นเดียวกัน เพื่อพิสูจน์ว่าวงจรแบบอสมวารมีการใช้พลังงานที่ต่ำกว่ามากน้อยเพียงใด

รายการอ้างอิง

1. Hauck, S. Asynchronous design methodologies: An overview. IEEE Transaction on Computer 83, 1 (January 1995): 69-93.
2. Xilinx Incorporation. The Programmable Logic Data Book 2000. United State of America, 2000.
3. Bhasker, J. A VHDL Primer. Prentice-Hall, 1992.
4. Park, S. B. Synthesis of Asynchronous VLSI Circuits from Signal Transition Graph Specifications. Doctoral dissertation, Department of Engineering-Computer Science, Tokyo Institute of Technology (TIT), 1996.
5. Martin, A. J. The Limitations to Delay-Insensitivity in Asynchronous Circuits. Proceedings on Advanced Research in VLSI 6 (1990): 263-278.
6. Nanya, T.; Ueno, Y.; Kagontani, H.; Kuwako, M; and Takamura, A. TITAC: Design of a Quasi-Delay-Insensitive Microprocessor. IEEE Design & Test of Computers 11, 2 (1994): 50-63.
7. Takamura, A.; Kuwako, M.; Imai, M.; Fujii, T.; Ozawa, M.; Fukasaku, I.; Ueno, Y.; Nanya, T. TITAC-2: An asynchronous 32-bit microprocessor based on Scalable-Delay-Insensitive model. Proceeding of ICCD, IEEE (October 1997): 288-294.
8. Muller, D.E.; Bartky, W.S. A Theory of Asynchronous Circuits. Proceedings on Theory and Switching (1959): 204-243.
9. Aker, S.B. Binary Decision Diagrams. IEEE Trans. On Computer 6, Vol. C-27 (1978): 509-516.
10. รัชดา นุตจรัส. การออกแบบวงจรตอบรับที่ไร้อุปกรณ์ชนิดซีสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้.หลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2543.
11. Hauck, S.; Burns, S.; Borriello, G.; Ebeling, C. An FPGA for Implementing Asynchronous Circuit. IEEE Design & Test of Computer Vol.11, No. 3, (1944): 60-69.
12. Exemplar Logic Incorporation. [Online], Available from: http://www.exemplar.com., [2002, Feb].

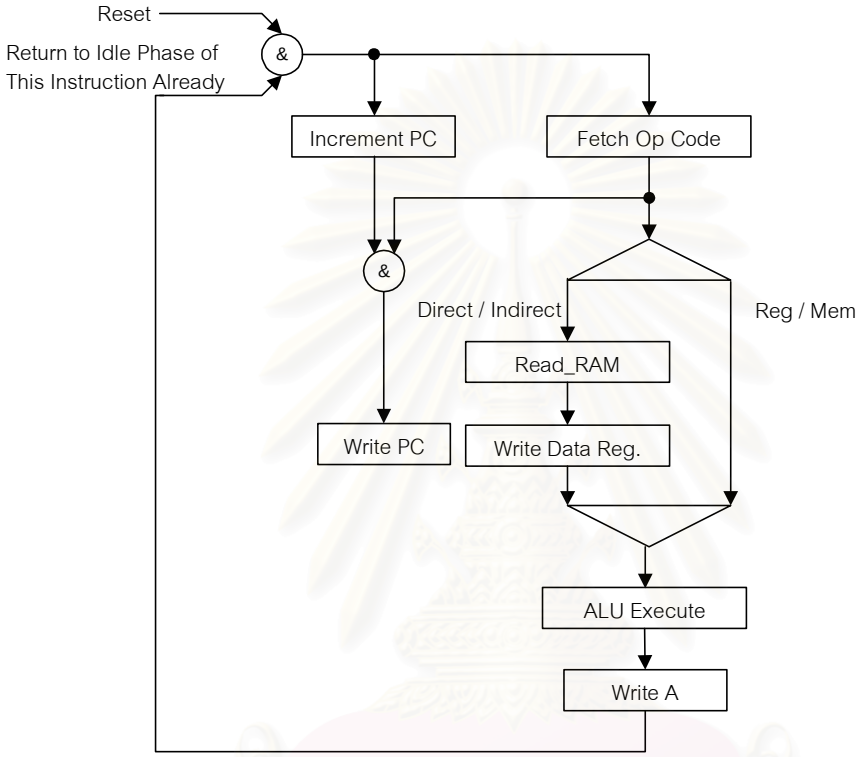
13. Xilinx Incorporation. [Online], Available from: http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+Foundation., [2002, Feb].
14. Model Technology. [Online], Available from: http://www.model.com., [2002, Feb].
15. Xilinx Incorporation. [Online], Available from: http://www.xilinx.com/products/software/moddes/moddes.htm., [2002, Feb].
16. Xilinx Incorporation. [Online], Available from: http://www.xilinx.com/xapp/xapp404.pdf., [2002, Feb].
17. Li, Y.; Chu, W. Implementation of Single Precision Floating Point Square Root on FPGAs. Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines. (1997): 226-234.

ภาคผนวก

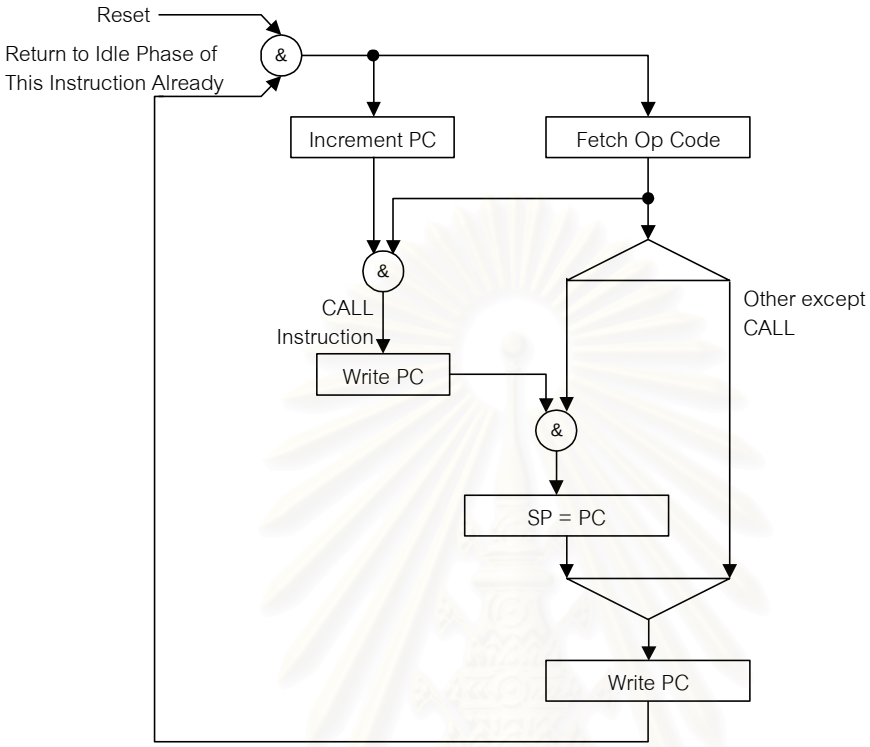
ภาคผนวก ก

โครงสร้างลำดับการทำงานของแต่ละคำสั่ง

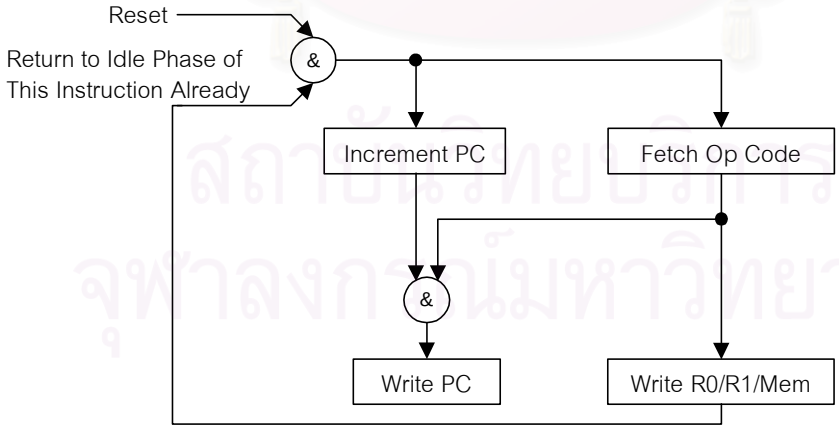
คำสั่ง LD, ADD, SUB, AND, OR, XOR, SL และ SR



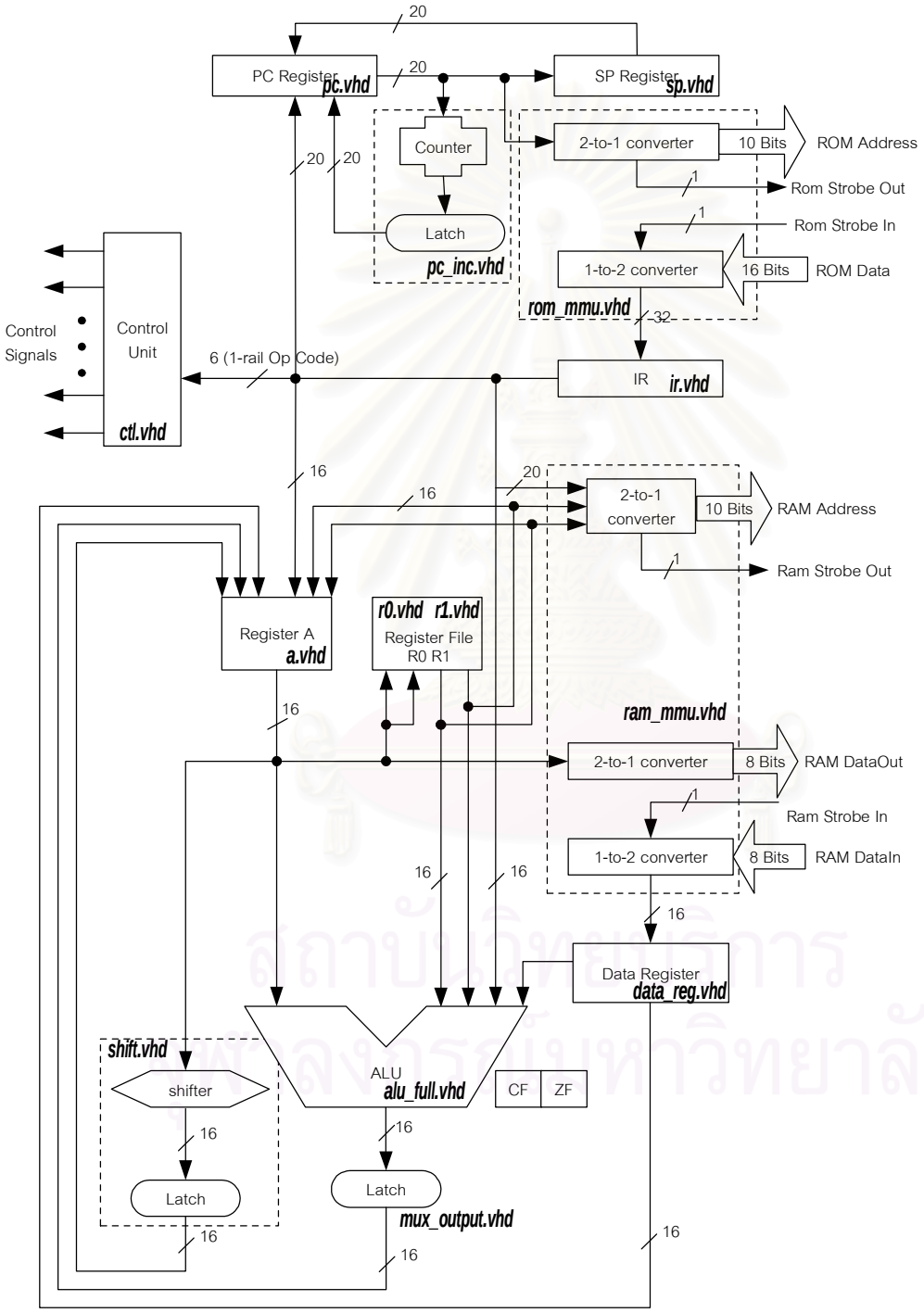
คำสั่ง JMP, CALL, JZ, JNZ, JC, JNC และ Ret



คำสั่ง ST



ภาคผนวก ข
การออกแบบไมโครโปรเซสเซอร์ด้วยภาษาวีเอชดีแอล



a.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity a is
        port(   reset_l         : in std_logic;
                a_alu_req       : in std_logic; -------------------------
                a_ram_req       : in std_logic; --                      --
                a_r0_req        : in std_logic; -- Read A Request --
                a_r1_req        : in std_logic; --                      --
                a_shf_req       : in std_logic; -------------------------
                alu_to_a        : in std_logic_vector(7 downto 0);
                alu_to_a_l      : in std_logic_vector(7 downto 0);
                r0_to_a         : in std_logic_vector(7 downto 0);
                r0_to_a_l       : in std_logic_vector(7 downto 0);
                r1_to_a         : in std_logic_vector(7 downto 0);
                r1_to_a_l       : in std_logic_vector(7 downto 0);
                data_to_a       : in std_logic_vector(7 downto 0);
                data_to_a_l     : in std_logic_vector(7 downto 0);
                ir_to_a         : in std_logic_vector(7 downto 0);
                ir_to_a_l       : in std_logic_vector(7 downto 0);
                shf_to_a        : in std_logic_vector(7 downto 0);
                shf_to_a_l      : in std_logic_vector(7 downto 0);
                a_to_alu        : out std_logic_vector(7 downto 0);
                a_to_alu_l      : out std_logic_vector(7 downto 0);
                a_to_r0         : out std_logic_vector(7 downto 0);
                a_to_r0_l       : out std_logic_vector(7 downto 0);
                a_to_r1         : out std_logic_vector(7 downto 0);
                a_to_r1_l       : out std_logic_vector(7 downto 0);
                a_to_ram        : out std_logic_vector(7 downto 0);
                a_to_ram_l      : out std_logic_vector(7 downto 0);
                a_to_shf        : out std_logic_vector(7 downto 0);
                a_to_shf_l      : out std_logic_vector(7 downto 0);
                wr_a_ack        : out std_logic;
                zero            : out std_logic;
                zero_l          : out std_logic);
end a;

Architecture rtl of a is
        signal x,x_l,lat,lat_l,buff     : std_logic_vector(7 downto 0);
        signal andbuff,orbuff,ack       : std_logic;
        signal chk_zero,chk_zero_l,tzero,tzero_l        : std_logic;
begin
        x     <= alu_to_a or r0_to_a or r1_to_a or data_to_a or ir_to_a or shf_to_a;
        x_l   <=  alu_to_a_l or r0_to_a_l or r1_to_a_l or data_to_a_l or ir_to_a_l or shf_to_a_l;

        gen_latch:
        for i in 0 to 7 generate
                lat(i)      <= (x(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (x_l(i) nor lat(i)) and reset_l;
        end generate;
        gen_buffer:
        for i in 0 to 7 generate
                buff(i)     <= (x(i) and lat_l(i)) or (x_l(i) and lat(i));
        end generate;

        andbuff <= buff(0) and buff(1) and buff(2) and buff(3) and
                   buff(4) and buff(5) and buff(6) and buff(7);

        orbuff     <=  buff(0) or buff(1) or buff(2) or buff(3) or
                       buff(4) or buff(5) or buff(6) or buff(7);
```

```
        ack         <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);

        gen_output:
        for i in 0 to 7 generate
                a_to_alu(i)             <= lat_l(i) and a_alu_req;
                a_to_alu_l(i)           <= (lat(i) and a_alu_req);
                a_to_r0(i)              <= lat_l(i) and a_r0_req;
                a_to_r0_l(i)            <= lat(i) and a_r0_req;
                a_to_r1(i)              <= lat_l(i) and a_r1_req;
                a_to_r1_l(i)            <= lat(i) and a_r1_req;
                a_to_ram(i)             <= lat_l(i) and a_ram_req;
                a_to_ram_l(i)           <= lat(i) and a_ram_req;
                a_to_shf(i)             <= lat_l(i) and a_shf_req;
                a_to_shf_l(i)           <= (lat(i) and a_shf_req);
        end generate;
        wr_a_ack<= ack;

        chk_zero        <= lat_l(0) or lat_l(1) or lat_l(2) or lat_l(3) or lat_l(4) or lat_l(5) or    lat_l(6) or lat_l(7);
        chk_zero_l      <= lat(0) and lat(1) and lat(2) and lat(3) and lat(4) and lat(5) and    lat(6) and lat(7);

  ------------------- Flag ----------------------
        tzero   <= ((chk_zero and ack) nor tzero_l) or (not reset_l);
        tzero_l  <= ((chk_zero_l and ack) nor tzero) and reset_l;

        zero        <= tzero;
        zero_l      <= tzero_l;
end rtl;
```

alu_full.vhd

```
library ieee;
use ieee.std_logic_1164.all;

Entity alu is
        port(   a_to_alu        : in std_logic_vector(7 downto 0);
                a_to_alu_l      : in std_logic_vector(7 downto 0);
                data_to_alu     : in std_logic_vector(7 downto 0);
                data_to_alu_l   : in std_logic_vector(7 downto 0);
                r0_to_alu       : in std_logic_vector(7 downto 0);
                r0_to_alu_l     : in std_logic_vector(7 downto 0);
                r1_to_alu       : in std_logic_vector(7 downto 0);
                r1_to_alu_l     : in std_logic_vector(7 downto 0);
                ir_to_alu       : in std_logic_vector(7 downto 0);
                ir_to_alu_l     : in std_logic_vector(7 downto 0);
                add             : in std_logic;
                sub             : in std_logic;
                orr             : in std_logic;
                andd            : in std_logic;
                xorr            : in std_logic;
                carry_out       : out std_logic;
                carry_out_l     : out std_logic;
                aluout          : out std_logic_vector(7 downto 0);
                aluout_l        : out std_logic_vector(7 downto 0));
end alu;

Architecture rtl of alu is
        signal in1,in1_l,in2,in2_l.or11,or12,res,res_l, tin2: std_logic_vector(7 downto 0);
        signal carry,carry_l : std_logic_vector(8 downto 0);
        signal res_and,res_and_l,res_or,res_or_l,res_xor : std_logic_vector(7 downto 0);
        signal results,results_l,outalu,outalu_l,tin2_l,xor11 : std_logic_vector(7 downto 0);
        signal t_carry,t_carry_l,cout,cout_l,all_control,ack            : std_logic;
        signal all_input,allsignal,allsignal1,xor12,res_xor_l: std_logic_vector(7 downto 0);

begin
        in1     <= a_to_alu;
        in1_l   <= a_to_alu_l;
        tin2    <= data_to_alu or r0_to_alu or r1_to_alu or ir_to_alu;
        tin2_l  <= data_to_alu_l or r0_to_alu_l or r1_to_alu_l or ir_to_alu_l;
        select_in2:
                for i in 0 to 7 generate
                        in2(i)  <= (tin2_l(i) and sub) or (tin2(i) and add);
                        in2_l(i)<= (tin2(i) and sub) or (tin2_l(i) and add);
                end generate;

-------------------------- ADD/SUB/INC MODULE -------------------------
        select_cin:
                carry(0)  <= sub;
                carry_l(0)<= add;
        module_or11_or12:
                for i in 0 to 7 generate
                        or11(i) <= (in1(i) and in2_l(i)) or (in1_l(i) and in2(i));
                        or12(i) <= (in1(i) and in2(i)) or (in1_l(i) and in2_l(i));
                end generate;
        module_res:
                for i in 0 to 7 generate
                        res(i)  <= (or11(i) and carry_l(i)) or (or12(i) and carry(i));
                        res_l(i)<= (or11(i) and carry(i)) or (or12(i) and carry_l(i));
                end generate;
        module_carry:
                for i in 0 to 7 generate
                        carry(i+1)  <= (or11(i) and carry(i)) or (in1(i) and in2(i));
```

```
                carry_l(i+1) <= (or11(i) and carry_l(i)) or (in1_l(i) and in2_l(i));
            end generate;
---------- AND MODULE ----------
        module_and:
            for i in 0 to 7 generate
                res_and(i)      <= andd and in1(i) and tin2(i);
                res_and_l(i)    <= andd and (in1_l(i) or (in1(i) and tin2_l(i)));
            end generate;

----------OR MODULE----------
        module_or:
            for i in 0 to 7 generate
                res_or(i)   <= orr and (in1(i) or (in1_l(i) and tin2(i))) ;
                res_or_l(i) <= orr and (in1_l(i) and tin2_l(i));
            end generate;

---------- XOR MODULE ----------
        module_xor:
            for i in 0 to 7 generate
                xor11(i)    <= (in1(i) and tin2_l(i)) or (in1_l(i) and tin2(i));
                xor12(i)    <= (in1(i) and tin2(i)) or (in1_l(i) and tin2_l(i));
                res_xor(i) <= xorr and xor11(i);
                res_xor_l(i) <= xorr and xor12(i);
            end generate;
        -- Generate Result of Operation
        results     <= res_and or res_or or res_xor or res;
        results_l   <= res_and_l or res_or_l or res_xor_l or res_l;
        t_carry     <= (add and carry(8)) or (sub and carry_l(8));
        t_carry_l   <= (add and carry_l(8)) or (sub and carry(8));

---------- Generate ACKNOWLEDGEMENT ----------
        or_all_input: -- select output of LUT that is OR-Gates of input circuit.
            for i in 0 to 7 generate
                all_input(i) <= in1(i) or in1_l(i) or in2(i) or in2_l(i) or tin2(i) or tin2_l(i);
            end generate;
        all_control <= add or sub or orr or xorr or andd;    -- select input circuit
        or_all_signal:
            for i in 0 to 7 generate
                allsignal(i) <= or11(i) or or12(i) or xor11(i) or xor12(i);
            end generate;
        or_all_signal1:
            for i in 0 to 7 generate
                allsignal1(i)   <= carry(i) or carry_l(i) or all_input(i) or allsignal(i);
            end generate;
        ack <= allsignal1(0) or allsignal1(1) or allsignal1(2) or allsignal1(3) or allsignal1(4) or allsignal1(5) or
               allsignal1(6) or allsignal1(7) or all_control or carry(8) or carry_l(8);

---------- Generate OUTPUT ----------
        gen_aluout:
          for i in 0 to 7 generate
              outalu(i)  <= (results(i) and ack) or (results(i) and outalu(i)) or (ack and  outalu(i));
              outalu_l(i) <= (results_l(i) and ack) or (results_l(i) and outalu_l(i)) or (ack    and outalu_l(i));
          end generate;
        gen_carryout:
          cout      <= (t_carry and ack) or (t_carry and cout) or (ack and cout);
          cout_l  <= (t_carry_l and ack) or (t_carry_l and cout_l) or (ack andcout_l);
        aluout      <= outalu;
        aluout_l    <= outalu_l;
        carry_out   <= cout;
        carry_out_l <= cout_l;

end rtl;
```

ctl.vhd

```
library ieee;
use ieee.std_logic_1164.all;

Entity ctl is
        port(   reset_l         : in std_logic;
                opcode          : in std_logic_vector(7 downto 0);
                carry_out       : in std_logic;
                carry_out_l     : in std_logic;
                zero            : in std_logic;
                zero_l          : in std_logic;
                ir_ctl_ack      : in std_logic;-- fetch finsih
                wr_a_ack        : in std_logic; -- Write A ACK
                wr_pc_ack       : in std_logic; -- Write PC ACK
                wr_inc_ack      : in std_logic; -- Write PC Inc. Value ACK
                pc_inc_ack      : in std_logic; -- Inc PC finish
                a_r0_ack        : in std_logic; -- A Write R0 Finish
                a_r1_ack        : in std_logic; -- A Write R1 Finish
                data_ram_ack    : in std_logic; -- Read RAM Finish
                pc_sp_ack       : in std_logic;-- PC Write SP Finish
                wr_ram_ack      : in std_logic; -- Write RAM Finish
                alu_ack         : in std_logic;-- ALU Execute Finish
                shift_ack       : in std_logic;
                add             : out std_logic;
                sub             : out std_logic;
                orr             : out std_logic;
                andd            : out std_logic;
                xorr            : out std_logic;
                out_to_a        : out std_logic; -- Send ALU Output to A
                pc_rom_req      : out std_logic; --------------------------------
                pc_inc_req      : out std_logic; -- Read PC Request   --
                pc_sp_req       : out std_logic; --------------------------------
                a_alu_req       : out std_logic; --------------------------------
                a_ram_req       : out std_logic; --                           --
                a_r0_req        : out std_logic; -- Read A Request    --
                a_r1_req        : out std_logic; --                           --
                a_shf_req       : out std_logic; --------------------------------
                shf_a_req       : out std_logic;
                do_shl          : out std_logic;
                do_shr          : out std_logic;
                send_to_pc      : out std_logic; -- PC_INC Request (Send Output to PC)
                r0_alu_req      : out std_logic; --------------------------------
                r0_a_req        : out std_logic; -- Read R0 Request   --
                r0_ram_req      : out std_logic; --------------------------------
                r1_alu_req      : out std_logic; --------------------------------
                r1_a_req        : out std_logic; -- Read R1 Request   --
                r1_ram_req      : out std_logic; --------------------------------
                data_alu_req    : out std_logic; -- Read RAM Data Request
                data_a_req      : out std_logic;
                sp_pc_req       : out std_logic; -- Read SP Request
                ir_alu_req      : out std_logic; ------------------------------------------
                ir_a_req        : out std_logic; -- Read Imm. Value Request  --
                ir_pc_req       : out std_logic; ---                           --
                ir_ram_req      : out std_logic; ------------------------------------------
                ram_read        : out std_logic);
end ctl;

Architecture beh of ctl is
        signal st_fetch,c_fetch,inc_pc,c_inc_pc, cnt_pc_wr,c_write_pc       : std_logic;
        signal imm_mode,reg_mode,drct_mode,idrct_mode, write_pc             : std_logic;
        signal req_r0_a,req_r1_a,req_r0_ram,req_r1_ram, call_op,c_rd_pc     : std_logic;
        signal req_a_r0,req_a_r1,req_a_ram, t_exec1,t_exec2,exe1,exe2       : std_logic;
```

```
        signal store,call,cal,load,jump,jmpcond,ret,jmp_rel,shift : std_logic;
        signal do_add,do_sub,do_and,do_or,do_xor,exe3,nt_jmprel : std_logic;
        signal read_ram,req_ram,c_rd_ram, exec,fin_exec,execute,c_exe      : std_logic;
        signal last_exe,write_fin,write_des,c_write_des,des_ack, carry,carry_l: std_logic;
begin

--*************** FETCH MODULE ******************
        --control_first fetch
        st_fetch  <= reset_l and (not des_ack);
        pc_rom_req <= st_fetch and (not c_fetch);
        c_fetch  <= (st_fetch and ir_ctl_ack) or (st_fetch and c_fetch) or (ir_ctl_ack and c_fetch);

--****************** INC PC *********************
        inc_pc    <= st_fetch and (not c_inc_pc);
        c_inc_pc  <= (st_fetch and pc_inc_ack) or (st_fetch and c_inc_pc) or (pc_inc_ack and c_inc_pc);
        write_pc  <= c_inc_pc and ((not opcode(7)) or (jmp_rel and (not jmpcond)) or call) and c_fetch;
        cnt_pc_wr   <= write_pc and (not c_write_pc);
        c_write_pc  <= (write_pc and wr_inc_ack) or (write_pc and c_write_pc) or (wr_inc_ack and
                        c_write_pc);

--******************** DECODE ******************
--        "0000"    = STORE
--        "0001"    = LOAD
--        "0010"    = SHIFT + '0' = SHIFT LEFT /  + '1' = SHIFT RIGHT
--        "0011"    = AND
--        "0100"    = OR
--        "0101"    = XOR
--        "0110"    = SUB
--        "0111"    = ADD
--        "1000"    = BRANCH WITH CONDITION + "00" = JZ
--                                                  "01" = JNZ
--                                                  "10" = JC
--                                                  "11" = JNC
--        "1001"    = BRANCH + "00" = JUMP
--                                 "01" = CALL
--                                 "10" = RET
-------------------------------------------------------------------------------------
        store      <= '1' when (opcode(7 downto 4) = "0000") else '0';          -- STORE
        load       <= '1' when (opcode(7 downto 4) = "0001") else '0';          -- LOAD
        shift      <= '1' when (opcode(7 downto 4) = "0010") else '0';
        cal         <= '1' when ((opcode(7 downto 6) = "01") or (opcode(7 downto 4) = "0011")) else '0'; --
                       AND,OR,XOR,SUB,ADD
        jmp_rel    <= '1' when (opcode(7 downto 4) = "1000") else '0';
        jmpcond    <= (imm_mode and zero and (not zero_l)) or (reg_mode and (not zero) and zero_l) or
                      (drct_mode and carry and (not carry_l)) or (idrct_mode and (not carry) and carry_l);
        jump       <= '1' when (opcode(7 downto 2) = "100100") else '0'; -- JMP
        call       <= '1' when (opcode(7 downto 2) = "100101") else '0'; -- CALL
        ret        <= '1' when (opcode(7 downto 2) = "100110") else '0'; -- RET
        -- What's Mode and Source ----
        imm_mode        <= not (opcode(3) or opcode(2));        -- immediate
        reg_mode        <= (not opcode(3)) and opcode(2);       -- register
        drct_mode       <= opcode(3) and (not opcode(2));       -- direct
        idrct_mode      <= opcode(3) and opcode(2);             -- indirect
        req_r0_a        <= reg_mode and opcode(0) and load; -- load register to A
        req_r1_a        <= reg_mode and opcode(1) and load;
        req_r0_ram      <= idrct_mode and opcode(0);             -- address
        req_r1_ram      <= idrct_mode and opcode(1);
        req_a_r0        <= store and reg_mode and opcode(0); -- store A to register
        req_a_r1        <= store and reg_mode and opcode(1);
        req_a_ram       <= store and opcode(3); -- store A to memory
        -- What's Operation ----------
        do_and     <= '1' when (opcode(7 downto 4) = "0011") else '0';
        do_or      <= '1' when (opcode(7 downto 4) = "0100") else '0';
```

```
	do_xor	<= '1' when (opcode(7 downto 4) = "0101") else '0';
	do_sub	<= '1' when (opcode(7 downto 4) = "0110") else '0';
	do_add	<= '1' when (opcode(7 downto 4) = "0111") else '0';

--******************** DEFER *********************
	-- direct / indirect addressing mode
	read_ram  <= opcode(3) and (cal or load) and c_fetch;
	req_ram	<= read_ram and (not c_rd_ram);
	c_rd_ram  <= (read_ram and data_ram_ack) or (read_ram and c_rd_ram) or (data_ram_ack and
		     c_rd_ram);
	-- call instrction (keep pc before execute)
	call_op	<= call and c_write_pc;
	pc_sp_req  <= call_op and (not c_rd_pc);		-- Call insturction
	c_rd_pc	<= (call_op and pc_sp_ack) or (call_op and c_rd_pc) or (pc_sp_ack and c_rd_pc);

--******************** ALU EXECUTE ******************
	t_exec1	<= ((cal and (not opcode(3))) or shift) and c_fetch;
	t_exec2	<= cal and c_rd_ram;
	exec	<= t_exec1 or t_exec2;
	fin_exec  <= alu_ack or shift_ack;
	execute	<= exec and (not c_exe);
	c_exe	<= (exec and fin_exec) or (exec and c_exe) or (fin_exec and c_exe);

--********************** DESTINATION WRITE BACK **************
	exe1	<= (store or jump or ret or (load and (not opcode(3))) or (jmp_rel and jmpcond)) and c_fetch;
	exe2	<= call and c_rd_pc;
	exe3	<= load and c_rd_ram;
	last_exe	<= exe1 or exe2 or exe3 or c_exe;
	write_fin <= wr_a_ack or wr_ram_ack or a_r0_ack or a_r1_ack or wr_pc_ack;
	write_des	<= last_exe and (not c_write_des);
	c_write_des  <= (last_exe and write_fin) or (last_exe and c_write_des) or (write_fin and c_write_des);
	nt_jmprel  <= jmp_rel and (not jmpcond) and c_write_pc;
	des_ack	<= (c_write_des and c_write_pc and (not opcode(7))) or
		     nt_jmprel or (c_write_des and opcode(7));

--********************** CONTROL SIGNAL  *************************
	send_to_pc		<= cnt_pc_wr;
	r0_ram_req		<= req_r0_ram and (req_ram or (write_des and store));
	r1_ram_req		<= req_r1_ram and (req_ram or (write_des and store));
	ir_ram_req		<= drct_mode and (req_ram or (write_des and store));
	ir_alu_req		<= execute and imm_mode;
	data_alu_req		<= execute and opcode(3);
	r0_alu_req		<= execute and reg_mode and opcode(0);
	r1_alu_req		<= execute and reg_mode and opcode(1);
	a_alu_req		<= execute and cal;
	a_r0_req		<= write_des and req_a_r0;
	a_r1_req		<= write_des and req_a_r1;
	a_ram_req		<= write_des and req_a_ram;
	ir_a_req		<= write_des and load and imm_mode;
	r0_a_req		<= write_des and req_r0_a;
	r1_a_req		<= write_des and req_r1_a;
	data_a_req		<= write_des and load and opcode(3);
	sp_pc_req		<= write_des and ret ;
	ir_pc_req		<= write_des and (call or jump or (jmp_rel and jmpcond));
	out_to_a		<= write_des and cal;
	a_shf_req		<= execute and shift;
	shf_a_req		<= write_des and shift;
	do_shl		<= execute and shift and (not opcode(3));
	do_shr		<= execute and shift and opcode(3);
	pc_inc_req		<= inc_pc;
	ram_read		<= req_ram;
	add		<= do_add and execute; ----------------------------------
	sub		<= do_sub and execute; --                                --
```

```
        andd            <= do_and and execute; -- ALU Mode for Execute --
        orr             <= do_or and execute  ; --                      --
        xorr            <= do_xor and execute;  ----------------------------------
        carry           <= (carry_out_l nor carry_l) and reset_l;
        carry_l         <= (carry_out nor carry) or (not reset_l);
end beh;
```

data_reg.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity data_reg is
        port(   reset_l             : in std_logic;
                ram_to_data         : in std_logic_vector(7 downto 0);
                ram_to_data_l       : in std_logic_vector(7 downto 0);
                data_alu_req        : in std_logic;
                data_a_req          : in std_logic;
                data_ram_ack        : out std_logic;
                data_to_a           : out std_logic_vector(7 downto 0);
                data_to_a_l         : out std_logic_vector(7 downto 0);
                data_to_alu         : out std_logic_vector(7 downto 0);
                data_to_alu_l       : out std_logic_vector(7 downto 0)
                );
end data_reg;

Architecture rtl of data_reg is
        signal lat,lat_l,buff : std_logic_vector(7 downto 0);
        signal andbuff,orbuff,ack : std_logic;
begin
        gen_latch:
        for i in 0 to 7 generate
                lat(i)      <= (ram_to_data(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (ram_to_data_l(i) nor lat(i)) and reset_l;
        end generate;

        gen_buffer:
        for i in 0 to 7 generate
                buff(i)     <= (ram_to_data(i) and lat_l(i)) or (ram_to_data_l(i) and lat(i));
        end generate;

        andbuff  <=  buff(0) and buff(1) and buff(2) and buff(3) and
                     buff(4) and buff(5) and buff(6) and buff(7);

        orbuff   <=  buff(0) or buff(1) or buff(2) or buff(3) or
                     buff(4) or buff(5) or buff(6) or buff(7);

        ack      <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);

        gen_output:
        for i in 0 to 7 generate
                data_to_alu(i)      <= lat_l(i) and data_alu_req;
                data_to_alu_l(i)    <= lat(i) and data_alu_req;
                data_to_a(i)        <= lat_l(i) and data_a_req;
                data_to_a_l(i)      <= lat(i) and data_a_req;
        end generate;
   data_ram_ack <= ack;

end rtl;
```

ir.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity ir is
        port(   reset_l         : in std_logic;
                data_rom        : in std_logic_vector(15 downto 0);
                data_rom_l      : in std_logic_vector(15 downto 0);
                ir_alu_req      : in std_logic; ----------------------------------------
                ir_a_req        : in std_logic; -- Read Imm. Value Request --
                ir_pc_req       : in std_logic; --                                    --
                ir_ram_req      : in std_logic; ----------------------------------------
                ir_ctl_ack      : out std_logic;
                ir_to_alu       : out std_logic_vector(7 downto 0);
                ir_to_alu_l     : out std_logic_vector(7 downto 0);
                ir_to_a         : out std_logic_vector(7 downto 0);
                ir_to_a_l       : out std_logic_vector(7 downto 0);
                ir_to_ram       : out std_logic_vector(9 downto 0);
                ir_to_ram_l     : out std_logic_vector(9 downto 0);
                ir_to_pc        : out std_logic_vector(9 downto 0);
                ir_to_pc_l      : out std_logic_vector(9 downto 0);
                opcode          : out std_logic_vector(7 downto 0));
end ir;
Architecture rtl of ir is
        signal lat,lat_l,buff : std_logic_vector(15 downto 0);
        signal andbuff,orbuff,ack       : std_logic;
begin
        gen_latch:
        for i in 0 to 15 generate
                lat(i)      <= (data_rom(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (data_rom_l(i) nor lat(i)) and reset_l;
        end generate;
        gen_buffer:
        for i in 0 to 15 generate
                buff(i)     <= (data_rom(i) and lat_l(i)) or (data_rom_l(i) and lat(i));
        end generate;
        andbuff  <=  buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and buff(5) and buff(6) and buff(7)
                     and buff(8) and buff(9) and buff(10) and buff(11) and buff(12) and buff(13) and buff(14)
                     and buff(15);
        orbuff      <=  buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or buff(5) or buff(6) or buff(7) or buff(8)
                            or buff(9) or buff(10) or buff(11) or buff(12) or buff(13) or buff(14) or buff(15);
        ack         <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);
        gen_output:
        for i in 0 to 7 generate
                ir_to_alu(i)        <= lat_l(i) and ir_alu_req;
                ir_to_alu_l(i)      <= lat(i) and ir_alu_req;
                ir_to_a(i)          <= lat_l(i) and ir_a_req;
                ir_to_a_l(i)        <= lat(i) and ir_a_req;
        end generate;
        gen_output1:
        for i in 0 to 9 generate
                ir_to_ram(i)        <= lat_l(i) and ir_ram_req;
                ir_to_ram_l(i)      <= lat(i) and ir_ram_req;
                ir_to_pc(i)         <= lat_l(i) and ir_pc_req;
                ir_to_pc_l(i)       <= lat(i) and ir_pc_req;
        end generate;
        ir_ctl_ack<= ack;
        opcode    <= lat_l(15 downto 10) & lat_l(1 downto 0);
end rtl;
```

mux_output.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity mux_output is
        port(   reset_l             : in std_logic;
                aluout              : in std_logic_vector(7 downto 0);
                aluout_l            : in std_logic_vector(7 downto 0);
                out_to_a            : in std_logic;
                alu_to_a            : out std_logic_vector(7 downto 0);
                alu_to_a_l          : out std_logic_vector(7 downto 0);
                alu_ack             : out std_logic
                );
end mux_output;

Architecture rtl of mux_output is
  signal lat,lat_l,buff     : std_logic_vector(7 downto 0);
  signal andbuff,orbuff,ack  : std_logic;
begin

  gen_lat:
  for i in 0 to 7 generate
    lat(i)    <= (aluout(i) nor lat_l(i)) or (not reset_l);
    lat_l(i)  <= (aluout_l(i) nor lat(i)) and reset_l;
  end generate;

  gen_buf:
  for i in 0 to 7 generate
    buff(i) <= (aluout(i) and lat_l(i)) or (aluout_l(i) and lat(i));
  end generate;

  andbuff <= buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and
             buff(5) and buff(6) and buff(7);

  orbuff <= buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or
            buff(5) or buff(6) or buff(7);

  ack  <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);
  alu_ack <= ack;

        gen_alu_to_a:
        for i in 0 to 7 generate
                alu_to_a(i)         <= out_to_a and lat_l(i);
                alu_to_a_l(i)       <= out_to_a and lat(i);
        end generate;

end rtl;
```

pc.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity pc is
        port(   reset_l             : in std_logic;
                pc_inc_req          : in std_logic;
                pc_sp_req           : in std_logic;
                pc_out              : in std_logic_vector(9 downto 0);
                pc_out_l            : in std_logic_vector(9 downto 0);
                sp_to_pc            : in std_logic_vector(9 downto 0);
                sp_to_pc_l          : in std_logic_vector(9 downto 0);
                ir_to_pc            : in std_logic_vector(9 downto 0);
                ir_to_pc_l          : in std_logic_vector(9 downto 0);
                pc_to_sp            : out std_logic_vector(9 downto 0);
                pc_to_sp_l          : out std_logic_vector(9 downto 0);
                pc_to_rom           : out std_logic_vector(9 downto 0);
                pc_to_inc           : out std_logic_vector(9 downto 0);
                pc_to_inc_l         : out std_logic_vector(9 downto 0);
                wr_inc_ack          : out std_logic; -- Write PC Inc. Value ACK
                wr_pc_ack           : out std_logic  -- Write PC ACK);
end pc;

Architecture rtl of pc is
        signal x,x_l,lat,lat_l,buff     : std_logic_vector(9 downto 0);
        signal andbuff,orbuff,ack_x,t_wr,ack_wr,t_inc,ack_inc     : std_logic;
begin
        x       <= sp_to_pc or ir_to_pc or pc_out;
        x_l     <= sp_to_pc_l or ir_to_pc_l or pc_out_l;
        gen_latch:
        for i in 0 to 9 generate
                lat(i)      <= (x(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (x_l(i) nor lat(i)) and reset_l;
        end generate;
        gen_buffer:
        for i in 0 to 9 generate
                buff(i)     <= (x(i) and lat_l(i)) or (x_l(i) and lat(i));
        end generate;
        andbuff <= buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and
                   buff(5) and buff(6) and buff(7) and buff(8) and buff(9);
        orbuff  <= buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or
                   buff(5) or buff(6) or buff(7) or buff(8) or buff(9);

        ack_x   <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);
        t_wr    <= sp_to_pc(0) or sp_to_pc_l(0) or ir_to_pc(0) or ir_to_pc_l(0);
        t_inc   <= pc_out(0) or pc_out_l(0);
        ack_wr  <= (t_wr and ack_x) or (t_wr and ack_wr) or (ack_x and ack_wr);
        ack_inc <= (t_inc and ack_x) or (t_inc and ack_inc) or (ack_x and ack_inc);
        ---------- Gen Output ----------
        gen_output:
        for i in 0 to 9 generate
                pc_to_rom(i)        <= lat_l(i);
                pc_to_inc(i)        <= lat_l(i) and pc_inc_req;
                pc_to_inc_l(i)      <= lat(i) and pc_inc_req;
                pc_to_sp(i)         <= lat_l(i) and pc_sp_req;
                pc_to_sp_l(i)       <= lat(i) and pc_sp_req;
        end generate;
        wr_pc_ack           <= ack_wr;
        wr_inc_ack          <= ack_inc;
end rtl;
```

pc_inc.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity pc_inc is
        port(   reset_l         : in std_logic;
                send_to_pc      : in std_logic;
                pc_to_inc       : in std_logic_vector(9 downto 0);
                pc_to_inc_l     : in std_logic_vector(9 downto 0);
                pc_inc_ack      : out std_logic;
                pc_out          : out std_logic_vector(9 downto 0);
                pc_out_l        : out std_logic_vector(9 downto 0));
end pc_inc;
Architecture rtl of pc_inc is
        signal sum,sum_l,buff,lat,lat_l         : std_logic_vector(9 downto 0);
        signal andbuff,orbuff,ack       : std_logic;
        signal c,c_l            : std_logic_vector(9 downto 1);
begin
        sum(0)      <= pc_to_inc_l(0);
        sum_l(0)    <= pc_to_inc(0);
        c(1)        <= pc_to_inc(0);
        c_l(1)      <= pc_to_inc_l(0);
        gen_sum:
        for i in 1 to 9 generate
                sum(i)      <= (pc_to_inc(i) and c_l(i)) or (pc_to_inc_l(i) and c(i));
                sum_l(i)    <= (pc_to_inc_l(i) and c_l(i)) or (pc_to_inc(i) and c(i));
        end generate;
        gen_carry:
        for i in 2 to 9 generate
                c(i)        <= pc_to_inc(i-1) and c(i-1);
                c_l(i)      <= pc_to_inc_l(i-1) or (pc_to_inc(i-1) and c_l(i-1));
        end generate;
        ---------- Latch ----------
        gen_latch:
        for i in 0 to 9 generate
                lat(i)      <= (sum(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (sum_l(i) nor lat(i)) and reset_l;
        end generate;
        ---------- Gen Acknowledgement ----------
        gen_buff:
        for i in 0 to 9 generate
                buff(i)     <= (sum(i) and lat_l(i)) or (sum_l(i) and lat(i));
        end generate;

        andbuff   <= buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and
                     buff(5) and buff(6) and buff(7) and buff(8) and buff(9);

        orbuff    <= buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or
                     buff(5) or buff(6) or buff(7) or buff(8) or buff(9);

        ack_x     <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);
        ack       <= pc_to_inc(0) or pc_to_inc_l(0);
        ack_inc   <= (ack_x and ack) or (ack_x and ack_inc) or (ack and ack_inc);
        ---------- Gen Output ----------
        gen_out_pc:
        for i in 0 to 9 generate
                pc_out(i)           <= lat_l(i) and send_to_pc;
                pc_out_l(i)         <= lat(i) and send_to_pc;
        end generate;
        pc_inc_ack          <= ack_inc;
end rtl;
```

r0.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity r0 is
        port(   reset_l             : in std_logic;
                r0_ram_req          : in std_logic;
                r0_alu_req          : in std_logic;
                r0_a_req            : in std_logic;
                a_to_r0             : in std_logic_vector(7 downto 0);
                a_to_r0_l           : in std_logic_vector(7 downto 0);
                a_r0_ack            : out std_logic;
                r0_to_a             : out std_logic_vector(7 downto 0);
                r0_to_a_l           : out std_logic_vector(7 downto 0);
                r0_to_ram           : out std_logic_vector(7 downto 0);
                r0_to_ram_l         : out std_logic_vector(9 downto 0);
                r0_to_alu           : out std_logic_vector(7 downto 0);
                r0_to_alu_l         : out std_logic_vector(7 downto 0));
end r0;

Architecture rtl of r0 is
        signal lat,lat_l,buff : std_logic_vector(7 downto 0);
        signal andbuff,orbuff,ack       : std_logic;
begin
        gen_latch:
        for i in 0 to 7 generate
                lat(i)      <= (a_to_r0(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (a_to_r0_l(i) nor lat(i)) and reset_l;
        end generate;

        gen_buffer:
        for i in 0 to 7 generate
                buff(i)     <= (a_to_r0(i) and lat_l(i)) or (a_to_r0_l(i) and lat(i));
        end generate;

        andbuff <=  buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and buff(5) and buff(6) and buff(7);
        orbuff  <=  buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or buff(5) or buff(6) or buff(7);
        ack     <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);

        gen_output:
        for i in 0 to 7 generate
                r0_to_alu(i)        <= lat_l(i) and r0_alu_req;
                r0_to_alu_l(i)      <= lat(i) and r0_alu_req;
                r0_to_a(i)          <= lat_l(i) and r0_a_req;
                r0_to_a_l(i)        <= lat(i) and r0_a_req;
                r0_to_ram(i)        <= lat_l(i) and r0_ram_req;
                r0_to_ram_l(i)      <= lat(i) and r0_ram_req;
        end generate;

        r0_to_ram_l(8)      <= r0_ram_req;
        r0_to_ram_l(9)      <= r0_ram_req;
        a_r0_ack <= ack;

end rtl;
```

r1.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity r1 is
        port(   reset_l         : in std_logic;
                r1_ram_req      : in std_logic;
                r1_alu_req      : in std_logic;
                r1_a_req        : in std_logic;
                a_to_r1         : in std_logic_vector(7 downto 0);
                a_to_r1_l       : in std_logic_vector(7 downto 0);
                a_r1_ack        : out std_logic;
                r1_to_a         : out std_logic_vector(7 downto 0);
                r1_to_a_l       : out std_logic_vector(7 downto 0);
                r1_to_ram       : out std_logic_vector(7 downto 0);
                r1_to_ram_l     : out std_logic_vector(9 downto 0);
                r1_to_alu       : out std_logic_vector(7 downto 0);
                r1_to_alu_l     : out std_logic_vector(7 downto 0));
end r1;

Architecture rtl of r1 is
        signal lat,lat_l,buff : std_logic_vector(7 downto 0);
        signal andbuff,orbuff,ack       : std_logic;
begin
        gen_latch:
        for i in 0 to 7 generate
                lat(i)      <= (a_to_r1(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (a_to_r1_l(i) nor lat(i)) and reset_l;
        end generate;

        gen_buffer:
        for i in 0 to 7 generate
                buff(i)     <= (a_to_r1(i) and lat_l(i)) or (a_to_r1_l(i) and lat(i));
        end generate;

        andbuff <=  buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and buff(5) and buff(6) and buff(7);
        orbuff  <=  buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or buff(5) or buff(6) or buff(7);
        ack     <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);

        gen_output:
        for i in 0 to 7 generate
                r1_to_alu(i)        <= lat_l(i) and r1_alu_req;
                r1_to_alu_l(i)      <= lat(i) and r1_alu_req;
                r1_to_a(i)          <= lat_l(i) and r1_a_req;
                r1_to_a_l(i)        <= lat(i) and r1_a_req;
                r1_to_ram(i)        <= lat_l(i) and r1_ram_req;
                r1_to_ram_l(i)      <= lat(i) and r1_ram_req;
        end generate;

        r1_to_ram_l(8)      <= r1_ram_req;
        r1_to_ram_l(9)      <= r1_ram_req;
        a_r1_ack <= ack;

end rtl;
```

ram_mmu.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity ram_mmu is
        port(   reset_l         : in std_logic;
                ir_to_ram       : in std_logic_vector(9 downto 0);
                ir_to_ram_l     : in std_logic_vector(9 downto 0);
                r0_to_ram       : in std_logic_vector(7 downto 0);
                r0_to_ram_l     : in std_logic_vector(9 downto 0);
                r1_to_ram       : in std_logic_vector(7 downto 0);
                r1_to_ram_l     : in std_logic_vector(9 downto 0);
                a_to_ram        : in std_logic_vector(7 downto 0);
                a_to_ram_l      : in std_logic_vector(7 downto 0);
                ram_read        : in std_logic;
                ack_ram_addr    : in std_logic;
                ack_ram_data    : in std_logic;
                ram_ack_addr    : out std_logic;
                ram_ack_data    : out std_logic;
                ram_to_data     : out std_logic_vector(7 downto 0);
                ram_to_data_l   : out std_logic_vector(7 downto 0);
                wr_ram_ack      : out std_logic;
                ram_addr        : out std_logic_vector(9 downto 0);
                ram_re          : out std_logic;
                ram_we          : out std_logic;
                ram_data        : in std_logic_vector(7 downto 0);
                ram_data_wr     : out std_logic_vector(7 downto 0));
end ram_mmu;
Architecture rtl of ram_mmu is
        signal lat,lat_l,x,x_l,buff : std_logic_vector(9 downto 0);
        signal andbuff,orbuff,ack_addr, anddat,ordat,ack_dat,wr     : std_logic;
        signal dat,latdat,latdat_l      : std_logic_vector(7 downto 0);
begin
------------------ 2-to-1 convert for address---------------------------------
        x(7 downto 0) <= r0_to_ram or r1_to_ram or ir_to_ram(7 downto 0);
        x(9 downto 8) <= ir_to_ram(9 downto 8);
        x_l <= r0_to_ram_l or r1_to_ram_l or ir_to_ram_l;
        gen_buff:
        for i in 0 to 9 generate
                buff(i)     <= x(i) or x_l(i);
        end generate;
        andbuff <= buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and
                   buff(5) and buff(6) and buff(7) and buff(8) and buff(9);
        orbuff   <= buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or
                    buff(5) or buff(6) or buff(7) or buff(8) or buff(9);
        ack_addr <= (andbuff and orbuff) or (andbuff and ack_addr) or (orbuff and ack_addr);
        ram_ack_addr        <= ack_addr;
---------------- 1-to-2 convert for data to read ----------------------------
        gen_data_ram:
        for i in 0 to 7 generate
                ram_to_data(i)      <= ack_ram_addr and ram_read and ram_data(i);
                ram_to_data_l(i)    <= ack_ram_addr and ram_read and (not ram_data(i));
        end generate;
------------------ 2-to-1 convert for data to write --------------------------------
        gen_buff_data:
        for i in 0 to 7 generate
                dat(i) <= a_to_ram(i) or a_to_ram_l(i);
        end generate;
        anddat    <= dat(0) and dat(1) and dat(2) and dat(3) and dat(4) and dat(5) and dat(6) and dat(7);
        ordat     <= dat(0) or dat(1) or dat(2) or dat(3) or dat(4) or dat(5) or dat(6) or dat(7);
```

```
        ack_dat  <= (anddat and ordat) or (anddat and ack_dat) or (ordat and ack_dat);
        ram_ack_data        <= ack_dat;
---------- RAM Control Signal ------------------------------
        wr  <= ack_addr and ack_dat and (not ack_ram_addr) and (not ack_ram_data);
        ram_addr            <= x;
        ram_we              <= wr;
        ram_re              <= '1';
        ram_data_wr         <= a_to_ram;
        wr_ram_ack          <= ack_ram_addr and ack_ram_data;

end rtl;
```

rom_mmu.vhd

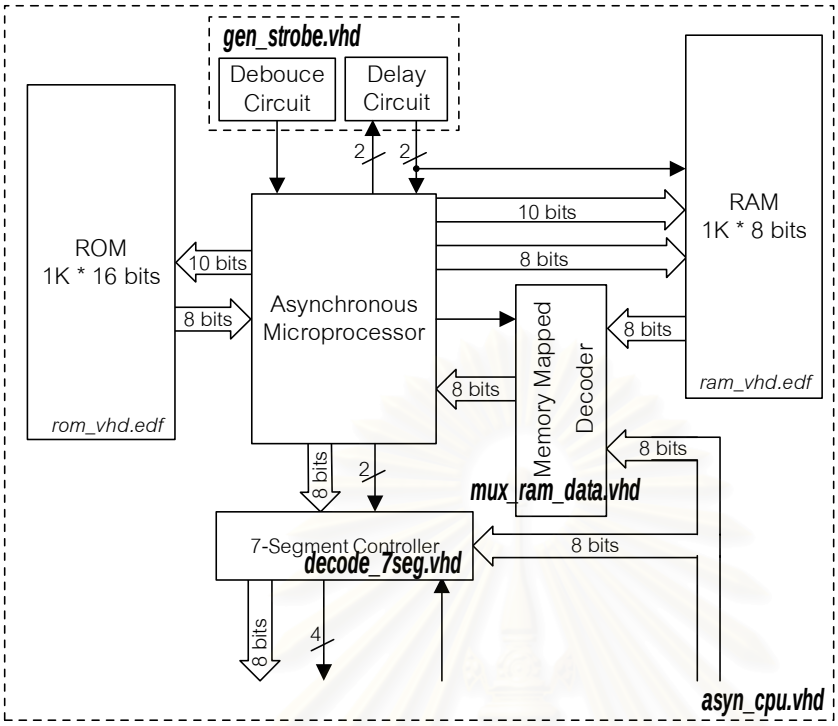
```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity rom_mmu is
        port(   reset_l         : in std_logic;
                pc_to_rom       : in std_logic_vector(9 downto 0);
                rom_data        : in std_logic_vector(15 downto 0);
                ack_rom_addr    : in std_logic;
                rom_re          : out std_logic;
                rom_addr        : out std_logic_vector(9 downto 0);
                data_rom        : out std_logic_vector(15 downto 0);
                data_rom_l      : out std_logic_vector(15 downto 0) );
end rom_mmu;

Architecture rtl of rom_mmu is
        signal lat,lat_l,buff : std_logic_vector(9 downto 0);
        signal andbuff,orbuff,ack_addr          : std_logic;
begin
---------- 1-to-2 convert for rom data ----------------------------
        gen_data_rom:
        for i in 0 to 15 generate
                data_rom(i)     <= ack_rom_addr and rom_data(i) ;
                data_rom_l(i)   <= ack_rom_addr and (not rom_data(i));
        end generate;

-------------------- ROM Control Signal -----------------------------
        rom_re   <= '1';
        rom_addr <= pc_to_rom;
end rtl;
```

shift.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity shift is
        port(   reset_l         : in std_logic;
                do_shr          : in std_logic;
                do_shl          : in std_logic;
                shf_a_req       : in std_logic;
                a_to_shf        : in std_logic_vector(7 downto 0);
                a_to_shf_l      : in std_logic_vector(7 downto 0);
                shift_ack       : out std_logic;
                shf_to_a        : out std_logic_vector(7 downto 0);
                shf_to_a_l      : out std_logic_vector(7 downto 0));
end shift;
Architecture rtl of shift is
        signal shf_res,shf_res_l.buff,lat,lat_l     : std_logic_vector(7 downto 0);
        signal andbuff,orbuff,ack       : std_logic;
begin
        shf_res(0)          <= do_shr and a_to_shf(1);
        shf_res_l(0)        <= do_shl or (do_shr and a_to_shf_l(1));
        gen_shift:
        for i in 1 to 6 generate
           shf_res(i) <= (do_shl and a_to_shf(i-1)) or (do_shr and a_to_shf(i+1));
           shf_res_l(i)<= (do_shl and a_to_shf_l(i-1)) or (do_shr and a_to_shf_l(i+1));
        end generate;

        shf_res(7)          <= do_shl and a_to_shf(6);
        shf_res_l(7)        <= do_shr or (do_shl and a_to_shf_l(6));

        ---------- Latch ----------
        gen_latch:
        for i in 0 to 7 generate
                lat(i)      <= (shf_res(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (shf_res_l(i) nor lat(i)) and reset_l;
        end generate;

        ---------- Gen Acknowledgement ----------
        gen_buff:
        for i in 0 to 7 generate
                buff(i)     <= (shf_res(i) and lat_l(i)) or (shf_res_l(i) and lat(i));
        end generate;

        andbuff <= buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and
                   buff(5) and buff(6) and buff(7);

        orbuff  <= buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or
                   buff(5) or buff(6) or buff(7);

        ack  <= (andbuff and orbuff) or (andbuff and ack_x) or (orbuff and ack_x);
        ---------- Gen Output ----------
        gen_out_pc:
        for i in 0 to 7 generate
                shf_to_a(i)         <= lat_l(i) and shf_a_req;
                shf_to_a_l(i)       <= lat(i) and shf_a_req;
        end generate;

        shift_ack <= ack;

end rtl;
```

sp.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity sp is
        port(   reset_l         : in std_logic;
                sp_pc_req       : in std_logic;
                pc_to_sp        : in std_logic_vector(9 downto 0);
                pc_to_sp_l      : in std_logic_vector(9 downto 0);
                pc_sp_ack       : out std_logic;
                sp_to_pc        : out std_logic_vector(9 downto 0);
                sp_to_pc_l      : out std_logic_vector(9 downto 0));
end sp;

Architecture rtl of sp is
        signal lat,lat_l,buff : std_logic_vector(9 downto 0);
        signal andbuff,orbuff,ack       : std_logic;
begin
        gen_latch:
        for i in 0 to 9 generate
                lat(i)      <= (pc_to_sp(i) nor lat_l(i)) or (not reset_l);
                lat_l(i)    <= (pc_to_sp_l(i) nor lat(i)) and reset_l;
        end generate;

        gen_buffer:
        for i in 0 to 9 generate
                buff(i)     <= (pc_to_sp(i) and lat_l(i)) or (pc_to_sp_l(i) and lat(i));
        end generate;

        andbuff <= buff(0) and buff(1) and buff(2) and buff(3) and buff(4) and
                   buff(5) and buff(6) and buff(7) and buff(8) and buff(9);

        orbuff      <= buff(0) or buff(1) or buff(2) or buff(3) or buff(4) or
                       buff(5) or buff(6) or buff(7) or buff(8) or buff(9);

        ack         <= (t_pc and ack_x) or (t_pc and ack_pc) or (ack_x and ack_pc);

        gen_output:
        for i in 0 to 9 generate
                sp_to_pc(i)         <= lat_l(i) and sp_pc_req;
                sp_to_pc_l(i)       <= lat(i) and sp_pc_req;
        end generate;
        pc_sp_ack  <= ack;

end rtl;
```



mux_ram_data.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity mux_ram_data is
        port(   switch          : in std_logic_vector(7 downto 0);
                ram_addr        : in std_logic_vector(9 downto 0);
                ram_we          : in std_logic;
                ram_data_rd     : in std_logic_vector(7 downto 0);
                ram_data        : out std_logic_vector(7 downto 0)
                );
end mux_ram_data;

Architecture rtl of mux_ram_data is
begin
        ram_data <= switch when ((ram_addr = "1111110000") and ram_we = '0') else ram_data_rd;
end rtl;
```

decode_7seg.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity decode_7seg is
   port(   reset_l              : in std_logic;
           clk                  : in std_logic;
           data_show            : in std_logic_vector(7 downto 0);
           position             : in std_logic_vector(9 downto 0);
           show                 : in std_logic;
           show_input           : in std_logic;
           debug_dat            : in std_logic_vector(15 downto 0);
           to_7seg              : out std_logic_vector(7 downto 0);
           seg_en               : out std_logic_vector(3 downto 0));
end decode_7seg;

Architecture rtl of decode_7seg is
           signal cnt     : unsigned(15 downto 0);
           signal state   : unsigned(1 downto 0);
           signal seg_dat,dat1,dat2,dat3,dat4  : std_logic_vector(7 downto 0);
           signal chk,t_chk,tt_chk         : std_logic;
           signal data0,data1,data2,data3              : std_logic_vector(7 downto 0);
           signal tmp             : std_logic_vector(15 downto 0);
begin
-- concurrent statements
           chk <= '1' when ((position(9 downto 2) = "11111111") and (show = '1')) else '0';
           latch_dat:
           process(reset_l,clk)
           begin
                   if (reset_l = '0') then
                           dat1        <= (others => '0');
                           dat2        <= (others => '0');
                           dat3        <= (others => '0');
                           dat4        <= (others => '0');
                   elsif rising_edge(clk) then
                           if (chk = '1') then
                                   case position(1 downto 0) is
                                           when "00" => dat1 <= data_show;
                                           when "01" => dat2 <= data_show;
                                           when "10" => dat3 <= data_show;
                                           when others => dat4 <= data_show;
                                   end case;
                           end if;
                   end if;
           end process latch_dat;
           latch_input:
           process(reset_l,clk)
           begin
                   if (reset_l = '0') then
                           tmp         <= (others => '0');
                   elsif rising_edge(clk) then
                           tmp         <= debug_dat;
                   end if;
           end process latch_input;

--------------------------------
-- Debug Show
--------------------------------
           with tmp(3 downto 0) select
                   data0 <=  "11000000" when "0000",
                           "11111001" when "0001",
```

```
                "10100100" when "0010",
                "10110000" when "0011",
                "10011001" when "0100",
                "10010010" when "0101",
                "10000010" when "0110",
                "11111000" when "0111",
                "10000000" when "1000",
                "10010000" when "1001",
                "10001000" when "1010",
                "10000011" when "1011",
                "10100111" when "1100",
                "10100001" when "1101",
                "10000110" when "1110",
                "10001110" when others;
with tmp(7 downto 4) select
        data1 <= "11000000" when "0000",
                "11111001" when "0001",
                "10100100" when "0010",
                "10110000" when "0011",
                "10011001" when "0100",
                "10010010" when "0101",
                "10000010" when "0110",
                "11111000" when "0111",
                "10000000" when "1000",
                "10010000" when "1001",
                "10001000" when "1010",
                "10000011" when "1011",
                "10100111" when "1100",
                "10100001" when "1101",
                "10000110" when "1110",
                "10001110" when others;
with tmp(11 downto 8) select
        data2 <= "11000000" when "0000",
                "11111001" when "0001",
                "10100100" when "0010",
                "10110000" when "0011",
                "10011001" when "0100",
                "10010010" when "0101",
                "10000010" when "0110",
                "11111000" when "0111",
                "10000000" when "1000",
                "10010000" when "1001",
                "10001000" when "1010",
                "10000011" when "1011",
                "10100111" when "1100",
                "10100001" when "1101",
                "10000110" when "1110",
                "10001110" when others;
with tmp(15 downto 12) select
        data3 <= "11000000" when "0000",
                "11111001" when "0001",
                "10100100" when "0010",
                "10110000" when "0011",
                "10011001" when "0100",
                "10010010" when "0101",
                "10000010" when "0110",
                "11111000" when "0111",
                "10000000" when "1000",
                "10010000" when "1001",
                "10001000" when "1010",
                "10000011" when "1011",
                "10100111" when "1100",
                "10100001" when "1101",
```

```
                                "10000110" when "1110",
                                "10001110" when others;
----------------------Display 7-Segment -------------------------
   divide_clock:
          Process(reset_l,clk)
          begin
                    if (reset_l = '0') then
                              cnt <= (others => '0');
                    elsif rising_edge(clk) then
                              cnt <= cnt+1;
                    end if;
          end process divide_clock;
   show_7seg:
   process(reset_l,clk)
   begin
          if (reset_l = '0') then
                    state     <= "00";
                    seg_en    <= "0000";
                    seg_dat   <= (others => '0');
          elsif rising_edge(clk) then
                     if (cnt = 65535) then
                    state <= state+1;

              If (show_input = '0') then
                    case state is
                       when "00" =>
                              seg_en  <= "0001";
                              seg_dat   <= data0;
                       when "01" =>
                              seg_en  <= "0010";
                              seg_dat   <= data1;
                       when "10" =>
                              seg_en  <= "0100";
                              seg_dat   <= data2;
                       when others =>
                              seg_en  <= "1000";
                              seg_dat   <= data3;
                    end case;
                else
                    case state is
                       when "00" =>
                              seg_en  <= "0001";
                              seg_dat   <= dat4;
                       when "01" =>
                              seg_en  <= "0010";
                              seg_dat   <= dat3;
                       when "10" =>
                              seg_en  <= "0100";
                              seg_dat   <= dat2;
                       when others =>
                              seg_en  <= "1000";
                              seg_dat   <= dat1;
                    end case;
                end if;
          else
                state  <= state;
                seg_dat <= seg_dat;
         end if;
       end if;
   end process show_7seg;
   to_7seg <= seg_dat;
end rtl;
```

gen_strobe.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity gen_strobe is
	port(ram_ack_addr : in std_logic;
		ram_ack_data	: in std_logic;
		rom_ack_addr	: in std_logic;
		clk		: in std_logic;
		reset_l		: in std_logic;
		rst_async	: out std_logic;
		ack_rom_addr	: out std_logic;
		ack_ram_addr	: out std_logic;
		ack_ram_data	: out std_logic
		);
end gen_strobe;

Architecture rtl of gen_strobe is
	signal ram_1x_addr,rom_1x_addr,ram_1x_data	: std_logic;
	signal tram_addr1,trom_addr1,tram_data1		: std_logic;
	signal cnt : unsigned(4 downto 0);
	signal state : std_logic_vector(1 downto 0);
begin
	gen_ram_1x_addr:
	process(ram_ack_addr,clk)
	begin
		if (ram_ack_addr = '0') then
			tram_addr1	<= '0';
			ram_1x_addr	<= '0';
		elsif rising_edge(clk) then
			tram_addr1	<= '1';
			ram_1x_addr	<= tram_addr1;
		end if;
	end process gen_ram_1x_addr;

	gen_rom_1x_addr:
	process(rom_ack_addr,clk)
	begin
		if (rom_ack_addr = '0') then
			trom_addr1	<= '0';
			rom_1x_addr	<= '0';
		elsif rising_edge(clk) then
			trom_addr1	<= '1';
			rom_1x_addr	<= trom_addr1;
		end if;
	end process gen_rom_1x_addr;

	gen_ram_1x_data:
	process(ram_ack_data,clk)
	begin
		if (ram_ack_data = '0') then
			tram_data1	<= '0';
			ram_1x_data	<= '0';
		elsif rising_edge(clk) then
			tram_data1	<= '1';
			ram_1x_data	<= tram_data1;
		end if;
	end process gen_ram_1x_data;

	ack_rom_addr	<= rom_1x_addr;
	ack_ram_addr	<= ram_1x_addr;
```

```
	ack_ram_data	<= ram_1x_data;
-----------------------------------------
-- Debault reset button
-----------------------------------------
	gen_state:
	process(clk)
	begin
		if rising_edge(clk) then
		 case state is
			when "00" =>
				if (reset_l = '0') then
					state <= "01";
				else
					state <= "00";
				end if;
			when "01" =>
				if (cnt(4) = '1') then
					state <= "10";
				else
					state <= "01";
				end if;
			when "10" =>
				if (reset_l = '1') then
					state <= "00";
				else
					state <= "10";
				end if;
			when others => state <= state;
		 end case;
		end if;
	end process gen_state;

	gen_cnt:
	process(clk)
	begin
		if rising_edge(clk) then
			case state is
			  when "00" => cnt <= (others => '0');
			  when "01" => cnt <= cnt+1;
			  when others => cnt <= cnt;
			end case;
		end if;
	end process gen_cnt;

	rst_async <= '1' when (state = "00") else '0';
end rtl;
```

asyn_cpu.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity asyn_cpu is
   port(reset_l              : in std_logic;
        gclk                 : in std_logic;
        switch               : in std_logic_vector(7 downto 0);
        show_input           : in std_logic;
        led                  : out std_logic_vector(3 downto 0);
        to_7seg              : out std_logic_vector(7 downto 0);
        seg_en               : out std_logic_vector(3 downto 0);
        ------ DEBUG -----------
        addr_rom             : out std_logic_vector(9 downto 0);
        dat_rom              : out std_logic_vector(15 downto 0);
        addr_ram             : out std_logic_vector(9 downto 0);
        we_ram               : out std_logic;
        datwr_ram            : out std_logic_vector(7 downto 0);
        datrd_ram            : out std_logic_vector(7 downto 0));
end asyn_cpu;

Architecture rtl of asyn_cpu is
---------------------- CPU -----------------------------------
        component a
        port(   reset_l              : in std_logic;
                a_alu_req            : in std_logic;
                a_ram_req            : in std_logic;
                a_r0_req             : in std_logic;
                a_r1_req             : in std_logic;
                a_shf_req            : in std_logic;
                alu_to_a             : in std_logic_vector(7 downto 0);
                alu_to_a_l           : in std_logic_vector(7 downto 0);
                r0_to_a              : in std_logic_vector(7 downto 0);
                r0_to_a_l            : in std_logic_vector(7 downto 0);
                r1_to_a              : in std_logic_vector(7 downto 0);
                r1_to_a_l            : in std_logic_vector(7 downto 0);
                data_to_a            : in std_logic_vector(7 downto 0);
                data_to_a_l          : in std_logic_vector(7 downto 0);
                ir_to_a              : in std_logic_vector(7 downto 0);
                ir_to_a_l            : in std_logic_vector(7 downto 0);
                shf_to_a             : in std_logic_vector(7 downto 0);
                shf_to_a_l           : in std_logic_vector(7 downto 0);
                a_to_alu             : out std_logic_vector(7 downto 0);
                a_to_alu_l           : out std_logic_vector(7 downto 0);
                a_to_r0              : out std_logic_vector(7 downto 0);
                a_to_r0_l            : out std_logic_vector(7 downto 0);
                a_to_r1              : out std_logic_vector(7 downto 0);
                a_to_r1_l            : out std_logic_vector(7 downto 0);
                a_to_ram             : out std_logic_vector(7 downto 0);
                a_to_ram_l           : out std_logic_vector(7 downto 0);
                a_to_shf             : out std_logic_vector(7 downto 0);
                a_to_shf_l           : out std_logic_vector(7 downto 0);
                wr_a_ack             : out std_logic;
                zero                 : out std_logic;
                zero_l               : out std_logic);
        end component;
        component alu
        port(   a_to_alu : in std_logic_vector(7 downto 0);
                a_to_alu_l           : in std_logic_vector(7 downto 0);
                data_to_alu          : in std_logic_vector(7 downto 0);
                data_to_alu_l        : in std_logic_vector(7 downto 0);
```

```
        r0_to_alu       : in std_logic_vector(7 downto 0);
        r0_to_alu_l     : in std_logic_vector(7 downto 0);
        r1_to_alu       : in std_logic_vector(7 downto 0);
        r1_to_alu_l     : in std_logic_vector(7 downto 0);
        ir_to_alu       : in std_logic_vector(7 downto 0);
        ir_to_alu_l     : in std_logic_vector(7 downto 0);
        add             : in std_logic;
        sub             : in std_logic;
        orr             : in std_logic;
        andd            : in std_logic;
        xorr            : in std_logic;
        carry_out       : out std_logic;
        carry_out_l     : out std_logic;
        aluout          : out std_logic_vector(7 downto 0);
        aluout_l        : out std_logic_vector(7 downto 0));
end component;
component ctl
port(   reset_l         : in std_logic;
        opcod           : in std_logic_vector(7 downto 0);
        carry_out       : in std_logic;
        carry_out_l     : in std_logic;
        zero            : in std_logic;
        zero_l          : in std_logic;
        ir_ctl_ack      : in std_logic;
        wr_a_ack        : in std_logic;
        wr_pc_ack       : in std_logic;
        wr_inc_ack      : in std_logic;
        pc_inc_ack      : in std_logic;
        a_r0_ack        : in std_logic;
        a_r1_ack        : in std_logic;
        data_ram_ack    : in std_logic;
        pc_sp_ack       : in std_logic;
        wr_ram_ack      : in std_logic;
        alu_ack         : in std_logic;
        shift_ack       : in std_logic;
        do_shl          : out std_logic;
        do_shr          : out std_logic;
        add             : out std_logic;
        sub             : out std_logic;
        orr             : out std_logic;
        andd            : out std_logic;
        xorr            : out std_logic;
        out_to_a        : out std_logic;
        pc_rom_req      : out std_logic;
        pc_inc_req      : out std_logic;
        pc_sp_req       : out std_logic;
        a_alu_req       : out std_logic;
        a_ram_req       : out std_logic;
        a_r0_req        : out std_logic;
        a_r1_req        : out std_logic;
        a_shf_req       : out std_logic;
        send_to_pc      : out std_logic;
        r0_alu_req      : out std_logic;
        r0_a_req        : out std_logic;
        r0_ram_req      : out std_logic;
        r1_alu_req      : out std_logic;
        r1_a_req        : out std_logic;
        r1_ram_req      : out std_logic;
        data_alu_req    : out std_logic;
        data_a_req      : out std_logic;
        sp_pc_req       : out std_logic;
        shf_a_req       : out std_logic;
        ir_alu_req      : out std_logic
```

```
            ir_a_req            : out std_logic;
            ir_pc_req           : out std_logic;
            ir_ram_req          : out std_logic;
            ram_read            : out std_logic);
end component;
component data_reg
port(       reset_l             : in std_logic;
            ram_to_data         : in std_logic_vector(7 downto 0);
            ram_to_data_l       : in std_logic_vector(7 downto 0);
            data_alu_req        : in std_logic;
            data_a_req          : in std_logic;
            data_ram_ack        : out std_logic;
            data_to_a           : out std_logic_vector(7 downto 0);
            data_to_a_l         : out std_logic_vector(7 downto 0);
            data_to_alu         : out std_logic_vector(7 downto 0);
            data_to_alu_l       : out std_logic_vector(7 downto 0));
end component;
component ir
port(       reset_l             : in std_logic;
            data_rom            : in std_logic_vector(15 downto 0);
            data_rom_l          : in std_logic_vector(15 downto 0);
            ir_alu_req          : in std_logic;
            ir_a_req            : in std_logic;
            ir_pc_req           : in std_logic;
            ir_ram_req          : in std_logic;
            ir_ctl_ack          : out std_logic;
            ir_to_alu           : out std_logic_vector(7 downto 0);
            ir_to_alu_l         : out std_logic_vector(7 downto 0);
            ir_to_pc            : out std_logic_vector(9 downto 0);
            ir_to_pc_l          : out std_logic_vector(9 downto 0);
            ir_to_a             : out std_logic_vector(7 downto 0);
            ir_to_a_l           : out std_logic_vector(7 downto 0);
            ir_to_ram           : out std_logic_vector(9 downto 0);
            ir_to_ram_l         : out std_logic_vector(9 downto 0);
            opcode              : out std_logic_vector(7 downto 0));
end component;
component mux_output
port(       reset_l             : in std_logic;
            aluout              : in std_logic_vector(7 downto 0);
            aluout_l            : in std_logic_vector(7 downto 0);
            out_to_a            : in std_logic;
            alu_to_a            : out std_logic_vector(7 downto 0);
            alu_to_a_l          : out std_logic_vector(7 downto 0);
            alu_ack             : out std_logic);
end component;
component pc
port(       reset_l             : in std_logic;
            pc_inc_req          : in std_logic;
            pc_sp_req           : in std_logic;
            pc_out              : in std_logic_vector(9 downto 0);
            pc_out_l            : in std_logic_vector(9 downto 0);
            sp_to_pc            : in std_logic_vector(9 downto 0);
            sp_to_pc_l          : in std_logic_vector(9 downto 0);
            ir_to_pc            : in std_logic_vector(9 downto 0);
            ir_to_pc_l          : in std_logic_vector(9 downto 0);
            pc_to_sp            : out std_logic_vector(9 downto 0);
            pc_to_sp_l          : out std_logic_vector(9 downto 0);
            pc_to_rom           : out std_logic_vector(9 downto 0);
            pc_to_inc           : out std_logic_vector(9 downto 0);
            pc_to_inc_l         : out std_logic_vector(9 downto 0);
            wr_inc_ack          :out std_logic;
            wr_pc_ack           : out std_logic);
end component;
```

```
component pc_inc
port(       reset_l             : in std_logic;
            send_to_pc          : in std_logic;
            pc_to_inc           : in std_logic_vector(9 downto 0);
            pc_to_inc_l         : in std_logic_vector(9 downto 0);
            pc_inc_ack          : out std_logic;
            pc_out              : out std_logic_vector(9 downto 0);
            pc_out_l            : out std_logic_vector(9 downto 0));
end component;
component r0
port(       reset_l             : in std_logic;
            r0_ram_req          : in std_logic;
            r0_alu_req          : in std_logic;
            r0_a_req            : in std_logic;
            a_to_r0             : in std_logic_vector(7 downto 0);
            a_to_r0_l           : in std_logic_vector(7 downto 0);
            a_r0_ack            : out std_logic;
            r0_to_a             : out std_logic_vector(7 downto 0);
            r0_to_a_l           : out std_logic_vector(7 downto 0);
            r0_to_ram           : out std_logic_vector(7 downto 0);
            r0_to_ram_l         : out std_logic_vector(9 downto 0);
            r0_to_alu           : out std_logic_vector(7 downto 0);
            r0_to_alu_l         : out std_logic_vector(7 downto 0));
end component;
component r1
port(       reset_l             : in std_logic;
            r1_ram_req          : in std_logic;
            r1_alu_req          : in std_logic;
            r1_a_req            : in std_logic;
            a_to_r1             : in std_logic_vector(7 downto 0);
            a_to_r1_l           : in std_logic_vector(7 downto 0);
            a_r1_ack            : out std_logic;
            r1_to_a             : out std_logic_vector(7 downto 0);
            r1_to_a_l           : out std_logic_vector(7 downto 0);
            r1_to_ram           : out std_logic_vector(7 downto 0);
            r1_to_ram_l         : out std_logic_vector(9 downto 0);
            r1_to_alu           : out std_logic_vector(7 downto 0);
            r1_to_alu_l         : out std_logic_vector(7 downto 0));
end component;
component ram_mmu
port(       reset_l             : in std_logic;
            ir_to_ram           : in std_logic_vector(9 downto 0);
            ir_to_ram_l         : in std_logic_vector(9 downto 0);
            r0_to_ram           : in std_logic_vector(7 downto 0);
            r0_to_ram_l         : in std_logic_vector(9 downto 0);
            r1_to_ram           : in std_logic_vector(7 downto 0);
            r1_to_ram_l         : in std_logic_vector(9 downto 0);
            a_to_ram            : in std_logic_vector(7 downto 0);
            a_to_ram_l          : in std_logic_vector(7 downto 0);
            ram_read            : in std_logic;
            ack_ram_addr        : in std_logic;
            ack_ram_data        : in std_logic;
            ram_ack_addr        : out std_logic;
            ram_ack_data        : out std_logic;
            ram_to_data         : out std_logic_vector(7 downto 0);
            ram_to_data_l       : out std_logic_vector(7 downto 0);
            wr_ram_ack          : out std_logic;
            ram_addr            : out std_logic_vector(9 downto 0);
            ram_re              : out std_logic;
            ram_we              : out std_logic;
            ram_data            : in std_logic_vector(7 downto 0);
            ram_data_wr         : out std_logic_vector(7 downto 0));
end component;
```

```
        component rom_mmu
        port(       reset_l             : in std_logic;
                    pc_to_rom           : in std_logic_vector(9 downto 0);
                    rom_data            : in std_logic_vector(15 downto 0);
                    ack_rom_addr        : in std_logic;
                    rom_re              : out std_logic;
                    rom_addr            : out std_logic_vector(9 downto 0);
                    data_rom            : out std_logic_vector(15 downto 0);
                    data_rom_l          : out std_logic_vector(15 downto 0));
        end component;
        component shift
        port(       reset_l             : in std_logic;
                    do_shr              : in std_logic;
                    do_shl              : in std_logic;
                    shf_a_req           : in std_logic;
                    a_to_shf            : in std_logic_vector(7 downto 0);
                    a_to_shf_l          : in std_logic_vector(7 downto 0);
                    shift_ack           : out std_logic;
                    shf_to_a            : out std_logic_vector(7 downto 0);
                    shf_to_a_l          : out std_logic_vector(7 downto 0));
        end component;
        component sp
        port(       reset_l             : in std_logic;
                    sp_pc_req           : in std_logic;
                    pc_to_sp            : in std_logic_vector(9 downto 0);
                    pc_to_sp_l          : in std_logic_vector(9 downto 0);
                    pc_sp_ack           : out std_logic;
                    sp_to_pc            : out std_logic_vector(9 downto 0);
                    sp_to_pc_l          : out std_logic_vector(9 downto 0));
        end component;

        signal a_to_alu,a_to_alu_l,a_to_r0,a_to_r0_l,a_to_shf,a_to_shf_l: std_logic_vector(7 downto 0);
        signal a_to_r1,a_to_r1_la_to_ram,a_to_ram_l,aluout,aluout_l : std_logic_vector(7 downto 0);
        signal wr_a_ack,zero,zero_l, carry_out,carry_out_l,alu_ack : std_logic;
        signal add,sub,orr,andd,xorr, out_to_a,pc_rom_req,pc_inc_req : std_logic;
        signal pc_sp_req,a_alu_req,a_ram_req,a_r0_req,a_r1_req   : std_logic;
        signal r0_ram_req,r1_alu_req,r1_a_req,r1_ram_req,ir_pc_req          : std_logic;
        signal send_to_pc,r0_alu_req,r0_a_req ,data_alu_req,data_a_req : std_logic;
        signal do_shl,do_shr,a_shf_req,shf_a_req, sp_pc_req,ir_alu_req : std_logic;
        signal data_ram_ack, ir_ram_req,ir_a_req,ram_read,ir_clt_ack          : std_logic;
        signal data_to_alu_l,data_to_a,data_to_a_l : std_logic_vector(7 downto 0);
        signal ir_to_alu,ir_to_alu_l,ir_to_a,ir_to_a_l       : std_logic_vector(7 downto 0);
        signal ir_to_ram,ir_to_ram_l,ir_to_pc,ir_to_pc_l  : std_logic_vector(9 downto 0);
        signal alu_to_a,alu_to_a_l,opcode,data_to_alu : std_logic_vector(7 downto 0);
        signal pc_to_sp,pc_to_sp_l,pc_to_rom,pc_to_inc : std_logic_vector(9 downto 0);
        signal pc_inc_ack,a_r0_ack,a_r1_ack. wr_pc_ack,wr_inc_ack          : std_logic;
        signal pc_out,pc_out_l      , pc_to_inc_l: std_logic_vector(9 downto 0);
        signal r0_to_a,r0_to_a_l,r1_to_a,r1_to_a_l : std_logic_vector(7 downto 0);
        signal r0_to_ram,r0_to_alu,r0_to_alu_l : std_logic_vector(7 downto 0);
        signal r0_to_ram_l,r1_to_ram_l : std_logic_vector(9 downto 0);
        signal r1_to_ram,r1_to_alu,r1_to_alu_l : std_logic_vector(7 downto 0);
        signal ram_to_data,ram_to_data_l        : std_logic_vector(7 downto 0);
        signal ram_addr      : std_logic_vector(9 downto 0);
        signal ram_ack_addr,ram_ack_data. wr_ram_ack,ram_re,ram_we : std_logic;
        signal rom_re,ack_rom_addr, shift_ack .pc_sp_ack             : std_logic;
        signal data_rom,data_rom_l   : std_logic_vector(15 downto 0);
        signal shf_to_a,shf_to_a_l. ram_data_wr : std_logic_vector(7 downto 0);
        signal sp_to_pc,sp_to_pc_l,rom_addr    : std_logic_vector(9 downto 0);
------------------------- MEMORY ----------------------
        component rom_vhd
        port (      clk         : in std_logic;
                    addr        : in std_logic_vector(9 downto 0);
                    en          : in std_logic;
```

```
                dout        : out std_logic_vector(15 downto 0));
        end component;
        component ram_vhd
        port (  clk         : IN std_logic;
                addr        : IN std_logic_VECTOR(9 downto 0);
                en          : IN std_logic;
                we          : IN std_logic;
                din         : IN std_logic_VECTOR(7 downto 0);
                dout        : OUT std_logic_VECTOR(7 downto 0));
        end component;
        component decode_7seg
        port(   reset_l     : in std_logic;
                clk                 : in std_logic;
                data_show           : in std_logic_vector(7 downto 0);
                position            : in std_logic_vector(9 downto 0);
                show                : in std_logic;
                show_input          : in std_logic;
                debug_dat           : in std_logic_vector(15 downto 0);
                to_7seg             : out std_logic_vector(7 downto 0);
                seg_en              : out std_logic_vector(3 downto 0));
        end component;
        component gen_strobe is
        port(   rom_ack_addr        : in std_logic;
                ram_ack_addr        : in std_logic;
                ram_ack_data        : in std_logic;
                clk                 : in std_logic;
                reset_l             : in std_logic;
                rst_async           : out std_logic;
                ack_rom_addr        : out std_logic;
                ack_ram_addr        : out std_logic;
                ack_ram_data        : out std_logic);
        end component;
        component mux_ram_data is
        port(   switch              : in std_logic_vector(7 downto 0);
                 ram_addr           : in std_logic_vector(9 downto 0);
                 ram_we             : in std_logic;
                 ram_data_rd        : in std_logic_vector(7 downto 0);
                 ram_data           : out std_logic_vector(7 downto 0));
        end component;
        component bufgp
        port(   i           : in std_logic;
                o           : out std_logic);
        end component;

        signal debug_dat,rom_data : std_logic_vector(15 downto 0);
        signal ram_data,ram_data_rd : std_logic_vector(7 downto 0);
        signal clk,ack_ram_addr,ack_ram_data : std_logic;
begin
   use_a: a port map(           rst_async,a_alu_req,a_ram_req,a_r0_req,a_r1_req,a_shf_req,
                                alu_to_a,alu_to_a_l,r0_to_a,r0_to_a_l,r1_to_a,r1_to_a_l,
                                data_to_a,data_to_a_l,ir_to_a,ir_to_a_l.shf_to_a,shf_to_a_l,
                                a_to_alu,a_to_alu_l,a_to_r0,a_to_r0_l,a_to_r1,a_to_r1_l,
                                a_to_ram,a_to_ram_l,a_to_shf,a_to_shf_l,wr_a_ack,zero,zero_l);
   use_alu: alu port map(       a_to_alu,a_to_alu_l,data_to_alu,data_to_alu_l,r0_to_alu,
                                r0_to_alu_l,r1_to_alu,r1_to_alu_l,ir_to_alu,ir_to_alu_l,add,sub,
                                orr,andd,xorr,carry_out,carry_out_l,aluout,aluout_l);
   use_ctl: ctl port map(       rst_async,opcode,carry_out,carry_out_l,zero,zero_l,ir_ctl_ack,
                                wr_a_ack,wr_pc_ack,wr_inc_ack,pc_inc_ack,a_ro_ack,a_r1_ack,
                                 data_ram_ack,pc_sp_ack,wr_ram_ack,alu_ack,shift_ack,add,
                                sub,orr,andd,xorr,out_to_a,pc_rom_req,pc_inc_req,pc_sp_req,
                                a_alu_req,a_ram_req,a_r0_req,a_r1_req,a_shf_req,shf_a_req,
                                do_shl,do_shr,send_to_pc,r0_alu_req,r0_a_req,r0_ram_req,
                                r1_alu_req,r1_a_req,r1_ram_req,data_alu_req,data_a_req,
```

```
                                    sp_pc_req,ir_alu_req,ir_a_req,ir_ram_req,ir_pc_req,ram_read);
    use_data_reg: data_reg port map(rst_async,ram_to_data,ram_to_data_l,
                                    data_alu_req,data_a_req,data_ram_ack,data_to_a,
                                    data_to_a_l,data_to_alu,data_to_alu_l,
    use_ir: ir port map(            rst_async,data_rom,data_rom_l,ir_alu_req,ir_a_req,ir_pc_req,
                                    ir_ram_req,ir_ctl_ack,ir_to_alu,ir_to_alu_l,ir_to_pc,ir_to_pc_l,
                                    ir_to_a,ir_to_a_l,ir_to_ram,ir_to_ram_l,ir_to_pc,ir_to_pc_l);
   use_mux_output: mux_output port map(rst_async,aluout,aluout_l,out_to_a,alu_to_a,
                                       alu_to_a_l,alu_ack);
   use_pc : pc port map(            rst_async,pc_inc_req,pc_sp_req,pc_out,pc_out_l,sp_to_pc,
                                    sp_to_pc_l,ir_to_pc,ir_to_pc_l,pc_to_sp,pc_to_sp_l,pc_to_rom,
                                    pc_to_inc,pc_to_inc_l,wr_inc_ack,wr_pc_ack);
   use_pc_inc : pc_inc port map(rst_async,send_to_pc,pc_to_inc,pc_to_inc_l,
                                pc_inc_ack,pc_out,pc_out_l);
   use_r0: r0 port map(             rst_async,r0_ram_req,r0_alu_req,r0_a_req,a_to_r0,a_to_r0_l,
                                    a_r0_ack,r0_to_a,r0_to_a_l,r0_to_ram,r0_to_ram_l,r0_to_alu,
                                    r0_to_alu_l);
   use_r1: r1 port map(             rst_async,r1_ram_req,r1_alu_req,r1_a_req,a_to_r1,a_to_r1_l,
                                    a_r1_ack,r1_to_a,r1_to_a_l,r1_to_ram,r1_to_ram_l,r1_to_alu,
                                    r1_to_alu_l);
   use_ram_mmu: ram_mmu port map(rst_async,ir_to_ram,ir_to_ram_l,r0_to_ram,
                                 r0_to_ram_l,r1_to_ram,r1_to_ram_l,a_to_ram,
                                 a_to_ram_l,ram_read,ack_ram_addr,
                                 ack_ram_data,ram_ack_addr,ram_ack_data,
                                 ram_to_data,ram_to_data_l,wr_ram_ack,
                                 ram_addr,ram_re,ram_we,ram_data,ram_data_wr);
   use_rom_mmu: rom_mmu port map(rst_async,pc_to_rom,rom_data,ack_rom_addr,
                                 rom_re,rom_addr,data_rom,data_rom_l);
   use_shift: shift port map(   rst_async,do_shr,do_shl,shf_a_req,a_to_shf,a_to_shf_l,
                                shift_ack,shf_to_a,shf_to_a_l);
   use_sp: sp port map(         rst_async,sp_pc_req,pc_to_sp,pc_to_sp_l,pc_sp_ack,
                                sp_to_pc,sp_to_pc_l);
-----------------------------------------------------
-- Out Side Asynchronous CPU
-----------------------------------------------------
   use_decode_7seg: decode_7seg port map(       reset_l,clk,ram_data_wr,ram_addr,
                                                ram_we,show_input,debug_dat,to_7seg,seg_en);
   debug_dat        <= "00000000" & switch;
   use_rom_vhd: rom_vhd port map(rom_addr,clk,rom_re,rom_data);
   use_ram_vhd: ram_vhd port map(ram_addr,clk,ram_data_wr,ram_re,ram_we,ram_data_rd);
   use_gen_strobe: gen_strobe port map(ram_ack_addr,ram_ack_data,pc_rom_req,clk,
                                       reset_l,rst_async,ack_rom_addr,ack_ram_addr,
                                       ack_ram_data);
   use_mux_ram_data: mux_ram_data port map(switch,ram_addr,ram_we,ram_data_rd,ram_data);
   use_bufgp: bufgp port map(gclk,clk);

   led(1 downto 0)  <= "11";
   led(3)  <= rst_async;
   led(2)  <= keep_fin;
-------------------------------------------------------------------------
-- DEBUG
-------------------------------------------------------------------------
   we_ram           <= ram_we;
   addr_ram         <= ram_addr;
   datwr_ram        <= ram_data_wr;
   datrd_ram        <= ram_data_rd;
   addr_rom         <= rom_addr;
   dat_rom          <= rom_data;
end rtl
```

ภาคผนวก ค

ตารางรหัสช่วยจำ

```
1,"Rs","R0","R1","Rx"

*; NUM START LENGTH    EXP    LOW    HIGH        COMMENT
      1,      8,          8,        #,       0,      255        ; Immediate Value
      2,      14,         2,        @1,      0,      3          ; Register
      3,      6,          10        #,       0,      1023       ; Direct Addressing

*
1,        A,#{1}^0000:                ; DES,#SRC (Immediate Operand)
2,        A,{2}^0400:                 ; DES,SRC  (Register Operand)
3,        A,@{3}^0800:                ; DES,@SRC (Direct Addressing Operand)
4,        A,@{2}^0C00:                ; DES,@SRC (Register Index Addressing Operand)
5,        {3}^0000:                   ; SRC (jnz,jz,jc,jnc)
6,        A^0000:                     ; SRC (Shift Left, Shift Right)

*
ST        |2^0000:          ; ST     A,R0
ST        |3^0000:          ; ST     A,@033H
ST        |4^0000:          ; ST     A,@R0

LD        |1^1000:          ; LD     A,#33H
LD        |2^1000:          ; LD     A,R0
LD        |3^1000:          ; LD     A,@033H
LD        |4^1000:          ; LD     A,@R0

SL        |6^2000:          ; SL     A
SR        |6^2800:          ; SR     A

AND       |1^3000:          ; AND    A,#33H
AND       |2^3000:          ; AND    A,R0
AND       |3^3000:          ; AND    A,@033H
AND       |4^3000:          ; AND    A,@R0

OR        |1^4000:          ; OR     A,#33H
OR        |2^4000:          ; OR     A,R0
OR        |3^4000:          ; OR     A,@033H
OR        |4^4000:          ; OR     A,@R0

XOR       |1^5000:          ; XOR    A,#33H
XOR       |2^5000:          ; XOR    A,R0
XOR       |3^5000:          ; XOR    A,@033H
XOR       |4^5000:          ; XOR    A,@R0

SUB       |1^6000:          ; SUB    A,#33H
SUB       |2^6000:          ; SUB    A,R0
SUB       |3^6000:          ; SUB    A,@033H
SUB       |4^6000:          ; SUB    A,@R0

ADD       |1^7000:          ; ADD    A,#33H
ADD       |2^7000:          ; ADD    A,R0
ADD       |3^7000:          ; ADD    A,@033H
ADD       |4^7000:          ; ADD    A,@R0

JZ        |5^8000:          ; JZ     LABEL2
JNZ       |5^8400:          ; JNZ    LABEL2
JC        |5^8800:          ; JC     LABEL2
JNC       |5^8C00:          ; JNC    LABEL2
JMP       |5^9000:          ; JMP    LABEL1
CALL      |5^9400:          ; CALL   LABEL1
RET       ^9800:           ; RET
```

ภาคผนวก ง

โปรแกรมหาค่ารากที่สอง

```
CPU     "CPU.TBL"
HOF     "BIN8"
WDLN    2
;M(0)   = TEMP OF DATA
;M(1)   = Qt
;M(2)   = LOOPCNT
;M(3)   = 03H
;R0     = RES (R)
;R1     = QUADIANT (Q)
;M(15)  = OLD DATA
ORG     0000H
START_PROGRAM:
        LD              A,@3F0H         ; A = DATA      ------------------------
        ST              A,@00H          ; M(00) = DATA  --              --
        ST              A,@0FH          ; M(15) = DATA  --              --
        LD              A,#07H          ;               --              --
        ST              A,@02H          ;               --              --
        LD              A,#00H          ;               --              --
        ST              A,R0            ; R = 00H       --      Initial --
        ST              A,R1            ; Q = 00H       --              --
        LD              A,#0FFH         ;               --              --
        ST              A,@01H          ; Qt = FFH      --              --
        LD              A,#03H          ;               --              --
        ST              A,@03H          ; M(03) = 03H   --              --
        CALL            KEEP_RES        ;               ------------------------
START_DATA_NOT_ZERO:
        LD              A,#80H
        AND             A,R0            ; RES >= 0?
        JNZ             RES_LESS_ZERO
        LD              A,R1            ; RES >= 0
        SL              A
        OR              A,#01H
        ST              A,R1            ; Q = Q[MSB-1..0] & "1"
        XOR             A,#0FFH         ; Q = NOT Q = Qt
        JMP             DO_SAME
RES_LESS_ZERO:
        LD              A,R1            ; RES < 0
        SL              A
        ST              A,R1            ; Q = Q[MSB-1..0] & "0" = Qt
DO_SAME:
        SL              A
        SL              A
        OR              A,@03H
        ST              A,@01H          ; Qt = Qt[MSB-2..0] & "11"
        CALL            KEEP_RES
        LD              A,@02H
        SR              A
        ST              A,@02H
        JNZ             START_DATA_NOT_ZERO
; **********************
; DATA = 00H ALREADY
; M(00) = LOOPCNT
; M(01) = Qt
; M(04)   = RES
; M(05) = Q
; R0 = 04H POINT TO RES
; R1 = 05H POINT TO Q
; **********************
        LD              A,#01H          ;
        ST              A,@00H          ; LOOPCNT = F0H
        LD              A,R0
```

```
        ST          A,@04H          ; M(04) = RES
        LD          A,#04H
        ST          A,R0            ; R0 = 04H POINT TO RES
        LD          A,R1
        ST          A,@05H          ; M(05) = Q
        LD          A,#05H
        ST          A,R1            ; R1 = 05H POINT TO Q
START_DATA_ZERO:
        LD          A,#80H
        AND         A,@R0           ; RES >= 0?
        JNZ         RES_LESS_0
        LD          A,@R1           ; RES >= 0
        SL          A
        OR          A,#01H
        ST          A,@R1           ; Q = Q[MSB-1..0] & "1"
        LD          A,#0FFH
        SUB         A,@R1
        JMP         FIND_RES
RES_LESS_0:
        LD          A,@R1           ; RES < 0
        SL          A
        ST          A,@R1           ; Q = Q[MSB-1..0] & "0"
FIND_RES:
        SL          A
        SL          A
        OR          A,#03H
        ST          A,@01H          ; Qt = Qt[MSB-2..0] & "11"
        LD          A,@R0
        SL          A
        SL          A
        ST          A,@R0           ; RES = RES[MSB-2..0] & "00"
        LD          A,@01H
        ADD         A,@R0
        ST          A,@R0           ; RES = RES + Qt
        LD          A,#01H
        XOR         A,@00H
        ST          A,@00H          ; LOOPCNT = "0" & LOOPCNT[MSB..1]
        JZ          START_DATA_ZERO
; ***************************
; M(00) = LOOPCNT
; M(01) = Qt(h)
; M(02) = Qt(l)
; M(05) = Q
; R1    = RES(h)
; R0    = RES(l)
; ***************************
        LD      A,#07H
        ST      A,@00H
        LD      A,@01H
        ST      A,@02H
        LD      A,@R0
        ST      A,R0
        AND     A,#80H
        JZ      SIGH_0
        LD      A,#0FFH
SIGH_0:
        ST      A,R1
START_16_BIT:
        LD      A,#80H
        AND     A,R1                ; RES >= 0?
        JNZ     RHIGH_1
        LD      A,@05H              ; RES >= 0
        SL      A
        OR      A,#01H              ; Q = Q[MSB-1..0] & "1"
        ST      A,@05H
        LD      A,#0FCH
        ST      A,@01H              ; Qt(h) = "FC"
        LD      A,#0FFH
        XOR     A,@05H              ; Qt(l) = NOT Q
```

```
        JMP     SHIFT_Qt
RHIGH_1:
        LD      A,#00H
        ST      A,@01H                  ; Qt(h) = "00"
        LD      A,@05H                  ; RES < 0
        SL      A
        ST      A,@05H                  ; Q = Q[MSB-1..0] & "0"
SHIFT_Qt:
        ST      A,@06H
        SL      A
        SL      A
        OR      A,#03H                  ; Qt(l) = Q[MSB-2..0] & "11"
        ST      A,@02H
        LD      A,@06H
        SR      A
        SR      A
        SR      A
        SR      A
        SR      A
        SR      A
        OR      A,@01H
        ST      A,@01H                  ; Qt(h) = Qt(l)[7..6]
ADD_RES:
        LD      A,R1
        SL      A
        SL      A
        ST      A,R1
        LD      A,R0
        SR      A
        SR      A
        SR      A
        SR      A
        SR      A
        SR      A
        OR      A,R1
        ST      A,R1
        LD      A,R0
        SL      A
        SL      A
        ADD     A,@02H                  ; RES(l) = RES(l) + Qt
        ST      A,R0
        JNC     ADD_RES_Qt_HIGH
        LD      A,#01H
        ADD     A,R1                    ; RES(h) = RES(h) + "1"
        ST      A,R1
ADD_RES_Qt_HIGH:
        LD      A,@01H
        ADD     A,R1                    ; RES(h) = RES(h) + Qt(h)
        ST      A,R1
CHK_ROUND:
        LD      A,@00H
        SR      A
        ST      A,@00H
        JNZ     START_16_BIT
        LD      A,#80H
        AND     A,R1    ; RES >= 0?
        JNZ     CONVERT_HEX
        LD      A,#01H
        ADD     A,@05H ; Q = Q+"1"
        ST      A,@05H
; **********************
; CONVERT HEX TO BCD
; AND SHOW 7 SEGMENT
; M(05) = Q
; R0 = POINT TO DEC OF Q
; M(06) = DEC OF Q (UNIT PART)
; M(07) = DEC OF Q (TEN PART)
; M(08) = DEC OF Q (Position1)
; M(09) = DEC OF Q (Position2)
```

```
; M(0A) = DEC OF Q (Position3)
; M(0B) = DEC OF Q (Position4)
; **********************
CONVERT_HEX:
        LD      A,#00H
        ST      A,@08H
        ST      A,@09H
        ST      A,@0AH
        ST      A,@0BH
        LD      A,@05H
        AND     A,#0F0H
        SR      A
        SR      A
        SR      A
        SR      A
        ST      A,@04H
        SUB     A,#0AH
        JC      Q_LESS_10
        CALL    DECODE_7SE
        ST      A,@07H          ; KEEP UNIT PART OF Q
        LD      A,#0F9H         ;
        ST      A,@06H          ; KEEP "1" AT TEN PART OF Q
        JMP     BCD_AFTER_POINT
Q_LESS_10:
        LD      A,@04H
        CALL    DECODE_7SEG
        ST      A,@07H          ; KEEP UNTI PART OF Q
        LD      A,#0FFH
        ST      A,@06H          ; KEEP "0" AT TEN PART OF Q
BCD_AFTER_POINT:
        LD      A,@05H
        ST      A,R1            ; R1 = Q
        AND     A,#08H
        JZ      CHK_BIT2
        LD      A,#05H
        ST      A,@08H          ; X = 0.5
CHK_BIT2:
        LD      A,#04H
        AND     A,R1
        JZ      CHK_BIT1
        LD      A,#05H
        ST      A,@09H          ; X = X + 0.05
        LD      A,#08H
        ST      A,R0            ; R0 = 08H
        LD      A,#02H
        XOR     A,@R0
        ST      A,@R0           ; X = X + 0.2
CHK_BIT1:
        LD      A,R1
        AND     A,#02H
        JZ      CHK_BIT0
        ST      A,R1            ; R1 = 02H
        LD      A,#05H
        ST      A,@0AH          ; X = X + 0.005
        LD      A,@09H
        ADD     A,R1
        ST      A,@09H          ; X = X + 0.02
        LD      A,#01H
        ADD     A,@08H
        ST      A,@08H          ; X = X + 0.1
CHK_BIT0:
        LD      A,#01H
        AND     A,@05H
        JZ      CONVERT_7SEG
        LD      A,#05H
        ST      A,@0BH          ; X = X + 0.0005
        LD      A,#02H
        ST      A,R1            ; R1 = 02H
        LD      A,@0AH
```

```
        XOR     A,R1
        ST      A,@0AH          ; X = X + 0.002
        LD      A,#0AH
        ST      A,R1            ; R1 = 0AH
        LD      A,@09H
        ADD     A,#06H
        ST      A,@09H          ; X = X + 0.06

        SUB     A,R1
        JC      CONVERT_7SEG
        ST      A,@09H
        LD      A,#01H
        ADD     A,@08H
        ST      A,@08H          ; X = X + 0.1 (OPTIONAL)
        LD      A,#07H
        ST      A,R0            ; R0 = 07H
CONVERT_7SEG:
        LD      A,@08H
        CALL    DECODE_7SEG
        ST      A,@08H
        LD      A,@09H
        CALL    DECODE_7SEG
        ST      A,@09H
; ************************************
; SHOW VALUE TO 7 SEGMENT
; ************************************
        ST      A,@3FFH
        LD      A,@08H
        ST      A,@3FEH
        LD      A,#7FH
        AND     A,@07H
        ST      A,@3FDH
        LD      A,@06H
        ST      A,@3FCH
HALT:
        LD      A,@3F0H
        SUB     A,@0FH
        JZ      HALT
        JMP     START_PROGRAM
; ***********************************
; RES = RES[MSB-2..0] & DATA[7..6]
; RES = RES + Qt
; ***********************************
KEEP_RES:
        LD      A,R0
        SL      A
        ST      A,R0            ; RES = RES[MSB-1..0] & "0"
        LD      A,#80H
        AND     A,@00H          ; DATA[MSB] = '1'?
        JZ      J1              ;
        LD      A,R0            ; ** DATA[MSB] = '1' **
        OR      A,#01H          ; A = RES[MSB-1..0] & "1"
        JMP     J11
J1:
        LD      A,R0            ; ** DATA[MSB] = '0' **
J11:
        SL      A
        ST      A,R0            ; RES = RES[MSB-1..0] & "0"
        LD      A,@00H          ;
        SL      A
        AND     A,#80H          ; DATA[MSB] = '1'?
        JZ      J2
        LD      A,#01           ; ** DATA[MSB] = '1' **
        OR      A,R0            ; RES = RES[MSB-1..0] & "1"
        JMP     J22
J2:
        LD      A,R0            ; ** DATA[MSB] = '0' **
J22:
        ADD     A,@01H          ;
```

```
        ST          A,R0        ; RES = RES + Qt
        LD          A,@00H
        SL          A
        SL          A
        ST          A,@00H      ; DATA = DATA[MSB-2..0] & "00";
        RET
; ***************************
; DECODE DATA TO 7SEGMENT
; M(0C) = TEMP
; ***************************
DECODE_7SEG:
        ST          A,@0CH
        SUB         A,#00H
        JNZ         VALUE_1
        LD          A,#0C0H
        JMP         END_DECODE
VALUE_1:
        LD          A,#01H
        SUB         A,@0CH
        JNZ         VALUE_2
        LD          A,#0F9H
        JMP         END_DECODE
VALUE_2:
        LD          A,@0CH
        SUB         A,#02H
        JNZ         VALUE_3
        LD          A,#0A4H
        JMP         END_DECODE
VALUE_3:
        LD          A,#03H
        SUB         A,@0CH
        JNZ         VALUE_4
        LD          A,#0B0H
        JMP         END_DECODE
VALUE_4:
        LD          A,#04H
        SUB         A,@0CH
        JNZ         VALUE_5
        LD          A,#99H
        JMP         END_DECODE
VALUE_5:
        LD          A,#05H
        SUB         A,@0CH
        JNZ         VALUE_6
        LD          A,#92H
        JMP         END_DECODE
VALUE_6:
        LD          A,@0CH
        SUB         A,#06H
        JNZ         VALUE_7
        LD          A,#82H
        JMP         END_DECODE
VALUE_7:
        LD          A,@0CH
        SUB         A,#07H
        JNZ         VALUE_8
        LD          A,#0F8H
        JMP         END_DECODE
VALUE_8:
        LD          A,@0CH
        SUB         A,#08H
        JNZ         VALUE_9
        LD          A,#80H
        JMP         END_DECODE
VALUE_9:
        LD          A,#90H
END_DECODE:
        RET
        END
```

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวปัญจภา เรืองสินทรัพย์ เกิดเมื่อวันที่ 16 ธันวาคม พ.ศ. 2520 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปีการศึกษา 2540 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2542

ในระหว่างการศึกษาได้รับรางวัลชนะเลิศการแข่งขันการประกวดการออกแบบวงจรรวมแห่งชาติครั้งที่ 1 (The National IC Design Contest 2000: NIC2000) ประเภทดิจิตอล ในปีการศึกษา 2543 และได้รับรางวัลชนะเลิศการแข่งขันการประกวดการออกแบบวงจรรวมแห่งชาติครั้งที่ 2 (The National IC Design Contest 2001: NIC2001) ประเภทดิจิตอล ในปีการศึกษา 2544 ซึ่งจัดขึ้นโดยศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ