การออกแบบหน่วยประมวลผลสัญญาณดิจิทัลขนาด 16 บิตที่ประกอบด้วยตัวกรองเอฟไออาร์ที่ปรับความยาวได้

นายรวิวร มะหะสิทธิ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2546

ISBN 974-17-3621-5

A DESIGN OF A 16-BIT DIGITAL SIGNAL PROCESSOR WITH A VARIABLE LENGTH FIR FILTER

Mr.Ravivon Mahasith

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

Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2003

ISBN 974-17-3621-5

หัวข้อวิทยานิพนธ์	การออกแบบหน่วยประมวลผลสัญญาณดิจิทัลขนาด 16 บิตที่ประกอบด้วยตัวกรองเอฟไออาร์ที่ปรับความยาวได้
โดย	นายรวิวร มะหะสิทธิ์
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	อาจารย์ ดร.วันเฉลิม โปรา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

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

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(รองศาสตราจารย์ ดร.เอกชัย ลีลารัศมี)

..... อาจารย์ที่ปรึกษา
(อาจารย์ ดร.วันเฉลิม โปรา)

.....กรรมการ
(อาจารย์ ดร.นิศาชล ตั้งเสงี่ยมวิสัย)

นายรวิวร มะหะสิทธิ์ : การออกแบบหน่วยประมวลผลสัญญาณดิจิทัลขนาด 16 บิตที่ประกอบด้วยตัวกรองเอฟไอาร์ที่ปรับความยาวได้. (A DESIGN OF A 16-BIT DIGITAL SIGNAL PROCESSOR WITH A VARIABLE LENGTH FIR FILTER) อ. ที่ปรึกษา : ดร.วันเฉลิม โปรา, 2 หน้า. ISBN 974-17-3621-5.

วิทยานิพนธ์นี้นำเสนอการออกแบบหน่วยประมวลผลสัญญาณดิจิทัลแบบทศนิยมคงที่ 16 บิต ที่มีตัวกรองเอฟไอาร์ภายใน และอุปกรณ์บริวารอื่นๆ เช่น ตัวตั้งเวลา วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S ดีเอ็มเอ และพอร์ตอินพุต-เอาต์พุต โครงสร้างของหน่วยประมวลผลสัญญาณดิจิทัลมีโครงสร้างแบบไปป์ไลน์ 5 ขั้นตอน และมีลักษณะของชุดคำสั่งแบบริสค์ ตัวกรองเอฟไอาร์ที่ออกแบบสามารถทำงานได้สองลักษณะ โดยสามารถทำงานขนานอย่างอิสระจากการทำงานของหน่วยประมวลผล หรือทำงานเป็นหน่วยคูณและสะสมที่เรียกใช้โดยตรงได้จากหน่วยประมวลผลกลาง เมื่อหน่วยประมวลผลกลางทำงานร่วมกับตัวกรองจะสามารถคำนวณตัวกรองเอฟไอาร์แบบปรับตัวแบบกำลังสองน้อยที่สุดได้ภายใน 1.5N+26 วงรอบคำสั่ง เมื่อ N เป็นความยาวของตัวกรอง หน่วยประมวลผลที่ออกแบบถูกนำมาจำลองการทำงานและสร้างตัวต้นแบบบนเอฟพีจีเอ แล้วจึงนำมาสร้างลายวงจรรวมบนเทคโนโลยีซีมอส 0.35 ไมครอน ผลการทดสอบคุณสมบัติของลายวงจรรวมก่อนนำไปเจือสาร ลายวงจรรวมใช้พื้นที่ประมาณ 5.23 ตารางมิลลิเมตร สามารถทำงานได้ที่ความถี่สูงสุด 120 MHz และกินกำลังไฟประมาณ 5.7 mW/MHz ที่แรงดันไฟเลี้ยง 3.3 โวลต์

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่ออาจารย์ที่ปรึกษา .....

ปีการศึกษา 2546

4470481121 : MAJOR ELECTRICAL ENGINEERING

KEY WORD : DSP PROCESSOR / VLSI DESIGN / COMPUTER ARCHITECTURE / FIR FILTER / FPGA PROTOTYPING

RAVIVON MAHASITH : A DESIGN OF A 16-BIT DIGITAL SIGNAL PROCESSOR WITH A VARIABLE LENGTH FIR FILTER. THESIS ADVISOR : WANCHALERM PORA, Ph.D., 2 pp. ISBN 974-17-3621-5.

This thesis presents a design which consists of a 16-b fixed point digital signal processor with a built-in FIR filter and some peripheral devices such as timer, I^2S interfacing circuit, DMA and I/O ports. The 5-stages pipelined architecture together with RISC based instruction set are employed. The filter can either operate in parallel with the processor or can be configured as a multiply and accumulate unit controlled by the processor. With the LMS adaptive FIR benchmark, the processor can finish the computation in 1.5N+26 cycles. All parts were simulated and implemented on FPGA. The chip is laid out using a 0.35-μm CMOS technology. The approximated chip area is about 5.23 mm^2 and its operation frequency is up to 120 MHz. The power consumption is estimated to be 5.7 mW/MHz at 3.3 V .

Department Electrical Engineering Student's signature .....

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

Academic year 2003 .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่งของอาจารย์ ดร.วันเฉลิม โปรา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านได้ให้คำแนะนำและข้อคิดเห็นต่างๆที่เป็นประโยชน์ ในการวิจัยด้วยดีมาโดยตลอด

ขอขอบคุณโครงการศิษย์ก้นกุฏิของภาควิชาวิศวกรรมไฟฟ้า จุฬาลงกรณ์มหาวิทยาลัย ที่ได้ให้ทุนวิจัยสนับสนุนการวิจัยนี้เป็นอย่างดี

ขอขอบคุณศูนย์พัฒนาธุรกิจออกแบบวงจรรวม (TIDI) ศูนย์อิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) ที่ความสนับสนุนด้านซอฟต์แวร์และเทคโนโลยีที่ใช้ในการออกแบบวงจรรวม และขอขอบคุณพี่ๆ ที่ศูนย์พัฒนาธุรกิจออกแบบวงจรรวม และบริษัท Silicon Craft ที่คอยดูแลและให้คำแนะนำเป็นอย่างดี ถึงจะเป็นเวลาไม่นานแต่ก็รู้สึกดีมากครับ

นอกจากนี้ยังมีเพื่อนๆ พี่ๆ และน้องๆ ทุกคนในห้องปฏิบัติการวิจัยออกแบบและประยุกต์ใช้งานวงจรรวม (IDAR) ที่คอยห่วงใย ให้คำแนะนำและความรื่นเริงมาโดยตลอด

ท้ายนี้ผู้วิจัยใคร่ขอกราบขอบพระคุณ บิดา-มารดา รวมทั้งพี่น้องทั้ง 5 คน ที่สนับสนุนในด้านการเงินและกำลังใจอย่างล้นเหลือ และต้อนรับผู้วิจัยกลับบ้านด้วยรอยยิ้มเสมอมา

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หน้า

สารบัญภาพ (ต่อ)

หน้า

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1 แนวเหตุผลในการทำวิทยานิพนธ์

ปัจจุบันกรรมวิธีสัญญาณดิจิทัลได้ถูกนำมาประยุกต์ใช้งานอย่างกว้างขวาง ตัวอย่างของการประยุกต์ใช้งานกรรมวิธีสัญญาณดิจิทัลได้แก่ การบีบอัดและการเข้ารหัสสัญญาณเสียง การตรวจสอบลายเซ็น และการกำจัดเสียงสะท้อน (Echo Cancellation) เป็นต้น หน่วยประมวลผลสัญญาณดิจิทัล (Digital Signal Processor) ถือได้ว่าเป็นส่วนประกอบหลักในระบบประมวลสัญญาณดิจิทัลเหล่านี้ หน่วยประมวลผลแบบนี้สามารถประมวลผลการดำเนินการทางคณิตศาสตร์ได้อย่างรวดเร็ว และมีชุดคำสั่งที่สนับสนุนการประมวลผลในระบบที่มีสัญญาณเข้ามาในอัตราสม่ำเสมอและต่อเนื่อง

การประยุกต์ใช้งานหลายๆอย่าง เช่น การกำจัดเสียงสะท้อน ต้องการการคำนวณส่วนย่อยๆ หลายส่วน ได้แก่ การคำนวณในส่วนของตัวกรอง และการคำนวณเพื่อปรับค่าสัมประสิทธิ์ เป็นต้น หากเลือกใช้หน่วยประมวลผลสัญญาณดิจิทัลเพียงอย่างเดียวอาจจำเป็นต้องใช้หน่วยประมวลผลที่มีประสิทธิภาพสูงมาก แต่ทว่าหากเลือกใช้ฮาร์ดแวร์เฉพาะงานเพียงอย่างเดียวก็อาจจะทำให้ขาดความยืดหยุ่นในการปรับปรุงการทำงาน ทางเลือกอีกทางหนึ่งคือการเพิ่มฮาร์ดแวร์เฉพาะงาน (Application Specific Hardware) ให้กับหน่วยประมวลผลสัญญาณดิจิทัล จะช่วยลดปริมาณงานของหน่วยประมวลผลลง ทำให้หน่วยประมวลผลมีเวลาว่างสำหรับประมวลผลงานอื่น หรืออีกนัยหนึ่งหน่วยประมวลไม่จำเป็นต้องมีประสิทธิภาพสูงมากก็ได้

การเพิ่มฮาร์ดแวร์ที่ไม่ตรงกับงานที่ใช้จะทำให้เปลืองทรัพยากรฮาร์ดแวร์โดยเปล่าประโยชน์ แต่ทว่าโครงสร้างแบบตัวกรองเอฟไอาร์จัดได้ว่าเป็นองค์ประกอบหนึ่งที่ถูกนำมาใช้งานในระบบประมวลผลสัญญาณดิจิทัลส่วนใหญ่ ไม่ว่าจะเป็นงานกรองสัญญาณ การหาค่าสหสัมพันธ์ (Correlation) หรือการตรวจคุณลักษณะ (Identification) ดังนั้นการเพิ่มเติมฮาร์ดแวร์ตัวกรองเอฟไอาร์เพื่อทำงานร่วมกับหน่วยประมวลผล จะทำให้สามารถใช้งานฮาร์ดแวร์เฉพาะงานได้อย่างกว้างขวาง

วิทยานิพนธ์นี้นำเสนอการพัฒนาหน่วยประมวลผลสัญญาณดิจิทัลแบบใช้งานทั่วไป (General Purpose Digital Signal Processor) ที่ประกอบด้วยฮาร์ดแวร์เฉพาะงานสำหรับการประมวลผลสัญญาณดิจิทัลได้แก่ ตัวกรองเอฟไอาร์ที่สามารถปรับความยาวและทำงานขนานกับหน่วยประมวลผลได้ นอกจากนี้หากใช้ในงานประมวลผลที่ไม่จำเป็นต้องใช้ตัวกรองดังกล่าว หน่วยประมวลผลกลางสามารถปรับโมดการทำงานให้องค์ประกอบของตัวกรองซึ่งได้แก่ หน่วยความจำและวงจรทางคณิตศาสตร์กลายเป็นองค์ประกอบหนึ่งของหน่วยประมวลผลได้ ทำให้หน่วยประมวลผลสามารถทำงานได้ทั้งในโมดการทำงานเฉพาะงานหรือการใช้งานทั่วไปได้ หน่วยประมวลผลที่ออกแบบสามารถประมวลผลสัญญาณที่มีแบนด์วิดธ์ในย่านเสียงได้

1.2 วัตถุประสงค์ของการวิจัย

1. เพื่อออกแบบและพัฒนาโครงสร้างของหน่วยประมวลผลสัญญาณดิจิทัลขนาด 16 บิตที่มีตัวกรองเอฟไอาร์ 64 แท็ปที่ปรับความยาวได้
2. เพื่อสร้างตัวต้นแบบของหน่วยประมวลผลสัญญาณดิจิทัลที่ออกแบบในรูปแบบวงจรรวม
3. เพื่อนำหน่วยประมวลผลที่ออกแบบไปใช้สำหรับการประมวลสัญญาณในย่านเสียง

1.3 ขอบเขตของการวิจัย

1. ออกแบบหน่วยประมวลผลสัญญาณดิจิทัลที่มีคุณสมบัติดังนี้
   - ขนาด 16 บิต มีสถาปัตยกรรมแบบฮาร์วาร์ด โครงสร้างแบบไปป์ไลน์ 5 ขั้นตอน
   - ประกอบด้วยอุปกรณ์บริวารได้แก่ วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S วงจรตั้งเวลาและพอร์ตอินพุต/เอาต์พุต
2. ออกแบบวงจรตัวกรองเอฟไอาร์ที่ออกแบบที่มีคุณสมบัติดังนี้
   - สามารถปรับโมดการทำงานให้มีความยาว 32 64 128 หรือ 256 แท็ป
   - คำนวณตัวอย่างอินพุตแบบมีเครื่องหมายขนาด 16 บิตและสัมประสิทธิ์แบบมีเครื่องหมายขนาด 16 บิต
   - สามารถทำงานขนานกับหน่วยประมวลผลที่ออกแบบ
3. ทดสอบการทำงานของหน่วยประมวลผลที่ออกแบบบนเอฟพีจีเอ
4. ออกแบบลายวงจรรวมของตัวต้นแบบที่สามารถนำไปเจือสารด้วยเทคโนโลยีวงจรรวม

1.4 วิธีดำเนินการวิจัย

1. ศึกษาการทำงานของหน่วยประมวลผลสัญญาณดิจิทัลและการออกแบบตัวกรองดิจิทัล
2. เขียนภาษาบรรยายฮาร์ดแวร์ VHDL เพื่อออกแบบ และจำลองการทำงานของหน่วยประมวลผลบนคอมพิวเตอร์
3. ออกแบบและสร้างบอร์ดที่มีเอฟพีจีเอ และอุปกรณ์สำหรับการทดสอบ
4. สังเคราะห์วงจรหน่วยประมวลผลและทดสอบการทำงานบนเอฟพีจีเอ
5. สังเคราะห์หน่วยประมวลที่ทดสอบบนเอฟพีจีเอลงในเซลล์มาตรฐานของวงจรรวมเทคโนโลยีซีมอส
6. เขียนแบบลายวงจรรวม และวัดประสิทธิภาพของวงจรรวม
7. สรุปผลการทดลองและเขียนวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ความรู้เกี่ยวกับขั้นตอนการออกแบบวงจรรวมขนาดใหญ่
2. หน่วยประมวลผลสัญญาณดิจิทัลขนาด 16 บิต ที่ประกอบด้วยตัวกรองดิจิทัลแบบเอฟไอาร์ที่สามารถปรับความยาวได้

1.6 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ผ่านการพิจารณาเป็นบทความทางวิชาการ ในหัวข้อ “การออกแบบวงจรรวมหน่วยประมวลผลสัญญาณดิจิทัลที่มีตัวกรองเอฟไอาร์” โดย รวิวร มะหะสิทธิ์ และวันเฉลิม โปรา ในงานประชุมวิชาการ “การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 26” (26th Electrical Engineering Conference : EECON26) ซึ่งจัดโดยภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ ระหว่างวันที่ 6-7 พฤศจิกายน 2546

บทที่ 2

ความรู้พื้นฐาน และปริทัศน์วรรณกรรม

2.1 หน่วยประมวลผลสัญญาณดิจิทัล

หน่วยประมวลผลสัญญาณดิจิทัล [2] เป็นหน่วยประมวลผลที่ถูกออกแบบสำหรับการประยุกต์ใช้ในงานกรรมวิธีสัญญาณดิจิทัล ซึ่งสามารถแบ่งได้เป็นหลายประเภทขึ้นอยู่กับหลักเกณฑ์ที่ใช้แบ่ง หากพิจารณาตามรูปแบบข้อมูลที่ใช้ประมวลผลจะสามารถแบ่งได้ 2 แบบได้แก่ หน่วยประมวลผลสัญญาณดิจิทัลแบบทศนิยมคงที่หรือ Fixed point และแบบ Floating Point

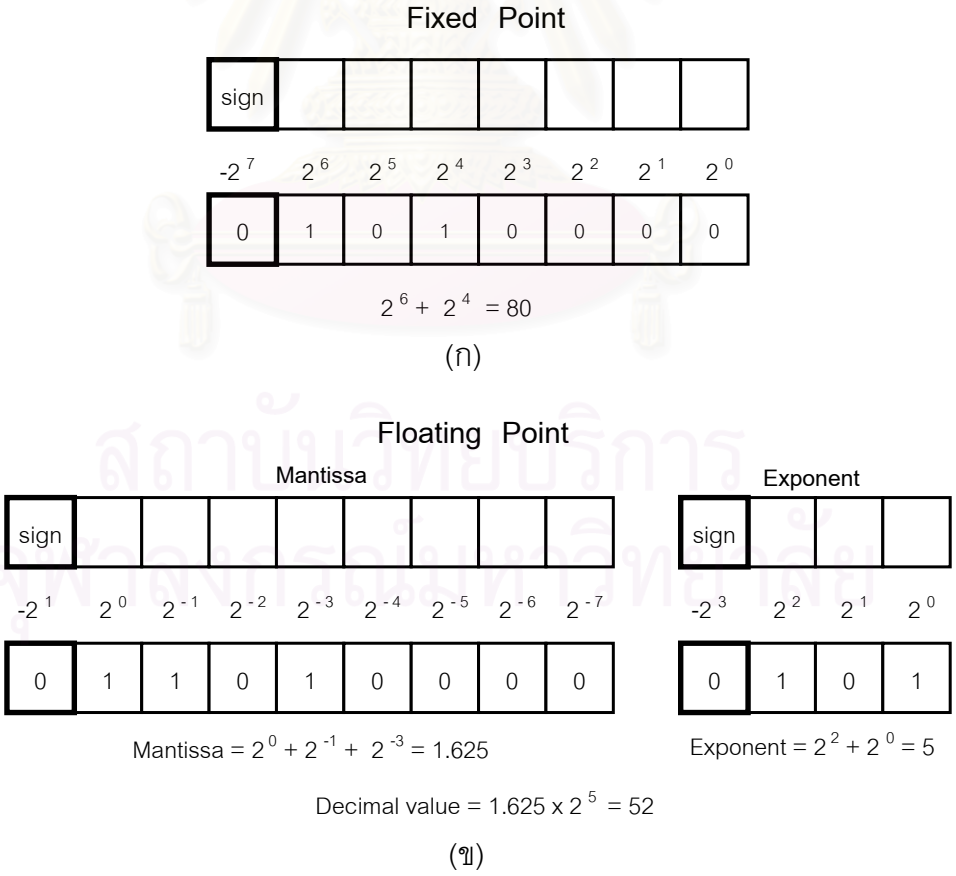
ในกรณีของหน่วยประมวลผลสัญญาณดิจิทัลแบบทศนิยมคงที่ ข้อมูลที่ใช้ในหน่วยประมวลผลจะอยู่ในรูปแบบ Fixed Point (ดูรูปที่ 2.1 (ก)) หน่วยประมวลผลประเภทนี้มีข้อดีตรงที่วิถีข้อมูล (Data Path) ไม่ซับซ้อน แต่ทว่ามีพิสัยพลวัต (Dynamic Range) แคบ ส่วนหน่วยประมวลผลสัญญาณดิจิทัลแบบ Floating Point ซึ่งค่าของข้อมูลที่ใช้คำนวณมาจากข้อมูลย่อยสองส่วนได้แก่ แมนทิสซา (Mantisa) และเลขยกกำลัง (Exponent) (ดูรูปที่ 2.1 (ข)) หน่วยประมวลผลประเภทนี้มีข้อดีตรงที่พิสัยพลวัตกว้าง แต่ทว่ามีวิถีข้อมูลค่อนข้างซับซ้อน โดยทั่วไปหน่วยประมวลผลสัญญาณดิจิทัลแบบ Floating Point ส่วนใหญ่จะสามารถประมวลผลข้อมูลได้ทั้งในรูปแบบ Fixed point และ Floating Point

หน่วยประมวลผลสัญญาณดิจิทัลมีคุณลักษณะที่สำคัญที่ต่างจากหน่วยประมวลผลทั่วไปหลายประการดังนี้

- ประมวลผลการดำเนินการทางคณิตศาสตร์ได้อย่างรวดเร็ว เนื่องจากในงานกรรมวิธีสัญญาณดิจิทัลส่วนใหญ่จะเป็นนำสมการทางคณิตศาสตร์มาดัดแปลงให้อยู่ในรูปที่เหมาะสมสำหรับการเขียนโปรแกรมควบคุมหน่วยประมวลผลต่อไป โดยการดำเนินการทางคณิตศาสตร์ที่ใช้มากในกรรมวิธีสัญญาณดิจิทัลได้แก่ การคูณและสะสม (Multiplication and Accumulation) กล่าวคือหน่วยประมวลผลสัญญาณดิจิทัลจะต้องสามารถประมวลผลการคูณและสะสมได้อย่างรวดเร็ว
- สามารถเข้าถึงข้อมูลในหน่วยความจำได้รวดเร็ว เนื่องจากข้อมูลที่ใช้ในการประมวลผลมักจะมีลักษณะเป็นกลุ่มของข้อมูลซึ่งอยู่ภายในหน่วยความจำ ดังนั้นหน่วยประมวลผลจึงจำเป็นต้องอ่านข้อมูลจากหน่วยความจำเพื่อนำมาคำนวณหา

ผลลัพธ์ และการดำเนินการบางอย่างจำเป็นต้องใช้ตัวถูกดำเนินการหลายชุดเพื่อมาประมวลผลพร้อมๆกัน กล่าวคือหน่วยประมวลผลสัญญาณดิจิทัลจะต้องสามารถเข้าถึงข้อมูลในหน่วยความจำได้หลายๆ ชุดในเวลาเดียวกันอย่างรวดเร็ว

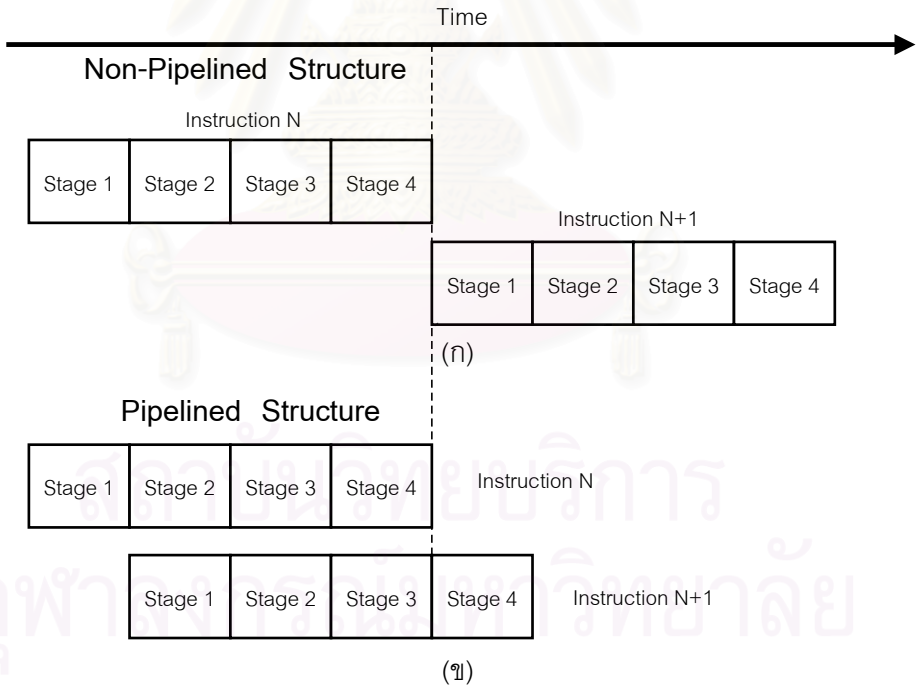
- ประกอบด้วยแอสเดรสซิงโมดพิเศษสำหรับกรรมวิธีสัญญาณดิจิทัล เนื่องจากข้อมูลในการประมวลผลมักมีลักษณะเข้าก่อน-ออกก่อน (First In - First Out : FIFO) หรือมีลักษณะเป็นอะเรย์ ดังนั้นหากหน่วยประมวลผลมีแอสเดรสซิงโมดที่สอดคล้องกับลักษณะของข้อมูลดังกล่าวจะทำให้สามารถประมวลผลอย่างได้มีประสิทธิภาพ
- ประกอบด้วยส่วนควบคุมโปรแกรมแบบพิเศษ เนื่องจากการประมวลผลในกรรมวิธีสัญญาณดิจิทัลมักจะมีการดำเนินการซ้ำๆ กับข้อมูลหลายๆ ชุด หรือทำงานในลักษณะวนลูป ดังนั้นหากหน่วยประมวลผลมีส่วนควบคุมเฉพาะเพื่อประมวลผลในรูปแบบดังกล่าว จะทำให้โปรแกรมที่ใช้ทำงานมีขนาดเล็กลง และทำงานได้รวดเร็วยิ่งขึ้น ทั้งนี้ส่วนควบคุมดังกล่าวอาจจะทำงานโดยผ่านทางคำสั่งเฉพาะเช่น คำสั่งวนลูป เป็นต้น



รูปที่ 2.1 ตัวอย่างข้อมูลในรูปแบบ Fixed Point (ก) และ Floating Point (ข)

2.2 โครงสร้างหน่วยประมวลผลแบบไปป์ไลน์

หน่วยประมวลผลที่มีโครงสร้างแบบไปป์ไลน์ [5, 6, 7] มีข้อดีเหนือกว่าหน่วยประมวลผลที่ไม่ใช้โครงสร้างไปป์ไลน์ เนื่องจากหน่วยประมวลผลที่ไม่ใช้โครงสร้างแบบไปป์ไลน์จะประมวลผลคำสั่งจนเสร็จทีละคำสั่ง แล้วจึงเริ่มประมวลผลคำสั่งต่อไป ส่วนหน่วยประมวลผลแบบไปป์ไลน์จะแบ่งการประมวลผลของแต่ละคำสั่งเป็นหลายๆขั้นตอน (Stage) โดยขณะที่หน่วยประมวลผลกำลังประมวลผลคำสั่งหนึ่งอยู่ ก็สามารถเริ่มประมวลผลคำสั่งถัดไปได้ทันที โดยแต่ละคำสั่งจะมีขั้นตอนการประมวลผลซ้อนทับกัน ดังนั้นหากมีการประมวลผลคำสั่งอย่างต่อเนื่องจะทำให้เวลาประสิทธิผลที่ใช้ประมวลผลคำสั่งของหน่วยประมวลผลแบบไปป์ไลน์มีค่าน้อยลงจากเวลาที่ประมวลผลจริง ตัวอย่างเช่น หน่วยประมวลผลหนึ่งแบ่งการประมวลผลคำสั่งออกเป็น 4 ขั้นตอน โดยพิจารณาว่าการประมวลผลแต่ละขั้นตอนใช้เวลาเท่ากับ 1 วงรอบการทำงาน หากหน่วยประมวลผลนั้นไม่ใช้โครงสร้างแบบไปป์ไลน์ จะต้องใช้เวลา 4 วงรอบการทำงานในการประมวลผลหนึ่งคำสั่ง (ดูรูปที่ 2.2 (ก)) แต่ถ้าหน่วยประมวลผลนั้นมีโครงสร้างแบบไปป์ไลน์จะใช้เวลาประมวลผลคำสั่งอย่างต่อเนื่องเพียงคำสั่งละ 1 วงรอบการทำงาน (ดูรูปที่ 2.2 (ข))



รูปที่ 2.2 การประมวลคำสั่งสำหรับโครงสร้างแบบไม่ใช่ไปป์ไลน์ (ก) และแบบไปป์ไลน์ (ข)

2.2.1 ปัญหาที่เกิดจากการใช้งานโครงสร้างแบบไปป์ไลน์

หน่วยประมวลผลที่มีโครงสร้างแบบไปป์ไลน์มีข้อดีคือ สามารถประมวลผลคำสั่งได้รวดเร็ว เนื่องจากขณะที่กำลังประมวลผลคำสั่งปัจจุบัน จะสามารถเริ่มประมวลผลคำสั่งถัดไปได้ทันที และทำให้ใช้งานวงจรและทรัพยากรภายในของหน่วยประมวลผลได้อย่างมีประสิทธิภาพ เพราะการแบ่งการประมวลผลคำสั่งหนึ่งเป็นหลายขั้นตอน ทำให้เมื่อประมวลผลคำสั่งอย่างต่อเนื่อง แต่ละคำสั่งจะแบ่งกันใช้งานวงจรภายในของหน่วยประมวลผลแต่ละส่วนในเวลาเดียวกัน อย่างไรก็ตาม การใช้งานโครงสร้างแบบไปป์ไลน์อาจเกิดอันตราย (Hazard) ซึ่งต้องพึงระวังในการออกแบบได้แก่

อันตรายเชิงโครงสร้าง (Structural Hazard) เกิดขึ้นเมื่อหน่วยประมวลผลประมวลผลคำสั่งสองคำสั่งซึ่งต้องการใช้ทรัพยากรเดียวกันในเวลาเดียวกัน เช่น คำสั่งหนึ่งพยายามเขียนข้อมูลลงในหน่วยความจำขณะที่อีกคำสั่งกำลังอ่านข้อมูลจากหน่วยความจำ

อันตรายเชิงข้อมูล (Data Hazard) เกิดขึ้นเมื่อคำสั่งหนึ่งต้องการข้อมูลที่กำลังประมวลผลหรือเป็นผลลัพธ์ของอีกคำสั่งก่อนหน้านั้นที่ยังประมวลผลไม่เสร็จ อันตรายเชิงข้อมูลจะทำให้ข้อมูลที่นำไปประมวลผลมีความคาดเคลื่อนจากความเป็นจริง

อันตรายเชิงการควบคุม (Control Hazard) เกิดขึ้นเมื่อส่วนควบคุมของหน่วยประมวลผลประเมินเงื่อนไขของโปรแกรมผิดพลาด เพราะว่าเงื่อนไขของการตัดสินใจยังประมวลผลไม่เสร็จ เช่น คำสั่งกระโดดอ่านข้อมูลตัวทด (Carry) จากรีจิสเตอร์สถานะ ในขณะที่คำสั่งก่อนหน้านั้นกำลังคำนวณผลลัพธ์ของตัวทดอยู่ ส่งผลให้การกระโดดของโปรแกรมผิดพลาด

การใช้งานโครงสร้างแบบไปป์ไลน์อาจทำให้เกิดอันตรายดังกล่าว การป้องกันและแก้ไขอันตรายที่จะเกิดขึ้นอาจอยู่ในรูปแบบของฮาร์ดแวร์หรือซอฟต์แวร์ การแก้ไขในรูปแบบของฮาร์ดแวร์ ผู้ออกแบบหน่วยประมวลผลอาจจะต้องเพิ่มเติมฮาร์ดแวร์ที่มีหน้าที่ตรวจสอบเงื่อนไขการเกิดอันตรายและแก้ไขอันตรายนั้นๆ ส่วนการแก้ไขในรูปแบบของซอฟต์แวร์ โปรแกรมเมอร์จำเป็นต้องเขียนโปรแกรมโดยหลีกเลี่ยงกรณีที่เกิดอันตราย หรืออาจจะใช้คอมไพล์เลอร์เพื่อช่วยเรียบเรียงโปรแกรมไม่ให้เกิดอันตรายขึ้น

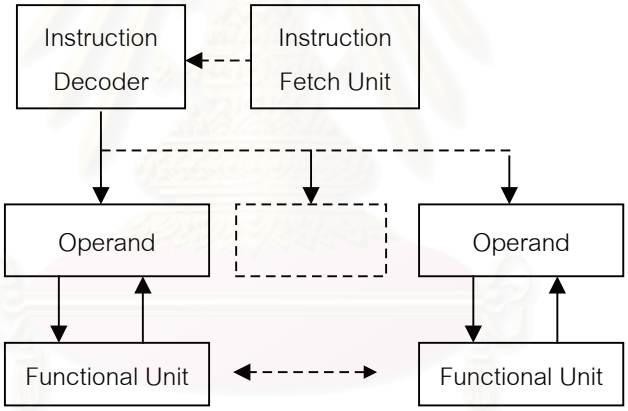
2.3 สถาปัตยกรรมของหน่วยประมวลผลแบบขนาน

การประมวลผลสัญญาณดิจิทัลเป็นการประยุกต์ใช้งานที่ต้องการหน่วยประมวลผลที่สามารถทำงานได้รวดเร็ว การนำโครงสร้างแบบไปป์ไลน์มาใช้ทำให้หน่วยประมวลผลใช้เวลาประมวลผลคำสั่งน้อยลง กล่าวคือเป็นการทำให้หน่วยประมวลผลสามารถทำงานที่ความถี่สัญญาณนาฬิกาสูงขึ้น นอกจากการใช้งานโครงสร้างไปป์ไลน์ที่มีการประมวลผลคำสั่งซ้อนทับกัน

แล้ว การที่หน่วยประมวลผลมีสถาปัตยกรรมแบบขนาน จะทำให้สามารถประมวลผลคำสั่งและข้อมูลได้มากกว่าหน่วยประมวลผลที่มีสถาปัตยกรรมแบบปรกติ เมื่อใช้เวลาประมวลผลเท่ากัน สถาปัตยกรรมของหน่วยประมวลผลแบบขนานมีหลายประเภท ได้แก่

2.3.1 สถาปัตยกรรมแบบ Single Instruction Multiple Data

สถาปัตยกรรมแบบ Single Instruction Multiple Data หรือ SIMD [1] เป็นสถาปัตยกรรมที่สามารถประมวลผลข้อมูลแบบขนาน โดยจะสามารถประมวลผลข้อมูลพร้อมกันหลายชุดด้วยการดำเนินการคำสั่งเพียงคำสั่งเดียว คุณลักษณะของสถาปัตยกรรมแบบ SIMD คือ โครงสร้างหน่วยประมวลผลจะมีหน่วยถอดรหัสคำสั่ง (Instruction Decoding Unit) และหน่วยควบคุม (Control Unit) หนึ่งชุดสำหรับควบคุมการทำงานของหน่วยดำเนินการ (Functional Unit) หลายชุด โดยข้อมูลที่ใช้ประมวลผลต้องมีความเป็นระเบียบสูง เนื่องจากข้อมูลหลายๆชุดจะต้องถูกดำเนินการในลักษณะเดียวกัน จึงจะสามารถประมวลผลขนานกันได้ โครงสร้างอย่างง่ายของสถาปัตยกรรมแบบ SIMD แสดงดังรูปที่ 2.3

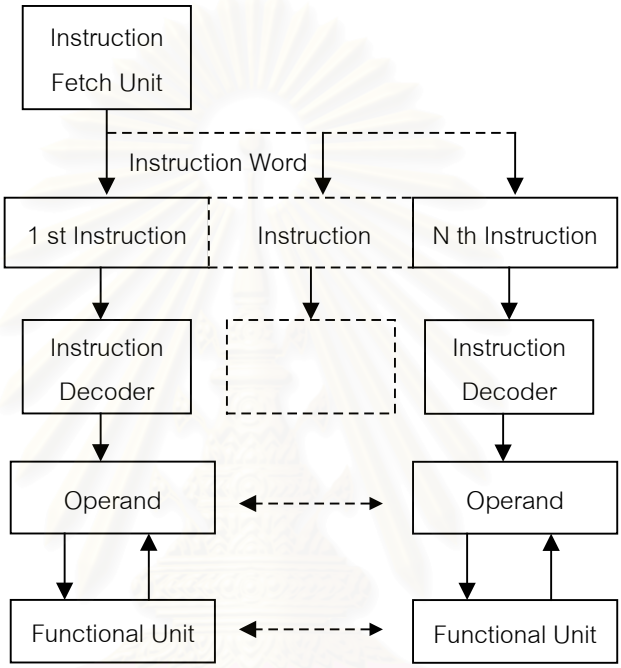


รูปที่ 2.3 โครงสร้างอย่างง่ายของสถาปัตยกรรมแบบ SIMD

2.3.2 สถาปัตยกรรมแบบ Very Long Instruction Word

สถาปัตยกรรมแบบ Very Long Instruction Word หรือ VLIW [1] เป็นสถาปัตยกรรมที่มีกลุ่มคำสั่ง (Instruction Word) ที่มีความยาวมาก โดยกลุ่มคำสั่งจะประกอบด้วยคำสั่งย่อย (Instruction Issue) ตั้งแต่ 2 สองคำสั่งขึ้นไป คุณลักษณะของสถาปัตยกรรมแบบ VLIW คือหน่วยประมวลผลจะอ่านข้อมูลของคำสั่งมาในลักษณะของกลุ่มคำสั่ง ซึ่งจะแบ่งออกเป็นคำสั่งย่อยแล้วส่งไปควบคุมหน่วยดำเนินการแต่ละส่วนเพื่อทำงานขนานกันต่อไป ดังนั้นหน่วยดำเนินการของของสถาปัตยกรรมแบบ VLIW จึงมักจะมีจำนวนเท่ากับจำนวนคำสั่งย่อย แนวคิดของสถาปัตยกรรมแบบ VLIW คือการย้ายความซับซ้อนของการจัดการโปรแกรมที่ถูกเขียนขึ้นอย่าง

เป็นลำดับให้สามารถทำงานขนานกันได้มาไว้ที่การเขียนซอฟต์แวร์ ซึ่งโปรแกรมเมอร์จะต้องเขียนโปรแกรมที่สามารถทำงานแบบขนานเองหรืออาจจะใช้คอมไพเลอร์เพื่อดัดแปลงโปรแกรมให้สามารถทำงานอย่างขนานและจัดรูปแบบคำสั่งให้อยู่ในลักษณะของกลุ่มคำสั่ง ดังนั้นการออกแบบหน่วยประมวลผลที่มีสถาปัตยกรรมแบบ VLIW จึงรวมไปถึงการออกแบบคอมไพเลอร์ที่มีประสิทธิภาพสูงด้วย ส่วนฮาร์ดแวร์สำหรับการจัดการและถอดรหัสคำสั่งของสถาปัตยกรรมแบบ VLIW มักจะไม่ซับซ้อนมากนัก

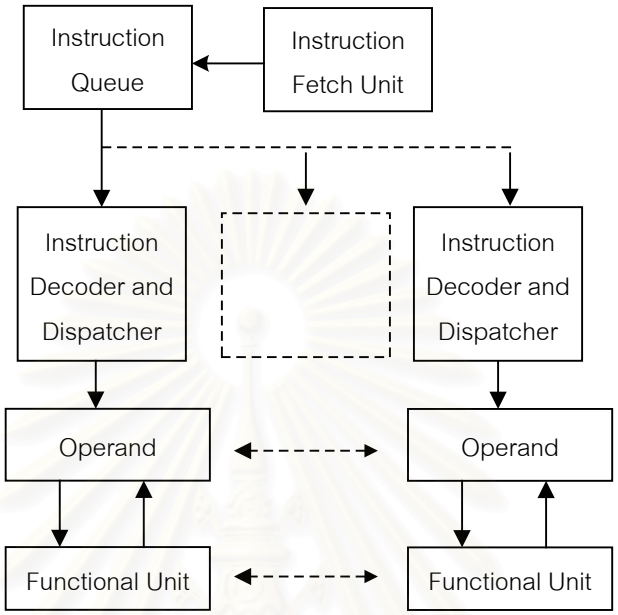


รูปที่ 2.4 โครงสร้างอย่างง่ายของสถาปัตยกรรมแบบ VLIW

2.3.3 <u>สถาปัตยกรรมแบบ Superscalar</u>

โครงสร้างหน่วยประมวลผลแบบ Superscalar [1] ประกอบด้วยฮาร์ดแวร์เฉพาะเพื่อจัดลำดับและกระจายคำสั่งเพื่อไปควบคุมหน่วยดำเนินการหลายๆตัวให้สามารถทำงานขนานกันได้ การเขียนโปรแกรมเพื่อใช้งานกับหน่วยประมวลผลอาจไม่จำเป็นต้องพิจารณาถึงโครงสร้างภายในของหน่วยประมวลผลเหมือนกับกรณีของสถาปัตยกรรมแบบ VLIW จึงส่งผลให้โปรแกรมมีความเข้ากันได้ (Compatability) สูง ขณะที่ประมวลผล หน่วยประมวลผลแบบ Superscalar จะมีการเก็บข้อมูล (Profiling) ของโปรแกรมที่ได้ประมวลผลเพื่อทำการตัดสินใจและควบคุมการทำงานของหน่วยประมวลผลแบบพลวัตอีกด้วย แนวคิดของสถาปัตยกรรมแบบ Superscalar คือการลดภาระของโปรแกรมเมอร์ในการเขียนโปรแกรมเพื่อให้สามารถทำงานแบบขนาน โดยการแก้ปัญหาของการเขียนโปรแกรมในระดับของฮาร์ดแวร์แทน ความยากของการออกแบบหน่วยประมวลผล

แบบ Superscalar อยู่ที่การออกแบบวงจรที่ทำหน้าที่จัดลำดับและกระจายคำสั่งที่มีความซับซ้อนสูง และเนื่องจากการที่มีฮาร์ดแวร์เพิ่มเติม ทำให้หน่วยประมวลผลใช้ทรัพยากรของวงจรและกินกำลังไฟมาก



รูปที่ 2.5 โครงสร้างอย่างง่ายของสถาปัตยกรรมแบบ Superscalar

2.4 สถาปัตยกรรมของชุดคำสั่ง

สถาปัตยกรรมชุดคำสั่ง [16] เป็นปัจจัยที่ส่งผลต่อโครงสร้างของหน่วยประมวลผลโดยตรง และเป็นตัวกำหนดความซับซ้อนของคำสั่ง ซึ่งเป็นตัวกลางในการควบคุมฮาร์ดแวร์ภายในของหน่วยประมวลผล หากพิจารณาโดยอาศัยเกณฑ์ความซับซ้อนของคำสั่งจะสามารถแบ่งสถาปัตยกรรมของชุดคำสั่งได้เป็น 2 ประเภท ได้แก่

2.4.1 หน่วยประมวลผลแบบริสค์

หน่วยประมวลผลแบบริสค์ (RISC: Reduced Instruction Set Computer) มีจุดเด่นตรงที่มีจำนวนคำสั่งน้อย โดยการดำเนินการของแต่ละคำสั่งจะไม่ซับซ้อน ทำให้สามารถประมวลผลแต่ละคำสั่งได้รวดเร็ว และมักจะมีโครงสร้างแบบไปป์ไลน์ เนื่องจากแต่ละคำสั่งของหน่วยประมวลผลไม่ซับซ้อน จึงสามารถแบ่งการประมวลผลของแต่ละคำสั่งเป็นหลายๆ ขั้นตอนจึงทำได้ง่าย และหากจำเป็นต้องประมวลผลการดำเนินการที่ซับซ้อน หน่วยประมวลผลแบบริสค์จะใช้การเขียนซอฟต์แวร์ที่ประกอบด้วยคำสั่งย่อยหลายๆ คำสั่งมาประมวลผลเป็นการดำเนินการที่ซับซ้อน

แนวคิดของหน่วยประมวลผลแบบริสค์คือการย้ายความซับซ้อนของฮาร์ดแวร์ประมวลผลคำสั่งไปยังซอฟต์แวร์หรือโปรแกรมที่ใช้ควบคุมหน่วยประมวลผล ดังนั้นโปรแกรมที่ใช้ในควบคุมการทำงานมักจะมีขนาดใหญ่ ส่งผลให้หน่วยความจำที่ใช้เก็บโปรแกรมมีขนาดใหญ่ตามไปด้วย และเนื่องจากหน่วยประมวลผลแบบริสค์มักจะทำงานได้เร็ว จึงหลีกเลี่ยงการติดต่อกับหน่วยความจำซึ่งจะทำให้เกิดความล่าช้า และมักจะมีรีจิสเตอร์ภายในจำนวนมากเพื่อเก็บข้อมูลที่ใช้ประมวลผล

2.4.2 หน่วยประมวลผลแบบซิสค์

หน่วยประมวลผลแบบซิสค์ (CISC: Complex Instruction Set Computer) เป็นหน่วยประมวลผลที่มีจำนวนคำสั่งค่อนข้างมาก โดยชุดคำสั่งจะประกอบด้วยคำสั่งที่มีความซับซ้อนน้อยซึ่งสามารถประมวลผลได้เร็ว และคำสั่งที่มีความซับซ้อนมากและมีการดำเนินการสอดคล้องกับภาษาระดับสูง (High Level Language : HLL) ซึ่งอาจต้องใช้เวลาประมวลผลมาก ข้อดีของหน่วยประมวลผลแบบซิสค์คือ จะมีวงจรภายในซึ่งทำหน้าที่จัดการประมวลผลคำสั่งที่ซับซ้อนให้เป็นการดำเนินการย่อยเพื่อนำไปควบคุมวงจรภายใน ทำให้การเขียนโปรแกรมเพื่อควบคุมหน่วยประมวลผลจึงสามารถทำได้สะดวกและรวดเร็ว

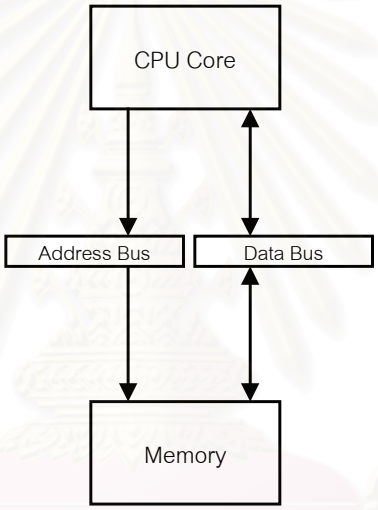
แนวคิดของหน่วยประมวลผลแบบซิสค์คือการย้ายความซับซ้อนของซอฟต์แวร์หรือโปรแกรมที่ใช้ควบคุมหน่วยประมวลผลไปยังฮาร์ดแวร์หน่วยประมวลผล ซึ่งมีหน้าที่แปลคำสั่งที่ซับซ้อนให้กลายเป็นรหัสคำสั่งย่อย (Microcode) หลายๆคำสั่งที่วงจรภายในหน่วยประมวลผลสามารถเข้าใจได้

2.5 สถาปัตยกรรมหน่วยความจำ

สถาปัตยกรรมหน่วยความจำ [3, 4] เป็นปัจจัยที่ส่งผลโดยตรงต่อประสิทธิภาพของหน่วยประมวลผล เนื่องจากการติดต่อกับหน่วยความจำมักจะมีความล่าช้ามาก ดังนั้นการออกแบบสถาปัตยกรรมหน่วยความจำให้เหมาะสมกับการใช้งานหน่วยประมวลผลจึงจำเป็นอย่างยิ่ง หน่วยความจำที่ใช้กับหน่วยประมวลผลอาจแบ่งได้เป็น 2 ประเภท ได้แก่ หน่วยความจำโปรแกรม ซึ่งมีหน้าที่เก็บโปรแกรมที่ใช้ควบคุมหน่วยประมวลผล และหน่วยความจำข้อมูล ซึ่งมีหน้าที่เก็บข้อมูลที่ใช้ประมวลผล หากพิจารณาจากการจัดโครงสร้างของหน่วยความจำทั้งสองประเภท จะสามารถแบ่งสถาปัตยกรรมของหน่วยความจำได้เป็น 2 ประเภทได้แก่ สถาปัตยกรรมแบบวอนนอยแมน (Von Neumann Architecture) และสถาปัตยกรรมแบบฮาร์วาร์ด (Harvard Architecture)

2.5.1 สถาปัตยกรรมแบบวอนนอยแมน

สถาปัตยกรรมแบบวอนนอยแมนเป็นโครงสร้างอย่างง่ายที่ประกอบด้วยหน่วยความจำหลักและบัสเพียงชุดเดียว (ดูรูปที่ 2.6) โดยหน่วยความจำจะมีหน้าที่เก็บโปรแกรมและข้อมูลที่ใช้ประมวลผล โดยทั่วไป หน่วยประมวลผลที่มีโครงสร้างแบบวอนนอยแมนจะสามารถเข้าถึงข้อมูลในหน่วยความจำได้เพียงหนึ่งครั้งในหนึ่งวงรอบคำสั่งเท่านั้น จึงทำให้การประมวลผลที่ใช้ข้อมูลจากหน่วยความจำทำได้ล่าช้า เพราะว่าหน่วยประมวลผลไม่สามารถอ่านโปรแกรมและข้อมูลได้ในเวลาเดียวกัน การแก้ไขปัญหาเนื่องจากความล่าช้าในการติดต่อหน่วยความจำของสถาปัตยกรรมแบบวอนนอยแมน อาจทำได้โดยการเพิ่มเติมหน่วยความจำแคช (Cache) เพื่อลดปริมาณการติดต่อกับหน่วยความจำภายนอก

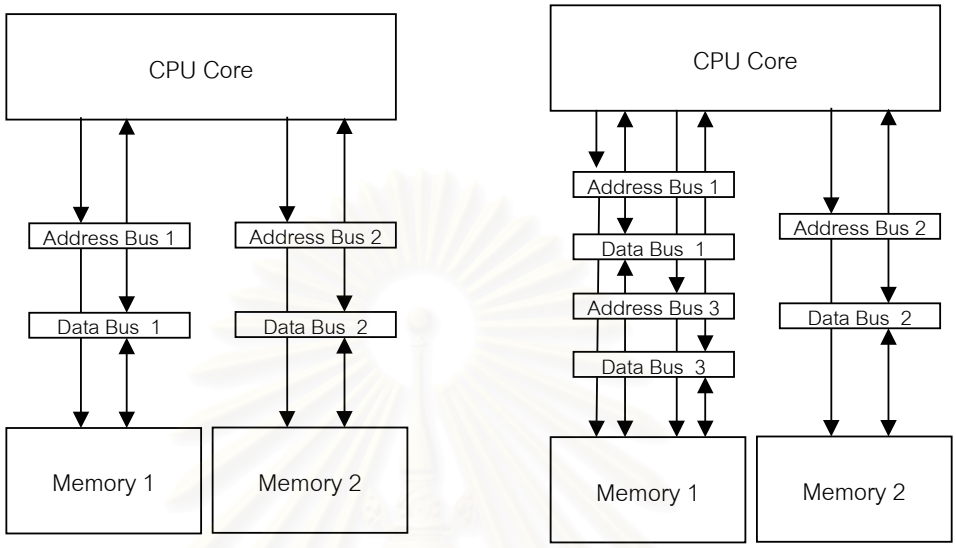


รูปที่ 2.6 สถาปัตยกรรแบบวอนนอยแมน

2.5.2 สถาปัตยกรรมแบบฮาร์วาร์ด

สถาปัตยกรรมแบบฮาร์วาร์ดเป็นโครงสร้างที่ประกอบด้วยหน่วยความจำหลักและบัสสองชุดที่อิสระต่อกัน (ดูรูปที่ 2.7 (ก)) โครงสร้างฮาร์วาร์ดทั่วไปจะใช้หน่วยความจำชุดหนึ่งเก็บโปรแกรมและอีกชุดหนึ่งเก็บข้อมูล แต่ในบางครั้ง หน่วยความจำชุดหนึ่งอาจจะใช้เก็บทั้งโปรแกรมและข้อมูลและอีกชุดหนึ่งจำทำการเก็บข้อมูลเพียงอย่างเดียว หรือหน่วยความจำหลักอาจมีมากกว่าสองชุด โครงสร้างดังกล่าวจะถูกเรียกรวมว่าสถาปัตยกรรมแบบฮาร์วาร์ดดัดแปลง (Modified Harvard Architecture) (ดูรูปที่ 2.7 (ข))

จุดเด่นของสถาปัตยกรรมแบบฮาร์วาร์ดคือ สามารถเข้าถึงหน่วยความจำหลายๆชุดได้ในเวลาเดียวกัน ทำให้สามารถเข้าถึงข้อมูลในหน่วยความจำได้เร็วว่าแบบวอนนอยแมน ด้วยคุณลักษณะดังกล่าวทำให้หน่วยประมวลผลสัญญาณดิจิทัลทั่วไปมีโครงสร้างแบบฮาร์วาร์ด



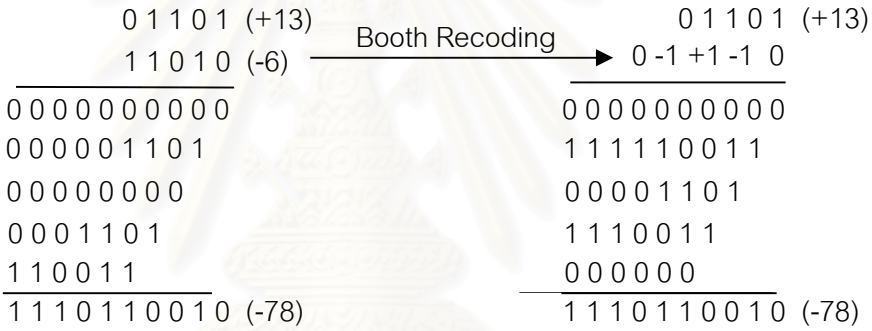
รูปที่ 2.7 สถาปัตยกรรมแบบฮาร์วาร์ด (ก) และฮาร์วาร์ดดัดแปลง (ข)

2.6 อัลกอริทึมของบู๊ธ

อัลกอริทึมของบู๊ธ (Booth Algorithm) [16] เป็นอัลกอริทึมสำหรับการดำเนินการการคูณ โดยจะถูกนำมาใช้สำหรับการเข้ารหัสซ้ำ (Recoding) ข้อมูลตัวคูณ (Multiplier) โดยจะพิจารณาบิตข้อมูลของตัวคูณทีละสองบิต แล้วทำการเข้ารหัสซ้ำตามด้วยเงื่อนไขตามตารางที่ 2.1 โดยผลลัพธ์ของการเข้ารหัสซ้ำจะนำมาดัดแปลงค่าของตัวถูกคูณ (Multiplicand) เพื่อนำไปคำนวณหาผลลัพธ์ของการคูณต่อไป ตัวอย่างการประยุกต์ใช้งานอัลกอริทึมของบู๊ธได้แก่ การคูณเลขฐานสองแบบมีเครื่องหมาย 2 จำนวนคือ 01101 และ 11010 หากนำมาคูณกันด้วยวิธีปรกติจะมีขั้นตอนดังรูปที่ 2.8 (ซ้าย) และเมื่อนำมาคูณโดยใช้อัลกอริทึมของบู๊ธจะมีขั้นตอนดังรูปที่ 2.8 (ขวา)

ตารางที่ 2.1 การเข้ารหัสซ้ำตัวคูณด้วยอัลกอริทึมของบู๊ธ

ตัวคูณ		ค่าที่นำไปคูณกับตัวถูกคูณ
บิตที่ i	บิตที่ i-1	
0	0	0
0	1	+1
1	0	-1
1	1	0

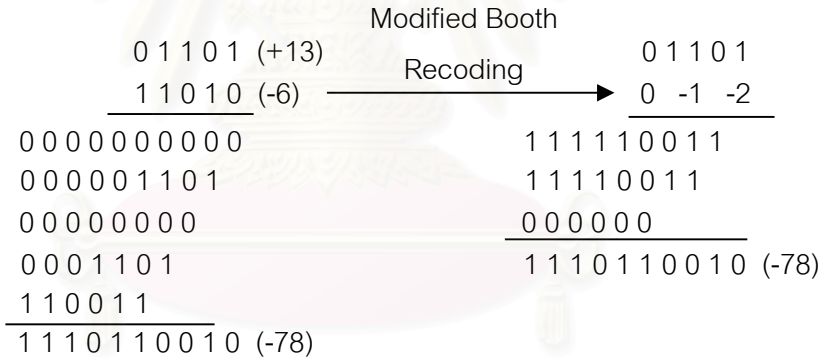


รูปที่ 2.8 ตัวอย่างการประยุกต์ใช้งานอัลกอริทึมของบู๊ธ

การใช้งานอัลกอริทึมของบู๊ธจะมีประสิทธิภาพมากในกรณีที่ค่าของตัวคูณประกอบด้วยบิตที่มีค่า 0 หรือ 1 ติดกันหลายบิต เนื่องจากจะทำให้ค่าที่นำไปดัดแปลงตัวถูกคูณมีค่าเป็น 0 ซึ่งหมายความว่าไม่มีการดำเนินการกับตัวถูกคูณ เนื่องจากตัวคูณที่ถูกเข้ารหัสในบิตนั้น นอกจากนี้อัลกอริทึมของบู๊ธยังสามารถนำมาดัดแปลงโดยการพิจารณาบิตข้อมูลของตัวคูณทีละสามบิตแล้วนำมาเข้ารหัสด้วยเงื่อนไขตามตารางที่ 2.2 ข้อดีของการใช้อัลกอริทึมของบู๊ธแบบดัดแปลงคือจะช่วยลดจำนวนของผลคูณบางส่วนที่ต้องนำมาบวกเป็นผลคูณได้ครึ่งหนึ่ง ตัวอย่างของการประยุกต์ใช้งานอัลกอริทึมของบู๊ธแบบดัดแปลงมีขั้นตอนดังรูปที่ 2.9 (ขวา)

ตารางที่ 2.2 การเข้ารหัสซ้ำตัวคูณด้วยอัลกอริทึมของบู๊ธ

ตัวคูณ			ค่าที่นำไปคูณกับตัวถูกคูณ
บิตที่ i+1	บิตที่ i	บิตที่ i-1	
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0



รูปที่ 2.9 ตัวอย่างการประยุกต์ใช้งานอัลกอริทึมของบู๊ธแบบดัดแปลง

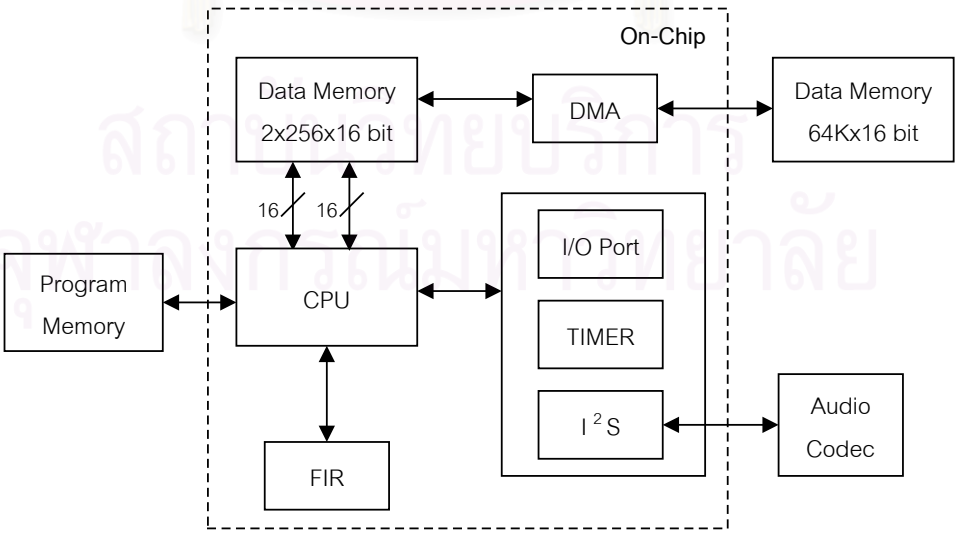
บทที่ 3

สถาปัตยกรรมของ D-CHIP

3.1 โครงสร้างของ D-CHIP

D-CHIP เป็นหน่วยประมวลผลสัญญาณดิจิทัลที่ถูกพัฒนาขึ้นในวิทยานิพนธ์นี้ องค์ประกอบหลักของ D-CHIP ได้แก่ หน่วยประมวลผลกลาง (Central Processing Unit :CPU) แบบทศนิยมคงที่ขนาด 16 บิต ที่มีโครงสร้างแบบไปป์ไลน์ 5 ขั้นตอน ตัวกรองเอฟไออาร์ (FIR) หน่วยความจำข้อมูล (Data Memory) ภายในขนาด 16 บิต 256 ตำแหน่ง จำนวน 2 ชุด และอุปกรณ์บริวารอื่นๆ เช่น พอร์ตอินพุต-เอาต์พุต (I/O Port) ตัวตั้งเวลา (Timer) และวงจรเชื่อมต่ออุปกรณ์มาตราฐาน I^2S (I^2S Interfacing Circuit) ดังแสดงในรูปที่ 3.1

หน่วยประมวลผลกลางมีสถาปัตยกรรมหน่วยความจำแบบฮาร์ดวาร์ดดัดแปลง ซึ่งประกอบด้วยหน่วยความจำโปรแกรม และหน่วยความจำข้อมูลภายใน 2 ชุด ทำให้สามารถประมวลผลการดำเนินการที่ใช้ข้อมูลจากหน่วยความจำข้อมูลทั้งสองชุดได้ภายในหนึ่งวงรอบคำสั่ง นอกจากนี้ยังมีหน่วยความจำภายนอกไว้สำหรับเก็บข้อมูลในกรณีที่หน่วยความจำภายในไม่เพียงพอ การโอนย้ายข้อมูลระหว่างหน่วยความจำภายในและภายนอกจะถูกควบคุมด้วยดีเอ็มเอ (DMA) โดยสามารถโอนย้ายอย่างขนานกับการทำงานของหน่วยประมวลผลกลางได้



รูปที่ 3.1 โครงสร้างของ D-CHIP

3.2 การออกแบบชุดคำสั่ง

3.2.1 คำสั่ง

แนวทางของการออกแบบวงจร คือ หน่วยประมวลผลจะต้องทำงานได้เร็วและมีความซับซ้อนน้อยเพื่อประหยัดทรัพยากรฮาร์ดแวร์ ดังนั้นชุดคำสั่งจึงถูกออกแบบด้วยแนวคิดของสถาปัตยกรรมชุดคำสั่งแบบริสค์ ซึ่งมีจำนวนคำสั่งน้อย และมีรูปแบบของรหัสดำเนินการที่เรียบง่าย รหัสดำเนินการของคำสั่งมีความยาวคงที่ 16 บิต ซึ่งจัดว่าสั้นเมื่อเทียบกับหน่วยประมวลผลสัญญาณดิจิทัลที่จำหน่ายในท้องตลาด [7, 22, 25] ข้อดีของรหัสดำเนินการที่มีความยาวน้อย คือ การออกแบบชุดคำสั่ง และการเขียนโปรแกรมสามารถทำได้ง่าย เนื่องจากมีแอดเดรสซิงโมดน้อย และการเข้ารหัสคำสั่งไม่ซับซ้อน ซึ่งส่งผลให้ฮาร์ดแวร์สำหรับการถอดรหัสคำสั่งไม่ยุ่งยาก เพื่อแลกกับความสามารถในการอ้างตัวถูกดำเนินการและมีการดำเนินการย่อยได้น้อย เป็นผลให้ต้องใช้จำนวนคำสั่งมากกว่าหน่วยประมวลผลที่มีรหัสดำเนินการยาว ชุดคำสั่งที่ออกแบบประกอบด้วยคำสั่งทั้งหมด 58 คำสั่ง ซึ่งสามารถแบ่งออกเป็น 4 กลุ่ม ได้แก่ กลุ่มคณิตศาสตร์และตรรกะ (Arithmetic and Logic Instructions) กลุ่มการคูณและสะสม (Multiplication and Accumulation Instructions) กลุ่มการโอนย้ายข้อมูล (Data Transfer Instructions) และกลุ่มการควบคุมโปรแกรม (Program Control Instructions)

1. กลุ่มคณิตศาสตร์และตรรกะ

คำสั่งในกลุ่มนี้แบ่งได้เป็นสองกลุ่มย่อยได้แก่ คำสั่งทางคณิตศาสตร์พื้นฐาน เช่น การบวก การลบ การเลื่อนข้อมูล และการเปรียบเทียบข้อมูล และคำสั่งทางตรรกะ เช่น AND OR Exclusive-OR และ NOT เป็นต้น (ดูรายละเอียดในตารางที่ 3.2) คำสั่งในกลุ่มถูกออกแบบให้มีความซ้ำซ้อนของการดำเนินการน้อย โดยการลดคำสั่งที่สามารถทดแทนด้วยคำสั่งอื่นได้ [10]

ตัวอย่างของการลดคำสั่งที่สามารถทดแทนด้วยคำสั่งอื่นได้เช่น การเลื่อนข้อมูลไปทางซ้าย (Logical Left Shift) สามารถแทนด้วยคำสั่งการบวก โดยการนำตัวถูกดำเนินการมาบวกด้วยตัวถูกดำเนินการเดียวกัน หรือกล่าวได้ว่าการเลื่อนข้อมูลในรีจิสเตอร์ R0 ไปทางซ้ายหนึ่งครั้ง เท่ากับการประมวลผลคำสั่ง ADD R0, R0

การหมุนข้อมูลไปทางซ้ายผ่านตัวทด (Left Rotate through Carry) สามารถแทนด้วยคำสั่งการบวกแบบมีตัวทด (Add with Carry) โดยการนำตัวถูกดำเนินการมาบวกด้วยตัวถูกดำเนินการเดียวกัน ตัวอย่างเช่นคำสั่ง ADDC R0, R0 คือการหมุนข้อมูลในรีจิสเตอร์ R0 ไปทางซ้ายหนึ่งครั้ง

โดยบิตที่มีความสำคัญน้อยสุด (Least Significant Bit : LSB) จะถูกแทนที่ด้วยค่าในตัวทด ส่วนบิตที่มีความสำคัญมากสุด (Most Significant Bit : MSB) ของ R0 จะเข้าไปแทนที่ค่าในตัวทด

การเคลียร์ข้อมูล (Clear) สามารถแทนด้วยคำสั่ง Exclusive-OR โดยตัวถูกดำเนินการทั้งสองเป็นตัวเดียวกัน ตัวอย่างเช่นคำสั่ง XORL R0, R0 ซึ่งจะทำให้ค่าในรีจิสเตอร์ R0 มีค่าเป็นศูนย์ทั้งหมด

การเปรียบเทียบข้อมูล (Compare) สามารถแทนด้วยการดำเนินการเช่นเดียวกับคำสั่งลบ โดยจะแตกต่างกับคำสั่งลบตรงที่ผลลัพธ์ที่ได้จากการลบไม่ถูกเขียนกลับลงในรีจิสเตอร์

2. <u>กลุ่มการคูณและสะสม</u>

คำสั่งในกลุ่มการคูณและสะสมเป็นคำสั่งสำหรับรองรับการดำเนินการพื้นฐานของการประมวลผลสัญญาณดิจิทัล ซึ่งได้แก่ การคูณ การคูณและการสะสม และการคูณและลด การดำเนินการคำสั่งในกลุ่มนี้สามารถแบ่งย่อยออกเป็นการดำเนินการของข้อมูลชนิดที่มีเครื่องหมาย ข้อมูลชนิดที่ไม่มีเครื่องหมาย หรือการดำเนินการระหว่างข้อมูลชนิดที่ไม่มีเครื่องหมายและชนิดที่มีเครื่องหมาย (ดูรายละเอียดในตารางที่ 3.3) อนึ่งการที่มีคำสั่งที่มีการดำเนินการระหว่างข้อมูลชนิดที่ไม่มีเครื่องหมายกับชนิดที่มีเครื่องหมายได้จะช่วยเพิ่มความเที่ยงตรงของข้อมูลได้ (Extended Precision) เช่น ทำให้การคูณข้อมูล 32 บิตชนิดมีเครื่องหมายเป็นไปได้

3. <u>กลุ่มการโอนย้ายข้อมูล</u>

คำสั่งในกลุ่มการโอนย้ายข้อมูล (ดูรายละเอียดในตารางที่ 3.4) จะเกี่ยวข้องกับการดำเนินการที่ต้องติดต่อกับแหล่งข้อมูลต่างๆ ซึ่งได้แก่ หน่วยความจำข้อมูล หน่วยความจำโปรแกรม และรีจิสเตอร์ใช้งานพิเศษ (Special Function Register) การดำเนินการสามารถแบ่งของคำสั่งในกลุ่มนี้ได้เป็นสองประเภทได้แก่ การอ่านข้อมูลจากแหล่งข้อมูล และการเขียนข้อมูลลงในแหล่งข้อมูล ในการอ่านข้อมูลจะใช้ข้อมูลจากแฟ้มรีจิสเตอร์เป็นตัวถูกดำเนินการ ส่วนการเขียนข้อมูลจะสามารถเลือกใช้ข้อมูลจากตัวถูกดำเนินการได้จากสองแหล่งได้แก่ ข้อมูลจากรีจิสเตอร์ผลลัพธ์ (Result Register :RREG) และข้อมูลจากตัวสะสม (Accumulator :ACC)

4. <u>กลุ่มการควบคุมโปรแกรม</u>

คำสั่งในกลุ่มการควบคุมโปรแกรม (ดูรายละเอียดในตารางที่ 3.5) ได้แก่ คำสั่งควบคุมการกระโดดของโปรแกรม และคำสั่งควบคุมโปรแกรมให้ทำงานตามจำนวนรอบที่กำหนด การดำเนินการของคำสั่งในกลุ่มนี้จะเกี่ยวข้องกับการเปลี่ยนแปลงค่าของตำแหน่งของหน่วยความจำ ซึ่งได้แก่ ค่าของตัวนับโปรแกรม (Program Counter :PC)

คำสั่งควบคุมการกระโดดของโปรแกรมสามารถแบ่งออกเป็นสองรูปแบบ ได้แก่การกระโดดของโปรแกรมแบบไม่มีเงื่อนไข และการกระโดดของโปรแกรมแบบมีเงื่อนไข ซึ่งเงื่อนไขของการกระโดดจะขึ้นอยู่กับแฟล็กสถานะว่าสอดคล้องกับเงื่อนไขของการกระโดดหรือไม่ การเปลี่ยนแปลงค่าของตัวนับโปรแกรมเนื่องจากคำสั่งการกระโดดจะเป็นแบบสัมพัทธ์ กล่าวคือการเปลี่ยนแปลงค่าของตัวนับโปรแกรมจะอ้างอิงจากตำแหน่งปัจจุบันแล้วปรับปรุงด้วยค่าการกระจัดของโปรแกรม

คำสั่งควบคุมโปรแกรมให้ทำงานตามจำนวนรอบที่กำหนดใช้ควบคุมการทำงานของโปรแกรมประเภทลูป FOR ซึ่งโปรแกรมจะทำงานซ้ำตามจำนวนรอบที่กำหนด โดยปราศจากโปรแกรมหรือตัวแปรส่วนที่ทำหน้าที่นับรอบการทำงานว่าครบตามจำนวนที่กำหนดแล้วหรือยัง

ชุดคำสั่งของหน่วยประมวลผลกลางถูกแสดงดังตารางที่ 3.2 ถึง 3.5 โดยแบ่งออกตามกลุ่มคำสั่งทั้งสี่กลุ่ม ส่วนสัญลักษณ์ต่างๆ ที่ใช้ในตารางคำสั่งและรหัสดำเนินการถูกอธิบายในตารางที่ 3.1

ตารางที่ 3.1 สัญลักษณ์ที่ใช้ในตารางคำสั่งและรหัสดำเนินการ

สัญลักษณ์	คำอธิบาย	หมายเหตุ
@dir	direct address	8 bit
#imm	immidiate value	8 bit
rel	relative program address	signed 9 bit
I	interrupt flag	
OPa	operand A	OPa= {R, *Xn}
OPb	operand B	OPb = {R, *Xn, *Yn}
OPb	operand C	OPb = {RREG, MCON}
Rd	destination register	
Rs	source register	
*Xn	data from pointer Xn	
*Yn	data from pointer Yn	
Xn	pointer Xn	
Yn	pointer Yn	
io_addr	I/O Register Address	4 bit
bit	bit number to set/clear	4 Bit
stsrc	data source for store	stsrc={RREG, ACCH, ACCL}
PC	program counter	
#loop	loop times	up to 512
loop_end	loop end address	
cond	flag clear or set condition	

ตารางที่ 3.2 ชุดคำสั่งในกลุ่มคณิตศาสตร์และตรรกะ

Mnenonic	Operand	Description	Operation	Flag	Cycle
Arithmetic and Logic Instructions					
shr	Rd, Rs	Logical Right Shift	C<-Rs[0], Rd <-' 0'&Rs[15:1]	C,Z,S	1
asr	Rd, Rs	Arithmatic Right Shift	C<-Rs[0], Rd <- Rs[15]&Rs[15:1]	C,Z,S	1
rrc	Rd, Rs	Right Rotate through Carry	C<-Rs[0], Rd <- C&Rs[15:1]	C,Z,S	1
setf	Rd, bit	Bit Set	Rd(bit) <- '1'	Z,S	1
clrf	Rd, bit	Bit Clear	Rd(bit) <-' 0'	Z,S	1
com	Rd, Rs	Complement	Rd <-NOT Rs	Z,S	1
andl	OPa, OPb	Logical AND	if OPa=Rd then (OPa <- OPa AND OPb) else (R0<-OPa AND OPb)	Z,S	1
orl	OPa, OPb	Logical OR	if OPa=Rd then (OPa <- OPa OR OPb) else (R0<-OPa OR OPb)	Z,S	1
xorl	OPa, OPb	Logical Exclusive OR	if OPa=Rd then (OPa <- OPa XOR OPb) else (R0<-OPa XOR OPb)	Z,S	1
add	OPa, OPb	Add	if OPa=Rd then (OPa <-OPa+Opb) else (R0<-OPa+OPb)	C,Z,S	1
addc	OPa, OPb	Add with Carry	if OPa=Rd then (OPa <- OPa+OPb+C) else (R0<-OPa+OPb+C)	C,Z,S	1
sub	OPa, OPb	Subtract	if OPa=Rd then OPa <- OPa-OPb else R0<-OPa-OPb	C,Z,S	1
subb	OPa, OPb	Subtract with Carry	if OPa=Rd then (OPa <- OPa-OPb-C) else (R0<-OPa-OPb-C)	C,Z,S	1
cp	OPa, OPb	Compare	Temp <- OPa-OPb	N,Z,S	1
swap	Rd, OPb	Swap Byte	if OPa=Rd then (OPa <- OPa+OPb) else (R0<-OPa+OPb)	Z,S	1

ตารางที่ 3.3 ชุดคำสั่งในกลุ่มการคูณและสะสม

Mnenonic	Operand	Description	Operation	Flag	Cycle
Multiplication and Accumulation Instructions					
mulu	OPa, OPb	Unsigned Multiply	ACC <- unsigned OPa * unsigned OPb	-	1
mulus	OPa, OPb	Unsigned to Signed Multiply	ACC <- unsigned OPa * signed OPb	-	1
muls	OPa, OPb	Signed Multiply	ACC <- signed OPa * signed OPb	-	1
madd	*Xn, *Yn, Opc	Multiply and Add	ACC <- signed MEM(Xn) + (signed MEM(Y)* signed OPc)	-	1
macu	OPa, OPb	Unsigned Multiply&Accumulate	ACC <- ACC+(unsigned OPa * unsigned OPb)	-	1
macus	OPa, OPb	Unsigned to Signed Multiply&Accumulate	ACC <- ACC+(unsigned OPa * signed OPb)	-	1
macs	OPa, OPb	Signed Multiply&Accumulate	ACC <- ACC+(signed OPa * signed OPb)	-	1
msbu	OPa, OPb	Unsigned Multiply&Subtract	ACC <- ACC-(unsigned OPa * unsigned OPb)	-	1
msbus	OPa, OPb	Unsigned to Signed Multiply&Subtract	ACC <- ACC-(unsigned OPa * signed OPb)	-	1
msbs	OPa, OPb	Signed Multiply&Subtract	ACC <- ACC-(signed OPa * signed OPb)	-	1

ตารางที่ 3.4 ชุดคำสั่งในกลุ่มการโอนย้ายข้อมูล

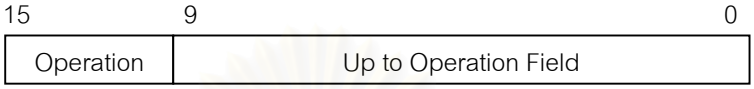
Mnenonic	Operand	Description	Operation	Flag	Cycle
Data Transfer Instructions					
ldkh	Rd, #imm	Load Constant to High Byte	Rd[15:8] <- imm	-	1
ldkl	Rd, #imm	Load Constant to Low Byte	Rd[7:0] <- imm	-	1
lddx	Rd, @dir	Direct Load from Memory X	Rd <-MEMX(dir)	-	1
lddy	Rd, @dir	Direct Load from Memory Y	Rd <-MEMY(dir)	-	1
ldix	Rd,*Xn	Indirect Load from Memory X	Rd <-MEMY(Xn)	-	1
ldiy	Rd,*Yn	Indirect Load from Memory Y	Rd <-MEMY(Yn)	-	1
lpm	Rd	Load from Program Memory	Rd <-MEMP(Prog_addr)	-	3
ldio	Rd, io_addr	Load from IO Register	Rd <- IO(io_addr)	-	1
stix	stsrc,*Xn	Indirect Store in Memory X	MEMX(Xn) <- stsrc	-	1
stiy	stsrc,*Yn	Indirect Store in Memory Y	MEMY(Yn) <- stsrc	-	1
stdx	stsrc, @dir	Direct Store in Memory X	MEMX(dir) <- stsrc	-	1
stdy	stsrc, @dir	Direct Store in Memory Y	MEMY(dir) <- stsrc	-	1
stio	RREG, io_addr	Store in IO register	IO(io_addr) <- stsrc	-	1
starp	stsrc, Xn, Yn	Store in ARP	arpX(n) <- stsrc[15:8], arpY(n)<- stsrc[7:0]	-	1
stfc	stsrc	Store in Coefficient Memory	FIRMEMC <- stsrc	-	1
stfs	stsrc	Store in Sample Memory	FIRMEMS <- stsrc	-	1
ldarp	Rd, Xn, Yn	Load ARP to Register	Rd[15:8] <- arpX(n), Rd[7:0] <- arpY(n)	-	1
mov	Rd , Opb	Copy to Register	Rd <- OPb	Z,S	1
mvbx	in/out	Move Block of Data with X	MEMZ(DMA_page&DMA_addr) <-> MEMX(DMA_addr)	-	-
mvby	in/out	Move Block of Data with Y	MEMZ(DMA_page&DMA_addr) <-> MEMY(DMA_addr)	-	-

ตารางที่ 3.5 ชุดคำสั่งในกลุ่มการควบคุมโปรแกรม

Mnenonic	Operand	Description	Operation	Flag	Cycle
Program Control Instructions					
srpt	# loop	Start Loop		-	2
erpt	loop_end	End Loop		-	1
reti		Return from Interrupt	I <-'1', PC <- PC_Backup	I	2
rjmp	rel	Relative Jump	PC <- PC+rel	-	2
brc	cond, rel	Jump if Carry set or clear	if C=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
brn	cond, rel	Jump if Negative set or clear	if N=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
brz	cond, rel	Jump if Zero set or clear	if Z=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
brs	cond, rel	Jump if Sign set or clear	if S=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
brnx	cond, rel	Jump if Negative_X set or clear	if NX=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
brzx	cond, rel	Jump if Zero_X set or clear	if ZX=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
brny	cond, rel	Jump if Negative_Y set or clear	if NY=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
brzy	cond, rel	Jump if Zero_Y set or clear	if ZY=cond then (PC <- PC+rel) else (PC <-PC+1)	-	2
firstart		Start Filter		-	-
nop		No Operation			

3.2.2 <u>รหัสดำเนินการ</u>

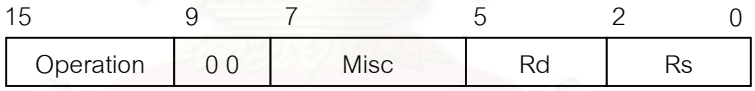
รูปแบบรหัสดำเนินการประกอบด้วยฟิลด์ย่อยตั้งแต่สามถึงห้าฟิลด์ โดยขึ้นอยู่กับฟิลด์การดำเนินการ (Operation Field) ซึ่งมีขนาดคงที่ 6 บิต จากบิตที่ 10 ถึง 15 (ดูรูปที่ 3.2) และรูปแบบของแอสเดรสซิงโมด ซึ่งมีทั้งหมด 5 รูปแบบ ได้แก่



รูปที่ 3.2 รูปแบบรหัสดำเนินการ

1. <u>แอดเดรสซิงโมดแบบรีจิสเตอร์</u>

แอดเดรสซิงโมดแบบรีจิสเตอร์เป็นการเรียกใช้รีจิสเตอร์เป็นตัวถูกดำเนินการ โดยที่ฟิลด์แอดเดรสซิงโมดในบิตที่ 8 และ 9 มีค่าเป็น 00 (ดูรูปที่ 3.3) ส่วนฟิลด์เลือกตัวถูกดำเนินการจะแบ่งออกเป็นสองฟิลด์ ได้แก่ ฟิลด์ของรีจิสเตอร์ต้นทาง (Source Register :Rs) ซึ่งกำหนดโดยข้อมูลในบิตที่ 0 ถึง 2 และฟิลด์ของรีจิสเตอร์ปลายทาง (Destination Register :Rd) ซึ่งกำหนดโดยข้อมูลในบิตที่ 3 ถึง 5



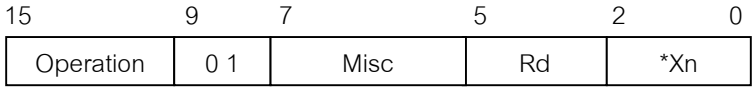
รูปที่ 3.3 รูปแบบรหัสดำเนินการที่มีแอดเดรสซิงโมดแบบรีจิสเตอร์

2. <u>แอดเดรสซิงโมดแบบโดยอ้อม</u>

เนื่องจากหน่วยประมวลผลประกอบด้วยหน่วยความจำข้อมูลภายใน 2 ชุด ได้แก่ หน่วยความจำ X และหน่วยความจำ Y ดังนั้นการอ้างแอดเดรสซิงโมดแบบโดยอ้อมจึงแบ่งได้เป็นหลายประเภทขึ้นอยู่กับการจัดหมู่ของการเลือกใช้รีจิเตอร์และหน่วยความจำ แอดเดรสซิงโมดแบบโดยอ้อมแบ่งได้ 3 ชนิด คือ

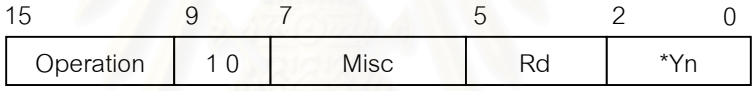
แอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 1 เป็นการเลือกใช้รีจิสเตอร์และข้อมูลจากหน่วยความจำ X เป็นตัวถูกดำเนินการ โดยที่ฟิลด์แอดเดรสซิงโมดในบิตที่ 8 และ 9 มีค่าเป็น 01 (ดูรูปที่ 3.4) ส่วนการเลือกตัวถูกดำเนินการจะแบ่งออกเป็นสองฟิลด์ ได้แก่ ฟิลด์ของรีจิสเตอร์ปลายทาง (Destination Register :Rd) ซึ่งกำหนดโดยข้อมูลในบิตที่ 3 ถึง 5 จะถูกนำมาเลือกรีจิสเตอร์ใช้งานทั่วไปที่ใช้ดำเนินการและเก็บผลลัพธ์ของการดำเนินการ และฟิลด์ตัวชี้ข้อมูลหน่วยความจำ X

(Memory X Data Pointer :*Xn) ซึ่งกำหนดโดยข้อมูลในบิตที่ 0 ถึง 2 จะถูกนำมาเลือกตัวชี้ข้อมูลของหน่วยความจำ X และการดำเนินการของตำแหน่งของตัวชี้นั้น



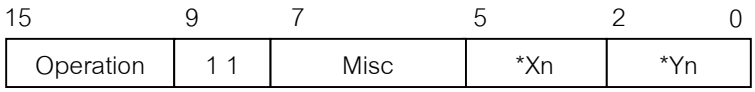
รูปที่ 3.4 รูปแบบรหัสดำเนินการที่มีแอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 1

แอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 2 เป็นการเลือกใช้รีจิสเตอร์และข้อมูลจากหน่วยความจำ Y เป็นตัวถูกดำเนินการ โดยที่ฟิลด์แอดเดรสซิงโมดในบิตที่ 8 และ 9 มีค่าเป็น 10 (ดูรูปที่ 3.5) ส่วนการเลือกตัวถูกดำเนินการจะแบ่งออกเป็นสองฟิลด์ ได้แก่ ฟิลด์ของรีจิสเตอร์ปลายทาง (Destination Register :Rd) ซึ่งกำหนดโดยข้อมูลในบิตที่ 3 ถึง 5 จะถูกนำมาเลือกรีจิสเตอร์ใช้งานทั่วไปที่ใช้ดำเนินการและเก็บผลลัพธ์ของการดำเนินการ และฟิลด์ตัวชี้ข้อมูลหน่วยความจำ Y (Memory Y Data Pointer :*Yn) ซึ่งกำหนดโดยข้อมูลในบิตที่ 0 ถึง 2 จะถูกนำมาเลือกตัวชี้ข้อมูลของหน่วยความจำ Y และการดำเนินการของตำแหน่งของตัวชี้นั้น



รูปที่ 3.5 รูปแบบรหัสดำเนินการที่มีแอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 2

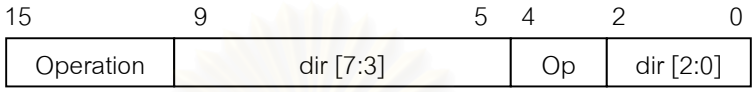
แอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 3 เป็นการเลือกใช้ข้อมูลจากหน่วยความจำ X และ Y เป็นตัวถูกดำเนินการ โดยที่ฟิลด์แอดเดรสซิงโมดในบิตที่ 8 และ 9 มีค่าเป็น 11 (ดูรูปที่ 3.6) ส่วนการเลือกตัวถูกดำเนินการจะแบ่งออกเป็นสองฟิลด์ ได้แก่ ฟิลด์ตัวชี้ข้อมูลหน่วยความจำ X (Memory X Data Pointer :*Xn) ซึ่งกำหนดโดยข้อมูลในบิตที่ 3 ถึง 5 จะถูกนำมาเลือกตัวชี้ข้อมูลของหน่วยความจำ X และการดำเนินการของตำแหน่งของตัวชี้นั้น และฟิลด์ตัวชี้ข้อมูลหน่วยความจำ Y (Memory Y Data Pointer :*Yn) ซึ่งกำหนดโดยข้อมูลในบิตที่ 0 ถึง 2 จะถูกนำมาเลือกตัวชี้ข้อมูลของหน่วยความจำ Y และการดำเนินการของตำแหน่งของตัวชี้นั้น ผลลัพธ์ของการดำเนินการที่มีแอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 3 จะถูกเก็บลงในรีจิสเตอร์ R0



รูปที่ 3.6 รูปแบบรหัสดำเนินการที่มีแอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 3

3. <u>แอดเดรสซิงโมดแบบโดยตรง</u>

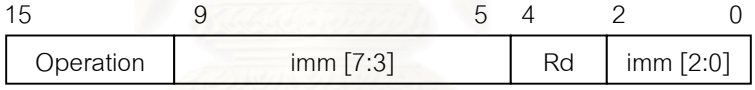
แอดเดรสซิงโมดแบบโดยตรงเป็นการอ้างข้อมูลในหน่วยความจำโดยใช้ตำแหน่งโดยตรงจากรหัสคำสั่ง ฟิลด์ที่ใช้อ้างตำแหน่งแบบโดยตรงมีขนาด 8 บิต โดยอยู่ในบิตที่ 0 ถึง 2 และบิตที่ 5 ถึง 9 (ดูรูปที่ 3.7) ส่วนฟิลด์ที่ใช้อ้างตัวถูกดำเนินการจะในบิตที่ 3 และ 4 การพิจารณาว่าแอดเดรสซิงโมดเป็นแบบโดยตรงหรือไม่ จะพิจารณาจากฟิลด์การดำเนินการ



รูปที่ 3.7 รูปแบบรหัสดำเนินการที่มีแอดเดรสซิงโมดแบบโดยตรง

4. <u>แอดเดรสซิงโมดแบบใช้ทันที</u>

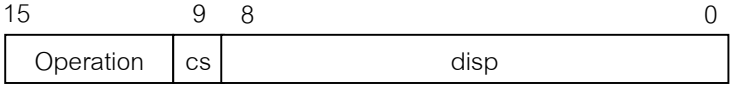
แอดเดรสซิงโมดแบบใช้ทันทีเป็นการอ้างค่าคงที่โดยตรงจากรหัสคำสั่ง ฟิลด์ใช้อ้างค่าคงที่แบบใช้ทันทีมีขนาด 8 บิต โดยอยู่ในบิตที่ 0 ถึง 2 และบิตที่ 5 ถึง 9 (ดูรูปที่ 3.8) ส่วนฟิลด์ที่ใช้อ้างตัวถูกดำเนินการจะในบิตที่ 3 และ 4 เช่นเดียวกับกรณีของแอดเดรสซิงโมดแบบโดยตรง การพิจารณาว่าแอดเดรสซิงโมดเป็นแบบใช้ทันทีหรือไม่ จะพิจารณาจากฟิลด์การดำเนินการ



รูปที่ 3.8 รูปแบบรหัสดำเนินการที่มีแอดเดรสซิงโมดแบบใช้ทันที

5. <u>แอดเดรสซิงโมดโปรแกรมแบบสัมพัทธ์</u>

แอดเดรสซิงโมดโปรแกรมแบบสัมพัทธ์ใช้เฉพาะการดำเนินการของคำสั่งประเภทการกระโดดของโปรแกรม สำหรับคำสั่งกระโดดแบบมีเงื่อนไขจะใช้ฟิลด์เงื่อนไข (cs) ซึ่งอยู่ในบิตที่ 9 ในการกำหนดว่ากระโดดจะเกิดขึ้นเมื่อเงื่อนไขของแฟล็กสถานะ (Status Flag) มีค่าเป็นศูนย์หรือหนึ่ง ส่วนฟิลด์การกระจัดจำนวน 9 บิต ที่อยู่ในบิตที่ 0 ถึง 8 (ดูรูปที่ 3.9) เป็นจำนวนแบบมีเครื่องหมายสำหรับใช้ในการปรับปรุงค่าของตัวนับโปรแกรม



รูปที่ 3.9 รูปแบบรหัสดำเนินการที่มีแอดเดรสซิงโมดแบบสัมพัทธ์

3.3 โครงสร้างไปป์ไลน์ของ D-CHIP

หน่วยประมวลผลสัญญาณดิจิทัลที่มีโครงสร้างแบบไปป์ไลน์ D-CHIP จะแบ่งการทำงานของคำสั่งเป็นหลายๆ ขั้นตอน เพื่อกระจายความล่าช้าของวงจรภายในไปในแต่ละขั้นตอน โดยการทำงานแต่ละขั้นตอนของ D-CHIP จะสามารถประมวลผลเสร็จภายในหนึ่งรอบสัญญาณนาฬิกา

หน่วยประมวลผลกลางจะแบ่งการทำงานแต่ละคำสั่งส่วนใหญ่ ยกเว้นคำสั่งในกลุ่มการคูณและสะสม ออกเป็น 5 ขั้นตอน (ดูรูปที่ 3.10) ได้แก่ ขั้นตอนเฟ็ทช์คำสั่ง (Instruction Fetch: F0) ขั้นตอนถอดรหัสคำสั่ง (Instruction Decode: D0) ขั้นตอนเข้าถึงหน่วยความจำ (Memory Access: M0) ขั้นตอนอ่านค่าตัวถูกดำเนินการ (Operand Fetch: D1) และขั้นตอนดำเนินการ (Execute: E1)

Instruction Fetch (F0)	Instruction Decode (D0)	Memory Access (M0)	Operand Fetch (D1)	Execute (E1)

รูปที่ 3.10 ขั้นตอนของการทำงานของไปป์ไลน์ของคำสั่งทั่วไป

ในขั้นตอนเฟ็ทช์คำสั่ง (ขั้นตอน F0) คำสั่งจะถูกเฟ็ทช์จากหน่วยความจำโปรแกรมเข้าสู่หน่วยประมวลผลกลาง ในรอบสัญญาณนาฬิกาที่สอง (ขั้นตอน D0) คำสั่งที่ถูกเฟ็ทช์ในขั้นตอนแรกจะถูกนำมาถอดรหัสเพื่อนำไปควบคุมการทำงานในขั้นตอนที่เหลือ ในรอบสัญญาณนาฬิกาที่สาม (ขั้นตอน M0) หากคำสั่งต้องการอ่านหรือเขียนข้อมูลในหน่วยความจำข้อมูล หน่วยประมวลผลกลางจะชี้ตำแหน่งของข้อมูลที่ต้องการ เพื่อเข้าถึงข้อมูลในหน่วยความจำ ข้อมูลที่ได้จากหน่วยความจำในขั้นตอนนี้จะถูกนำไปใช้ในขั้นตอนอ่านค่าตัวถูกดำเนินการ ในรอบสัญญาณนาฬิกาที่สี่ (ขั้นตอน D1) จะเป็นการเลือกชนิดของตัวถูกดำเนินการว่ามาจาก รีจิสเตอร์หรือหน่วยความจำ เพื่อนำไปดำเนินการในขั้นตอนสุดท้ายคือขั้นตอนดำเนินการ (ขั้นตอน E1) ในรอบสัญญาณนาฬิกาที่ห้า

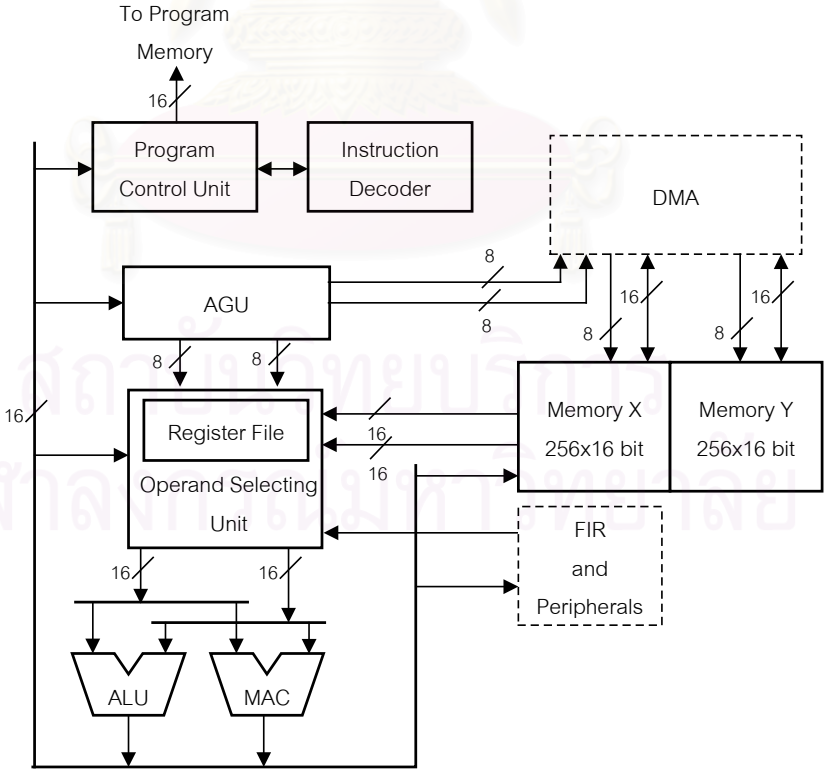
คำสั่งทั่วไปเกือบทุกคำสั่งสามารถทำงานเสร็จใน 5 ขั้นตอนดังที่กล่าวข้างต้น ยกเว้นคำสั่งในกลุ่มการคูณและสะสม ที่มีขั้นตอนคำสั่งเพิ่มเป็น 7 ขั้นตอน (ดูรูปที่ 3.11) โดยจะมีขั้นตอนดำเนินการย่อยเพิ่มขึ้นเป็น 3 ขั้นตอน ได้แก่ ขั้นตอนการคูณที่ 1 และ 2 (Multiply1: E1, Multiply2 : E2) และขั้นตอนการสะสม (Accumulate: E3)

Instruction Fetch (F0)	Instruction Decode (D0)	Memory Access (M0)	Operand Fetch (D1)	Multiply 1 (E1)	Multiply 2 (E2)	Accumulate (E3)

รูปที่ 3.11 ขั้นตอนของการทำงานของไปป์ไลน์ของคำสั่งในกลุ่มการคูณและสะสม

3.4 สถาปัตยกรรมของหน่วยประมวลผลกลาง

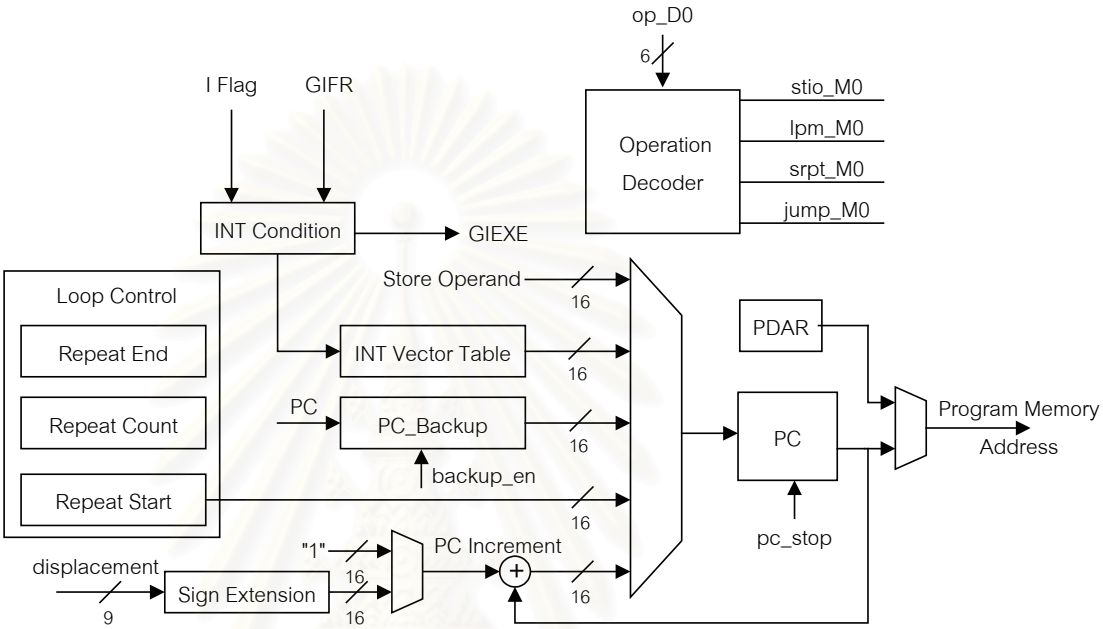
หน่วยประมวลผลกลางถูกแบ่งออกเป็นวงจรย่อยหลายๆ ส่วน การแบ่งวงจรจะพิจารณาจากขั้นตอนการทำงานไปป์ไลน์ โดยวงจรย่อยแต่ละส่วนจะรับผิดชอบงานแต่ละขั้นตอนของไปป์ไลน์ หน่วยควบคุมของหน่วยประมวลผลจะไม่กระจุกตัวเป็นวงจรย่อยเพียงวงจรเดียว แล้วควบคุมวงจรย่อยอื่นๆ แต่ทว่าจะแบ่งหน่วยควบคุมออกเป็นส่วนย่อยๆ ควบคุมวงจรย่อยอยู่ภายในวงจรย่อยนั้น [8] หน่วยประมวลผลกลางสามารถแบ่งได้เป็นวงจรย่อยๆ ได้ทั้งหมด 6 ส่วน ได้แก่ หน่วยควบคุมโปรแกรม (Program Control Unit) ตัวถอดรหัสคำสั่ง (Instruction Decoder) หน่วยเลือกตัวถูกดำเนินการและแฟ้มรีจิสเตอร์ (Operand Selector and Register File) หน่วยคำนวนและตรรกะ (Arithmetic and Logic Unit) หน่วยคูณและสะสม (Multiply and Accumulate Unit :MAC) หน่วยกำเนิดตำแหน่งข้อมูล (Address Generation Unit :AGU) สถาปัตยกรรมของหน่วยประมวลผลกลางแสดงดังในรูปที่ 3.12



รูปที่ 3.12 สถาปัตยกรรมของหน่วยประมวลผลกลาง

3.4.1 หน่วยควบคุมโปรแกรม

หน่วยควบคุมโปรแกรมมีหน้าที่ปรับปรุงค่าของตัวนับโปรแกรม และสร้างตำแหน่งข้อมูลของหน่วยความจำโปรแกรม นอกจากนี้หน่วยควบคุมโปรแกรมยังมีหน้าที่จัดการคำสั่งประเภทวนลูป ซึ่งจะทำให้โปรแกรมทำงานตามจำนวนรอบที่กำหนด



รูปที่ 3.13 โครงสร้างของหน่วยควบคุมโปรแกรม

โครงสร้างของหน่วยควบคุมโปรแกรมประกอบด้วยส่วนประกอบหลัก 4 ส่วน (ดูรูปที่ 3.13) ได้แก่ ตัวนับโปรแกรมขนาด 16 บิต (Program Counter :PC) ส่วนปรับปรุงตัวนับโปรแกรม ส่วนควบคุมการวนลูป (Loop Control) และส่วนถอดรหัสการดำเนินการ (Operation Decoder)

ตัวนับโปรแกรมเป็นรีจิสเตอร์ขนาด 16 บิตที่ทำหน้าที่เก็บค่าตำแหน่งของหน่วยความจำโปรแกรม ค่าของตัวนับโปรแกรมจะถูกปรับปรุงทุกๆรอบสัญญาณนาฬิกา โดยการปรับปรุงค่าของตัวนับจะขึ้นอยู่กับเงื่อนไขการทำงานของหน่วยประมวลผลในขณะนั้น เช่น เกิดการขัดจังหวะหรือการวนลูปของโปรแกรมขึ้น เป็นต้น เงื่อนไขการปรับปรุงตัวนับโปรแกรมแสดงในตารางที่ 3.6 โดยเงื่อนไขการปรับปรุงตัวนับโปรแกรมจะถูกนำไปถอดรหัส เพื่อนำไปเลือกข้อมูลที่นำไปเขียนลงในรีจิสเตอร์ตัวนับโปรแกรม

ตารางที่ 3.6 เงื่อนไขการปรับปรุงตัวนับโปรแกรม

ค่าของตัวนับโปรแกรม	เงื่อนไข	ลำดับความสำคัญ
PC <- INT Vector	เกิดการขัดจังหวะหน่วยประมวลผล	1
PC <- PC_Backup	ดำเนินการคำสั่ง RETI	2
PC <- Repeat Start	เกิดการวนลูปตามจำนวนรอบที่กำหนด	3
PC <- Store Operand	เขียนข้อมูลลงในรีจิสเตอร์ PC ด้วยคำสั่ง STIO	4
PC <- PC+PC Increment	ดำเนินการคำสั่งปรกติ	5

เมื่อเกิดการขัดจังหวะหน่วยประมวลผลขึ้น ค่าของรีจิสเตอร์ GIFR และแฟล็กสถานะ I จะถูกนำมาถอดรหัสเพื่อพิจารณาว่าเกิดการขัดจังหวะเนื่องจากแหล่งใด แล้วจึงนำค่าที่ได้มาเลือกตำแหน่งของเวคเตอร์การขัดจังหวะ (Interrupt Vector) จากตารางเวคเตอร์การขัดจังหวะ (Interrupt Vector Table) และส่งสัญญาณ GIEXE เพื่อแจ้งไปยังวงจรย่อยอื่นให้ทราบว่ามีการขัดจังหวะเกิดขึ้น ส่วนค่าของตัวนับโปรแกรมในปัจจุบันจะถูกนำมาเก็บในรีจิสเตอร์ PC_Backup

เมื่อโปรแกรมทำงานแบบวนลูป จำนวนรอบของการทำงานจะถูกนำมาเก็บที่รีจิสเตอร์ Repeat Count ด้วยคำสั่ง SRPT หลังจากนั้นเมื่อดำเนินการคำสั่ง ERPT ตำแหน่งของจุดสิ้นสุดการวนลูปจะถูกเก็บที่รีจิสเตอร์ Repeat End และตำแหน่งของจุดเริ่มต้นของการวนลูปซึ่งก็คือตำแหน่งของโปรแกรมที่อยู่ถัดจากคำสั่ง ERPT จะถูกเก็บในรีจิสเตอร์ Repeat Start เมื่อโปรแกรมทำงานมาถึงตำแหน่งสิ้นสุดของลูปซึ่งตำแหน่งของโปรแกรมตรงกับค่าในรีจิสเตอร์ Repeat End จะทำให้ค่าในรีจิสเตอร์ Repeat Count ลดลงทีละหนึ่ง แล้วตำแหน่งของโปรแกรมก็จะถูกปรับปรุงให้มีค่าเท่ากับรีจิสเตอร์ Repeat Start การทำงานแบบวนลูปจะเกิดซ้ำไปเรื่อยๆ จนครบตามจำนวนรอบที่กำหนด

เมื่อหน่วยประมวลผลดำเนินการคำสั่งปรกติ การปรับปรุงค่าของตัวนับโปรแกรมจะแบ่งออกเป็นสองรูปแบบ ได้แก่ การปรับปรุงเนื่องจากคำสั่งปรกติและการปรับปรุงเนื่องจากคำสั่งการกระโดด กรณีการปรับปรุงค่าของตัวนับโปรแกรมเนื่องจากคำสั่งปรกติ ค่าของตัวนับโปรแกรมจะเพิ่มขึ้นทีละหนึ่ง ส่วนในกรณีการปรับปรุงเนื่องจากคำสั่งการกระโดด ค่าของตัวนับโปรแกรมที่ถูกเพิ่มล่วงหน้าด้วยหนึ่งจะถูกบวกด้วยค่าสัมพัทธ์ซึ่งเป็นจำนวนมีเครื่องหมายแบบ 2's Complement ขนาด 16 บิตซึ่งถูกดำเนินการขยายแบบมีเครื่องหมาย (Sign Extension) จากค่าการกระจัด (Displacement) ขนาด 9 บิต ซึ่งหากคำสั่งกระโดดเป็นชนิดที่มีเงื่อนไขและเงื่อนไขเป็น

เท็จ การปรับปรุงค่าของตัวนับโปรแกรมถูกบวกด้วยค่าสัมพัทธ์จะถูกยกเลิก เนื่องจากสัญญาณ PC_stop จะทำให้ค่าของตัวนับโปรแกรมไม่เปลี่ยนแปลง

เมื่อหน่วยประมวลผลดำเนินการคำสั่ง LPM ซึ่งเป็นการอ่านข้อมูลจากหน่วยความจำโปรแกรม ตำแหน่งของหน่วยความจำโปรแกรมจะใช้ค่าจากรีจิสเตอร์ PDAR (Program Data Address Register) ซึ่งเป็นรีจิสเตอร์ใช้งานพิเศษ (ดูรายละเอียดในหัวข้อที่ 4.1.9) ที่สามารถเขียนค่าได้ด้วยคำสั่ง STIO

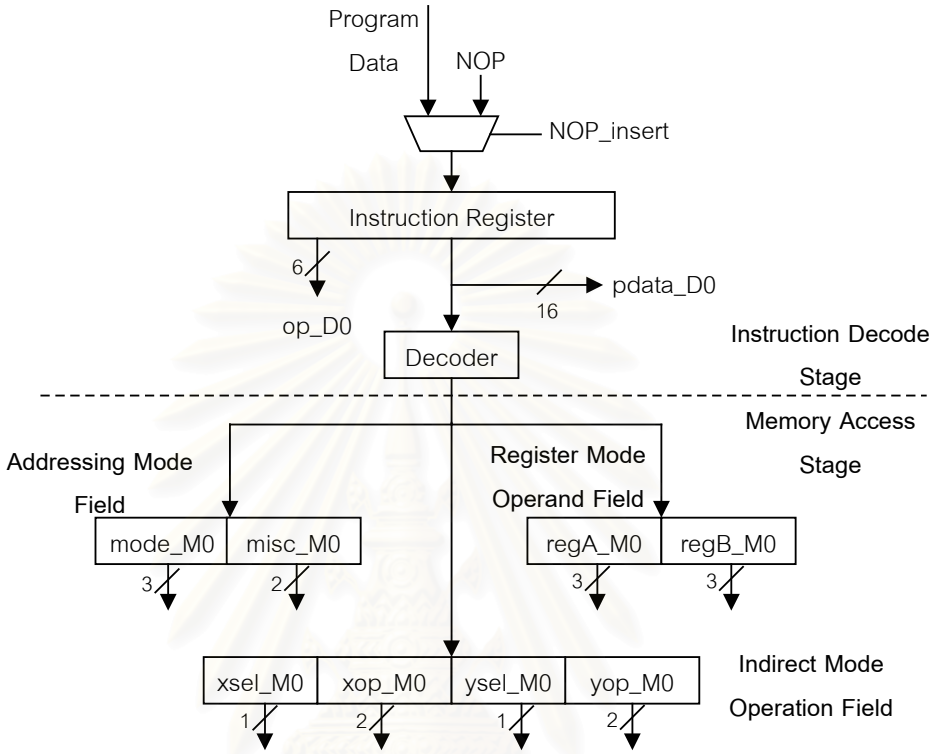
ส่วนถอดรหัสการดำเนินการมีหน้าที่อ่านข้อมูลจากฟิลด์การดำเนินการแล้วนำมาถอดรหัสคำสั่งล่วงหน้าก่อนที่จะถึงขั้นตอนที่ต้องพิจารณาคำสั่งนั้นๆ โดยสัญญาณที่ถอดรหัสแล้วจะมีขนาดเพียงหนึ่งบิตและถูกเก็บไว้ในรีจิสเตอร์ ทำให้หน่วยควบคุมสามารถเข้าใจคำสั่งได้อย่างรวดเร็วและมีความล่าช้าน้อยเช่น ถ้าสัญญาณ stio_M0 มีค่าเป็น 1 แสดงว่าคำสั่งในไปป์ไลน์ขั้นตอนการเข้าถึงหน่วยความจำ (ขั้นตอน M0) คือคำสั่ง STIO เป็นต้น ส่วนถอดรหัสการดำเนินการจะถูกใช้กับวงจรย่อยส่วนใหญ่ของหน่วยประมวลผล โดยคำสั่งที่ถูกนำมาถอดรหัสจะเป็นคำสั่งที่ใช้ควบคุมการทำงานของวงจรย่อยนั้น

3.4.2 ตัวถอดรหัสคำสั่ง

เมื่อคำสั่งถูกเฟ็ทช์เข้ามาในหน่วยประมวลผลกลางจะเก็บค่าในรีจิสเตอร์และถอดรหัสที่ตัวถอดรหัสคำสั่ง หน้าที่ของตัวถอดรหัสคำสั่งคือ การถอดรหัสคำสั่ง และแยกฟิลด์ของรหัสดำเนินการ ให้อยู่ในรูปแบบที่วงจรย่อยภายในสามารถเข้าใจได้ เนื่องจากในแอดเดรสซิงโมดแต่ละแบบจะมีฟิลด์ของรหัสดำเนินการซ้อนทับที่ตำแหน่งเดียวกัน

โครงสร้างของตัวถอดรหัสคำสั่งจะประกอบด้วยสองส่วน (ดูรูปที่ 3.14) ได้แก่ ส่วนที่เป็นวงจรเชิงจัดหมู่ที่ทำหน้าที่ถอดรหัสคำสั่ง (Decoder) โดยพิจารณาเงื่อนไขของการดำเนินการและแอดเดรสซิงโมดต่างๆ และส่วนที่เป็นรีจิสเตอร์ ซึ่งเก็บข้อมูลของฟิลด์ต่างๆ ที่ถูกถอดรหัสแล้ว ฟิลด์ของคำสั่งที่ถูกถอดรหัสแล้ว จะถูกนำมาใช้ในขั้นตอนการเข้าถึงหน่วยความจำ (ขั้นตอน M0) ซึ่งแบ่งได้เป็น 3 ฟิลด์ได้แก่ ฟิลด์แอดเดรสซิงโมด (Addressing Mode Field) ฟิลด์ตัวถูกดำเนินการของแอดเดรสซิงโมดแบบรีจิสเตอร์ (Register Mode Operand Field) และฟิลด์การดำเนินการสำหรับแอดเดรสซิงโมดแบบโดยอ้อม (Indirect Mode Operation Field) เนื่องจากรหัสดำเนินการของหน่วยประมวลผลมีความยาว 16 บิต แต่ทว่าโครงสร้างภายในของหน่วยประมวลผลมีวงจรย่อยหลายส่วน และมีสัญญาณควบคุมเป็นจำนวนมาก ดังนั้นจึงมีการเข้ารหัสฟิลด์ของรหัสดำเนินการให้มีการซ้อนทับกัน เนื่องจากวงจรย่อยแต่ละส่วนไม่ได้ทำงานพร้อมกันตลอดเวลา ตัว

อย่างเช่น คำสั่งที่มีแอดเดรสซิงโมดแบบรีจิสเตอร์จะไม่ได้ใช้งานหน่วยกำเนิดตำแหน่งข้อมูล ดังนั้น สัญญาณควบคุมของหน่วยกำเนิดตำแหน่งข้อมูลต้องสั่งการให้หน่วยกำเนิดตำแหน่งข้อมูลหยุดทำงาน



รูปที่ 3.14 โครงสร้างของตัวถอดรหัสคำสั่ง

การถอดรหัสของฟิลด์แอดเดรสซิงโมดจะพิจารณาคำสั่งที่มีแอดเดรสซิงโมดแบบโดยตรงและแบบใช้ทันที ได้แก่ คำสั่ง LDKH LDKL STDX และ STDY โดยการถอดรหัสคำสั่งเหล่านี้จำเป็นต้องกำหนดแอดเดรสซิงโมดที่สอดคล้องกับการดำเนินการด้วย ตัวอย่างเช่น คำสั่ง LDKH และ LDKL จะมีฟิลด์แอดเดรสซิงโมดที่ถูกถอดรหัสแล้ว (mode_M0) เป็นแบบรีจิสเตอร์ ส่วนคำสั่ง STDX และ STDY จะมีฟิลด์แอดเดรสซิงโมดที่ถูกถอดรหัสแล้วเป็นแอดเดรสซิงโมดแบบโดยอ้อมชนิดที่ 1 และ 2 ตามลำดับ

การถอดรหัสของฟิลด์ตัวถูกดำเนินการของแอดเดรสซิงโมดแบบรีจิสเตอร์ แบ่งเป็นการถอดรหัสของสัญญาณ regA_M0 ซึ่งนำไปเลือกตัวถูกดำเนินการและรีจิสเตอร์สำหรับเก็บผลลัพธ์ของการดำเนินการในแฟ้มรีจิสเตอร์ และการถอดรหัสของสัญญาณ regB_M0 ซึ่งนำไปเลือกตัวถูกดำเนินการจากแฟ้มรีจิสเตอร์เท่านั้น การถอดรหัสของสัญญาณ regA_M0 จะพิจารณาว่าหากเป็นคำสั่ง LDKH LDKL STDX หรือ STDY ฟิลด์เลือกตัวถูกดำเนินการเพียงสองบิตสำหรับเลือกรีจิส

เตอร์ R0 ถึง R3 จะถูกแปลงให้กลายเป็นสามบิตด้วยการเติมศูนย์เข้าด้านหน้าบิตที่มีความสำคัญมากที่สุด (Most Significant Bit :MSB) และถ้าหากเป็นคำสั่งในกลุ่มคณิตศาสตร์และตรรกะที่มีแอดเดรสซิงโมดเป็นแบบโดยอ้อมชนิดที่ 3 ซึ่งข้อมูลของตัวถูกดำเนินการมาจากหน่วยความจำทั้งคู่ จะให้ผลลัพธ์ของการดำเนินการถูกที่รีจิสเตอร์ R0 คือสัญญาณ regA_M0 มีค่าเท่ากับ 000 ส่วนการถอดรหัสของสัญญาณ regB_M0 จะพิจารณาว่าหากเป็นคำสั่ง LDKH หรือ LDKL จะให้ผลลัพธ์ของสัญญาณ regB_M0 มีค่าเท่ากับสัญญาณ regA_M0 เนื่องจากคำสั่งเหล่านี้จำเป็นต้องนำค่าของตัวถูกดำเนินการที่ถูกเลือกด้วยสัญญาณ regB_M0 มาแก้ไข แล้วจึงเขียนค่ากลับลงไปที่ตำแหน่งของรีจิสเตอร์ที่ถูกเลือกด้วยสัญญาณ regA_M0

ฟิลด์การดำเนินการสำหรับแอดเดรสซิงโมดแบบโดยอ้อมมีหน้าที่ควบคุมการทำงานของวงจรย่อยที่เกี่ยวข้องกับการทำงานของแอดเดรสซิงโมดแบบโดยอ้อม ซึ่งได้แก่ หน่วยกำเนิดตำแหน่งข้อมูล ให้ทำงานเฉพาะในกรณีที่มีแอดเดรสซิงโมดแบบโดยอ้อมเท่านั้น สัญญาณควบคุมจะแบ่งเป็นสัญญาณควบคุมสำหรับวงจรที่เกี่ยวข้องกับหน่วยความจำ X เช่น xsel_M0, xop_M0 และสัญญาณควบคุมสำหรับวงจรที่เกี่ยวข้องกับหน่วยความจำ Y เช่น ysel_M0, yop_M0 สัญญาณ xsel_M0 และ ysel_M0 ใช้สำหรับเลือกตัวชี้โปรแกรมที่ใช้อ้างตำแหน่งข้อมูล (ARP) ส่วนสัญญาณ xop_M0 และ yop_M0 ใช้สำหรับเลือกประเภทของการดำเนินการที่ใช้ปรับปรุงตัวชี้ข้อมูล การถอดรหัสของสัญญาณ xop_M0 และ yop_M0 จะใช้หลักการว่าคำสั่งที่ไม่ได้มีแอดเดรสซิงโมดแบบโดยอ้อมจะต้องไม่ทำให้ค่าของตัวชี้ข้อมูลเปลี่ยนแปลง ซึ่งสัญญาณ xop_M0 และ yop_M0 จะมีค่าเท่ากับ 00 ซึ่งหมายความว่าจะไม่มีการดำเนินการกับตัวชี้ข้อมูลของหน่วยกำเนิดตำแหน่งข้อมูลนั้นๆ

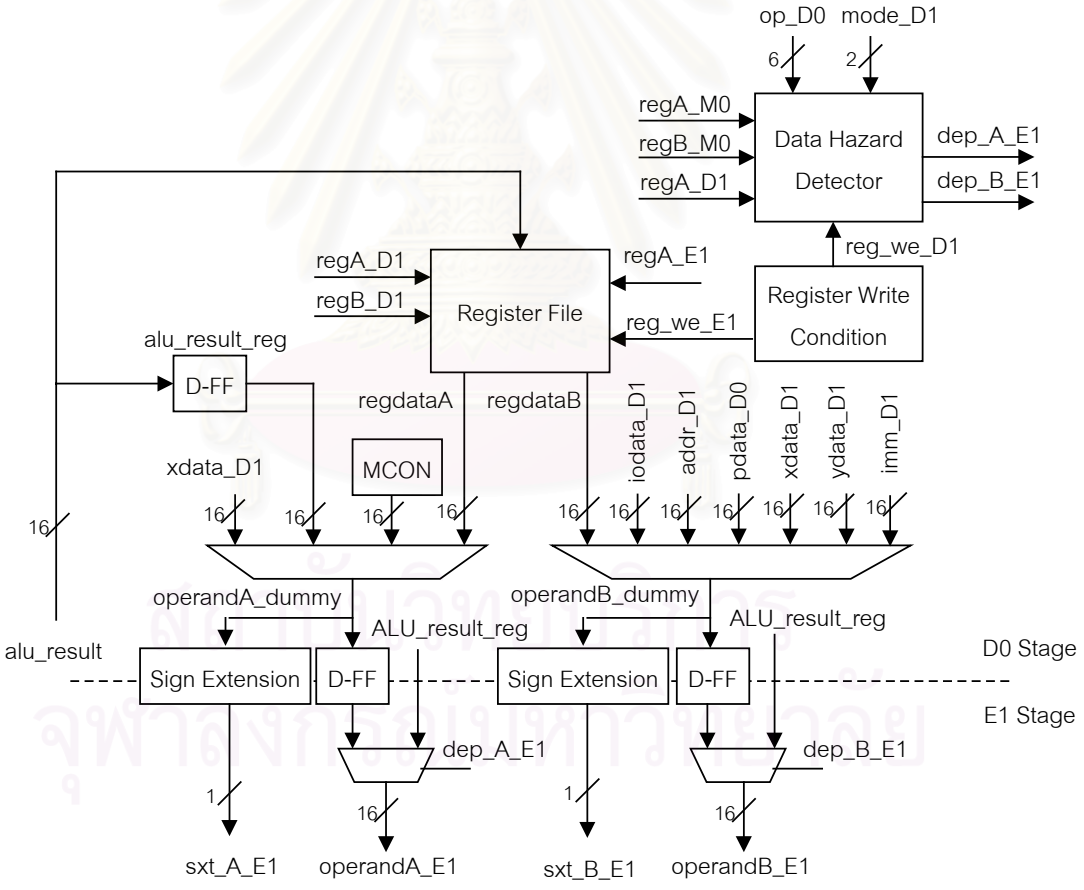
สัญญาณ pdata_D0 เป็นสัญญาณที่อยู่ในขั้นตอนการถอดรหัสคำสั่ง (ขั้นตอน D0) ซึ่งเก็บข้อมูลโปรแกรมที่ยังไม่ได้ถอดรหัส เพื่อส่งผ่านข้อมูลที่อยู่ภายในหน่วยความจำโปรแกรม ตำแหน่งข้อมูลสำหรับแอดเดรสซิงโมดแบบโดยตรง ข้อมูลค่าคงที่สำหรับแอดเดรสซิงโมดแบบใช้ทันที หรือค่าสัมพัทธ์สำหรับคำสั่งการกระโดด ไปยังวงจรย่อยส่วนต่างๆ

สัญญาณ op_D0 เป็นสัญญาณของฟิลด์การดำเนินการที่อยู่ในขั้นตอนการถอดรหัสคำสั่ง โดยจะถูกส่งไปถอดรหัสด้วยส่วนถอดรหัสการดำเนินการของวงจรย่อยแต่ละส่วน

สัญญาณ NOP_insert ใช้ควบคุมให้หน่วยประมวลผลเฟ็ทช์คำสั่ง NOP สำหรับป้องกันการเกิดอันตรายเชิงการควบคุมของหน่วยประมวลผลแบบไปป์ไลน์อันเนื่องมาจากการขัดจังหวะหน่วยประมวลผล หรือการกระโดดของโปรแกรม เป็นต้น

3.4.3 หน่วยเลือกตัวถูกดำเนินการและแฟ้มรีจิสเตอร์

หน่วยเลือกตัวถูกดำเนินการมีหน้าที่เลือกประเภทของตัวถูกดำเนินการในขั้นตอนอ่านค่าตัวถูกดำเนินการ (ขั้นตอน D1) เพื่อข้อมูลตัวถูกดำเนินการส่งต่อไปยังหน่วยดำเนินการอื่นๆ ในขั้นตอนดำเนินการ (ขั้นตอน E1) ตัวถูกดำเนินการแบ่งได้เป็น 6 ประเภท ได้แก่ ข้อมูลจากแฟ้มรีจิสเตอร์ (สัญญาณ regdataA และ regdataB) ข้อมูลจากหน่วยความจำข้อมูล (สัญญาณ xdata_D1 และ ydata_D1) ข้อมูลจากหน่วยความจำโปรแกรม (สัญญาณ pdata_D0) ข้อมูลตำแหน่งจากตัวชี้ข้อมูล (สัญญาณ addr_D1) ข้อมูลค่าคงที่สำหรับแอดเดรสซิงโมดแบบใช้ทันที (สัญญาณ imm_D1) และข้อมูลจากรีจิสเตอร์ใช้งานพิเศษ (สัญญาณ iodata_D1) นอกจากนี้หน่วยเลือกตัวถูกดำเนินการยังมีหน้าที่ตรวจสอบและแก้ไขอันตรายเชิงข้อมูลอันเนื่องมาจากการที่หน่วยประมวลผลมีโครงสร้างแบบไปป์ไลน์ด้วย โครงสร้างของหน่วยเลือกตัวถูกดำเนินการและแฟ้มรีจิสเตอร์แสดงในรูปที่ 3.15



รูปที่ 3.15 โครงสร้างของหน่วยเลือกตัวถูกดำเนินการ

การเลือกตัวถูกดำเนินการแบ่งได้เป็นสองส่วนได้แก่ ส่วนที่เลือกตัวถูกดำเนินการของสัญญาณ operandA_E1 ในขั้นตอน E1 ซึ่งรับค่ามาจากสัญญาณ operandA_dummy และส่วน

ที่เลือกตัวถูกดำเนินการของสัญญาณ operandB_E1 ในขั้นตอน E1 ซึ่งรับค่ามาจากสัญญาณ operandB_dummy เงื่อนไขของการเลือกตัวถูกดำเนินการของสัญญาณ operandA_dummy และ operandB_dummy แสดงในตารางที่ 3.7 และ 3.8 ตามลำดับ

ตารางที่ 3.7 เงื่อนไขของการเลือกตัวถูกดำเนินการ operandA_dummy

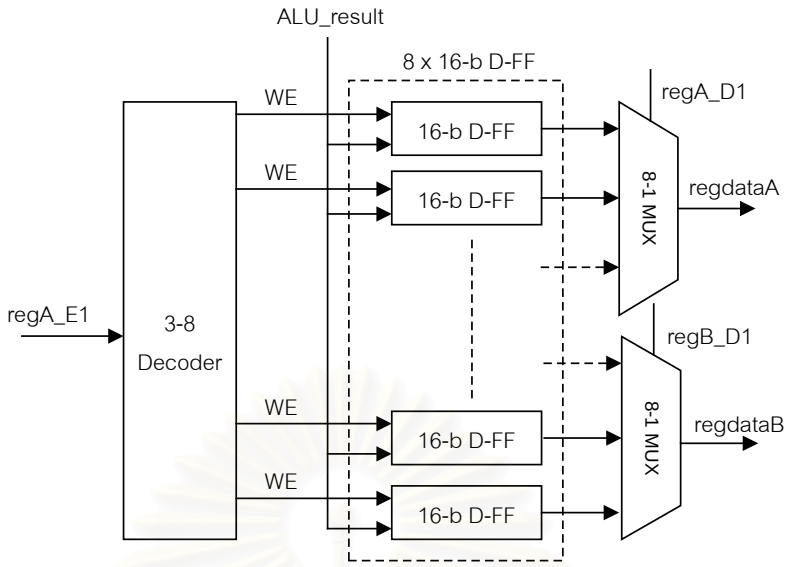
ตัวถูกดำเนินการที่เลือก	เงื่อนไข	คำอธิบายของตัวถูกดำเนินการ
ALU_result_reg	ดำเนินการคำสั่ง MADD และ Opc=RREG	รีจิสเตอร์ผลลัพธ์ของ ALU
MCON	ดำเนินการคำสั่ง MADD และ Opc=MCON	รีจิสเตอร์เฉพาะ MCON *
xdata_D1	อ้างถึงข้อมูลจากหน่วยความจำ X	ข้อมูลจากหน่วยความจำ X
regdataA	อ้างถึงข้อมูลจากแฟ้มรีจิสเตอร์	ข้อมูลจากแฟ้มรีจิสเตอร์

* ดูรายละเอียดในหัวข้อที่ 4.1.9

ตารางที่ 3.8 เงื่อนไขของการเลือกตัวถูกดำเนินการ operandB_dummy

ตัวถูกดำเนินการที่เลือก	เงื่อนไข	คำอธิบายของตัวถูกดำเนินการ
iodata_D1	ดำเนินการคำสั่ง LDIO	ข้อมูลจากรีจิสเตอร์ใช้งานพิเศษ
addr_D1	ดำเนินการคำสั่ง LDARP	ค่าของตัวชี้ข้อมูลของหน่วยความจำ X และ Y
imm_D1	ดำเนินการคำสั่ง LDKH หรือ LDKL	ค่าคงที่จากตัวถอดรหัสคำสั่ง
pdata_D0	ดำเนินการคำสั่ง LPM	ข้อมูลจากหน่วยความจำโปรแกรม
xdata_D1	อ้างถึงข้อมูลจากหน่วยความจำ X	ข้อมูลจากหน่วยความจำ X
ydata_D1	อ้างถึงข้อมูลจากหน่วยความจำ Y	ข้อมูลจากหน่วยความจำ Y
regdataB	อ้างถึงข้อมูลจากแฟ้มรีจิสเตอร์	ข้อมูลจากแฟ้มรีจิสเตอร์

แฟ้มรีจิสเตอร์ประกอบด้วยรีจิสเตอร์ใช้งานทั่วไปจำนวน 8 ชุด สำหรับใช้เก็บตัวถูกดำเนินการในกรณีที่แอดเดรสซิงโมดมีการเรียกใช้ค่าในรีจิสเตอร์ โครงสร้างของแฟ้มรีจิสเตอร์ถูกออกแบบให้สามารถอ่านข้อมูลสำหรับดำเนินการได้พร้อมกันทีละสองชุด (สำหรับตัวถูกดำเนินการ A และ B) และสามารถเขียนผลลัพธ์ของการดำเนินการลงในรีจิสเตอร์ภายในแฟ้มในขณะที่กำลังอ่านข้อมูลได้



รูปที่ 3.16 โครงสร้างของแฟ้มรีจิสเตอร์

คำสั่งในกลุ่มการคูณและสะสมสามารถดำเนินการกับข้อมูลทั้งแบบที่มีและไม่มีเครื่องหมาย ข้อมูลที่ใช้เป็นตัวถูกดำเนินการจะถูกขยายให้มีขนาดเป็น 17 บิตก่อนที่จะส่งไปยังหน่วยคูณและสะสม การขยายบิตข้อมูลจะเป็นแบบมีเครื่องหมาย เมื่อการดำเนินการเป็นแบบมีเครื่องหมาย และการขยายบิตข้อมูลจะเป็นแบบไม่มีเครื่องหมาย เมื่อการดำเนินการเป็นแบบไม่มีเครื่องหมาย โดยข้อมูลบิตที่ 17 ที่ถูกขยายจะแทนด้วยสัญญาณ sxt_A_E1 สำหรับตัวถูกดำเนินการ operandA_E1 และแทนด้วยสัญญาณ sxt_B_E1 สำหรับตัวถูกดำเนินการ operandB_E1

การป้องกันความผิดพลาดจากอันตรายเชิงข้อมูลเนื่องมาจากการที่หน่วยประมวลผลมีโครงสร้างแบบไปป์ไลน์จะใช้ส่วนตรวจจับอันตรายเชิงข้อมูล (Data Hazard Detector) ซึ่งเป็นวงจรเชิงจัดหมู่ (Combinational Circuit) ที่มีหน้าที่ตรวจสอบเงื่อนไขของการเกิดอันตรายเชิงข้อมูล แล้วจึงส่งสัญญาณควบคุมเพื่อให้เกิดการส่งผ่านข้อมูลที่ถูกต้องไปแทน

อันตรายเชิงข้อมูลเกิดเนื่องมาจากคำสั่งปัจจุบันอ่านค่าจากรีจิสเตอร์ที่ถูกดำเนินการด้วยคำสั่งก่อนหน้าแต่ว่ายังไม่ได้ถูกเขียนกลับลงในแฟ้มรีจิสเตอร์ ดังนั้นเงื่อนไขการเกิดอันตรายเชิงข้อมูลจะพิจารณาว่า แอดเดรสซิงโมดของคำสั่งก่อนหน้าว่ามีการเขียนค่าลงในรีจิสเตอร์หรือไม่ และรีจิสเตอร์ที่ถูกดำเนินการในคำสั่งปัจจุบันและก่อนหน้าว่าเป็นตัวเดียวกันหรือไม่ การตรวจสอบเงื่อนไขของอันตรายเชิงข้อมูลจะพิจารณาเมื่อคำสั่งปัจจุบันอยู่ในขั้นตอนการเข้าถึงหน่วยความจำ (ขั้นตอน M0) และคำสั่งก่อนหน้าอยู่ในขั้นตอนการอ่านค่าตัวถูกดำเนินการ (ขั้นตอน D1) โดยแบ่งเป็นสองรูปแบบได้แก่ อันตรายเชิงข้อมูลที่เกิดกับสัญญาณ operandA_E1 และอันตรายเชิงข้อมูลที่เกิดกับสัญญาณ operandB_E1 เงื่อนไขของการเกิดอันตรายเชิงข้อมูลแสดงในตารางที่ 3.9

ตารางที่ 3.9 เงื่อนไขของการเกิดอันตรายเชิงข้อมูล

สัญญาณที่เกิดอันตรายเชิงข้อมูล	เงื่อนไขการเกิดอันตรายเชิงข้อมูล
operandA_E1	1. คำสั่งในขั้นตอน D1 มีการเขียนค่าลงในแฟ้มรีจิสเตอร์ 2. คำสั่งในขั้นตอน D1 มีแอดเดรสซิงโมดแบบรีจิสเตอร์หรือแบบโดยอ้อมชนิดที่ 1 หรือ 2 3. สัญญาณเลือกแฟ้มรีจิสเตอร์ regA_D1 เท่ากับ regA_M0
operandB_E1	1. คำสั่งในขั้นตอน D1 มีการเขียนค่าลงในแฟ้มรีจิสเตอร์ 2. คำสั่งในขั้นตอน D1 มีแอดเดรสซิงโมดแบบรีจิสเตอร์ 3. สัญญาณเลือกแฟ้มรีจิสเตอร์ regA_D1 เท่ากับ regB_M0

3.4.4 หน่วยคณิตศาสตร์และตรรกะ

หน่วยคณิตศาสตร์และตรรกะมีหน้าที่ดำเนินการคำสั่งทางคณิตศาสตร์ ซึ่งได้แก่ การบวกและการเปรียบเทียบข้อมูล และคำสั่งทางตรรกะ นอกจากนี้คำสั่งประเภทการโหลดข้อมูลจะใช้หน่วยคณิตศาสตร์และตรรกะเป็นทางผ่านในการย้ายข้อมูลไปยังแฟ้มรีจิสเตอร์ด้วย การดำเนินการของหน่วยคณิตศาสตร์และตรรกะจะส่งผลให้แฟล็กสถานะ (Status Flag) ซึ่งได้แก่ แฟล็กตัวทด (Carry Flag) แฟล็กศูนย์ (Zero Flag) แฟล็กเครื่องหมาย (Sign Flag) และแฟล็กนิเสธ (Negative Flag) เปลี่ยนแปลง

โครงสร้างของหน่วยคณิตศาสตร์และตรรกะแบ่งได้เป็นสองส่วน (ดูรูปที่ 3.17) ได้แก่ ส่วนดำเนินการทางตรรกะ (Logic Path) และส่วนดำเนินการทางคณิตศาสตร์ (Arithmetic Path) โดยผลลัพธ์ของหน่วยคณิตศาสตร์และตรรกะ ซึ่งได้แก่ สัญญาณ alu_result จะถูกเลือกจากเอาต์พุตของวงจรทั้งสองส่วน ซึ่งได้แก่ สัญญาณ logic_out และสัญญาณ arith_out ด้วยสัญญาณ arith_sel_E1

ส่วนดำเนินการทางตรรกะมีหน้าที่ประมวลผลการดำเนินการทางตรรกะ โดยผลลัพธ์ของส่วนการดำเนินการทางตรรกะ ซึ่งได้แก่ สัญญาณ logic_out จะถูกเลือกจากมัลติเพล็กเซอร์ชนิด 8 เลือกออก 1 ด้วยสัญญาณควบคุม logic_op_E1 ที่ได้จากวงจรถอดรหัสคำสั่ง การทำงานของส่วนดำเนินการทางตรรกะแสดงในตารางที่ 3.10

ตารางที่ 3.10 การเลือกผลลัพธ์ของส่วนการดำเนินการทางตรรกะ

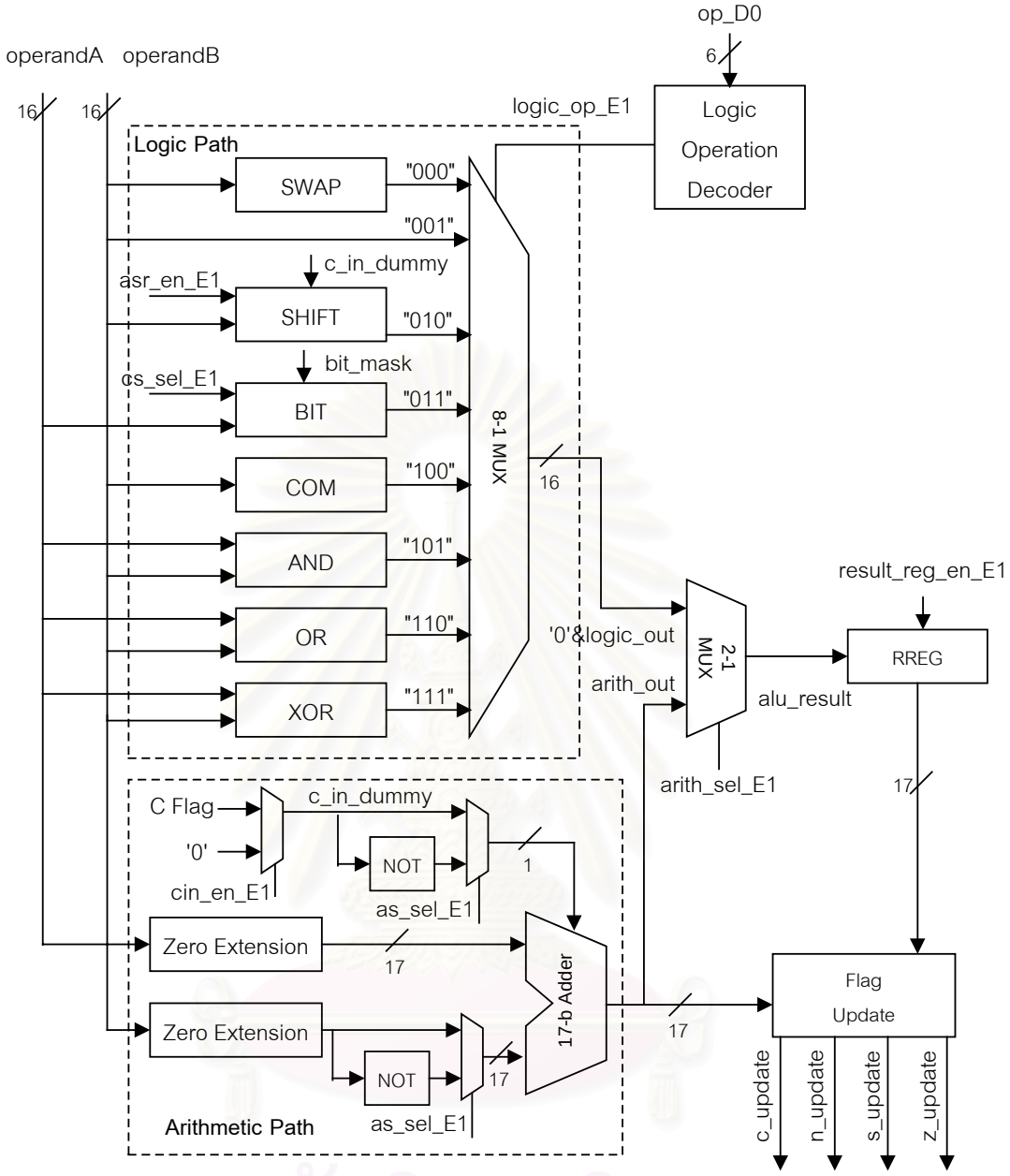
สัญญาณควบคุม logic_op_E1	การดำเนินการของส่วนดำเนินการทางตรรกะ
000	operandB [7:0]&operandB [15:8]
001	operandB [15:0]
010	SHR/ ASR/ RRC (ดูตารางที่ 3.11)
011	SETF/ CLRF (ดูตารางที่ 3.12)
100	NOT operandB[15:0]
101	operandA [15:0] AND operandB [15:0]
110	operandA [15:0] OR operandB [15:0]
111	operandA [15:0] XOR operandB [15:0]

เมื่อสัญญาณควบคุม logic_op_E1 มีค่าเท่ากับ 010 จะเป็นการเลือกผลลัพธ์จากวงจรเลื่อนข้อมูล (วงจร SHIFT) ซึ่งควบคุมด้วยสัญญาณอินพุต asr_en_E1 และ c_in_dummy การทำงานของวงจรเลื่อนข้อมูลแสดงในตารางที่ 3.11

ตารางที่ 3.11 การทำงานของวงจรเลื่อนข้อมูล

สัญญาณ asr_en_E1	ผลลัพธ์ของวงจรเลื่อนข้อมูล	การดำเนินการคำสั่ง
0	operandB[15]&operand[15:1]	ASR
1	c_in_dummy&operand[15:1]	RRC หรือ SHR

เมื่อสัญญาณ asr_en_E1 มีค่าเป็น 1 วงจรเลื่อนข้อมูลจะดำเนินการคำสั่ง ASR ซึ่งเป็นการเลื่อนข้อมูลไปทางขวาแบบมีเครื่องหมาย และถ้าสัญญาณ asr_en_E1 มีค่าเป็น 0 การทำงานของวงจรเลื่อนข้อมูลจะขึ้นอยู่ว่าถ้าสัญญาณ c_in_dummy มีค่าเท่ากับแฟล็กตัวทด วงจรเลื่อนข้อมูลจะดำเนินการคำสั่ง RRC แต่ถ้าสัญญาณ c_in_dummy มีค่าเป็น 0 วงจรเลื่อนข้อมูลจะดำเนินการคำสั่ง SHR



รูปที่ 3.17 โครงสร้างของหน่วยคณิตศาสตร์และตรรกะ

เมื่อสัญญาณควบคุม logic_op_E1 มีค่าเท่ากับ 011 เป็นการเลือกผลลัพธ์จากวงจรจัดการบิตข้อมูล (BIT) ซึ่งควบคุมด้วยสัญญาณ cs_sel_E1 และ bit_mask การทำงานของวงจรจัดการบิตข้อมูลแสดงในตารางที่ 3.12

ตารางที่ 3.12 การทำงานของวงจรจัดการบิตข้อมูล

สัญญาณ cs_sel_E1	ผลลัพธ์ของวงจรจัดการบิตข้อมูล	การดำเนินการคำสั่ง
0	operandA AND (NOT bit_mask)	CLRF
1	operandA OR bit_mask	SETF

เมื่อสัญญาณ cs_sel_E1 มีค่าเป็น 0 วงจรจัดการบิตข้อมูลจะดำเนินการคำสั่ง CLRF ซึ่งจะเคลียร์บิตข้อมูลของสัญญาณ operandA โดยการ AND สัญญาณ operandA กับนิเสธของสัญญาณ bit_mask ซึ่งจะมีค่าเป็น 1 เฉพาะบิตที่สนใจเพียงบิตเดียว เมื่อสัญญาณ cs_sel_E1 มีค่าเป็น 1 วงจรจัดการบิตข้อมูลดำเนินการคำสั่ง SETF ซึ่งจะเซ็ตบิตข้อมูลของสัญญาณ operandA โดยการ OR สัญญาณ operandA กับสัญญาณ bit_mask ซึ่งจะมีค่าเป็น 1 เฉพาะบิตที่สนใจเพียงบิตเดียว

เมื่อสัญญาณควบคุม logic_op_E1 มีค่าเท่ากับ 001 จะเป็นการส่งผ่านสัญญาณ operandB ไปยังสัญญาณ logic_out โดยตรง ซึ่งการดำเนินการในรูปแบบนี้จะถูกนำมาใช้สำหรับคำสั่ง MOV และคำสั่งประเภทโหลดข้อมูล (คำสั่ง LDxx) เนื่องจากคำสั่งเหล่านี้จะใช้หน่วยคณิตศาสตร์และตรรกะเป็นทางผ่านของข้อมูลที่รับมาจากหน่วยเลือกตัวถูกดำเนินการเพื่อส่งไปยังแฟ้มรีจิสเตอร์

ส่วนดำเนินการทางคณิตศาสตร์ ประกอบด้วยวงจรบวก-ลบข้อมูลขนาด 17 บิต สำหรับการดำเนินการบวก ลบ และเปรียบเทียบข้อมูล ข้อมูลตัวถูกดำเนินการขนาด 16 บิตจะถูกขยายแบบไม่มีเครื่องหมายให้มีขนาด 17 บิตแล้วจึงส่งไปยังวงจรบวก การทำงานของส่วนดำเนินการทางคณิตศาสตร์ถูกควบคุมด้วยสัญญาณ as_sel_E1 และ cin_en_E1 mask การทำงานของส่วนดำเนินการทางคณิตศาสตร์แสดงในตารางที่ 3.13

ตารางที่ 3.13 การทำงานของส่วนดำเนินการทางคณิตศาสตร์

สัญญาณ as_sel_E1	สัญญาณ cin_en_E1	ผลลัพธ์ของวงจรเลื่อนข้อมูล	การดำเนินการคำสั่ง
0	0	operandA + operandB	ADD
0	1	operandA + operandB + cin	ADDC
1	0	operandA - operandB	SUB
1	1	operandA - operandB - cin	SUBB

สัญญาณ alu_result จะถูกส่งไปยังรีจิสเตอร์ RREG ที่มีหน้าที่เก็บผลลัพธ์จากส่วนดำเนินการ การเขียนข้อมูลลงรีจิสเตอร์ RREG จะถูกควบคุมด้วยสัญญาณ result_reg_en_E1 ซึ่งจะควบคุมให้มีการเขียนค่าลงใน RREG เฉพาะผลลัพธ์ของคำสั่งที่เป็นการดำเนินการในกลุ่มคณิตศาสตร์และตรรกะ และคำสั่งประเภทโหลดข้อมูล

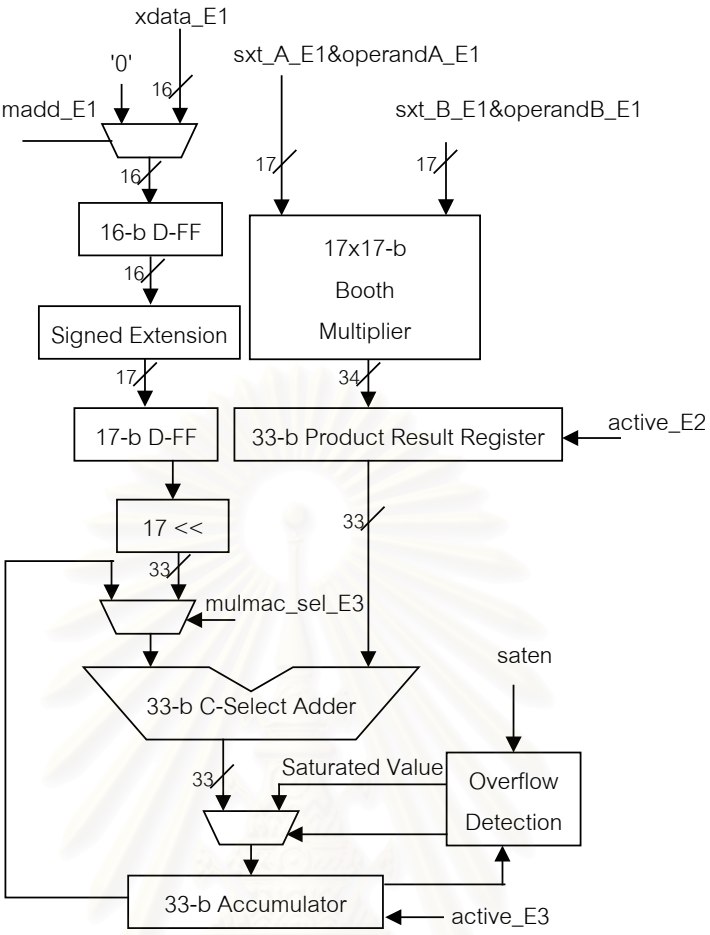
วงจรปรับปรุงแฟล็กสถานะ (Flag Update) ทำหน้าที่สร้างค่าปรับปรุงของแฟล็กสถานะสำหรับป้อนไปยังรีจิสเตอร์สถานะ โครงสร้างของวงจรปรับปรุงแฟล็กเป็นวงจรเชิงจัดหมู่ที่ทำหน้าที่ถอดรหัสสัญญาณอินพุตจากรีจิสเตอร์ RREG และสัญญาณ arith_out เงื่อนไขของสัญญาณปรับปรุงแฟล็กสถานะแสดงในตารางตารางที่ 3.14 สัญญาณปรับปรุงแฟล็กสถานะทั้งหมดจะถูกส่งไปยังรีจิสเตอร์สถานะ (SREG) และถูกนำไปปรับค่าในรีจิสเตอร์เฉพาะแฟล็กที่สอดคล้องกับการดำเนินการของคำสั่งเท่านั้น

ตารางที่ 3.14 สัญญาณปรับปรุงแฟล็กสถานะ

แฟล็กสถานะ	สัญญาณ	ข้อมูลที่นำมาถอดรหัสและเงื่อนไขที่ตรวจสอบ
ตัวทด	c_update	c_update <- arith_out [16]
นิเสธ	n_update	n_update <- RREG [16]
ศูนย์	z_update	z_update <- ‘1’ if RREG [15:0]=0 else ‘0’
เครื่องหมาย	s_update	s_update <- RREG [15]

3.4.5 <u>หน่วยคูณและสะสม</u>

หน่วยคูณและสะสมเป็นวงจรย่อยที่มีหน้าที่ดำเนินการคำสั่งที่เกี่ยวข้องกับการประมวลผลสัญญาณดิจิทัล ซึ่งได้แก่ การคูณ การคูณและสะสม และการคูณและลด หน่วยคูณและสะสมสามารถดำเนินการข้อมูลได้ทั้งแบบที่มีเครื่องหมาย และไม่มีเครื่องหมาย โครงสร้างของหน่วยคูณและสะสม (ดูรูปที่ 3.18) ประกอบด้วย วงจรคูณแบบมีเครื่องหมาย (Signed Multiplier) ขนาด 17 คูณ 17 บิต ตัวสะสม (Accumulator) ขนาด 33 บิต และส่วนตรวจจับการล้นเกิน (Overflow Detector) หน่วยคูณและสะสมรับอินพุตเป็นข้อมูลแบบมีเครื่องหมายขนาด 17 บิตจากหน่วยเลือกตัวถูกดำเนินการ ซึ่งทำหน้าที่ขยายข้อมูลจาก 16 บิตให้กลายเป็น 17 บิต ตามลักษณะของการดำเนินการของคำสั่งในกลุ่มการคูณและสะสม



รูปที่ 3.18 โครงสร้างของหน่วยคูณและสะสม

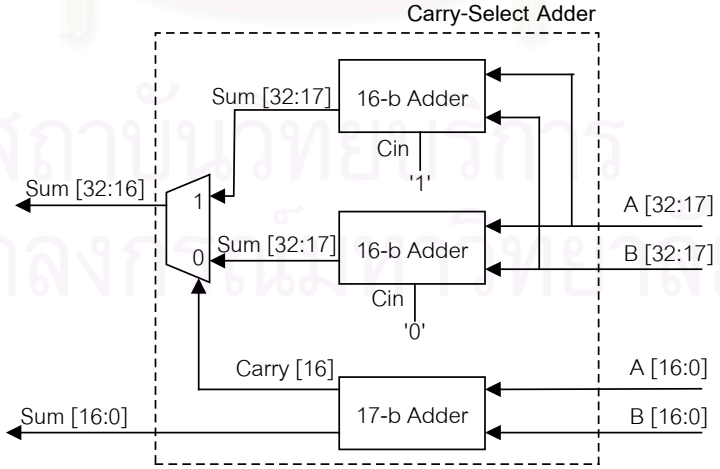
การทำงานของหน่วยคูณและสะสมในกรณีของคำสั่งการคูณ (คำสั่ง MULx) เริ่มตั้งแต่ข้อมูลอินพุตขนาด 17 บิตสองชุด (sxt_A_E1&operandA_E1 และ sxt_A_E1&operandB_E1) ถูกป้อนเข้าไปที่วงจรคูณแบบไปป์ไลน์สองขั้นตอน ผลลัพธ์ของวงจรคูณจะถูกส่งไปเก็บยังรีจิสเตอร์ผลลัพธ์การคูณ (Product Result Register) ข้อมูลของรีจิสเตอร์ผลลัพธ์การคูณจะส่งไปยังวงจรบวก 33 บิต เพื่อบวกกับข้อมูลอีกชุดหนึ่ง ซึ่งในกรณีของคำสั่งคูณจะมีค่าเป็นศูนย์ หลังจากนั้นผลลัพธ์ของการบวกจะถูกนำมาเก็บที่ตัวสะสม กล่าวคือผลลัพธ์ของการคูณจะเก็บที่ตัวสะสมนั่นเอง

การทำงานของหน่วยคูณและสะสมในกรณีของคำสั่งการคูณและสะสม (คำสั่ง MACx และ MSBx) จะเหมือนกับคำสั่งการคูณ แต่จะแตกต่างตรงที่ข้อมูลของรีจิสเตอร์ผลลัพธ์การคูณจะถูกนำมาบวกหรือลบกับค่าของตัวสะสมเอง แล้วจึงนำผลลัพธ์ไปเก็บที่ตัวสะสม ส่วนการทำงานในกรณีของคำสั่งการคูณและบวก (คำสั่ง MADD) ข้อมูลของรีจิสเตอร์ผลลัพธ์การคูณจะถูกนำมาบวกกับค่าของสัญญาณ xdata_E1 ซึ่งเป็นข้อมูลจากหน่วยความจำ X ขนาด 16 บิตที่ถูกนำมา

ขยายแบบมีเครื่องหมายและถูกเลื่อนไปทางซ้าย 17 บิต จนกลายเป็นข้อมูลขนาด 33 บิต แล้วจึงนำผลลัพธ์ไปเก็บที่ตัวสะสมเช่นเดียวกัน

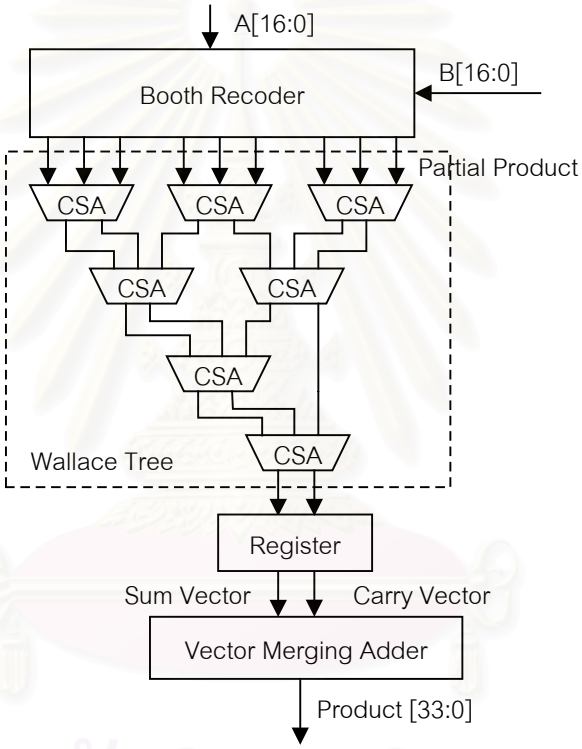
ส่วนตรวจจับการล้นเกิน มีหน้าที่ตรวจสอบเงื่อนไขของการเกิดการล้นเกินและปรับค่าในตัวสะสม เพื่อให้อิ่มตัวในกรณีที่เกิดการล้นเกิดขึ้น ส่วนตรวจจับการล้นเกินจะทำงานเมื่อฟิลด์ saten ซึ่งอยู่ในรีจิสเตอร์ใช้งานพิเศษ MCTRL (ดูรายละเอียดในหัวข้อที่ 4.1.2) มีค่าเป็น 1 การล้นเกินของตัวสะสมแบ่งได้เป็นสองแบบ ได้แก่ การล้นเกินด้านบวก (Positive Overflow) และการล้นเกินด้านลบ (Negative Overflow) โดยการตรวจสอบการเกิดการล้นเกิน จะพิจารณาข้อมูลบิตที่ 32 และ 31 ของตัวสะสม เมื่อข้อมูลบิตที่ 32 และ 31 ของตัวสะสมมีค่าเป็น 01 แสดงว่าเกิดการล้นเกินด้านบวก ส่วนตรวจจับการล้นเกินจะป้อนค่าคงที่ขนาด 33 บิต 001 … 11 ไปแทนที่ค่าที่อยู่ในตัวสะสม และเมื่อข้อมูลบิตที่ 32 และ 31 ของตัวสะสมมีค่าเป็น 01 แสดงว่าเกิดการล้นเกินด้านบวก ส่วนตรวจจับการล้นเกินจะป้อนค่าคงที่ขนาด 33 บิต 110 … 00 ไปแทนที่ค่าที่อยู่ในตัวสะสม

หน่วยคูณและสะสมเป็นวงจรย่อยที่มีความล่าช้ามากและถือเป็นเส้นทางวิกฤตของหน่วยประมวลผล ดังนั้นการออกแบบหน่วยคูณและสะสมให้สามารถทำงานได้เร็วจะส่งผลให้หน่วยประมวลผลสามารถทำงานได้ความถี่สูงขึ้น องค์ประกอบหลักของหน่วยคูณและสะสม ได้แก่ วงจรคูณ และวงจรบวก ในที่นี้การออกแบบจะเลือกใช้โครงสร้างของวงจรคูณชนิดมีเครื่องหมายแบบบู๊ธ (Booth Multiplier) และเลือกใช้โครงสร้างของวงจรบวกแบบเลือกตัวทด [11] (ดูรูปที่ 3.19) (Carry-Select Adder)

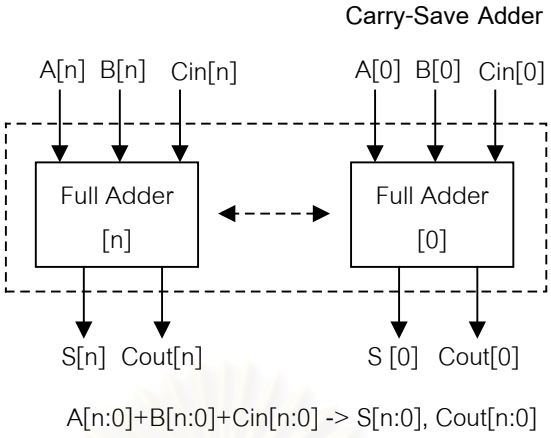


รูปที่ 3.19 วงจรบวกแบบเลือกตัวทด

วงจรคูณแบบบู๊ธ (ดูรูปที่ 3.20) เป็นการประยุกต์ใช้อัลกอริทึมของบู๊ธในการเข้ารหัสซ้ำเพื่อลดจำนวนของผลคูณบางส่วน (Partial Product) ส่วนการรวมผลคูณบางส่วนที่ได้จากกการเข้ารหัสด้วยอัลกอริทึมของบู๊ธจะใช้โครงสร้างแบบวอลเลซทรี (Walace Tree Structure) ซึ่งประกอบด้วยวงจรบวกแบบยกเว้นตัวทด (ดูรูปที่ 3.21) (Carry-Save Adder : CSA) ต่อเรียงกัน แล้วจึงนำผลลัพธ์จากวอลเลซทรี ซึ่งได้แก่ เวคเตอร์ผลรวม (Sum Vector) และเวคเตอร์ตัวทด (Carry Vector) มาเก็บไว้ในรีจิสเตอร์ก่อนที่จะนำมารวมกันด้วย วงจรบวกในขั้นตอนสุดท้ายอีกที การใช้รีจิสเตอร์มาแบ่งการทำงานของวงจรคูณให้มีโครงสร้างแบบไปป์ไลน์จะช่วยแบ่งเวลาล่าช้าของวงจรให้เป็นหลายขั้นตอน แต่ว่าจะทำให้ผลลัพธ์ของวงจรคูณล่าช้าไปหนึ่งรอบสัญญาณนาฬิกา



รูปที่ 3.20 วงจรคูณแบบบู๊ธ



รูปที่ 3.21 วงจรบวกแบบยกเว้นตัวทด

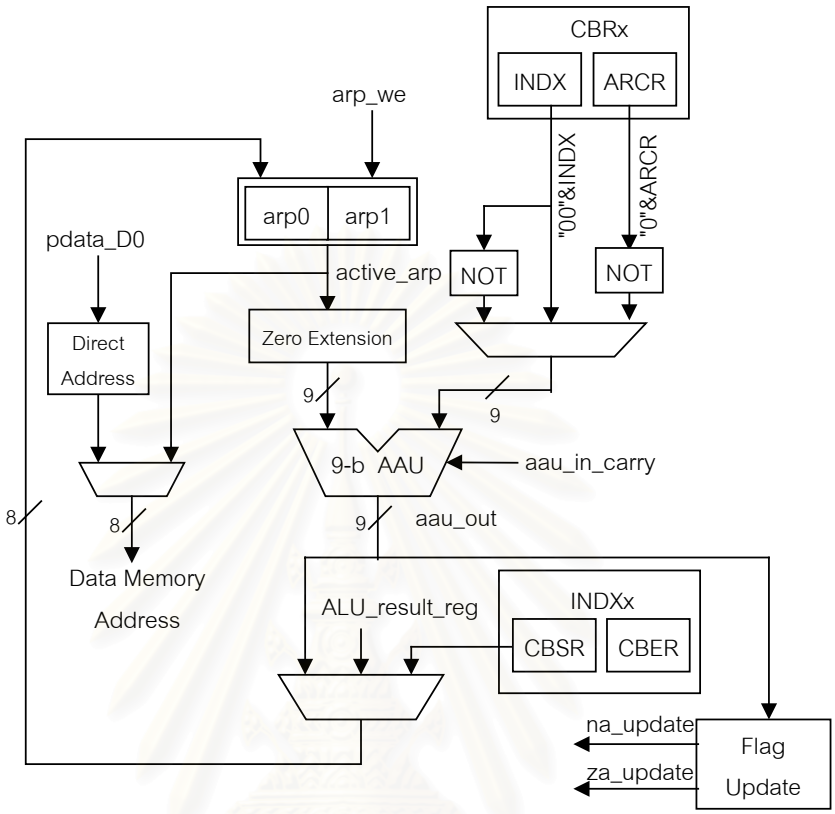
3.4.6 <u>หน่วยกำเนิดตำแหน่งข้อมูล</u>

หน่วยกำเนิดตำแหน่งข้อมูลมีหน้าที่เก็บและปรับปรุงตัวชี้ข้อมูลของหน่วยความจำ โดยสามารถกำหนดการปรับปรุงของตัวชี้ข้อมูลได้ 3 รูปแบบ เช่น การเพิ่มตัวชี้ด้วยค่าดัชนี (Index) การลดตัวชี้ด้วยค่าดัชนี และการเปรียบเทียบตัวชี้กับค่าคงที่ เป็นต้น นอกจากนี้สามารถให้ตัวชี้ทำงานในลักษณะของ Circular ได้ เนื่องจากหน่วยประมวลผลกลางมีหน่วยความจำข้อมูลภายในสองชุด ดังนั้นจึงมีหน่วยกำเนิดตำแหน่งข้อมูลสองชุดที่มีโครงสร้างคล้ายคลึงกันได้แก่ หน่วยกำเนิดตำแหน่งข้อมูล X (AGUX) และหน่วยกำเนิดตำแหน่งข้อมูล Y (AGUY) สำหรับอ้างตำแหน่งของหน่วยความจำ โดยแต่ละชุดจะมีตัวชี้ข้อมูลแยกออกจากกัน และสามารถทำงานขนานอย่างอิสระจากการทำงานของหน่วยกำเนิดข้อมูลอีกชุดหนึ่ง

โครงสร้างของหน่วยกำเนิดตำแหน่งข้อมูล (ดูรูปที่ 3.22) ประกอบด้วย รีจิสเตอร์สำหรับเก็บตัวชี้ข้อมูล ซึ่งได้แก่ รีจิสเตอร์ arp0 และ arp1 หน่วยคำนวณตำแหน่งข้อมูล (Address Arithmetic Unit : AAU) และรีจิสเตอร์ใช้งานพิเศษ CBRx และ INDXx (ดูรายละเอียดในหัวข้อ 4.1.13 และ 4.1.14) โครงสร้างของหน่วยกำเนิดตำแหน่งข้อมูลของหน่วยความจำทั้งสองชุดมีความคล้ายคลึงกัน โดยจะแตกต่างกันตรงที่สัญญาณควบคุมที่มาจากคนละแหล่ง และทำงานกับคำสั่งเกี่ยวข้องกับหน่วยความจำคนละชุด

การอ้างตำแหน่งของหน่วยความจำข้อมูลสามารถกระทำได้สองรูปแบบ ได้แก่ การอ้างตำแหน่งด้วยตัวชี้ข้อมูลสำหรับแอดเดรสซิงโมดแบบโดยอ้อม และการอ้างตำแหน่งด้วยตำแหน่ง

โดยตรง (Direct Address) ซึ่งมาจากข้อมูลของหน่วยความจำโปรแกรม (สัญญาณ pdata_D0) สำหรับคำสั่งที่มีแอดเดรสซิงโมดแบบโดยตรง



รูปที่ 3.22 โครงสร้างของหน่วยกำเนิดข้อมูล

หน่วยกำเนิดตำแหน่งข้อมูลจะถูกควบคุมด้วยสัญญาณจากตัวถอดรหัสคำสั่งและทำงานในขั้นตอนเข้าถึงหน่วยความจำ (ขั้นตอน M0) สัญญาณควบคุม xop_M0 (หรือ yop_M0) ทำหน้าที่กำหนดการดำเนินการของตัวชี้ข้อมูล ส่วนสัญญาณ xsel_M0 (หรือ ysel_M0) จะถูกนำมาเลือกตัวชี้ข้อมูลว่าจะเป็น arp0 หรือ arp1 สำหรับนำมาอ้างตำแหน่งแบบโดยอ้อมและปรับปรุงค่าของตัวชี้ข้อมูลหลังจากที่ได้อ้างตำแหน่งแล้ว ส่วนการอ้างตำแหน่งข้อมูลแบบโดยตรงจะใช้ข้อมูลตำแหน่งโดยตรงจากสัญญาณ pdata_D0 ที่มาจากตัวถอดรหัสคำสั่ง

การปรับปรุงตัวชี้ข้อมูลจะถูกดำเนินการด้วยหน่วยคำนวณตำแหน่งข้อมูล ซึ่งเป็นวงจรบวก-ลบขนาด 9 บิต ที่มีข้อมูลอินพุตมาจากสองแหล่งได้แก่ ตัวชี้ข้อมูล และค่าที่ใช้ปรับปรุงตัวชี้ข้อมูล ซึ่งขึ้นอยู่กับการดำเนินการของตัวชี้ข้อมูล ค่าที่ใช้ปรับปรุงตัวชี้ข้อมูลสำหรับการดำเนินการแบบต่างๆ แสดงในตารางที่ 3.15

ตารางที่ 3.15 ค่าที่ใช้ปรับปรุงตัวชี้ข้อมูลสำหรับการดำเนินการแบบต่างๆ

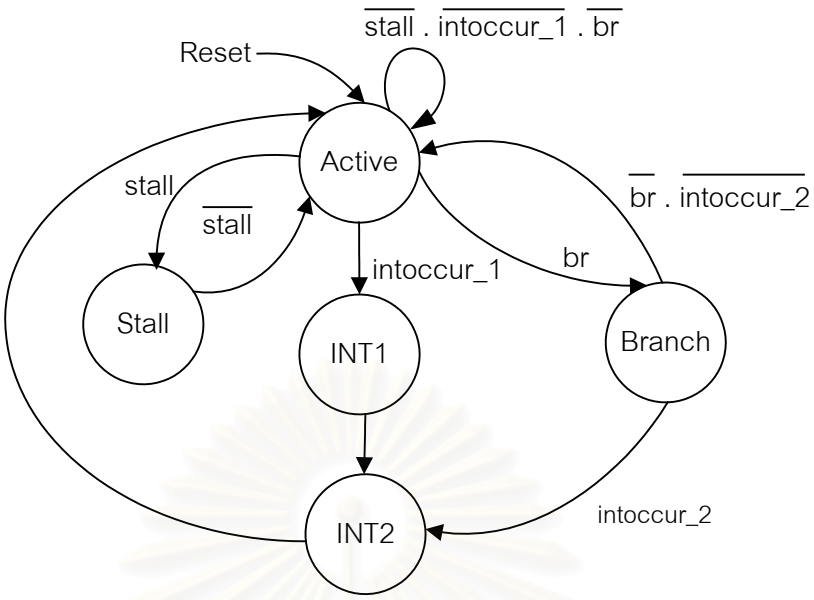
การดำเนินการของตัวชี้	xop_M0	ค่าที่ใช้ปรับปรุงตัวชี้ข้อมูล (9 บิต)	arp_we
ไม่เปลี่ยนแปลง	00	'1'& NOT (ARCR [7:0])	0
เพิ่มค่าด้วย INDX	01	"00"& INDX [6:0]	1
ลดค่าด้วย INDX	10	"11"& NOT (INDX [6:0])	1
เปรียบเทียบกับ ARCR	11	'1'& NOT (ARCR [7:0])	0

ผลลัพธ์ของหน่วยคำนวณตำแหน่งข้อมูลจะถูกนำไปปรับปรุงค่าของตัวชี้ข้อมูล โดยการปรับปรุงค่าของตัวชี้ข้อมูลจะถูกควบคุมด้วยสัญญาณ arp_we โดยค่าของตัวชี้ข้อมูลจะถูกปรับปรุงเมื่อสัญญาณ arp_we มีค่าเป็น 1 ในกรณีที่หน่วยกำเนิดตำแหน่งข้อมูลทำงานแบบ Circular กล่าวคือ เมื่อบิต CBEN (Circular Buffer Enable) ซึ่งอยู่ในรีจิสเตอร์ใช้งานพิเศษ INDX (ดูรายละเอียดในหัวข้อที่ 4.1.4) มีค่าเป็น 1 ผลลัพธ์จากหน่วยคำนวณตำแหน่งข้อมูล (aau_out [7:0]) จะถูกนำมาเปรียบเทียบกับค่าในรีจิสเตอร์ CBER (Circular Buffer End Register) ซึ่งถ้าค่าทั้งสองเท่ากัน ค่าของรีจิสเตอร์ CBSR (Circular Buffer Start Register) จะถูกนำไปเขียนลงในตัวชี้ข้อมูล นอกจากนี้เมื่อหน่วยประมวลผลดำเนินการคำสั่ง STARP จะสามารถเขียนข้อมูลจากแฟ้มรีจิสเตอร์ลงในตัวชี้ข้อมูลได้โดยตรง

ส่วนปรับปรุงแฟล็กสถานะของหน่วยกำเนิดตำแหน่งข้อมูลจะทำหน้าที่ถอดรหัสสัญญาณผลลัพธ์จากหน่วยคำนวณตำแหน่งข้อมูลในกรณีที่ดำเนินการเปรียบเทียบตัวชี้ข้อมูลกับรีจิสเตอร์ ARCR โดยจะตรวจสอบว่าค่าของตัวชี้ข้อมูลมีค่าน้อยกว่าหรือเท่ากับค่าในรีจิสเตอร์ ARCR เพื่อที่จะนำไปถอดรหัสเป็นสัญญาณปรับปรุงแฟล็กนิเสธ และแฟล็กศูนย์ของตัวชี้ตำแหน่ง (สัญญาณ na_update และ za_update) ตามลำดับ

3.5 สถานะการทำงานของหน่วยประมวลผลกลาง

การทำงานของหน่วยประมวลผลกลางสามารถอธิบายในรูปของเครื่องจักรสถานะจำกัด (Finite State Machine) ได้ดังรูปที่ 3.23 วงจรเครื่องจักรสถานะจำกัดประกอบขึ้นจากวงจรหลายส่วนทำงานประสานกัน โดยวงจรแต่ละส่วนจะรับสัญญาณควบคุมที่ถูกถอดรหัสจากสถานะการทำงานของเครื่องจักร และสัญญาณที่มาจากวงจรย่อยของหน่วยประมวลผล เช่น หน่วยควบคุมโปรแกรม เป็นต้น



รูปที่ 3.23 แผนผังสถานะการทำงานของหน่วยประมวลผลกลาง

สถานะการทำงานของหน่วยประมวลผลกลางแบ่งได้เป็น 5 สถานะ ได้แก่ สถานะ Active สถานะ Stall สถานะ Branch สถานะ INT1 และสถานะ INT2 สถานะ Active เป็นสถานะการทำงานปรกติของหน่วยประมวลผล ซึ่งจะเฟ็ทช์คำสั่งมาดำเนินการอย่างต่อเนื่อง การทำงานของหน่วยประมวลผลจะเข้าสู่สถานะ Stall เมื่อเกิดเงื่อนไขของอันตรายเชิงโครงสร้างขึ้น (สัญญาณ stall มีค่าเท่ากับ 1) ในสถานะนี้หน่วยประมวลผลจะหยุดอ่านคำสั่งจากภายนอกจนกระทั่งการทำงานไม่เกิดอันตราย เมื่อเกิดการกระโดดของโปรแกรมขึ้น (สัญญาณ br มีค่าเท่ากับ 1) การทำงานจะเข้าสู่สถานะ Branch เพื่อปรับปรุงค่าของตำแหน่งโปรแกรม เมื่อเกิดการขัดจังหวะหน่วยประมวลผลขึ้น (สัญญาณ intoccur_1 มีค่าเท่ากับ 1) การทำงานจะเข้าสู่สถานะ INT1 และ INT2 ตามลำดับ ในกรณีที่เกิดการขัดจังหวะขณะกำลังเกิดการกระโดดของโปรแกรมขึ้น (สัญญาณ intoccur_2 มีค่าเท่ากับ 1) หน่วยประมวลผลจะรอให้ปรับปรุงค่าตำแหน่งของโปรแกรมให้เสร็จก่อนแล้วจึงเข้าสู่สถานะ INT2

สัญญาณ Reset เป็นสัญญาณภายนอกที่มีลำดับความสำคัญสูงสุด เมื่อสัญญาณ Reset ถูกกระตุ้น การทำงานของหน่วยประมวลผลจะเริ่มต้นที่สถานะ Active

สัญญาณ intoccur_1 เป็นสัญญาณที่ถูกกระตุ้นเมื่อเกิดการขัดจังหวะหน่วยประมวลผลขึ้น ทำให้การทำงานเข้าสู่สถานะ INT1 โดยในรอบการทำงานถัดมาสัญญาณ intoccur_2 จะถูกกระตุ้นต่อจากสัญญาณ intoccur_1 เพื่อให้การทำงานเข้าสู่สถานะ INT2 ต่อไป

สัญญาณ br เป็นสัญญาณที่มีความสำคัญรองลงมาจากสัญญาณ Reset โดยสัญญาณ br จะถูกสร้างจากหน่วยควบคุมโปรแกรมที่มีหน้าที่ตรวจสอบเงื่อนไขของการกระโดดของโปรแกรม ในกรณีที่สัญญาณ br ถูกกระตุ้นพร้อมกับสัญญาณ intoccur_1 การทำงานจะเข้าสู่สถานะ Branch ก่อนแล้วจึงเข้าสู่สถานะ INT2 ในรอบการทำงานถัดมา

สัญญาณ stall เป็นสัญญาณที่มีลำดับความสำคัญต่ำสุด โดยจะถูกกระตุ้นเมื่อต้องการหยุดการทำงานของหน่วยประมวลผลชั่วคราว เงื่อนไขของการหยุดการทำงานของหน่วยประมวลผล ได้แก่ กรณีที่หน่วยประมวลผลดำเนินการคำสั่งกระโดด การทำงานจะเข้าสู่สถานะ Stall หนึ่งรอบสัญญาณนาฬิกา สำหรับการตรวจสอบเงื่อนไขของการกระโดดว่าเป็นจริงหรือไม่ โดยถ้าเงื่อนไขเป็นจริง จะกระตุ้นให้สัญญาณ br มีค่าเป็น 1 ในรอบสัญญาณนาฬิกาถัดมา

3.6 สรุปท้ายบท

ในบทนี้ได้กล่าวถึงรายละเอียดการออกแบบหน่วยประมวลผลกลาง ซึ่งเป็นส่วนประกอบหลักของ D-CHIP ได้แก่ การออกแบบชุดคำสั่ง โครงสร้างไปป์ไลน์ของหน่วยประมวลผลกลาง สถาปัตยกรรมของหน่วยประมวลผลกลาง และสถานะการทำงานของหน่วยประมวลผลกลาง สำหรับรายละเอียดของส่วนประกอบอื่นๆ ของ D-CHIP ซึ่งได้แก่ ตัวกรองเอฟไอาร์ และอุปกรณ์บริวาร จะได้กล่าวถึงในบทที่ 4

บทที่ 4

รีจิสเตอร์ใช้งานพิเศษและอุปกรณ์บริวาร

4.1 รีจิสเตอร์ใช้งานพิเศษ

รีจิสเตอร์ใช้งานพิเศษมีหน้าที่ควบคุมการทำงานของวงจรย่อยและอุปกรณ์บริวารต่างๆ การเข้าถึงข้อมูลในรีจิสเตอร์ใช้งานพิเศษจะกระทำผ่านคำสั่ง LDIO สำหรับการอ่านข้อมูล และคำสั่ง STIO สำหรับการเขียนข้อมูล รีจิสเตอร์ใช้งานพิเศษมีทั้งหมด 16 ตำแหน่ง โดยแต่ละตำแหน่งของรีจิสเตอร์ใช้งานพิเศษจะมีการอ่านและเขียนข้อมูลแตกต่างกัน โดยข้อมูลบางตำแหน่งจะสามารถเขียนข้อมูลได้เพียงอย่างเดียว หรือข้อมูลบางตำแหน่ง การอ่านและเขียนข้อมูลจะกระทำบนรีจิสเตอร์คนละตัว ตำแหน่งของรีจิสเตอร์ใช้งานพิเศษแสดงในตารางที่ 4.1

ตารางที่ 4.1 ตำแหน่งของรีจิสเตอร์ใช้งานพิเศษ

ตำแหน่ง	อ่าน	เขียน	หมายเหตุ
0000b	SREG	SREG	-
0001b	MCTRL	MCTRL	-
0010b	CBRX	ACCH	-
0011b	CBRY	ACCL	-
0100b	INDXX	FACCH	-
0101b	INDXY	FACCL	-
0110b	LDATAI	LDATAO	-
0111b	RDATAI	RDATAO	-
1000b	PININ	PINOUT	-
1001b	TCNT	TCNT	-
1010b	GIFR	GIFR	-
1011b	PC	-	เขียนเท่านั้น
1100b	PDAR	-	เขียนเท่านั้น
1101b	DMAR	-	เขียนเท่านั้น
1110b	PINDIR	-	เขียนเท่านั้น
1111b	MCON	-	เขียนเท่านั้น

4.1.1 <u>รีจิสเตอร์สถานะ</u>

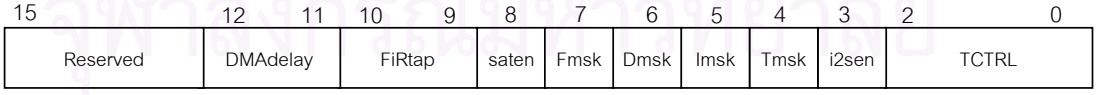
รีจิสเตอร์สถานะ (Status Register :SREG) (ดูรูปที่ 4.1) มีหน้าที่เก็บข้อมูลของแฟล็กสถานะต่างๆ ซึ่งได้แก่ แฟล็กตัวทด (C) แฟล็กนิเสธ (N) แฟล็กเครื่องหมาย (S) แฟล็กศูนย์ (Z) แฟล็กนิเสธของตัวชี้ตำแหน่งหน่วยความจำ X (NAX) แฟล็กศูนย์ของตัวชี้ตำแหน่งหน่วยความจำ X (ZAX) แฟล็กนิเสธของตัวชี้ตำแหน่งหน่วยความจำ Y (NAY) แฟล็กศูนย์ของตัวชี้ตำแหน่งหน่วยความจำ Y (ZAY) และแฟล็กขัดจังหวะ (I) โดยการเปลี่ยนแปลงค่าภายในรีจิสเตอร์สถานะจะขึ้นอยู่กับคำสั่งที่ใช้ดำเนินการ (ดูรายละเอียดในตารางที่ 3.2 ถึง 3.5)

รูปที่ 4.1 ข้อมูลภายในรีจิสเตอร์สถานะ

เมื่อเกิดการขัดจังหวะหน่วยประมวลผล ข้อมูลในรีจิสเตอร์สถานะจะถูกเก็บไว้ในรีจิสเตอร์ชั่วคราว และแฟล็กขัดจังหวะจะถูกเซ็ตให้มีค่าเป็น 1 ซึ่งจะส่งผลให้โปรแกรมกระโดดไปทำงานที่โปรแกรมบริการการขัดจังหวะ (Interrupt Service Routine) และเมื่อการทำงานในโปรแกรมบริการการขัดจังหวะสิ้นสุด กล่าวคือเมื่อหน่วยประมวลผลดำเนินการคำสั่ง RETI ข้อมูลของรีจิสเตอร์สถานะที่ถูกเก็บไว้ในรีจิสเตอร์ชั่วคราวจะถูกโหลดกลับมาในรีจิสเตอร์สถานะ และแฟล็กขัดจังหวะจะถูกเคลียร์ให้มีค่าเป็น 0 ด้วยฮาร์ดแวร์

4.1.2 <u>รีจิสเตอร์ควบคุมหลัก</u>

รีจิสเตอร์ควบคุมหลัก (Main Control Register :MCTRL) มีหน้าที่ควบคุมการทำงานของวงจรย่อย และอุปกรณ์บริวารต่างๆ ภายในหน่วยประมวลผล ข้อมูลในรีจิสเตอร์ควบคุมหลักแบ่งออกเป็นหลายฟิลด์ (ดูรูปที่ 4.2) ได้แก่



รูปที่ 4.2 ข้อมูลภายในรีจิสเตอร์ควบคุมหลัก

ฟิลด์ควบคุมตัวตั้งเวลา หรือ TCTRL (Timer Control) ในบิตที่ 0 ถึง 2 ทำหน้าที่ควบคุมการเริ่มทำงานของตัวตั้งเวลา รวมไปถึงการกำหนดการสเกลความถี่สัญญาณนาฬิกาที่ป้อนเข้ามายังตัวตั้งเวลา (ดูรายละเอียดในหัวข้อ 4.4)

ฟิลด์กำหนดการเริ่มทำงานของวงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S หรือ I2Sen (I^2S Enable) ในบิตที่ 3 มีหน้าที่ควบคุมการป้อนสัญญาณนาฬิกาเข้าสู่วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S ซึ่งเป็นการกำหนดการเริ่มการทำงานของวงจร โดยเมื่อฟิลด์ I2Sen มีค่าเป็น 1 จะทำให้วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S เริ่มทำงาน

ฟิลด์ Tmsk (Timer Overflow Interrupt Mask) ในบิตที่ 4 มีหน้าที่ควบคุมให้สามารถเกิดการขัดจังหวะหน่วยประมวลผล เมื่อเกิดการล้นเกิดของตัวนับเวลาได้ โดยเมื่อฟิลด์ Tmsk มีค่าเป็น 1 และเกิดการล้นเกินของตัวนับเวลาขึ้น จะทำให้เกิดการขัดจังหวะเนื่องจากตัวตั้งเวลาขึ้น

ฟิลด์ Imsk (I^2S Transfer Complete Interrupt Mask) ในบิตที่ 5 มีหน้าที่ควบคุมให้สามารถเกิดการขัดจังหวะหน่วยประมวลผล เมื่อเกิดการทำงานครบรอบของวงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S โดยเมื่อฟิลด์ Imsk มีค่าเป็น 1 และวงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S ทำงานครบรอบจะทำให้เกิดการขัดจังหวะเนื่องจากการเสร็จสิ้นการรับส่งข้อมูลกับอุปกรณ์ I^2S ขึ้น

ฟิลด์ Dmsk (DMA Transfer Complete Interrupt Mask) ในบิตที่ 6 มีหน้าที่ควบคุมให้สามารถเกิดการขัดจังหวะหน่วยประมวลผล เมื่อดีเอ็มเอโอนย้ายข้อมูลครบตามที่กำหนด โดยเมื่อฟิลด์ Dmsk มีค่าเป็น 1 และดีเอ็มเอโอนย้ายข้อมูลครบตามที่กำหนด จะทำให้เกิดการขัดจังหวะเนื่องจากดีเอ็มเอขึ้น

ฟิลด์ Fmsk (Filter Complete Interrupt Mask) ในบิตที่ 7 มีหน้าที่ควบคุมให้สามารถเกิดการขัดจังหวะหน่วยประมวลผล เมื่อตัวกรองเอฟไอออาร์คำนวณผลลัพธ์เสร็จ โดยเมื่อฟิลด์ Fmsk มีค่าเป็น 1 และตัวกรองเอฟไอออาร์คำนวณผลลัพธ์เสร็จ จะทำให้เกิดการขัดจังหวะเนื่องจากตัวกรองเอฟไอออาร์ขึ้น

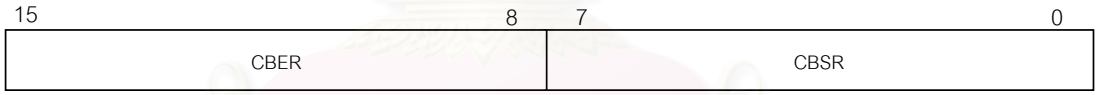
ฟิลด์ saten (Saturation Enable) ในบิตที่ 8 มีหน้าที่ควบคุมให้สามารถเกิดการอิ่มตัวของตัวสะสมของหน่วยคูณและสะสมและตัวกรองเอฟไอออาร์ เมื่อตัวสะสมเกิดการล้นเกินขึ้น โดยเมื่อฟิลด์ saten มีค่าเป็น 1 จะทำให้วงจรตรวจสอบการล้นเกินของตัวสะสมของหน่วยประมวลผลกลาง และตัวกรองเอฟไอออาร์ทำงาน

ฟิลด์ FIRtap (FIR Length Selection) ในบิตที่ 9 ถึง 10 มีหน้าที่กำหนดความยาวของตัวกรองเอฟไอออาร์ โดยสามารถปรับความยาวของตัวกรองเป็น 32 64 128 หรือ 256 แท็ป เมื่อกำหนดให้ฟิลด์ FIRtap มีค่า 00 01 10 หรือ 11 ตามลำดับ

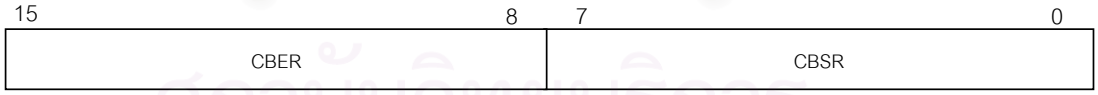
ฟิลด์ DMAdelay (DMA Delay Control) ในบิตที่ 11 ถึง 12 มีหน้าที่กำหนดค่าความหน่วงเวลาของการโอนย้ายข้อมูลระหว่างหน่วยความจำภายในและภายนอกด้วยดีเอ็มเอ เนื่องจากหน่วยความจำภายนอกอาจทำงานช้ากว่าหน่วยประมวลผลกลาง โดยค่าของฟิลด์ DMAdelay จะเป็นตัวกำหนดจำนวนวงรอบของสัญญาณนาฬิกาในการโอนย้ายข้อมูลแต่ละชุด โดยสามารถกำหนดความหน่วงเวลาของการโอนย้ายข้อมูลให้เท่ากับ 1 2 3 หรือ 4 รอบสัญญาณนาฬิกา เมื่อฟิลด์ DMAdelay มีค่า 00 01 10 หรือ 11 ตามลำดับ

4.1.3 <u>รีจิสเตอร์ควบคุมการอ้างตำแหน่งแบบ Circular</u>

รีจิสเตอร์ควบคุมการอ้างตำแหน่งแบบ Circular มีหน้าที่เก็บค่าตำแหน่งเริ่มต้นและสิ้นสุดของการอ้างตำแหน่งแบบ Circular โดยจะแบ่งเป็นรีจิสเตอร์สำหรับหน่วยความจำ X (CBRX) (ดูรูปที่ 4.3) และหน่วยความจำ Y (CBRY) (ดูรูปที่ 4.4) ข้อมูลในรีจิสเตอร์แบ่งออกเป็นสองฟิลด์ ได้แก่ ฟิลด์ CBSR ในบิตที่ 0 ถึง 7 ซึ่งมีหน้าที่เก็บค่าตำแหน่งเริ่มต้นของการอ้างตำแหน่งแบบ Circular Buffer และฟิลด์ CBER ในบิตที่ 8 ถึง 15 ซึ่งมีหน้าที่เก็บค่าตำแหน่งสิ้นสุดของการอ้างตำแหน่งแบบ Circular โดยการทำงานแบบ Circular จะทำการเปรียบตำแหน่งของตัวชี้ข้อมูลว่ามีค่าเท่ากับค่าใน CBER หรือไม่ ซึ่งถ้าเท่ากันค่าของตัวชี้ข้อมูลจะถูกปรับปรุงใหม่เท่ากับค่าใน CBSR



รูปที่ 4.3 ข้อมูลภายในรีจิสเตอร์ควบคุมCircular Buffer ของหน่วยความจำ X

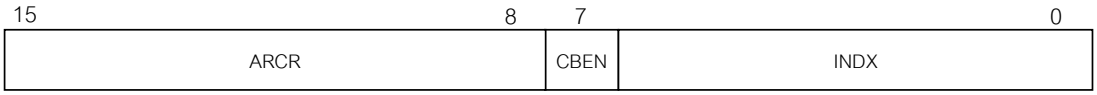


รูปที่ 4.4 ข้อมูลภายในรีจิสเตอร์ควบคุมCircular Buffer ของหน่วยความจำ Y

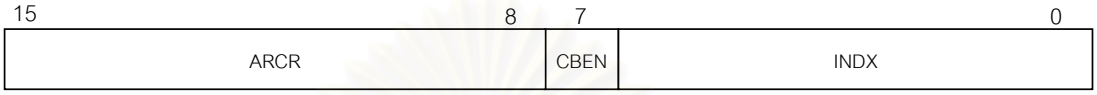
4.1.4 <u>รีจิสเตอร์ดัชนีและตัวเปรียบเทียบของตัวชี้ตำแหน่งข้อมูล</u>

รีจิสเตอร์ดัชนีและตัวเปรียบเทียบของตัวชี้ตำแหน่งข้อมูล มีหน้าที่เก็บค่าดัชนี สำหรับใช้ในการปรับปรุงค่าของตัวชี้ข้อมูล และตัวเปรียบเทียบ สำหรับใช้ในการตรวจสอบค่าของตัวชี้ข้อมูล โดยจะแบ่งเป็นรีจิสเตอร์สำหรับหน่วยความจำ X (INDXX) (ดูรูปที่ 4.5) และหน่วยความจำ Y (INDXY) (ดูรูปที่ 4.6) ข้อมูลในรีจิสเตอร์แบ่งออกเป็นสามฟิลด์ ได้แก่ ฟิลด์ INDX ในบิตที่ 0 ถึง 6 ซึ่งมีหน้าที่เก็บค่าดัชนี ฟิลด์ ARCR ในบิตที่ 8 ถึง 15 ซึ่งใช้ในการเก็บค่าของตัวเปรียบเทียบ และ

ฟิลด์ CBEN ในบิตที่ 7 ซึ่งมีหน้าที่ควบคุมการทำงานของหน่วยกำเนิดตำแหน่งข้อมูลว่าให้ทำงานแบบ Circular



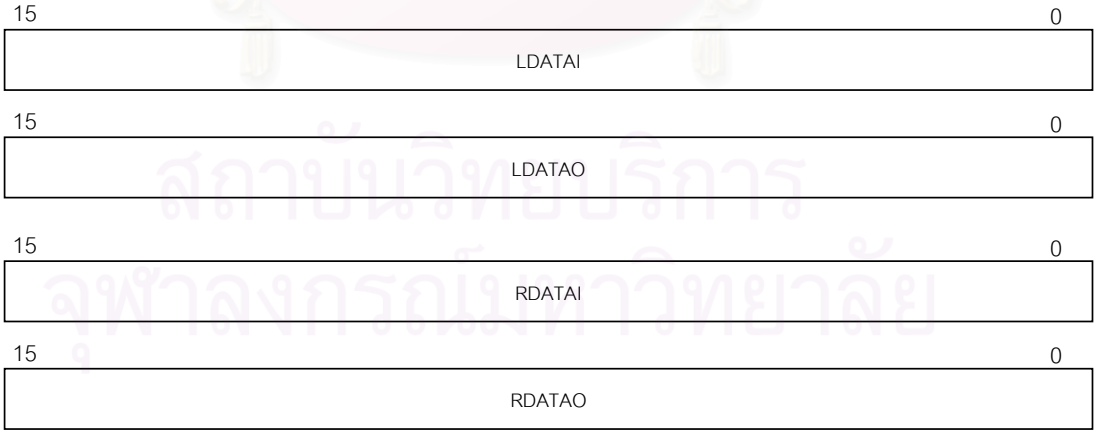
รูปที่ 4.5 ข้อมูลภายในรีจิสเตอร์ดัชนีและตัวเปรียบเทียบของหน่วยความจำ X



รูปที่ 4.6 ข้อมูลภายในรีจิสเตอร์ดัชนีและตัวเปรียบเทียบของหน่วยความจำ Y

4.1.5 <u>รีจิสเตอร์รับส่งข้อมูลของวงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S</u>

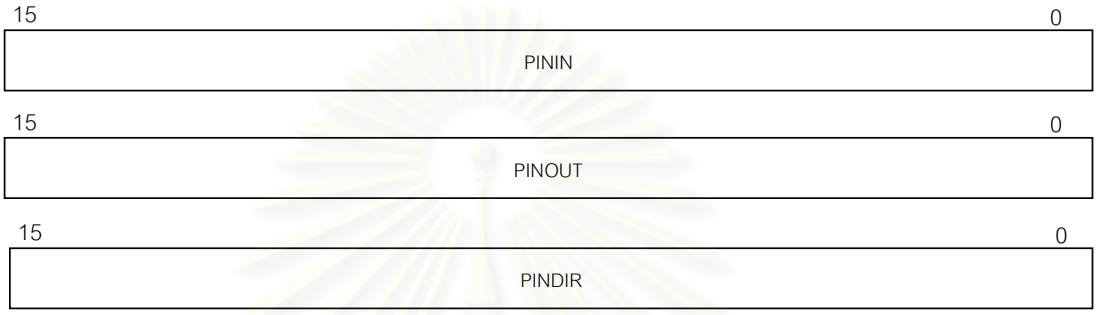
การรับส่งข้อมูลกับอุปกรณ์มาตรฐาน I^2S ซึ่งได้แก่ ตัวเข้า-ถอดรหัสสัญญาณเสียงแบบสเตอริโอ (Audio Stereo Codec) จะกระทำผ่านทางรีจิสเตอร์ใช้งานเฉพาะ 4 ตัว (ดูรูปที่ 4.7) ได้แก่ รีจิสเตอร์ LDATAI สำหรับการอ่านข้อมูลอินพุตของสัญญาณเสียงด้านซ้ายจากอุปกรณ์ I^2S รีจิสเตอร์ RDATAI สำหรับการอ่านข้อมูลอินพุตของสัญญาณเสียงด้านขวาจากอุปกรณ์ I^2S รีจิสเตอร์ LDATAO สำหรับการเขียนข้อมูลเอาต์พุตของสัญญาณเสียงด้านซ้ายออกไปยังอุปกรณ์ I^2S และรีจิสเตอร์ RDATAO สำหรับการเขียนข้อมูลเอาต์พุตของสัญญาณเสียงด้านขวาออกไปยังอุปกรณ์ I^2S



รูปที่ 4.7 ข้อมูลภายในรีจิสเตอร์รับส่งข้อมูลของวงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S

4.1.6 รีจิสเตอร์สำหรับควบคุมพอร์ตอินพุต-เอาต์พุต

การสื่อสารด้วยพอร์ตอินพุต-เอาต์พุตจะถูกควบคุมผ่านทางรีจิสเตอร์ใช้งานเฉพาะ 3 ตัว (ดูรูปที่ 4.8) ได้แก่ รีจิสเตอร์ PININ สำหรับการอ่านข้อมูลจากพอร์ตอินพุต รีจิสเตอร์ PINOUT สำหรับการเขียนข้อมูลไปยังพอร์ตเอาต์พุต และรีจิสเตอร์ PINDIR สำหรับควบคุมทิศทางของพอร์ตว่าให้ทำงานในโมดอินพุตหรือเอาต์พุต



รูปที่ 4.8 ข้อมูลภายในรีจิสเตอร์สำหรับควบคุมพอร์ตอินพุต-เอาต์พุต

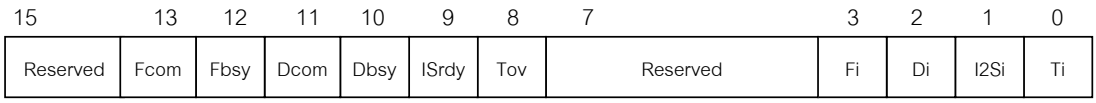
4.1.7 รีจิสเตอร์ตัวนับเวลา

รีจิสเตอร์ตัวนับเวลา (TCNT) (ดูรูปที่ 4.9) มีหน้าที่นับจำนวนของสัญญาณนาฬิกาที่ป้อนเข้ามายังตัวตั้งเวลา รีจิสเตอร์ตัวนับเวลาจะเก็บข้อมูลของตัวนับเวลาแบบนับขึ้น ซึ่งเมื่อค่าในรีจิสเตอร์ตัวนับเวลาเกิดการล้นเกิน กล่าวคือมีการเปลี่ยนแปลงจากค่า FFFFh เป็น 0000h วงจรควบคุมจะส่งสัญญาณว่ามีการล้นเกินเกิดขึ้น และทำให้ค่าของรีจิสเตอร์ตัวนับเวลามีค่าเป็นศูนย์

รูปที่ 4.9 ข้อมูลภายในรีจิสเตอร์ตัวนับเวลา

4.1.8 รีจิสเตอร์แฟล็กการขัดจังหวะรวม

รีจิสเตอร์แฟล็กการขัดจังหวะรวม หรือ GIFR (Global Interrupt Flag Register) (ดูรูปที่ 4.10) มีหน้าที่เก็บแฟล็กเงื่อนไขของการเกิดการขัดจังหวะ เช่น การล้นเกินของตัวตั้งเวลา หรือการเสร็จสิ้นการรับส่งข้อมูลกับอุปกรณ์มาตรฐาน I^2S เป็นต้น และแฟล็กการขัดจังหวะ ซึ่งจะถูกกำหนดด้วยค่าของแฟล็กเงื่อนไขของการเกิดการขัดจังหวะ และแฟล็กที่ควบคุมให้สามารถเกิดการขัดจังหวะ รีจิสเตอร์แฟล็กการขัดจังหวะรวมแบ่งออกเป็นหลายฟิลด์ ได้แก่



รูปที่ 4.10 ข้อมูลภายในรีจิสเตอร์แฟล็กการขัดจังหวะรวม

ฟิลด์การขัดจังหวะเนื่องจากตัวตั้งเวลา หรือ Ti ในบิตที่ 0 มีหน้าที่กำหนดให้เกิดการขัดจังหวะหน่วยประมวลผล เมื่อเกิดการล้นเกินของตัวตั้งเวลา โดยเมื่อ Ti มีค่าเป็น 1 จะเกิดการขัดจังหวะขึ้น และเมื่อโปรแกรมกระโดดไปยังเวคเตอร์การขัดจังหวะแล้ว Ti จะถูกเคลียร์อย่างอัตโนมัติด้วยฮาร์ดแวร์

ฟิลด์การขัดจังหวะเนื่องจากอุปกรณ์มาตรฐาน I^2S หรือ I2Si ในบิตที่ 1 มีหน้าที่กำหนดให้เกิดการขัดจังหวะหน่วยประมวลผล เมื่อเสร็จสิ้นการรับส่งข้อมูลกับอุปกรณ์มาตรฐาน I^2S โดยเมื่อ I2Si มีค่าเป็น 1 จะเกิดการขัดจังหวะขึ้น และเมื่อโปรแกรมกระโดดไปยังเวคเตอร์การขัดจังหวะแล้ว I2Si จะถูกเคลียร์อย่างอัตโนมัติด้วยฮาร์ดแวร์

ฟิลด์การขัดจังหวะเนื่องจากดีเอ็มเอ หรือ Di ในบิตที่ 2 มีหน้าที่กำหนดให้เกิดการขัดจังหวะหน่วยประมวลผล เมื่อดีเอ็มเอเสร็จสิ้นการโอนย้ายข้อมูล โดยเมื่อ Di มีค่าเป็น 1 จะเกิดการขัดจังหวะขึ้น และเมื่อโปรแกรมกระโดดไปยังเวคเตอร์การขัดจังหวะแล้ว Di จะถูกเคลียร์อย่างอัตโนมัติด้วยฮาร์ดแวร์

ฟิลด์การขัดจังหวะเนื่องจากตัวกรองเอฟไออาร์ หรือ Fi ในบิตที่ 3 มีหน้าที่กำหนดให้เกิดการขัดจังหวะหน่วยประมวลผล เมื่อตัวกรองเอฟไออาร์คำนวณผลลัพธ์เสร็จ โดยเมื่อ Fi มีค่าเป็น 1 จะเกิดการขัดจังหวะขึ้น และเมื่อโปรแกรมกระโดดไปยังเวคเตอร์การขัดจังหวะแล้ว Fi จะถูกเคลียร์อย่างอัตโนมัติด้วยฮาร์ดแวร์

ฟิลด์การล้นเกินของตัวตั้งเวลา หรือ Tov ในบิตที่ 8 มีหน้าที่แสดงสถานะการทำงานของตัวตั้งเวลาว่าเกิดการล้นเกินขึ้น โดย Tov จะถูกเซ็ตด้วยฮาร์ดแวร์ เมื่อมีตัวตั้งเวลาเกิดการล้นเกินขึ้น และสามารถเคลียร์ได้ด้วยซอฟต์แวร์

ฟิลด์ ISrdy ในบิตที่ 9 มีหน้าที่แสดงสถานะการทำงานของวงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S ว่ารับส่งข้อมูลเสร็จแล้ว โดย ISrdy จะถูกเซ็ตด้วยฮาร์ดแวร์ เมื่อการรับส่งข้อมูลสิ้นสุด และสามารถเคลียร์ได้ด้วยซอฟต์แวร์

ฟิลด์ Dbsy ในบิตที่ 10 มีหน้าที่แสดงสถานะการทำงานของดีเอ็มเอว่ากำลังโอนย้ายข้อมูลอยู่ โดย Dbsy จะถูกเซ็ตด้วยฮาร์ดแวร์เมื่อดีเอ็มเอกำลังโอนย้ายข้อมูล และถูกเคลียร์ด้วยฮาร์ดแวร์เมื่อดีเอ็มเอโอนย้ายข้อมูลเสร็จ หรือหยุดทำงาน

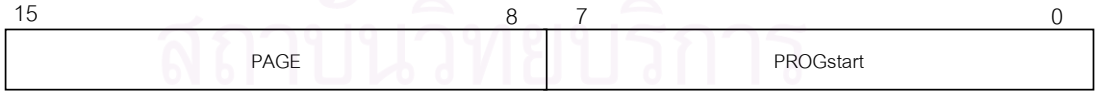
ฟิลด์ Dcom ในบิตที่ 11 มีหน้าที่แสดงสถานะการทำงานของดีเอ็มเอว่าโอนย้ายข้อมูลเสร็จแล้ว โดย Dcom จะถูกเซ็ตด้วยฮาร์ดแวร์เมื่อโอนย้ายข้อมูลเสร็จ และสามารถเคลียร์ได้ด้วยซอฟต์แวร์

ฟิลด์ Fbsy ในบิตที่ 12 มีหน้าที่แสดงสถานะการทำงานของตัวกรองเอฟไอาร์ว่ากำลังคำนวณผลลัพธ์อยู่ โดย Fbsy จะถูกเซ็ตด้วยฮาร์ดแวร์เมื่อตัวกรองเอฟไอาร์กำลังคำนวณผลลัพธ์ และถูกเคลียร์ด้วยฮาร์ดแวร์เมื่อตัวกรองเอฟไอาร์คำนวณผลลัพธ์เสร็จหรือหยุดทำงาน

ฟิลด์ Fcom ในบิตที่ 13 มีหน้าที่แสดงสถานะการทำงานของตัวกรองเอฟไอาร์ว่าคำนวณผลลัพธ์เสร็จแล้ว โดย Fcom จะถูกเซ็ตด้วยฮาร์ดแวร์เมื่อตัวกรองคำนวณผลลัพธ์เสร็จและสามารถเคลียร์ได้ด้วยซอฟต์แวร์

4.1.9 <u>รีจิสเตอร์ตำแหน่งโปรแกรมและดีเอ็มเอ</u>

รีจิสเตอร์ตำแหน่งโปรแกรมและดีเอ็มเอ หรือ PDAR (Program and DMA Address Register) (ดูรูปที่ 4.11) มีหน้าที่เก็บตัวชี้ตำแหน่งของโปรแกรมในกรณีที่อ้างถึงข้อมูลในหน่วยความจำข้อมูลด้วยคำสั่ง LPM (Load From Program Memory) และเก็บข้อมูลตำแหน่งของหน่วยความจำในกรณีที่โอนย้ายข้อมูลด้วยดีเอ็มเอ รีจิสเตอร์ตำแหน่งโปรแกรมและดีเอ็มเอแบ่งออกเป็นสองฟิลด์ ได้แก่ ฟิลด์ PROGstart และฟิลด์ PAGE



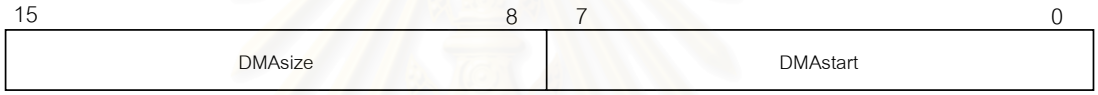
รูปที่ 4.11 ข้อมูลภายในรีจิสเตอร์ตำแหน่งโปรแกรมและดีเอ็มเอ

ฟิลด์ PROGstart ในบิตที่ 0 ถึง 7 มีหน้าที่เก็บค่าของตำแหน่งของโปรแกรมบิตที่ 0 ถึง 7 สำหรับการอ่านข้อมูลด้วยคำสั่ง LPM โดยค่าของ PROGstart จะเพิ่มขึ้นอย่างอัตโนมัติทีละหนึ่งทุกๆรอบของการดำเนินการคำสั่ง LPM

ฟิลด์ PAGE ในบิตที่ 8 ถึง 15 มีหน้าที่เก็บค่าตำแหน่งของโปรแกรมบิตที่ 8 ถึง 15 และข้อมูลในฟิลด์นี้จะถูกนำมาใช้เป็นตำแหน่งของข้อมูลบิตที่ 8 ถึง 15 ของหน่วยความจำภายนอกในกรณีที่โอนย้ายข้อมูลด้วยดีเอ็มเอ

4.1.10 รีจิสเตอร์ควบคุมดีเอ็มเอ

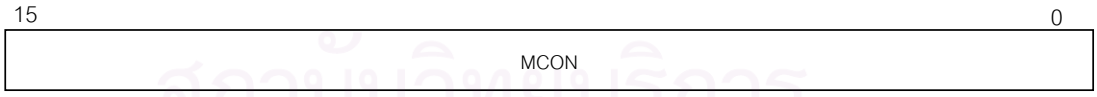
รีจิสเตอร์ควบคุมดีเอ็มเอ หรือ DMAR (DMA Control Register) (ดูรูปที่ 4.12) มีหน้าที่กำหนดตำแหน่งเริ่มต้นของข้อมูลและขนาดของกลุ่มข้อมูลที่โอนย้ายด้วยดีเอ็มเอ รีจิสเตอร์ควบคุมดีเอ็มเอแบ่งออกเป็นสองฟิลด์ ได้แก่ ฟิลด์ DMAstart ในบิตที่ 0 ถึง 7 ซึ่งมีหน้าที่เก็บค่าตำแหน่งเริ่มต้นของข้อมูลในการโอนย้ายด้วยดีเอ็มเอ และฟิลด์ DMAsize ในบิตที่ 8 ถึง 15 ซึ่งมีหน้าที่กำหนดขนาดของกลุ่มข้อมูลที่โอนย้าย



รูปที่ 4.12 ข้อมูลภายในรีจิสเตอร์ควบคุมดีเอ็มเอ

4.1.11 รีจิสเตอร์ค่าคงที่สำหรับหน่วยคูณและสะสม

รีจิสเตอร์ค่าคงที่สำหรับหน่วยคูณและสะสม หรือ MCON (ดูรูปที่ 4.13) มีหน้าที่เก็บค่าคงที่สำหรับการดำเนินการคำสั่ง MADD (Multiply and Add) โดยจะใช้ค่าจากรีจิสเตอร์ MCON เป็นตัวถูกดำเนินการร่วมกับข้อมูลจากหน่วยความจำทั้งสอง



รูปที่ 4.13 ข้อมูลภายในรีจิสเตอร์ค่าคงที่สำหรับหน่วยคูณและสะสม

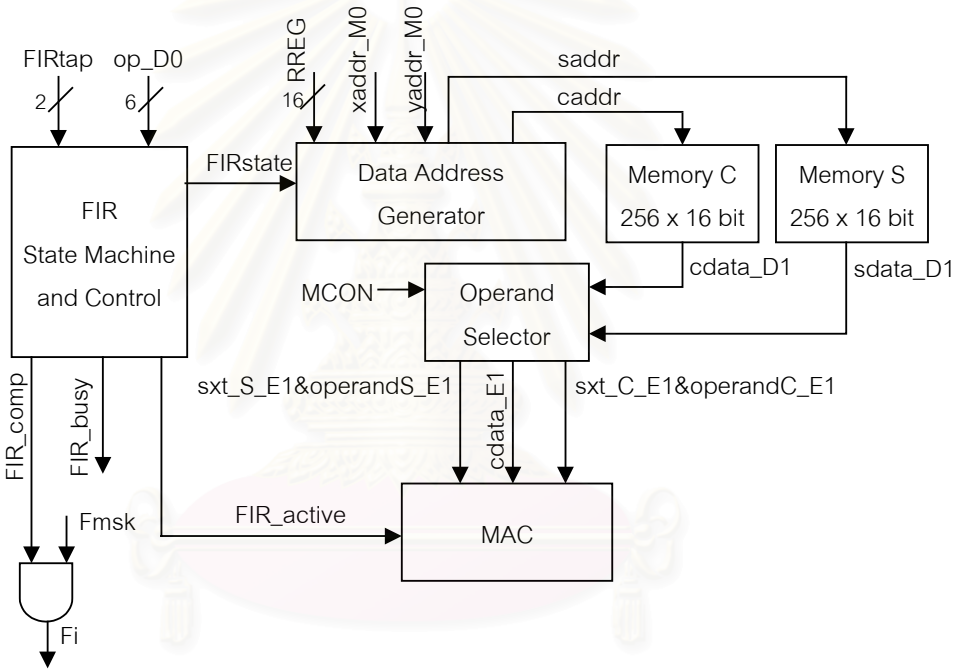
4.1.12 รีจิสเตอร์การอ่านข้อมูลตัวสะสม

รีจิสเตอร์การอ่านข้อมูลตัวสะสมมีหน้าที่ส่งผ่านค่าของตัวสะสมของหน่วยคูณและสะสมของหน่วยประมวลผลกลางและตัวกรองเอฟไอเออาร์ โดยรีจิสเตอร์ ACC จะถูกใช้สำหรับอ่านข้อมูลจากตัวสะสมของหน่วยประมวลผลกลาง ส่วนรีจิสเตอร์ FACC จะถูกใช้สำหรับอ่านข้อมูลจากตัวสะสมของตัวกรองเอฟไอเออาร์ เนื่องจากตัวสะสมมีขนาด 32 บิต ดังนั้นการเก็บข้อมูลจะแบ่งเป็น

สองส่วน ได้แก่ ข้อมูลบิตที่ 16 ถึง 31 อ่านได้จากรีจิสเตอร์ ACCH และ FACCH และข้อมูลบิตที่ 0 ถึง 15 อ่านได้จากรีจิสเตอร์ ACCL และ FACCL

4.2 ตัวกรองเอฟไออาร์

โครงสร้างของตัวกรองเอฟไออาร์ (ดูรูปที่ 4.14) ประกอบด้วย หน่วยคูณและสะสม (MAC) หน่วยความจำขนาด 16 บิต 256 ตำแหน่งจำนวนสองชุด สำหรับเก็บค่าสัมประสิทธิ์ (Memory C) และตัวอย่าง (Memory S) หน่วยเลือกตัวถูกดำเนินการ (Operand Selector) ตัวสร้างตำแหน่งข้อมูลของหน่วยความจำ (Data Address Generator) และวงจรควบคุมตัวกรอง (FIR State Machine and Control)



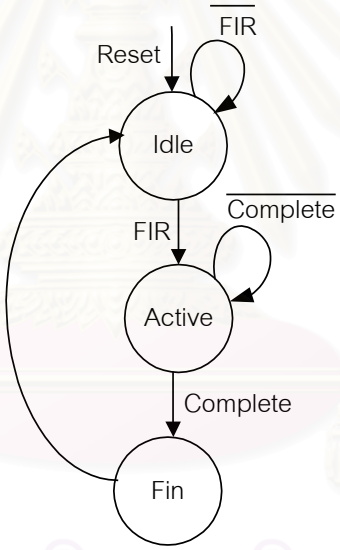
รูปที่ 4.14 โครงสร้างของตัวกรองเอฟไออาร์

หน่วยคูณและสะสมของตัวกรองมีโครงสร้างคล้ายกับหน่วยคูณและสะสมของหน่วยประมวลผลกลาง โดยประกอบไปด้วยวงจรคูณแบบบู๊ธ และวงจรบวกแบบเลือกตัวทดเช่นเดียวกับหน่วยประมวลผลกลาง นอกจากนี้ยังประกอบด้วยวงจรตรวจจับการล้นเกิน ซึ่งจะทำให้ค่าของตัวสะสมเกิดการอิ่มตัว เมื่อเกิดการล้นเกินขึ้น

หน่วยเลือกตัวถูกดำเนินการมีหน้าที่รับข้อมูลจากหน่วยความจำตัวอย่างและสัมประสิทธิ์ แล้วนำมาเลือกตัวถูกดำเนินการที่เหมาะสม รวมไปถึงการขยายจำนวนบิตข้อมูลให้มีขนาด 17 บิต ให้สอดคล้องกับการดำเนินการและชนิดของข้อมูลด้วย

ตัวสร้างตำแหน่งข้อมูลของหน่วยความจำมีหน้าที่สร้างตำแหน่งข้อมูลของภายในหน่วยความจำตัวอย่างและสัมประสิทธิ์ โดยโครงสร้างจะประกอบด้วยรีจิสเตอร์สำหรับเก็บค่าตำแหน่งข้อมูลของหน่วยความจำ ที่จะเพิ่มค่าอย่างอัตโนมัติทีละหนึ่งทุกครั้งที่ดำเนินการคำสั่ง STFC (Store in Coefficient Memory) หรือ STFS (Store in Sample Memory) นอกจากนี้การอ้างตำแหน่งข้อมูลของหน่วยความจำของตัวกรองเอฟไอาร์สามารถใช้ตำแหน่งจากหน่วยกำเนิดตำแหน่งข้อมูลของหน่วยประมวลผลกลางได้ โดยหน่วยความจำตัวอย่างจะใช้ตัวชี้ข้อมูลร่วมกับหน่วยความจำ Y ส่วนหน่วยความจำสัมประสิทธิ์จะใช้ตัวชี้ข้อมูลร่วมกับหน่วยความจำ X

วงจรควบคุมตัวกรองมีหน้าที่ควบคุมลำดับการทำงานของตัวกรอง โดยประกอบด้วยเครื่องจักรสถานะจำกัด และวงจรถอดรหัสสถานะการทำงานของตัวกรองเพื่อสร้างสัญญาณควบคุมส่งไปยังวงจรส่วนอื่นๆ เครื่องจักรสถานะจำกัดที่ออกแบบสามารถเขียนเป็นแผนผังสถานะการทำงานได้ดังรูปที่ 4.15



รูปที่ 4.15 แผนผังสถานะการทำงานของตัวกรองเอฟไอาร์

ในสถานะ Idle ตัวกรองเอฟไอาร์จะหยุดทำงาน และรอคำสั่งจากหน่วยประมวลผลเพื่อเริ่มทำงาน และขณะเดียวกันตัวกรองจะอ่านข้อมูลเช่น ความยาวของตัวกรอง จากรีจิสเตอร์ใช้งานพิเศษล่วงหน้าก่อนที่จะเริ่มทำงาน การเขียนข้อมูลลงในหน่วยความจำของตัวกรองด้วยหน่วยประมวลผลสามารถกระทำได้ในสถานะนี้ เมื่อหน่วยประมวลผลดำเนินการคำสั่ง firstart การทำงานของตัวกรองจะเข้าสู่สถานะ Active ในสถานะนี้ตัวกรองจะทำงานด้วยจำนวนรอบเท่ากับความยาวของตัวกรองที่เลือกไว้ โดยการทำงานแต่ละรอบ ตำแหน่งข้อมูลของหน่วยความจำตัวอย่างและสัมประสิทธิ์จะเพิ่มขึ้นทีละหนึ่ง เมื่อตัวกรองทำงานครบตามจำนวนรอบหรือความยาวที่

กำหนดแล้วจะเข้าสู่สถานะ Fin ซึ่งเป็นสถานะที่แสดงว่าตัวกรองทำงานเสร็จ แล้วจึงกลับสถานะ Idle

ขณะที่การทำงานของตัวกรองเอฟอาร์อยู่ในสถานะ Active จะส่งผลให้สัญญาณ FIR_busy และฟิลด์ Fbsy ของรีจิสเตอร์ใช้งานพิเศษ GIFR มีค่าเป็น 1 เพื่อแจ้งให้หน่วยประมวลผลทราบว่าตัวกรองกำลังทำงานอยู่ และขณะที่ตัวกรองได้คำนวณผลลัพธ์เสร็จแล้ว การทำงานของตัวกรองจะเข้าสู่สถานะ Fin ซึ่งจะส่งผลให้สัญญาณ FIR_comp และฟิลด์ Fcom มีค่าเป็น 1 เพื่อแจ้งให้หน่วยประมวลผลทราบว่าตัวกรองทำงานเสร็จแล้ว และถ้าฟิลด์ Fmsk ของรีจิสเตอร์ใช้งานพิเศษ MCTRL มีค่าเป็น 1 จะส่งผลให้เกิดการจังหวะหน่วยประมวลผล เนื่องจากตัวกรองเอฟไออาร์คำนวณผลลัพธ์เสร็จแล้ว

4.2.1 <u>การทำงานของตัวกรองเอฟไออาร์</u>

การใช้งานตัวกรองเอฟไออาร์ร่วมกับหน่วยประมวลผลกลางแบ่งออกได้เป็น 2 ลักษณะได้แก่ การใช้งานตัวกรองเพื่อการกรองสัญญาณแบบเอฟไออาร์ และการใช้งานตัวกรองเป็นองค์ประกอบหนึ่งของหน่วยประมวลผลกลาง

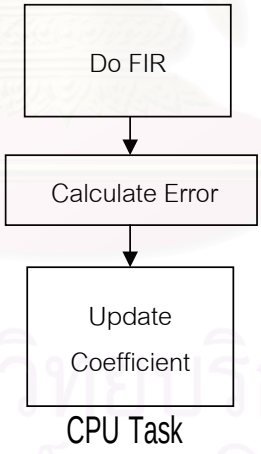
การใช้งานตัวกรองเพื่อการกรองสัญญาณแบบเอฟไออาร์ จะเป็นการใช้งานตัวกรองให้ทำงานขนานไปกับหน่วยประมวลผลกลาง โดยวงจรภายในของตัวกรองจะถูกควบคุมด้วยวงจรควบคุมของตัวกรองเอง เมื่อตัวกรองคำนวณผลลัพธ์เสร็จจะส่งสัญญาณไปขัดจังหวะหน่วยประมวลผล หรือสามารถตรวจสอบสถานะการทำงานของตัวกรองได้ที่แฟล็กสถานะการทำงาน Fbsy และ Fcom ที่รีจิสเตอร์แฟล็กการขัดจังหวะรวม GIFR

การใช้งานตัวกรองเป็นองค์ประกอบหนึ่งของหน่วยประมวลผลกลาง สามารถทำได้ในขณะที่ไม่ได้ใช้งานตัวกรองเพื่อการกรองสัญญาณแบบเอฟไออาร์ หน่วยประมวลผลจะสามารถควบคุมองค์ประกอบภายในตัวกรอง เช่น หน่วยคูณและสะสม และหน่วยความจำข้อมูล ให้ทำงานด้วยโปรแกรมของหน่วยประมวลผลโดยตรง การทำงานในลักษณะนี้ หน่วยประมวลผลจะสามารถดำเนินการข้อมูลได้พร้อมกันสองชุดด้วยคำสั่งเฉพาะเพียงคำสั่งเดียว โดยจะสามารถประมวลผลข้อมูลที่อยู่ในหน่วยความจำของตัวกรอง (Memory C และ Memory S) ควบคู่ไปกับข้อมูลที่อยู่ในหน่วยความจำของหน่วยประมวลผล (Memory X และ Memory Y) โดยใช้ทรัพยากรวงจร ได้แก่ ตัวชี้ข้อมูล ซึ่งมาจากหน่วยกำเนิดตำแหน่งข้อมูล และวงจรควบคุมร่วมกัน การทำงานในลักษณะนี้จะสอดคล้องกับแนวคิดของสถาปัตยกรรมแบบ SIMD ซึ่งจะทำให้หน่วยประมวลผลและตัวกรองสามารถทำงานร่วมกันได้อย่างอย่างประสิทธิภาพ โดยที่ใช้ทรัพยากรฮาร์ดแวร์เพิ่มเติมเพียงเล็ก

น้อย คำสั่งเฉพาะที่สามารถควบคุมตัวกรองพร้อมกับหน่วยประมวลผลแสดงในตารางที่ 4.2 โดยคำสั่งเฉพาะส่วนใหญ่จะเป็นการดำเนินการในกลุ่มการคูณและสะสม

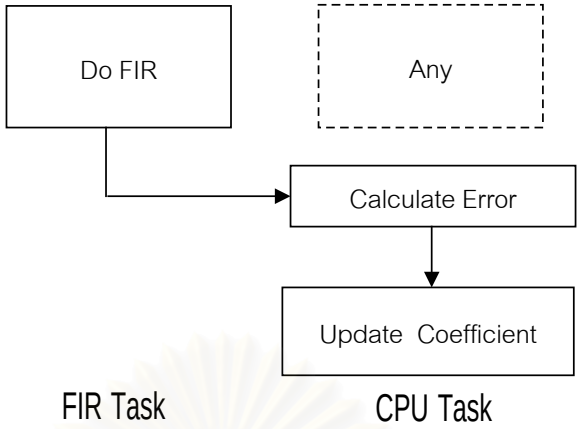
การทำงานของคำสั่งเฉพาะที่ควบคุมองค์ประกอบภายในตัวกรอง จะอาศัยการที่โครงสร้างหน่วยคูณและสะสมของหน่วยประมวลผลกลางมีความคล้ายคลึงกับหน่วยคูณและสะสมของตัวกรองเอฟไออาร์ ดังนั้นตอนที่ดำเนินการคำสั่งเฉพาะ การทำงานของหน่วยควบคุมตัวกรองเอฟไออาร์จะถูกแทนที่ด้วยวงจรควบคุมของหน่วยประมวลผลกลาง ซึ่งจะควบคุมการดำเนินการของหน่วยคูณและสะสมของหน่วยประมวลผลกลางและตัวกรองให้ทำงานพร้อมกัน ในที่นี้จะยกตัวอย่างการใช้งานหน่วยประมวลผลเพื่อการกรองสัญญาณแบบปรับตัวด้วยระเบียบวิธีกำลังสองน้อยที่สุด (Least Mean Square Method :LMS)

การกรองสัญญาณแบบปรับตัวด้วยระเบียบวิธีกำลังสองน้อยที่สุดประกอบด้วยการดำเนินการหลัก ได้แก่ การคำนวณตัวกรอง (Do FIR) การหาค่าความผิดพลาดของสัญญาณเอาต์พุตของตัวกรองเทียบกับสัญญาณอ้างอิง (Calculate Error) และการปรับปรุงค่าสัมประสิทธิ์ของตัวกรอง (Update Coefficient) ผังแสดงภาระงานของการกรองแบบปรับตัวกรณีที่ใช้งานเพียงหน่วยประมวลผลกลาง (ดูรูปที่ 4.16) แสดงให้เห็นว่าหน่วยประมวลผลมีหน้าที่ประมวลผลงานทุกส่วน



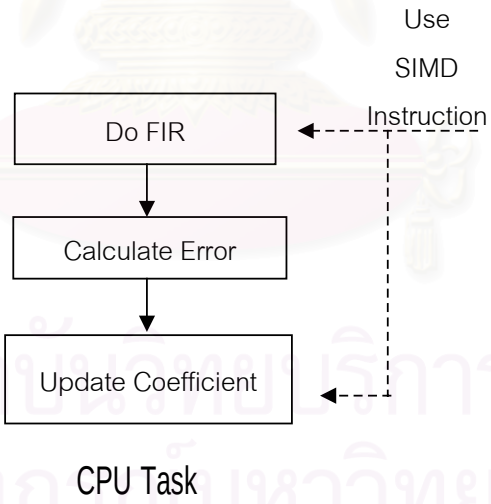
รูปที่ 4.16 ผังงานของการกรองสัญญาณแบบปรับตัวด้วยหน่วยประมวลผล

เมื่อใช้งานตัวกรองเอฟไออาร์ในลักษณะที่ 1 (ดูรูปที่ 4.17) จะทำให้ภาระงานของหน่วยประมวลผลกลางสำหรับการคำนวณการกรองแบบเอฟไออาร์ (Do FIR) ย้ายไปให้ฮาร์ดแวร์ตัวกรองทำการประมวลผลแทน ซึ่งทำให้หน่วยประมวลผลมีเวลาเหลือสำหรับการประมวลผลงานอื่นหรือสามารถทำงานในลักษณะขนานกับตัวกรองเอฟไออาร์ได้



รูปที่ 4.17 การกรองสัญญาณแบบปรับตัวเมื่อใช้ตัวกรองในลักษณะที่ 1

เมื่อใช้งานตัวกรองเอฟไออาร์ในลักษณะที่ 2 (ดูรูปที่ 4.18) การประยุกต์ใช้งานคำสั่งเฉพาะที่แสดงในตารางที่ 4.2 จะทำให้หน่วยประมวลผลจะสามารถคำนวณการกรองแบบเอฟไออาร์ได้ทีละสองแท็ปต่อหนึ่งคำสั่ง ซึ่งส่งผลให้จำนวนวงจรรอบคำสั่งในการคำนวณตัวกรองลดลงครึ่งหนึ่ง และในทำนองเดียวกันจะสามารถปรับปรุงค่าสัมประสิทธิ์ของตัวกรองได้พร้อมกันทีละสองชุดในหนึ่งคำสั่ง ซึ่งส่งผลให้เวลาในการปรับปรุงสัมประสิทธิ์ลดลงครึ่งหนึ่งเช่นเดียวกัน



รูปที่ 4.18 การกรองสัญญาณแบบปรับตัวเมื่อใช้ตัวกรองในลักษณะที่ 2

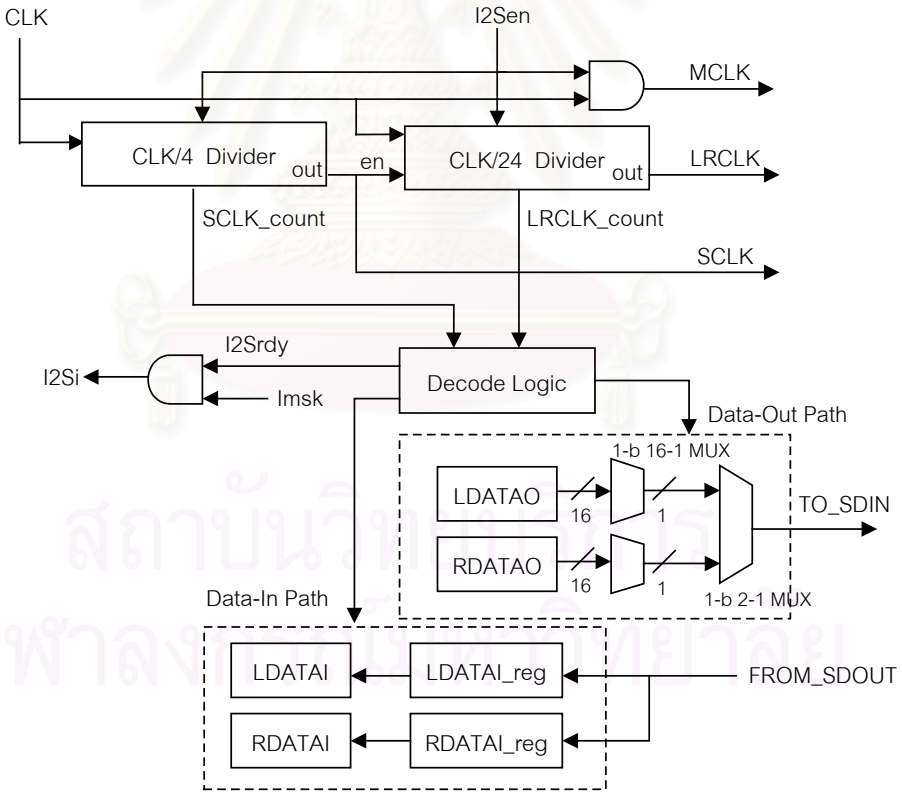
ตารางที่ 4.2 คำสั่งเฉพาะที่สามารถควบคุมตัวกรองพร้อมกับหน่วยประมวลผล

Mnenonic	Operand	Description	Operation	Flag	Cycle
SIMD instruction					
dstix	stsrc,*Xn	Indirect Store in Memory X and C	MEMX(Xn) <- stsrc, MEMC(Xn) <- stsrc	-	1
dstiy	stsrc,*Yn	Indirect Store in Memory Y and S	MEMY(Yn) <- stsrc, MEMS(Yn) <- stsrc	-	1
dmulu	*Xn, *Yn	Unsigned Multiply	ACC <- u MEMX(Xn) * u MEMX(Yn), FACC <- u MEMC(Xn) * u MEMS(Yn)	-	1
dmulus	*Xn, *Yn	Unsigned to Signed Multiply	ACC <- u MEMX(Xn) * s MEMX(Yn), FACC <- u MEMC(Xn) * s MEMS(Yn)	-	1
dmuls	*Xn, *Yn	Signed Multiply	ACC <- s MEMX(Xn) * s MEMX(Yn), FACC <- s MEMC(Xn) * s MEMS(Yn)	-	1
dmadd	*Xn, *Yn, Opc	Multiply and Add	ACC <- s MEMX(Xn) + (s MEMY(Yn)* s OPc), FACC <- s MEMC(Xn) + (s MEMS(Yn)* s OPc)	-	1
dmacu	*Xn, *Yn	Unsigned Multiply&Accumulate	ACC <- ACC + (u MEMX(Xn)* u MEMY(Yn)), FACC <- FACC + (u MEMC(Xn)* u MEMS(Yn))	-	1
dmacus	*Xn, *Yn	Unsigned to Signed Multiply&Acc	ACC <- ACC + (u MEMX(Xn)* s MEMY(Yn)), FACC <- FACC + (u MEMC(Xn)* s MEMS(Yn))	-	1
dmacs	*Xn, *Yn	Signed Multiply&Accumulate	ACC <- ACC + (s MEMX(Xn)* s MEMY(Yn)), FACC <- FACC + (s MEMC(Xn)* s MEMS(Yn))	-	1
dmsbu	*Xn, *Yn	Unsigned Multiply&Subtract	ACC <- ACC - (u MEMX(Xn)* u MEMY(Yn)), FACC <- FACC - (u MEMC(Xn)* u MEMS(Yn))	-	1
dmsbus	*Xn, *Yn	Unsigned to Signed Multiply&Subtract	ACC <- ACC - (u MEMX(Xn)* s MEMY(Yn)), FACC <- FACC - (u MEMC(Xn)* s MEMS(Yn))	-	1
dmsbs	*Xn, *Yn	Signed Multiply&Subtract	ACC <- ACC - (s MEMX(Xn)* s MEMY(Yn)), FACC <- FACC - (s MEMC(Xn)* s MEMS(Yn))	-	1

4.3 วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I2S

วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S มีหน้าที่เชื่อมต่อเพื่อรับส่งข้อมูลกับอุปกรณ์ภายนอกตามมาตรฐาน I^2S [12] ซึ่งได้แก่ อุปกรณ์เข้าและถอดรหัสสัญญาณเสียงแบบสเตอริโอ (Stereo Audio Codec) ที่มีหน้าที่แปลงสัญญาณแอนะล็อกจากภายนอกให้เป็นข้อมูลดิจิทัลแบบมีเครื่องหมายเพื่อป้อนให้กับหน่วยประมวลผล และรับข้อมูลดิลิทัลแบบมีเครื่องหมายจากหน่วยประมวลผลมาแปลงให้เป็นสัญญาณแอนะล็อกออกไปยังภายนอก

จังหวะการทำงานของอุปกรณ์มาตรฐาน I^2S จะถูกควบคุมด้วยสัญญาณนาฬิกา 3 ชุด ได้แก่ สัญญาณ MCLK (Master Clock) ซึ่งมีหน้าที่ควบคุมจังหวะการทำงานของวงจรภายในอุปกรณ์ สัญญาณ SCLK (Shift Clock) ซึ่งมีหน้าที่ควบคุมจังหวะการเลื่อนข้อมูลแบบอนุกรมให้เข้าและออกจากอุปกรณ์ และสัญญาณ LRCLK (Left/Right Clock) ซึ่งมีหน้าที่ควบคุมการรับส่งข้อมูลว่ากำลังกระทำกับข้อมูลด้านซ้ายหรือด้านขวาของอุปกรณ์



รูปที่ 4.19 โครงสร้างของวงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S

วงจรสร้างสัญญาณนาฬิกาจะสร้างสัญญาณ MCLK SCLK และ LRCLK จากสัญญาณนาฬิกาหลัก โดยจะควบคุมการป้อนสัญญาณนาฬิกาจากฟิลด์ I2Sen ของรีจิสเตอร์ใช้งานพิเศษ MCTRL ที่ควบคุมให้อุปกรณ์ I^2S ทำงานหรือไม่ สัญญาณ MCLK จะถูกสร้างด้วยสัญญาณนาฬิกาของระบบ สัญญาณ SCLK จะถูกสร้างด้วยสัญญาณนาฬิกาที่ถูกลดทอนความถี่ไป 4 เท่า ส่วนสัญญาณ LRCLK จะถูกสร้างด้วยสัญญาณนาฬิกาที่ถูกลดทอนความถี่ไป 24 เท่า วงจรส่วนนี้ถูกออกแบบให้ทำงานแบบสมวาร (Synchronous) ทั้งหมด เนื่องจากไม่จำเป็นต้องพิจารณาความล่าช้าของวงจรส่วนที่ทำงานแบบอสมวาร (Asynchronous Path) เช่น ตัวหารความถี่ ซึ่งอาจทำให้การทำงานของอุปกรณ์ I^2S เกิดผิดพลาดได้ แต่ทว่าวงจรแบบสมวารจะมีขนาดใหญ่กว่าแบบทำงานแบบอสมวาร

การรับส่งข้อมูลของอุปกรณ์มาตรฐาน I^2S จะกระทำผ่านขาสัญญาณ SDIN (Serial Data In) ซึ่งมีหน้าที่รับข้อมูลดิจิทัลจากหน่วยดำเนินการเพื่อแปลงเป็นสัญญาณแอนะล็อก และขาสัญญาณ SDOUT (Serial Data Out) ซึ่งมีหน้าที่ส่งข้อมูลดิจิทัลที่ถูกแปลงจากสัญญาณแอนะล็อก โดยการรับส่งข้อมูลจากขาสัญญาณทั้งสองจะกระทำในช่วงเวลาเดียวกัน

ข้อมูลที่ส่งไปยังขาสัญญาณ SDIN ของอุปกรณ์มาตรฐาน I^2S จะมาจากขาสัญญาณ TO_SDIN โดยการเลือกที่มาของข้อมูลว่าจะมาจากรีจิสเตอร์ LDATAO หรือ RDATAO ขึ้นอยู่กับสัญญาณ LRCLK ในขณะนั้น ถ้าสัญญาณ LRCLK มีค่าเป็น 0 แสดงว่าเป็นการส่งข้อมูลของสัญญาณด้านซ้าย และถ้าสัญญาณ LRCLK มีค่าเป็น 1 แสดงว่าเป็นการส่งข้อมูลของสัญญาณด้านขวา การแปลงข้อมูลจากรีจิสเตอร์ LDATAO และ RDATAO ที่มีขนาด 16 บิตให้กลายเป็นสัญญาณอนุกรมจะใช้มัลติเพล็กเซอร์แบบ 16 เลือกออก 1 ขนาด 1 บิตจำนวนสองชุดในการเลือกข้อมูลออก โดยจังหวะการเลือกข้อมูลออกจะถูกควบคุมโดยวงจรถอดรหัสจากสัญญาณ SCLK และ LRCLK

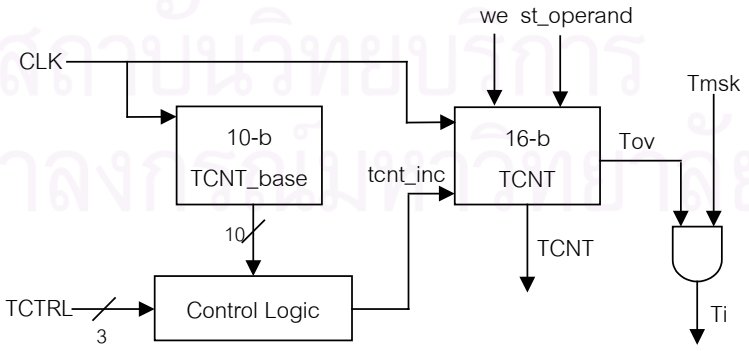
ข้อมูลที่รับมาจากขาสัญญาณ SDOUT ของอุปกรณ์มาตรฐาน I^2S จะมาจากขาสัญญาณ FROM_SDOUT โดยการเลือกข้อมูลว่าจะไปยังรีจิสเตอร์ LDATAI_REG หรือ RDATAI_REG ขึ้นอยู่กับสัญญาณ LRCLK ในขณะนั้น ถ้าสัญญาณ LRCLK มีค่าเป็น 0 แสดงว่าเป็นการรับข้อมูลจากสัญญาณด้านซ้าย และถ้าสัญญาณ LRCLK มีค่าเป็น 1 แสดงว่าเป็นการรับข้อมูลจากสัญญาณด้านขวา โดยข้อมูลที่เข้ามาจะถูกเลื่อนมาที่รีจิสเตอร์ LDATAI_REG และ RDATAI_REG ตามจังหวะของสัญญาณ SCLK และ LRCLK เมื่อการรับข้อมูลเสร็จสิ้นข้อมูลใน LDATAI_REG และ RDATAI_REG ถูกส่งไปที่รีจิสเตอร์ LDATAI และ RDATAI ตามลำดับ

เมื่อการรับส่งข้อมูลกระทำจนครบรอบ วงจรถอดรหัสจากสัญญาณ SCLK และ LRCLK จะให้สัญญาณ I2Srdy และฟิลด์ ISrdy ของรีจิสเตอร์ใช้งานพิเศษ GIFR มีค่าเป็น 1 เพื่อแจ้งหน่วยประมวลผลให้ทราบว่าการรับส่งข้อมูลเสร็จสิ้นแล้ว และถ้าฟิลด์ Imsk ของรีจิสเตอร์ใช้งานพิเศษ MCTRL มีค่าเป็น 1 จะส่งผลให้เกิดการจังหวะหน่วยประมวลผล เนื่องจากเกิดการสิ้นสุดการรับส่งข้อมูลขึ้น

4.4 ตัวตั้งเวลา

ตัวตั้งเวลามีหน้าที่สร้างฐานเวลาให้กับหน่วยประมวลผล โดยโครงสร้างของตัวตั้งเวลาประกอบด้วย ตัวนับเวลาแบบนับขึ้นขนาด 10 บิต (TCNT_base) สำหรับใช้เป็นฐานเวลาให้กับตัวตั้งเวลา ตัวนับเวลาแบบนับขึ้นขนาด 16 บิต (TCNT) สำหรับใช้นับสัญญาณที่ถูกส่งมาจากตัวนับเวลาขนาด 10 บิต และวงจรควบคุมที่ทำหน้าที่กำหนดจังหวะการเพิ่มขึ้นของตัวนับเวลา (ดูรูปที่ 4.20)

การทำงานตัวตั้งเวลาจะให้ตัวนับเวลา TCNT_base ทำการนับขึ้นอย่างอิสระ และให้วงจรควบคุมตรวจสอบค่าของตัวนับในขณะนั้นว่าสอดคล้องกับเงื่อนไขของสัญญาณควบคุมจากรีจิสเตอร์ใช้งานเฉพาะ TCTRL ที่ทำหน้าที่กำหนดการอัตราการเพิ่มค่าของรีจิสเตอร์ TCNT ว่าเพิ่มขึ้นด้วยความถี่เท่าใด เมื่อค่าของตัวนับเวลา TCNT_base สอดคล้องกับเงื่อนไขวงจรควบคุมที่ถูกกำหนดโดยฟิลด์ TCTRL ของรีจิสเตอร์ใช้งานพิเศษ MCTRLจะทำให้สัญญาณ tcnt_inc มีค่าเป็น 1 ซึ่งจะทำให้ตัวนับ TCNT เพิ่มค่าขึ้นหนึ่ง หรืออีกนัยหนึ่งคืออัตราการนับของตัวนับ TCNT จะขึ้นกับความถี่ที่สัญญาณ tcnt_inc มีค่าเป็น 1 เงื่อนไขของการทำงาน ของวงจรควบคุมเมื่อฟิลด์ TCTRL มีค่าต่างๆ แสดงในตารางที่ 4.3



รูปที่ 4.20 โครงสร้างของตัวตั้งเวลา

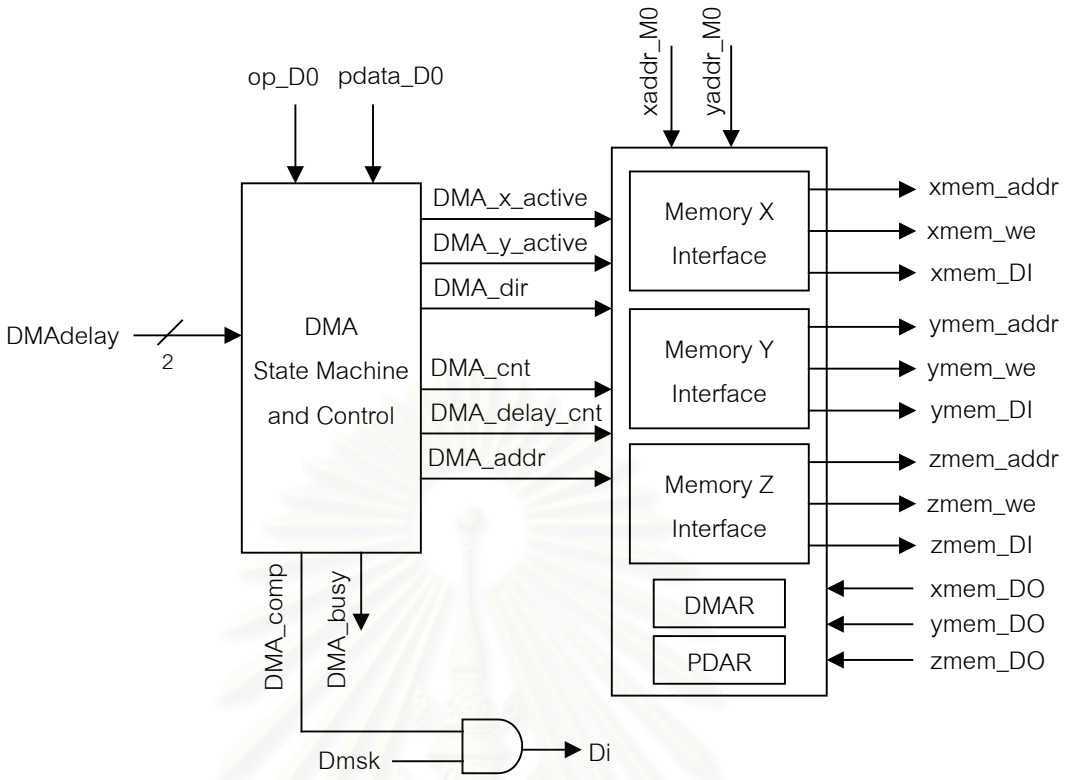
ตารางที่ 4.3 เงื่อนไขการทำงานของวงจรควบคุมของตัวตั้งเวลา

ค่าของฟิลด์ TCTRL	อัตราของการเพิ่มขึ้นของตัวนับ TCNT
001	ความถี่สัญญาณนาฬิกา
010	ความถี่สัญญาณนาฬิกาหารด้วย 8
011	ความถี่สัญญาณนาฬิกาหารด้วย 64
100	ความถี่สัญญาณนาฬิกาหารด้วย 256
101	ความถี่สัญญาณนาฬิกาหารด้วย 1024
อื่นๆ	ตัวนับเวลาหยุดทำงาน

เมื่อตัวนับเวลา TCNT เกิดการล้นเกินขึ้นจะส่งผลให้สัญญาณ Tov และฟิลด์ Tov ของรีจิสเตอร์ใช้งานพิเศษ GIFR มีค่าเป็น 1 เพื่อแจ้งให้หน่วยประมวลผลทราบ และถ้าฟิลด์ Tmsk ของรีจิสเตอร์ใช้งานพิเศษ MCTRL มีค่าเป็น 1 จะส่งผลให้เกิดการจังหวะหน่วยประมวลผล เนื่องจากเกิดการล้นเกินของตัวนับเวลาขึ้น

4.5 ดีเอ็มเอ

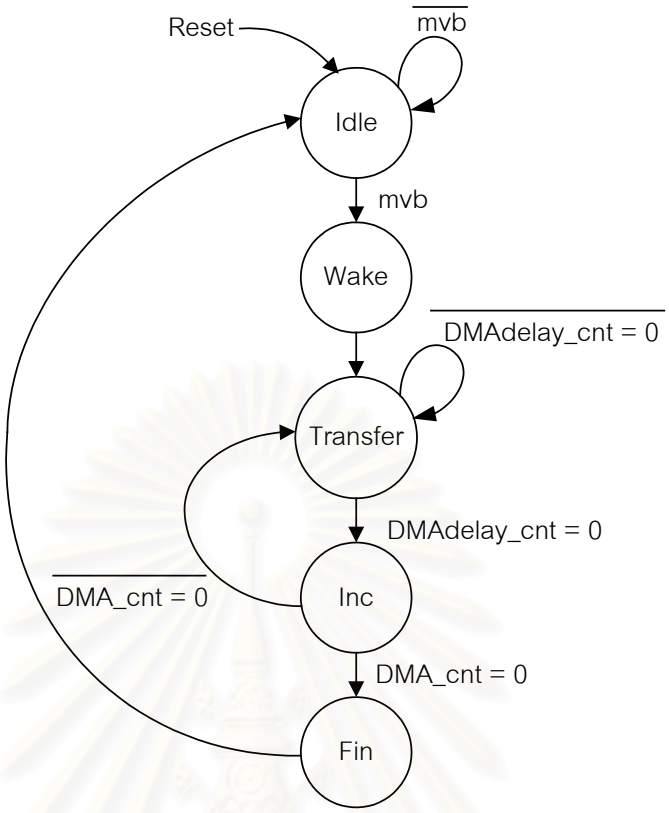
ดีเอ็มเอ (Direct Memory Access :DMA) มีหน้าที่เป็นตัวกลางเชื่อมโยงระหว่างหน่วยประมวลผลและหน่วยความจำ และเป็นตัวกลางในการโอนย้ายข้อมูลระหว่างหน่วยความจำภายในซึ่งได้แก่ หน่วยความจำ X หรือ Y และหน่วยความจำภายนอก Z โครงสร้างของดีเอ็มเอ (ดูรูปที่ 4.21) ประกอบด้วย วงจรควบคุมและเครื่องจักรสถานะจำกัด (DMA Stage Machine and Control) วงจรเชื่อมต่อหน่วยความจำภายใน X (Memory X Interface) วงจรเชื่อมต่อหน่วยความจำภายใน Y (Memory Y Interface) และวงจรเชื่อมต่อหน่วยความจำภายนอก Z (Memory Z Interface)



รูปที่ 4.21 โครงสร้างของดีเอ็มเอ

การเขียนข้อมูลจากหน่วยประมวลผลลงในหน่วยความจำ X (หรือหน่วยความจำ Y) วงจรเชื่อมต่อหน่วยความจำภายใน X (หรือ Y) จะผ่านค่าตำแหน่งข้อมูล xaddr_M0 (หรือ yaddr_M0) ที่ส่งมาจากหน่วยกำเนิดตำแหน่งข้อมูลไปยังขาสัญญาณ xmem_addr (หรือ ymem_addr) ซึ่งต่อกับขาตำแหน่งของหน่วยความจำ และอ่านข้อมูลรหัสดำเนินการจากสัญญาณ op_D0 และ pdata_D0 เพื่อถอดรหัสคำสั่งและนำไปควบคุมสัญญาณ xmem_we (หรือ ymem_we) เพื่อควบคุมให้มีการเขียนข้อมูลลงในหน่วยความจำต่อไป ส่วนการอ่านข้อมูลจากหน่วยความจำ X ข้อมูลจากหน่วยความจำจะผ่านเข้ามาทางขาสัญญาณ xmem_D0 (หรือ ymem_D0)

การโอนย้ายกลุ่มข้อมูลระหว่างหน่วยความจำภายในกับภายนอกจะแบ่งการทำงานได้เป็นสองทิศทาง ได้แก่ การย้ายข้อมูลจากหน่วยความจำภายในไปยังหน่วยความจำภายนอก และการย้ายข้อมูลจากหน่วยความจำภายนอกไปยังหน่วยความจำภายใน โดยการทำงานของดีเอ็มเอถูกควบคุมด้วยเครื่องจักรสถานะจำกัดดังแสดงในรูปที่ 4.22



รูปที่ 4.22 แผนผังสถานะการทำงานของดีเอ็มเอ

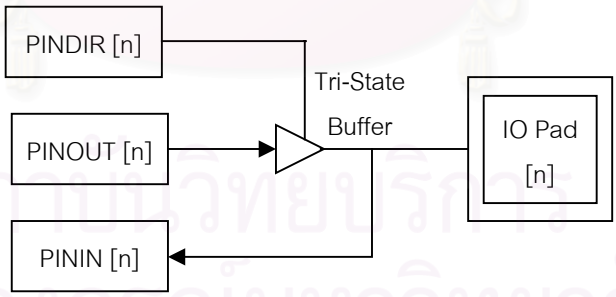
ในสถานะ Idle ดีเอ็มเอจะหยุดทำงานและรอคำสั่งจากหน่วยประมวลผลเพื่อเริ่มทำงาน ในสถานะนี้ หน่วยประมวลผลจะสามารถติดต่อกับหน่วยความจำภายในได้ด้วยตำแหน่งของข้อมูลจากหน่วยกำเนิดตำแหน่งข้อมูลโดยตรง เมื่อหน่วยประมวลผลดำเนินการคำสั่ง MVBx การทำงานของดีเอ็มเอจะเข้าสู่สถานะ Wake ในสถานะนี้ ดีเอ็มเอจะพิจารณาการโอนย้ายข้อมูลว่ากระทำกับหน่วยความจำ X หรือ Y และมีการโอนย้ายข้อมูลไปในทิศทางใด แล้วจึงอ่านข้อมูลจากรีจิสเตอร์ใช้งานเฉพาะ เช่น ฟิลด์ DMAdelay ของรีจิสเตอร์ใช้งานพิเศษ MCTRL ซึ่งกำหนดความเร็วในการโอนย้ายข้อมูลให้เหมาะสมกับหน่วยความจำภายนอกจากรีจิสเตอร์ควบคุมหลัก ฟิลด์ DMAstart ของรีจิสเตอร์ใช้งานพิเศษ DMAR ซึ่งกำหนดตำแหน่งเริ่มต้นของการโอนย้ายข้อมูล และฟิลด์ DMAsize ของรีจิสเตอร์ใช้งานพิเศษ DMAR ซึ่งกำหนดจำนวนชุดข้อมูลที่โอนย้ายจากรีจิสเตอร์ควบคุมดีเอ็มเอ หลังจากนั้นการทำงานของดีเอ็มเอจะเข้าสู่สถานะ Transfer ในสถานะนี้ดีเอ็มเอจะอ่านข้อมูลจากหน่วยความจำหนึ่งแล้วส่งไปเขียนยังหน่วยความจำอีกหน่วยความจำหนึ่ง ซึ่งระยะเวลาการโอนย้ายข้อมูลแต่ละชุดจะขึ้นอยู่กับฟิลด์ DMAdelay โดยถ้าฟิลด์ DMAdelay มีค่า 00 จะใช้ระยะเวลาในการโอนย้ายหนึ่งคาบสัญญาณนาฬิกา และถ้าฟิลด์ DMAdelay มีค่า 11 จะใช้ระยะเวลาในการโอนย้ายสี่คาบสัญญาณนาฬิกาเป็นต้น เมื่อดีเอ็มเออยู่ในสถานะ Transfer จนครบระยะเวลาที่กำหนดแล้วการทำงานจะเข้าสู่สถานะ Inc ในสถานะนี้ ดีเอ็มเอจะเพิ่มค่าของ

ตำแหน่งข้อมูลในหน่วยความจำขึ้นหนึ่ง แล้วกลับสู่สถานะ Transfer เพื่อโอนย้ายข้อมูลชุดต่อไป จนกระทั่งโอนย้ายข้อมูลครบตามที่กำหนดแล้วจะเข้าสู่สถานะ Fin และ Idle ตามลำดับ

เมื่อการทำงานของดีเอ็มเออยู่ในสถานะ Wake Transfer และ Inc จะส่งผลให้สัญญาณ DMA_busy และฟิลด์ Dbsy ของรีจิสเตอร์ใช้งานพิเศษ GIFR มีค่าเป็น 1 เพื่อแจ้งให้หน่วยประมวลผลทราบว่าดีเอ็มเอกำลังทำงานอยู่ เมื่อการทำงานของดีเอ็มเออยู่ในสถานะ Fin แสดงว่าการโอนย้ายข้อมูลได้เสร็จสิ้นแล้ว จะส่งผลให้สัญญาณ DMA_comp และฟิลด์ Dcom มีค่าเป็น 1 เพื่อแจ้งให้หน่วยประมวลผลทราบและถ้าฟิลด์ Dmsk ของรีจิสเตอร์ใช้งานพิเศษ MCTRL มีค่าเป็น 1 จะส่งผลให้เกิดการจังหวะหน่วยประมวลผล เนื่องจากการโอนย้ายข้อมูลด้วยดีเอ็มเอสิ้นสุด

4.6 พอร์ตอินพุต-เอาต์พุต

พอร์ตอินพุต-เอาต์พุตมีหน้าที่สำหรับใช้สื่อสารข้อมูลระหว่างหน่วยประมวลผลกับอุปกรณ์ภายนอก หรือแสดงสถานะการทำงานของหน่วยประมวลผล โครงสร้างพอร์ตอินพุต-เอาต์พุตเป็นพอร์ตแบบสองทิศทาง (Bidirectional Port) ขนาด 16 บิต โดยประกอบด้วย รีจิสเตอร์พิเศษ PININ สำหรับเก็บข้อมูลที่อ่านเข้ามา รีจิสเตอร์ใช้งานพิเศษ PINOUT สำหรับเก็บข้อมูลที่ส่งออกไปภายนอก และรีจิสเตอร์พิเศษ PINDIR สำหรับกำหนดทิศทางการทำงานของพอร์ตตำแหน่งนั้นๆ (ดูรูปที่ 4.23) เมื่อค่าในรีจิสเตอร์ PINDIR บิตใดมีค่าเป็น 0 จะเป็นการกำหนดให้พอร์ตนั้นทำงานเป็นพอร์ตเอาต์พุต หากรีจิสเตอร์ PINDIR บิตใดมีค่าเป็น 1 จะเป็นการกำหนดให้พอร์ตนั้นทำงานเป็นพอร์ตอินพุต



รูปที่ 4.23 โครงสร้างของพอร์ตอินพุต-เอาต์พุต

4.7 สรุปท้ายบท

ในบทนี้ได้กล่าวถึงรายละเอียดของรีจิสเตอร์ใช้งานเฉพาะ การออกแบบตัวกรองเอฟไอาร์และอุปกรณ์บริวารอื่นๆ เช่น วงจรเชื่อมต่ออุปกรณ์มาตราฐาน I^2S ตัวตั้งเวลา ดีเอ็มเอ และพอร์ตอินพุต-เอาต์พุต รวมถึงรูปแบบการทำงานของตัวกรองเอฟไอาร์

บทที่ 5

การวัดสมรรถนะของหน่วยประมวลผลสัญญาณดิจิทัล

5.1 การวัดสมรรถนะของ D-CHIP ด้วยบรรทัดฐาน

การวัดสมรรถนะของ D-CHIP ในที่นี้จะใช้การเขียนโปรแกรมควบคุมหน่วยประมวลผลให้ทำงานตามบรรทัดฐาน (Benchmark) ของงานกรรมวิธีสัญญาณดิจิทัล ได้แก่ การคูณค่าจริง การปรับปรุงค่าจริง การหาค่าสหสัมพันธ์ การหาค่าผลรวมของกำลังสอง การคำนวณตัวกรองเอฟไออาร์ การคำนวณตัวกรองเอฟไอออาร์แบบปรับตัวด้วยระเบียบวิธีกำลังสองน้อยที่สุด และการคำนวณตัวกรองไอไออาร์ เป็นต้น โดยสมรรถนะของหน่วยประมวลผลจะพิจารณาที่จำนวนรอบของสัญญาณนาฬิกา (Clock Cycles) ที่ใช้ในการประมวลผลบรรทัดฐานที่กำหนด

5.1.1 การคูณจำนวนจริง

การคูณจำนวนจริง (Real Multiply) เป็นการดำเนินการพื้นฐานของกรรมวิธีสัญญาณดิจิทัล โดยใช้ตัวถูกดำเนินการเป็นจำนวนจริงสองจำนวน (A และ B) การดำเนินการของการคูณค่าจริงแสดงดังสมการที่ 5.1

$$C = A \times B \quad \text{.....(5.1)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการคูณจำนวนจริง ปรากฏว่าหน่วยประมวลผลใช้เวลาดำเนินการ 6 รอบสัญญาณนาฬิกา สาเหตุที่การคูณจำนวนจริงใช้เวลาประมวลผล 6 รอบสัญญาณนาฬิกา เนื่องจากการที่วงจรคูณมีโครงสร้างแบบไปป์ไลน์ ทำให้มีช่วงเวลาว่างระหว่างที่กำลังดำเนินการการคูณ

5.1.2 การคูณจำนวนจริง N จำนวน

การคูณจำนวนจริง N จำนวน (N Real Multiply) เป็นการคูณจำนวนจริงจำนวน N ชุด โดยข้อมูลแต่ละชุดจะอยู่ภายในหน่วยความจำ และผลลัพธ์ของการคูณจะถูกเก็บลงในหน่วยความจำ การดำเนินการของการคูณค่าจริงแสดงดังสมการที่ 5.2

$$C(n) = A(n) \times B(n) \quad \text{..... (5.2)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการคูณจำนวนจริง N จำนวน ปรากฏว่าหน่วยประมวลผลใช้เวลาดำเนินการ 2N+4 รอบสัญญาณนาฬิกา การคูณจำนวนจริง N จำนวนจะใช้เวลาดำเนินการต่อข้อมูลหนึ่งชุดน้อยกว่าการคูณเพียงจำนวนเดียว เนื่องจากการดำเนินการแบบไปป์ไลน์ที่สามารถใช้ช่วงเวลาว่างจากการคูณข้อมูลก่อนหน้ามาดำเนินการการคูณข้อมูลชุดต่อไปได้

5.1.3 <u>การปรับปรุงจำนวนจริง</u>

การปรับปรุงจำนวนจริง (Real Update) เป็นการดำเนินการที่ประกอบด้วยการคูณจำนวนจริงสองจำนวน (A และ B) จากหน่วยความจำ และนำผลลัพธ์ของการคูณมาบวกกับจำนวนจริงอีกจำนวนหนึ่ง (C) แล้วจึงเก็บผลลัพธ์ของการบวก (D) ลงในหน่วยความจำ การดำเนินการของการปรับปรุงค่าจริงแสดงดังสมการที่ 5.3

$$D = C + A \times B \quad \text{.....(5.3)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการปรับปรุงจำนวนจริงปรากฏว่าหน่วยประมวลผลใช้เวลาดำเนินการ 8 รอบสัญญาณนาฬิกา สาเหตุที่การคูณจำนวนจริงใช้เวลาประมวลผล 8 รอบสัญญาณนาฬิกา เนื่องจากความล่าช้าจากการคูณ

5.1.4 <u>การปรับปรุงจำนวนจริง N จำนวน</u>

การปรับปรุงจำนวนจริง N จำนวน (N Real Update) เป็นการดำเนินการปรับปรุงจำนวนจริงกับข้อมูล N ชุด การดำเนินการของการปรับปรุงจำนวนจริง N จำนวนแสดงดังสมการที่ 5.4

$$D(n) = C(n) + A(n) \times B(n) \quad \text{.....(5.4)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการปรับปรุงจำนวนจริง N จำนวน ปรากฏว่าหน่วยประมวลผลใช้เวลาดำเนินการ 3N+6 รอบสัญญาณนาฬิกา การปรับปรุงจำนวนจริง N จำนวนจะใช้เวลาดำเนินการต่อข้อมูลหนึ่งชุดน้อยกว่าการปรับปรุงเพียงจำนวนเดียวเนื่องจากสามารถใช้ช่วงเวลาว่างจากการคูณข้อมูลก่อนหน้ามาดำเนินการการอ่านและคูณข้อมูลชุดต่อไปได้

5.1.5 <u>การหาค่าสหสัมพันธ์ของจำนวนจริง</u>

การหาค่าสหสัมพันธ์ของจำนวนจริง (Real Correlation) N จำนวนเป็นการดำเนินการการหาค่าสหสัมพันธ์ของข้อมูล (A และ B) ที่เป็นจำนวนจริง N ชุดที่อยู่ภายในหน่วยความจำ การดำเนินการของการหาค่าสหสัมพันธ์ของจำนวนจริงแสดงดังสมการที่ 5.5

$$C(n) = \sum_{i=0}^{N-1} A_i(n) \cdot B(n-i) \quad \text{..... (5.5)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการหาค่าสหสัมพันธ์ หากใช้เพียงหน่วยประมวลผลกลางจะใช้เวลาประมวลผลเท่ากับ N+8 รอบสัญญาณนาฬิกา แต่ถ้าใช้งานหน่วยประมวลผลกลางร่วมกับตัวกรองเอฟไอาร์ในลักษณะที่ 1 จะใช้เวลาประมวลผลเท่ากับ 0.5N+10 รอบสัญญาณนาฬิกา

5.1.6 <u>การหาค่าผลรวมของกำลังสองของจำนวนจริง</u>

การหาค่าผลรวมของกำลังสอง (Sum of Square) ของจำนวนจริง N จำนวนเป็นการดำเนินกการหาค่าผลรวมของกำลังสองของข้อมูล (A) ที่เป็นจำนวนจริง N ชุดที่อยู่ภายในหน่วยความจำ การดำเนินการของการหาค่าผลรวมของกำลังสองของจำนวนจริงดังสมการที่ 5.6

$$B(n) = \sum_{i=0}^{N-1} A^2(i) \quad \text{.....(5.6)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการหาค่าผลรวมของกำลังสองของจำนวนจริง ปรากฏว่าหน่วยประมวลผลใช้เวลาดำเนินการ 2N+11 รอบสัญญาณนาฬิกา

5.1.7 <u>การคำนวณตัวกรองเอฟไอาร์ความยาว N แท็ป</u>

การคำนวณตัวกรองเอฟไอาร์ความยาว N แท็ปของจำนวนจริง (N-Tap Real FIR) เป็นการดำเนินการกับข้อมูลตัวอย่าง ($x(n)$) และสัมประสิทธ์ ($h(n)$) อย่างละ N ชุด ที่เป็นจำนวนจริง การคำนวณตัวกรองเอฟไอาร์ของจำนวนจริงแสดงดังสมการที่ 5.7

$$y(n) = \sum_{i=0}^{N-1} h_i(n) \cdot x(n-i) \quad \text{.....(5.7)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการคำนวณตัวกรองเอฟไอาร์ หากใช้เพียงหน่วยประมวลผลกลางจะใช้เวลาประมวลผลเท่ากับ N+11 รอบสัญญาณนาฬิกา แต่ถ้าใช้

งานหน่วยประมวลผลกลางร่วมกับตัวกรองเอฟไอออาร์ไม่ว่าจะเป็นในลักษณะที่ 1 หรือ 2 จะใช้เวลาประมวลผลเท่ากับ 0.5N+14 รอบสัญญาณนาฬิกา

5.1.8 <u>การคำนวณตัวกรองไอไอออาร์แบบไบควอดอันดับสอง</u>

การคำนวณตัวกรองไอไอออาร์แบบไบควอดอันดับสอง (2nd Order Biquad IIR) เป็นการคำนวณตัวกรองไอไอออาร์ที่มีโครงสร้างโดยตรงชนิดที่ 2 (Direct Form II) โดยมีข้อมูลตัวอย่าง ($x(n)$) ข้อมูลผลลัพธ์ของการคำนวณในรอบผ่านมา ($w(n)$) และข้อมูลสัมประสิทธิ์ (a และ b) ที่เป็นจำนวนจริง การคำนวณตัวกรองเอฟไอออาร์ของจำนวนจริงแสดงดังสมการที่ 5.8 และ 5.9

$$w(n) = x(n) - a_1 \times w(n-1) - a_2 \times w(n-2) \quad \text{.....(5.8)}$$

$$y(n) = w(n) - b_1 \times w(n-1) - b_2 \times w(n-2) \quad \text{.....(5.9)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการคำนวณตัวกรองไอไอออาร์แบบไบ-ควอดอันดับสอง ปรากฏว่าหน่วยประมวลผลใช้เวลาดำเนินการ 14 รอบสัญญาณนาฬิกา

5.1.9 <u>การคำนวณตัวกรองไอไอออาร์แบบไบควอดต่อเรียง N ชุด</u>

การคำนวณตัวกรองไอไอออาร์แบบไบควอดอันดับสองต่อเรียง N ชุด (N Cascaded Biquad IIR) เป็นการคำนวณตัวกรองไอไอออาร์แบบไบควอดอย่างต่อเนื่อง โดยการนำผลลัพธ์ของตัวกรองในขั้นตอนก่อนหน้าไปเป็นข้อมูลอินพุตของตัวกรองในชุดถัดไป

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการคำนวณตัวกรองไอไอออาร์แบบไบ-ควอดอันดับสองต่อเรียง N ชุด ปรากฏว่าหน่วยประมวลผลใช้เวลาดำเนินการ 7N+10 รอบสัญญาณนาฬิกา

5.1.10 <u>การคำนวณตัวกรองเอฟไอออาร์แบบปรับตัวด้วยระเบียบวิธีกำลังสองน้อยที่สุด</u>

การคำนวณตัวกรองเอฟไอออาร์แบบปรับตัวด้วยระเบียบวิธีกำลังสองน้อยที่สุด (Least Mean Square Adaptive FIR) ความยาว N แท็ป เป็นการดำเนินการที่ประกอบด้วยการคำนวณหลักสองส่วน ได้แก่ การคำนวณตัวกรองเอฟไอออาร์ และการปรับปรุงค่าสัมประสิทธิ์ ซึ่งในที่นี้จะใช้ระเบียบวิธีกำลังสองน้อยที่สุดในการปรับปรุงสัมประสิทธิ์ การคำนวณตัวกรองเอฟไอออาร์แบบปรับตัวแสดงดังสมการที่ 5.10 ส่วนการปรับปรุงค่าสัมประสิทธิ์แสดงดังสมการที่ 5.11 และ 5.12 ตามลำดับ โดย $e(n)$ เป็นค่าความผิดพลาดของผลลัพธ์ของตัวกรองเทียบกับสัญญาณอ้างอิง $s(n)$

$$y(n) = \sum_{i=0}^{N-1} h_i(n) \cdot x(n-i) \quad \text{.....(5.10)}$$

$$e(n) = s(n) - y(n) \quad \text{.....(5.11)}$$

$$h_i(n+1) = h_i(n) + \mu \cdot e(n) \cdot x_i(n-i) \quad \text{.....(5.12)}$$

การเขียนโปรแกรมควบคุมให้ทำงานตามบรรทัดฐานการคำนวณตัวกรองเอฟไออาร์แบบปรับตัว หากใช้เพียงหน่วยประมวลผลกลางจะใช้เวลาประมวลผลเท่ากับ 3N+23 รอบสัญญาณนาฬิกา แต่ถ้าใช้งานหน่วยประมวลผลกลางร่วมกับตัวกรองเอฟไออาร์ในลักษณะที่ 2 จะใช้เวลาประมวลผลเท่ากับ 1.5N+26 รอบสัญญาณนาฬิกา

5.2 การเปรียบเทียบสมรรถนะของ D-CHIP กับหน่วยประมวลผลในท้องตลาด

การเปรียบเทียบสมรรถนะของ D-CHIP กับหน่วยประมวลผลสัญญาณดิจิทัลในท้องตลาด ซึ่งได้แก่ DSP56300 ของบริษัท Motorola [22] TMS320C6200 และ TMS320C5400 ของบริษัท Texas Instruments [23, 24] และ DSP16000 ของบริษัท Lucent Technology [25] จะพิจารณาตามบรรทัดฐานแต่ละอันที่กล่าวในหัวข้อที่ 5.1 โดยข้อมูลของสมรรถนะที่มีจะแตกต่างกันไปขึ้นอยู่กับแต่ละหน่วยประมวลผลและข้อมูลที่เปิดเผยของแต่ละบริษัท การเปรียบเทียบสมรรถนะของ D-CHIP กับหน่วยประมวลผลในท้องตลาดแสดงดังตารางที่ 5.1 โดยข้อมูลที่เป็นช่องว่างในตารางแสดงว่าเป็นเป็นข้อมูลที่ไม่ได้เปิดเผย

ข้อมูลในตารางที่ 5.1 แสดงให้เห็นว่าสมรรถนะของการประมวลผลบรรทัดฐานประเภทการกรองสัญญาณแบบเอฟไออาร์ และการกรองสัญญาณแบบปรับตัวของ D-CHIP อยู่ในอันดับเดียวกันหน่วยประมวลผลสัญญาณดิจิทัลในท้องตลาด ซึ่งแสดงให้เห็นว่า D-CHIP สามารถประมวลผลงานประเภทการกรองสัญญาณแบบเอฟไออาร์ได้เป็นอย่างดี แต่อย่างไรก็ตามการประมวลผลบรรทัดฐานประเภทการกรองสัญญาณแบบไอไออาร์ของ D-CHIP จะใช้จำนวนวงรอบคำสั่งมากกว่าหน่วยประมวลผลอื่นๆ เนื่องจากการที่วงจรคูณของ D-CHIP มีโครงสร้างแบบไปป์ไลน์ ทำให้การประมวลผลงานที่ต้องใช้ผลลัพธ์จากการคูณในขั้นตอนก่อนหน้ามาประมวลผลทำได้ไม่ดีนัก

5.3 สมรรถนะของ D-CHIP ในแง่ความถี่การทำงานสูงสุด

เมื่อพิจารณาสมรรถนะของ D-CHIP ในแง่ความถี่สามารถสูงสุดที่วงจรรวมสามารถทำงานได้ การพิจารณาจะแบ่งเป็นสองรูปแบบ ได้แก่ สมรรถนะของวงจรบนเทคโนโลยีเอฟพีจีเอ ซึ่งได้แก่ เทคโนโลยีเอฟพีจีเอ Spartan II ของบริษัท Xilinx และสมรรถนะของวงจรบนเทคโนโลยีเซลล์มาตราฐานซีมอส ซึ่งได้แก่ เทคโนโลยีซีมอส CUB ขนาด 0.8 ไมครอน และ C35 ขนาด 0.35

ไมครอนของบริษัท Austriamicrosystems หรือ AMS และเทคโนโลยีซีมอส MTC35000 ขนาด 0.5 ไมครอน ของบริษัท Alcatel โดยความถี่การทำงานสูงสุดที่ได้เป็นค่าที่ได้จากการสังเคราะห์ (Synthesis) วงจรลงบนเทคโนโลยีต่างๆ การเปรียบความถี่การทำงานสูงสุดของ D-CHIP ที่เทคโนโลยีสารกึ่งตัวนำต่างๆ แสดงดังตารางที่ 5.2

ตารางที่ 5.1 การเปรียบเทียบสมรรถนะของ D-CHIP กับหน่วยประมวลผลในท้องตลาด

บรรทัดฐาน	TI TMS320C6200	TI TMS320C5400	Lucent DSP16000	Motorola DSP56300	D-CHIP
Real Multiply	-	-	-	4	6
N Real Multiply	-	-	-	2N+8	2N+4
Real Update	-	-	-	5	8
N Real Update	-	-	-	2N+8	3N+6
Real Correlation	0.5N+22	N+4	0.5N+K*	N+14	0.5N+10
Sum of Square	0.5N+19	-	-	-	2N+11
N-Tap Real FIR	0.5N+22	N+4	0.5N+K*	N+14	0.5N+14
2^{nd} Order Biquad IIR	-	-	-	9	14
N Cascaded Biquad IIR	-	-	3N+K*	5N+10	7N+10
Exact LMS Adaptive FIR	1.5N+26	2N+14	1.5N+K*	3N+16	1.5N+26

* เมื่อ K เป็นค่าคงที่ซึ่งทางผู้ผลิตไม่ได้เปิดเผย

ตารางที่ 5.2 การเปรียบความถี่การทำงานสูงสุดของ D-CHIP ที่เทคโนโลยีสารกึ่งตัวนำต่างๆ

เทคโนโลยีสารกึ่งตัวนำ	ความถี่สูงสุดที่ทำงานได้ (MHz)
เอฟพีจีเอ Spartan II ของ Xilinx	44
เทคโนโลยีซีมอส 0.8 ไมครอน ของ AMS	40
เทคโนโลยีซีมอส 0.5 ไมครอน ของ Alcatel	53
เทคโนโลยีซีมอส 0.35 ไมครอน ของ AMS	130

การเปรียบเทียบสมรรถนะของ D-CHIP กับหน่วยประมวลผลสัญญาณดิจิทัลในท้องตลาด ในที่นี้จะพิจารณาที่บรรทัดฐาน ซึ่งเป็นการเปรียบเทียบสมรรถนะของหน่วยประมวลผลในแง่สถาปัตยกรรมเป็นหลักซึ่งแสดงให้เห็นว่าหากหน่วยประมวลผลแต่ละตัวทำงานที่ความถี่เท่ากัน หน่วยประมวลผลใดจะประมวลผลงานได้มากกว่า นอกจากนี้การเปรียบเทียบสมรรถนะของหน่วยประมวลผลโดยพิจารณาความถี่สูงสุดที่สามารถทำงานได้ จะทำให้ผลการเปรียบเทียบที่ได้ไม่มีความหมายในเชิงการออกแบบวงจร เนื่องจากหน่วยประมวลผลสัญญาณดิจิทัลแต่ละตัวถูกสร้างขึ้นด้วยสถาปัตยกรรม พื้นที่ของวงจรรวม และเทคโนโลยีวงจรรวมที่แตกต่างกัน ซึ่งทางผู้ผลิตมักจะไม่ได้เปิดเผยข้อมูลภายในของวงจรรวม จึงทำให้หน่วยประมวลผลแต่ละตัวทำงานได้ที่ความถี่สูงสุดแตกต่างกัน ตัวอย่างเช่น วงจรรวม D-CHIP ที่มีโครงสร้างแบบเดียวกัน แต่ถูกผลิตด้วยเทคโนโลยีสารกึ่งตัวตัวนำแตกต่างกัน จะสามารถทำงานได้ที่ความถี่สูงสุดแตกต่างกัน (ดูตารางที่ 5.2)

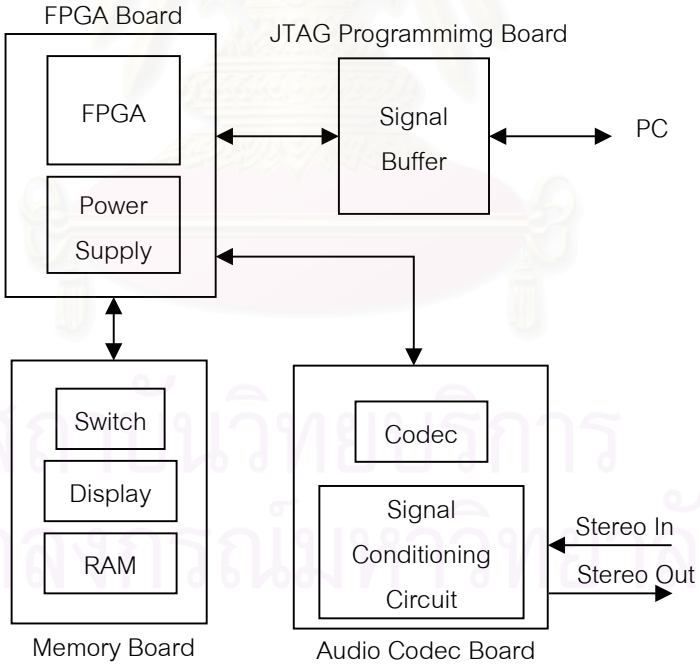
บทที่ 6

การสร้างตัวต้นแบบและการออกแบบวงจรรวม

6.1 การสร้างตัวต้นแบบ

6.1.1 บอร์ดตัวต้นแบบของหน่วยประมวลผลสัญญาณดิจิทัล

แบบจำลองของหน่วยประมวลผลสัญญาณดิจิทัลที่ถูกออกแบบ และจำลองการทำงานด้วยภาษา VHDL จะถูกนำมาสังเคราะห์เพื่อสร้างตัวต้นแบบบนเอฟพีจีเอ การออกแบบบอร์ดตัวต้นแบบจะแบ่งออกเป็นหลายบอร์ดขึ้นกับหน้าที่การทำงาน โดยจะประกอบไปด้วย บอร์ดเอฟพีจีเอ (FPGA Board) บอร์ดหน่วยความจำ (Memory Board) บอร์ดของตัวเข้า-ถอดรหัสสัญญาณเสียง (Audio Codec Board) และบอร์ดโปรแกรมเอฟพีจีเอด้วยการเชื่อมต่อแบบ JTAG (JTAG Programming Board) (ดูรูปที่ 6.1)



รูปที่ 6.1 โครงสร้างของบอร์ดตัวต้นแบบของหน่วยประมวลผลสัญญาณดิจิทัล

บอร์ดเอฟพีจีเอมีส่วนประกอบหลัก ได้แก่ เอฟพีจีเอ Spartan II เบอร์ XC2S150PQ208-5C ของบริษัท Xilinx [9] ซึ่งมีความซับซ้อนของวงจรภายในเทียบเท่า 150,000 เกต สำหรับการทำงาน

เป็นหน่วยประมวลผลและวงจรดิจิทัลอื่นๆที่นำมาสังเคราะห์ และวงจรส่วนจ่ายไฟ ซึ่งมีหน้าที่สร้างแรงดัน 2.5 3.3 และ 5 โวลต์ สำหรับป้อนให้กับเอฟพีจีเอและอุปกรณ์ในบอร์ดอื่นๆ

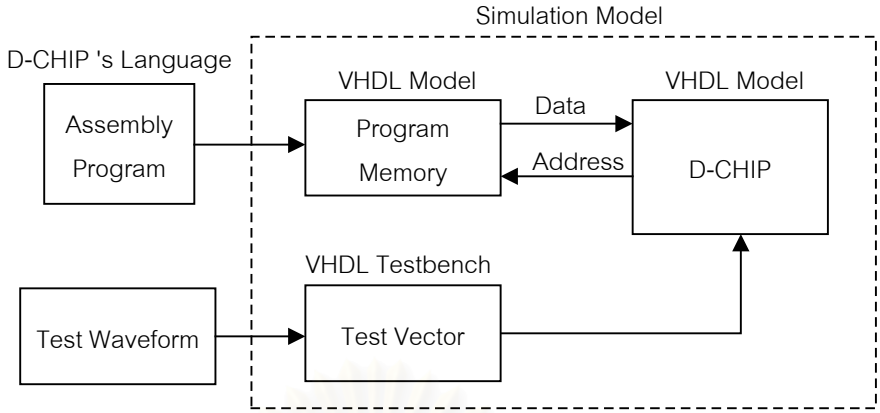
บอร์ดหน่วยความจำมีส่วนประกอบหลัก ได้แก่ หน่วยความจำแบบ SRAM เบอร์ IDT71V256SA ของบริษัท Integrated Device Technology [21] ที่มีเวลาเข้าถึง (Access Time) 20 นาโนวินาที ขนาด 8 บิต 32 กิโลไบต์ ไดโอดเปล่งแสง (LED) และสวิตซ์

บอร์ดของตัวเข้า-ถอดรหัสสัญญาณเสียงมีส่วนประกอบหลัก ได้แก่ ตัวเข้า-ถอดรหัสเสียงแบบสเตอริโอเบอร์ TLC320AD77C ของบริษัท Texas Instruments [20] สำหรับแปลงสัญญาณแอนะล็อกเป็นข้อมูลดิจิทัลเพื่อส่งให้หน่วยประมวลผล และรับข้อมูลดิจิทัลจากหน่วยประมวลผลเพื่อแปลงเป็นสัญญาณแอนะล็อก และวงจรสำหรับปรับสภาพสัญญาณให้เหมาะสมกับการเชื่อมต่อกับอุปกรณ์ภายนอก และบอร์ดโปรแกรมเอฟพีจีเอที่ทำหน้าที่เป็นวงจรกันชน (Buffer) ระหว่างเอฟพีจีเอและคอมพิวเตอร์ที่ใช้โปรแกรม

6.1.2 <u>การทดสอบการทำงานของตัวต้นแบบ</u>

การทดสอบการทำงานของตัวต้นแบบแบ่งออกได้เป็นสองส่วน ได้แก่ การตรวจสอบความถูกต้องของหน่วยประมวลผลด้วยการจำลองการทำงาน และการทดลองสังเคราะห์วงจรลงในบอร์ดตัวต้นแบบที่ได้ออกแบบ

การตรวจสอบความถูกต้องของหน่วยประมวลผล (ดูรูปที่ 6.2) จะใช้การจำลองการทำงานของแบบจำลองหน่วยประมวลผลที่ถูกออกแบบด้วยภาษา VHDL [17, 18] ร่วมกับแบบจำลองของหน่วยความจำโปรแกรมที่บรรจุโปรแกรมที่ถูกเขียนด้วยภาษาของหน่วยประมวลผลที่ออกแบบแล้วจึงป้อนเวคเตอร์ทดสอบ (Test Vector) ที่เตรียมไว้สำหรับการทดสอบการทำงานแต่ละเงื่อนไข เมื่อหน่วยประมวลผลสามารถทำงานได้ถูกต้องตามที่กำหนดแล้ว จึงนำแบบจำลองไปสังเคราะห์เพื่อทดสอบการทำงานบนเอฟพีจีเอ



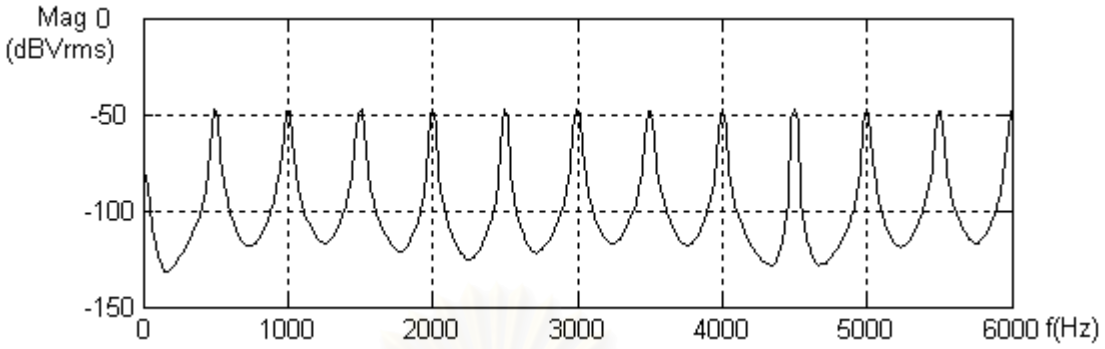
รูปที่ 6.2 ขั้นตอนการจำลองการทำงานของหน่วยประมวลผล

การทดลองสังเคราะห์วงจรลงในบอร์ดตัวต้นแบบ [8] จะนำแบบจำลองของหน่วยประมวลผลมาสังเคราะห์และโปรแกรมลงบนเอฟพีจีเอ และนำโปรแกรมทดสอบที่ถูกเขียนเพื่อทดสอบการทำงานมาโปรแกรมลงในหน่วยความจำ หลังจากนั้นจะป้อนสัญญาณอินพุตเพื่อทดสอบการทำงานของตัวต้นแบบ แล้วจึงวัดสัญญาณเอาต์พุตด้วยเครื่องมือวัดเพื่อพิจารณาความถูกต้องของการทำงาน ในที่นี้จะยกตัวอย่างการทดสอบการทำงานของหน่วยประมวลผลสำหรับงานกรองสัญญาณ

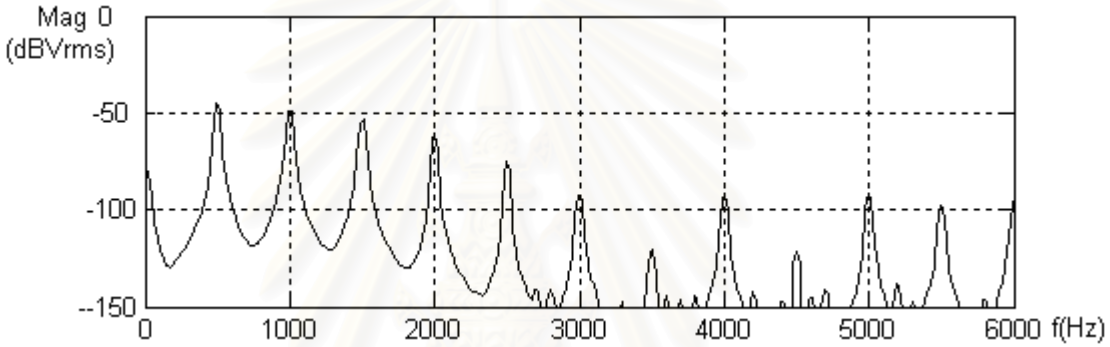
การทดสอบการทำงานของหน่วยประมวลผลสำหรับงานกรองสัญญาณ [13] ในที่นี้จะให้หน่วยประมวลผลสัญญาณดิจิทัลทำงานเป็นวงจรกรองผ่านต่ำ (Low Pass Filter) ที่มีแถบผ่านความถี่ (Pass Band) ตั้งแต่ 0 ถึง 1.5 kHz และมีแถบหยุดความถี่ (Stop Band) ตั้งแต่ 3 kHz เป็นต้นไป โดยมีอัตราการลดทอนสัญญาณที่แถบหยุดความถี่ –60 dB วงจรกรองที่ออกแบบมีโครงสร้างแบบเอฟไอาร์ที่มีความยาว 32 แท็ป การทดสอบการทำงานของตัวกรองในที่นี้จะแบ่งออกเป็นสองส่วน ได้แก่ การจำลองการทำงานของตัวกรองด้วยคอมพิวเตอร์ และการทดสอบการทำงานด้วยตัวต้นแบบที่สร้างขึ้น ผลลัพธ์จากการทดสอบการทำงานทั้งสองส่วนจะถูกนำมาเปรียบเทียบกัน เพื่อพิจารณาว่าหน่วยประมวลผลที่ออกแบบสามารถทำงานได้ถูกต้องหรือไม่

การจำลองการทำงานของตัวกรองด้วยคอมพิวเตอร์ จะนำสัญญาณอินพุตซึ่งเป็นไฟล์สัญญาณเสียงป้อนเข้าไปในโปรแกรม MATLAB ของบริษัท The Mathworks เพื่อนำมาคำนวณสัญญาณเอาต์พุตของตัวกรองเอฟไอาร์ที่ออกแบบเมื่อป้อนสัญญาณอินพุตเข้าไป โดยสัญญาณอินพุตที่ป้อนให้กับวงจรกรองสามารถเขียนได้ดังสมการที่ 6.1 หลังจากนั้นจึงนำสัญญาณอินพุตและเอาต์พุตมาวิเคราะห์หาสเปกตรัมของสัญญาณ โดยสเปกตรัมของสัญญาณอินพุตและเอาต์พุตของตัวกรองที่ได้จากการคำนวณแสดงดังรูปที่ 6.3 และ 6.4 ตามลำดับ

$$x(t) = A(1 + \sin(1000\pi \times t) + \sin(2000\pi \times t) + \sin(3000\pi \times t) + ...) \quad \text{.....(6.1)}$$

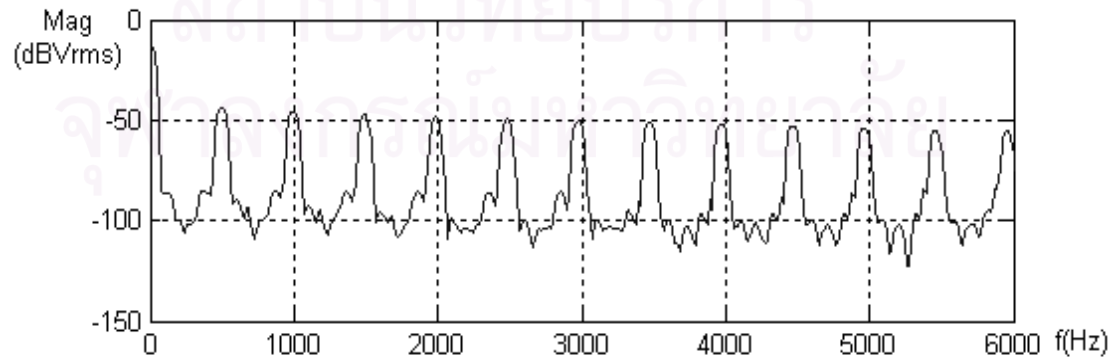


รูปที่ 6.3 สเปกตรัมของสัญญาณอินพุตของวงจรกรองเอฟไอาร์ที่ได้จากการคำนวณ

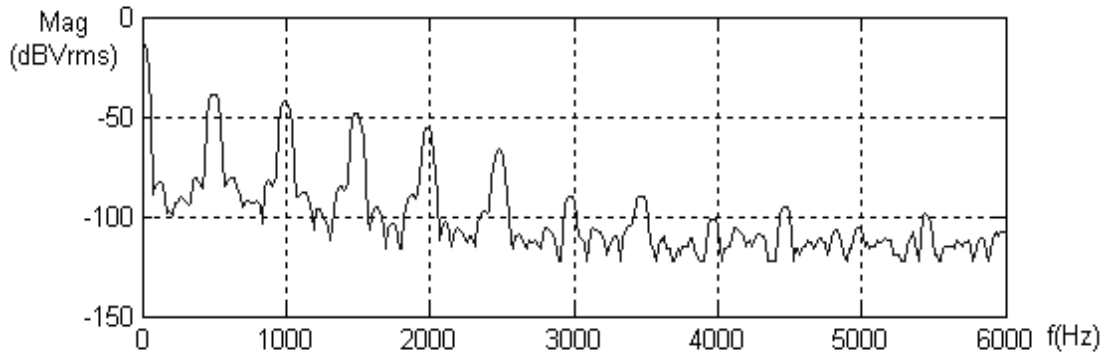


รูปที่ 6.4 สเปกตรัมของสัญญาณเอาต์พุตของวงจรกรองเอฟไอาร์ที่ได้จากการคำนวณ

การทดสอบการทำงานของวงจรกรองด้วยตัวต้นแบบ จะใช้เครื่องวิเคราะห์สเปกตรัมวัดสัญญาณอินพุตที่ป้อนเข้าไปในตัวต้นแบบของหน่วยประมวลผล และวัดสัญญาณเอาต์พุตของตัวกรอง โดยสัญญาณอินพุตที่ป้อนเป็นสัญญาณเดียวกันกับที่ใช้จำลองการทำงาน สัญญาณอินพุตและเอาต์พุตของตัวกรองเมื่อใช้เครื่องวิเคราะห์สเปกตรัมวัดแสดงดังรูปที่ 6.5 และ 6.6 ตามลำดับ



รูปที่ 6.5 สเปกตรัมของสัญญาณอินพุตของวงจรกรองเอฟไอาร์ที่ได้จากการวัด



รูปที่ 6.6 สเปกตรัมของสัญญาณเอาต์พุตของวงจรกรองเอฟไออาร์ที่ได้จากการวัด

เมื่อพิจารณาสเปกตรัมของสัญญาณที่ได้จากการทดสอบทั้งสองส่วน พบว่าสเปกตรัมจากการคำนวณและสเปกตรัมที่ได้จาการวัดจากตัวต้นแบบมีลักษณะคล้ายคลึงกันมาก โดยมีข้อแตกต่างกันเนื่องจากสัญญาณที่ได้จากการวัดจะมีสัญญาณรบกวนขนาดประมาณ –100 dBVrms ปนอยู่ด้วย ซึ่งสัญญาณรบกวนอาจมาจากวงจรที่ใช้ทดสอบ หรือสภาพแวดล้อม จึงทำให้สเปกตรัมของสัญญาณที่วัดคลาดเคลื่อนจากการคำนวณ อย่างไรก็ตามเมื่อพิจารณาผลลัพธ์จากการทดสอบแล้ว พบว่าตัวต้นแบบของหน่วยประมวลผลสัญญาณดิจิทัลสามารถทำงานได้ถูกต้องตามที่ออกแบบ

6.2 ขั้นตอนการออกแบบลายวงจรรวม

แบบจำลองของ D-CHIP ที่ผ่านการจำลองการทำงานและทดสอบด้วยตัวต้นแบบแล้ว จะถูกนำมาออกแบบลายวงจรรวม โดยการออกแบบลายวงจรรวมจะใช้ซอฟต์แวร์สำหรับการออกแบบวงจรรวมของบริษัท Mentor Graphic และ Cadence โดยขั้นตอนการออกแบบ (Design Flow) [11] จะเริ่มต้นตั้งแต่การสังเคราะห์วงจรบนเทคโนโลยีวงจรรวม (Synthesis) การวางและกำหนดเส้นทางลายวงจร (Place and Route : P&R) การตรวจสอบความถูกต้องของวงจรตามกฎการออกแบบ (Design Rule Check : DRC) และการเปรียบเทียบระหว่างลายวงจรรวมและแผนภาพ (Layout Versus Schematic : LVS) ตามลำดับ

6.2.1 การสังเคราะห์วงจรบนเทคโนโลยีวงจรรวม

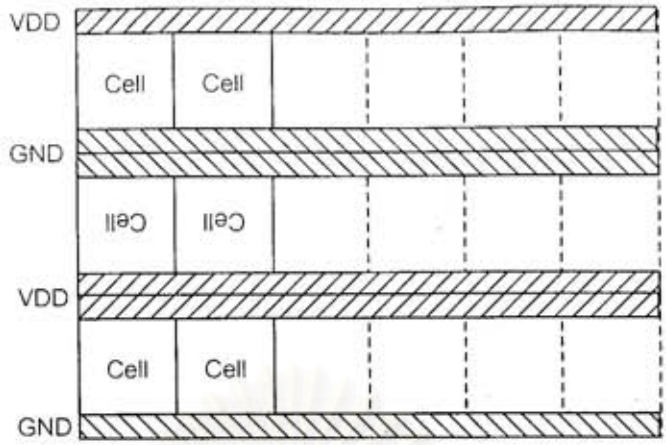
แบบจำลองของ D-CHIP ที่ผ่านการทดสอบความถูกต้องแล้ว จะถูกนำมาสังเคราะห์ลงบนเซลล์มาตรฐานของเทคโนโลยีสารกึ่งตัวนำซีมอสขนาด 0.35 ไมครอน [14] ที่แรงดันไฟเลี้ยง 3.3 โวลต์ ของบริษัท Austriamicrosystems ด้วยโปรแกรม Leonardo Spectrum ของบริษัท Mentor Graphic การสังเคราะห์วงจรของ D-CHIP จะเป็นวงจรออกเป็นสองส่วน ได้แก่ วงจรส่วนที่เป็น

หน่วยประมวลผล และวงจรส่วนที่เป็นหน่วยความจำ การสังเคราะห์วงจรส่วนที่เป็นหน่วยความจำจะถูกนำมาสังเคราะห์ลงในบล็อกมาตรฐานของหน่วยความจำที่ถูกเตรียมขึ้นโดยเฉพาะด้วยโปรแกรมสังเคราะห์หน่วยความจำของบริษัท Austriamicrosystems ดังนั้นแบบจำลองของ D-CHIP ที่นำมาสังเคราะห์จึงไม่จำเป็นต้องอธิบายรายละเอียดภายในของหน่วยความจำ โดยอธิบายเพียงการเชื่อมต่อระหว่างหน่วยประมวลผลและหน่วยความจำเท่านั้น หลังจากสังเคราะห์วงจรลงในเทคโนโลยีสารกึ่งตัวนำแล้ว จะนำผลลัพธ์ที่ได้ซึ่งเป็นไฟล์ในรูปแบบของ Verilog Netlist ไปใช้ในขั้นตอนการวางและกำหนดเส้นทางลายวงจรต่อไป

6.2.2 การวางและกำหนดเส้นทางลายวงจร

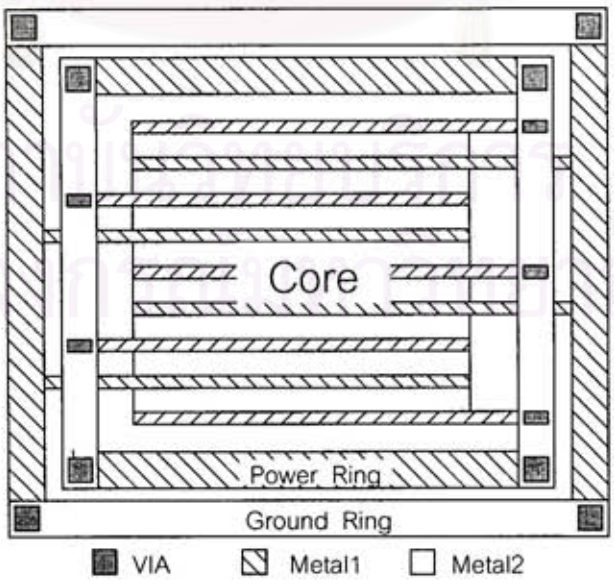
ไฟล์ในรูปแบบของ Verilog Netlist ที่ได้จากการสังเคราะห์วงจรจะถูกนำมาวางและกำหนดเส้นทางลายวงจรด้วยโปรแกรม Silicon Ensembler ของบริษัท Cadence ขั้นตอนการวางและกำหนดเส้นทางลายวงจรจะเริ่มต้นตั้งแต่การนำเข้า (Import) ไฟล์ Verilog Netlist ของ D-CHIP เข้าไปในโปรแกรม โดยจำเป็นต้องนำเข้าไฟล์รายละเอียดของเซลล์มาตรฐานของเทคโนโลยีซีมอส 0.35 ไมครอนควบคู่ไปด้วย หลังจากนำเข้าไฟล์แล้วจะเป็นการกำหนดพารามิเตอร์โดยรวมของวงจรรวม ซึ่งได้แก่ อัตราส่วนระหว่างความสูงและความกว้างของวงจรรวม (Aspect Ratio) อัตราส่วนการใช้งานพื้นที่ของแถวของการวางเซลล์มาตรฐาน (Row Utilization) ระยะห่างระหว่างแถวของเซลล์มาตรฐาน (Row Spacing) ระยะห่างระหว่างอินพุต-เอาต์พุตและแกนของวงจรรวม (I/O to Core Spacing) และลักษณะการจัดเรียงเซลล์มาตรฐานเป็นต้น เมื่อกำหนดพารามิเตอร์ของวงจรรวมแล้วจึงเข้าสู่ขั้นตอนการวางและกำหนดเส้นทางลายวงจรตามลำดับ

เนื่องจากเทคโนโลยีสารกึ่งตัวนำที่เลือกใช้สามารถลากลายวงจรด้วยชั้นโลหะถึง 4 ชั้น โดยปรกติการเชื่อมต่อภายในเซลล์มาตรฐานจะใช้การลากลายวงจรบนชั้นโลหะที่หนึ่ง และชั้นโพลีซิลิคอน (Metal 1 และ Polysilicon) เท่านั้น ดังนั้นการจัดเรียงเซลล์มาตรฐานจะให้ส่วนจ่ายไฟซึ่งอยู่ด้านบนและส่วนกราวนด์ ซึ่งอยู่ด้านล่างของเซลล์มาตรฐานในแต่ละแถวเชื่อมต่อกันโดยตรง (ดูรูปที่ 6.7) โดยที่ไม่ได้เว้นระยะระหว่างแถวของเซลล์สำหรับการเชื่อมต่อลายวงจรระหว่างเซลล์มาตรฐาน โดยการเชื่อมต่อลายวงจรระหว่างเซลล์มาตรฐานจะถูกกระทำที่ชั้นโลหะที่สอง สาม และสี่ (Metal 2 3 และ 4) ซึ่งคร่อมอยู่บนเซลล์มาตรฐาน และบนลายวงจรของส่วนจ่ายไฟเลี้ยงและกราวนด์เท่านั้น การวางเซลล์มาตรฐานแบบนี้จะทำให้การใช้พื้นที่ของวงจรลดลงไปมาก



รูปที่ 6.7 ลักษณะการวางเซลล์มาตรฐานของ D-CHIP

การวางเซลล์มาตรฐานในส่วนของหน่วยประมวลผลจะให้โปรแกรม Silicon Ensembler วางให้อย่างอัตโนมัติ ส่วนบล็อกมาตรฐานของหน่วยความจำผู้ออกแบบจะเป็นผู้กำหนดการวางด้วยตนเอง โดยพิจารณาจากระยะทางและการเชื่อมต่อกับเซลล์รอบข้าง หลังจากวางเซลล์มาตรฐานแล้ว จะเป็นการสร้างลายวงจรส่วนจ่ายไฟซึ่งมีลักษณะเป็นวงแหวนล้อมรอบแกนของวงจรรวม (Core Ring) โดยแบ่งเป็นวงแหวนของไฟเลี้ยง (Power Ring) และวงแหวนของกราวนด์ (Ground Ring) (ดูรูปที่ 6.8) เมื่อสร้างลายวงจรส่วนจ่ายไฟแล้วจะเป็นการกำหนดเส้นทางและเชื่อมต่อวงจรส่วนที่เหลือจนครบ โดยหลังจากที่วางและกำหนดเส้นทางลายวงจรเสร็จแล้ว จะนำผลลัพธ์ที่ได้ส่งออก (Export) เป็นไฟล์ในรูปแบบ GDSII เพื่อนำไปใช้ในโปรแกรมสำหรับออกแบบและแก้ไขลายวงจรรวมอื่นๆ ต่อไป



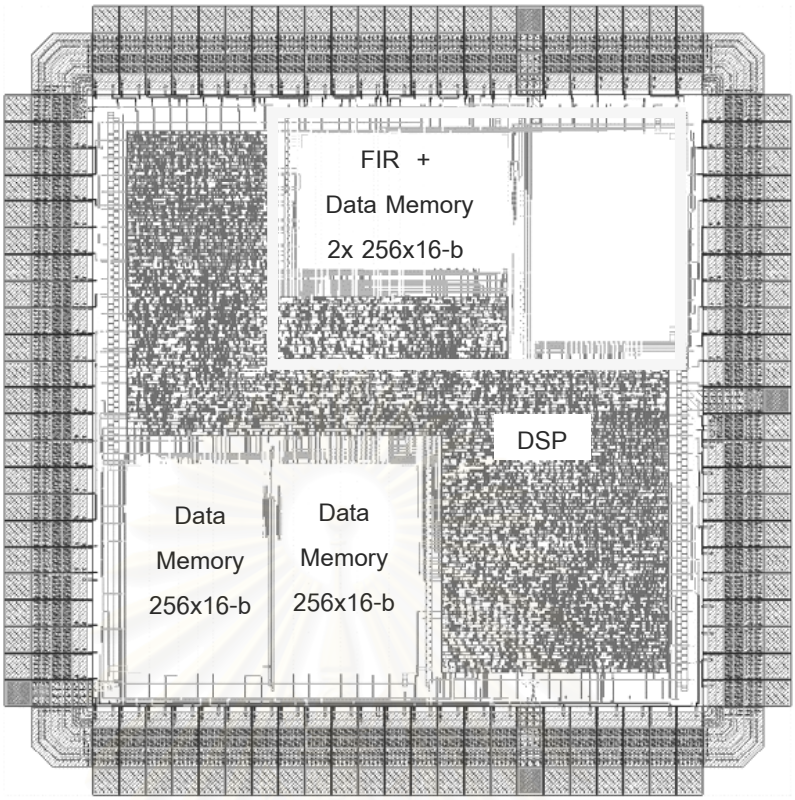
รูปที่ 6.8 ตัวอย่างลายวงจรส่วนจ่ายไฟที่มีลักษณะเป็นวงแหวน

6.2.3 การเปรียบเทียบระหว่างลายวงจรรวมและแผนภาพ

ไฟล์ในรูปแบบ GDSII ที่ได้จากขั้นตอนการวางและกำหนดเส้นทางลายวงจรจะถูกนำเข้าสู่โปรแกรม Virtuoso ของบริษัท Cadence ซึ่งเป็นโปรแกรมสำหรับออกแบบและแก้ไขลายวงจรรวม ลายวงจรรวมที่นำเข้าจะถูกนำมาสกัด (Extract) เอา Netlist ของลายวงจรรวมออกมา โดยการสกัดจะลงไปถึงระดับของเซลล์มาตรฐาน หรือระดับมาโคร (Macro Level) เท่านั้น เนื่องจากบริษัท Austriamicrosystems ไม่ได้เปิดเผยรายละเอียดภายในเซลล์มาตรฐาน หลังจากนั้นก็จะนำเข้าข้อมูลแผนภาพของ D-CHIP ซึ่งในที่นี้จะใช้ไฟล์ Verilog Netlist ที่ได้จากขั้นตอนการสังเคราะห์วงจรแทน หลังจากนั้นจะนำข้อมูล Netlist ของลายวงจรรวมและแผนภาพมาเปรียบเทียบกัน โดยการเปรียบเทียบจะพิจารณาการเชื่อมต่อระหว่างเซลล์มาตรฐาน และชนิดของเซลล์ว่าแตกต่างกันเพียงใด หากไม่มีความแตกต่างแสดงว่าลายวงจรที่ได้มาจากขั้นตอนที่ผ่านมามีความถูกต้อง หากผลการเปรียบเทียบแสดงว่า Netlist ทั้งสองมีความแตกต่างกัน ผู้ออกแบบจำเป็นต้องพิจารณาและแก้ไขด้วยตนเองจนกระทั่งผลการเปรียบเทียบออกมาตรงกัน

6.2.4 การตรวจสอบความถูกต้องของวงจรตามกฎการออกแบบ

หลังจากลายวงจรผ่านการเปรียบเทียบความถูกต้องของการเชื่อมต่อ จะถูกนำมาวางและเชื่อมต่อเข้ากับแพ็ดอินพุตและเอาต์พุต (I/O Pad) หลังจากนั้นจะถูกนำมาตรวจสอบความถูกต้องตามกฎการออกแบบ [15] กฎการออกแบบจะขึ้นอยู่กับกระบวนการผลิตของวงจรรวมที่เลือกใช้ การตรวจสอบความถูกต้องจะกระทำจนถึงระดับของเซลล์มาตรฐานเช่นเดียวกับการเปรียบระหว่างลายวงจรรวมกับแผนภาพ ความผิดพลาดที่เกิดขึ้นในขั้นตอนนี้ ผู้ออกแบบจำเป็นต้องแก้ไขลายวงจรรวมด้วยตนเองทั้งหมด นอกจากนี้ลายวงจรที่ถูกแก้ไขแล้วจะถูกนำมาเปรียบเทียบกับแผนภาพ (LVS) อีกที เพื่อป้องกันความผิดพลาดที่เกิดจากการแก้ไขลายวงจร ลายวงจรรวมของ D-CHIP ที่ถูกแก้ไขจนผ่านการตรวจสอบความถูกต้องของวงจรตามกฎการออกแบบแสดงดังรูปที่ 6.9



รูปที่ 6.9 ลายวงจรรวมของ D-CHIP

6.3 การวิเคราะห์ลายวงจรรวม

6.3.1 พื้นที่ของลายวงจรรวม

พื้นที่ของลายวงจรรวมของ D-CHIP ทั้งหมดประมาณ 5.23 ตารางมิลลิเมตร (รวมหน่วยความจำ แต่ไม่รวมแพ็ด) ซึ่งสามารถประมาณความซับซ้อนของวงจรได้เทียบเท่า 34,000 เกต (ไม่รวมหน่วยความจำ) พื้นที่ของวงจรรวมแบ่งได้เป็นสองส่วนใหญ่ๆ ได้แก่ พื้นที่ของหน่วยประมวลผล ซึ่งประกอบด้วยวงจรย่อยหลายๆส่วน พื้นที่ของหน่วยความจำ ส่วนที่เหลือเป็นพื้นที่เนื่องจากลายวงจรส่วนจ่ายไฟ ลายวงจรสำหรับการเชื่อมต่อระหว่างเซลล์ และระยะห่างระหว่างอินพุต-เอาต์พุตและแกนของวงจรรวม พื้นที่เซลล์มาตรฐานของวงจรย่อยต่างๆ ในหน่วยประมวลผลและเซลล์หน่วยความจำแสดงรายละเอียดดังตารางที่ 6.1

ตารางที่ 6.1 พื้นที่ของเซลล์มาตรฐานของวงจรย่อยต่างๆ และเซลล์หน่วยความจำ

เซลล์ของวงจร	พื้นที่ของวงจร (ตารางมิลลิเมตร)	เปอร์เซ็นต์ต่อพื้นที่เซลล์ทั้งหมด
หน่วยควบคุมโปรแกรม และตัวถอดรหัสคำสั่ง	0.083	3.01
หน่วยกำเนิดตำแหน่งข้อมูล	0.084	3.05
ตัวกรองเอฟไออาร์ (ไม่รวมหน่วยความจำ)	0.313	11.38
หน่วยคูณและสะสม	0.246	8.94
ดีเอ็มเอ	0.050	1.81
หน่วยคณิตศาสตร์และตรรกะ	0.047	1.70
รีจิสเตอร์ใช้งานพิเศษและอุปกรณ์บริวาร	0.141	5.13
หน่วยเลือกตัวถูกดำเนินการและแฟ้มรีจิสเตอร์	0.130	4.72
หน่วยความจำ	1.656	60.21

เนื่องจากในขั้นตอนการวางเซลล์มาตรฐานได้กำหนดให้อัตราส่วนการใช้งานพื้นที่ของแถวของการวางเซลล์มาตรฐานมีค่าประมาณ 70 เปอร์เซ็นต์ โดยเว้นที่ว่างสำหรับการลากลายของวงจรเชื่อมต่อระหว่างเซลล์มาตรฐาน และได้เตรียมพื้นที่สำหรับการลากลายวงจรเป็นวงแหวนสำหรับการจ่ายไฟ โดยกำหนดให้ความกว้างของลายวงจรวงแหวนที่ล้อมรอบแกนของวงจร (Core Ring Width) มีขนาด 15 ไมครอน และกำหนดให้ความกว้างของลายวงจรวงแหวนที่ล้อมรอบบล็อกหน่วยความจำมีขนาด 7 ไมครอน นอกจากนี้ได้กำหนดให้ระยะห่างระหว่างอินพุต-เอาต์พุตและแกนของวงจรรวมมีค่า 100 ไมครอนเพื่อเว้นระยะให้วงแหวนและสายสัญญาณจากแกนของวงจรออกไปยังอินพุต-เอาต์พุตสำหรับนำไปเชื่อมต่อกับแพ็ดต่อไป

6.3.2 <u>เวลาล่าช้า</u>

ความถี่การทำงานของหน่วยประมวลผล D-CHIP ขึ้นกับเส้นทางวิกฤติของวงจร โดยจากการพิจารณาเส้นทางที่มีความล่าช้ามากที่สุดของหน่วยประมวลผลจำนวน 2000 เส้นทาง พบว่าเส้นทางทั้งหมดอยู่ภายในหน่วยคูณและสะสม โดยเวลาล่าช้าของเส้นทางดังกล่าวจะมีค่าตั้งแต่ประมาณ 8.29 นาโนวินาที ซึ่งถือว่าเป็นเส้นทางวิกฤติของหน่วยประมวลผล จนถึงประมาณ 6.98 นาโนวินาที

ความล่าช้าเนื่องจากหน่วยความจำจะขึ้นเวลาของวงรอบการทำงาน (Cycle Time) และเวลาเข้าถึงข้อมูล (Access Time) ของหน่วยความจำ เนื่องจากการทำงานของหน่วยความจำจะต้องสามารถเข้าถึงข้อมูลได้อย่างต่อเนื่องในทุกๆรอบของสัญญาณนาฬิกา ดังนั้นความถี่หรือคาบเวลาของสัญญาณนาฬิกาที่ใช้ต้องมีค่าไม่เกินเวลาของวงรอบการทำงาน แต่ทว่าหน่วยความจำที่ใช้อยู่บนชิป จึงทำให้มีเวลาของวงรอบการทำงาน และเวลาเข้าถึงข้อมูลน้อยเพียง 4.89 และ 2.83 นาโนวินาทีตามลำดับ เนื่องจากวงจรที่เชื่อมต่อกับหน่วยความจำ ซึ่งได้แก่ หน่วยเลือกตัวถูกดำเนินการและแฟ้มรีจิสเตอร์ และดีเอ็มเอ มีความล่าช้าเนื่องจากเส้นทางที่มีการเชื่อมต่อกับหน่วยความจำไม่เกิน 1 นาโนวินาที ดังนั้นเวลาล่าช้ารวมของเส้นทางที่มีหน่วยความจำจะมีค่าไม่เกินเวลาล่าช้าของเส้นทางวิกฤติของหน่วยประมวลผล

6.3.3 <u>การกินกำลังงาน</u>

การกินกำลังงานของวงจรรวมจะสามารถคำนวณได้จากข้อมูลการกินกำลังงานของเซลล์มาตรฐานที่แรงดันไฟเลี้ยง 3.3 โวลต์ ของเทคโนโลยีสารกึ่งตัวนำซีมอส C35 ขนาด 0.35 ไมครอน ของบริษัท Austriamicrosystems [14] ข้อมูลการกินกำลังงานอยู่ในรูปแบบของอัตราการกินกำลังงานของเซลล์ที่ความถี่การสวิตช์อินพุต 1 MHz ข้อมูลการใช้เซลล์มาตรฐานของ D-CHIP สามารถแบ่งเซลล์มาตรฐานได้เป็นสองประเภท ได้แก่ เซลล์ประเภทวงจรเชิงจัดหมู่ เช่น เกตชนิดต่างๆ และเซลล์ประเภทวงจรเชิงลำดับ เช่น ฟลิปฟล็อป และรีจิสเตอร์ การกินพลังงานของเซลล์มาตรฐานประเภทวงจรเชิงจัดหมู่จะพิจารณาจากความถี่ของการเปลี่ยนแปลงอินพุต ส่วนการกินพลังงานของเซลล์มาตรฐานประเภทวงจรเชิงลำดับจะพิจารณาจากความถี่ของสัญญาณนาฬิกาที่ป้อนให้เซลล์ ส่วนการกินกำลังงานของหน่วยความจำจะพิจารณาได้จากความถี่ของการเข้าถึงข้อมูลภายในหน่วยความจำ การกินกำลังงานของเซลล์แต่ละประเภทแสดงดังตารางที่ 6.2

ตารางที่ 6.2 อัตราการกินกำลังงานของเซลล์แต่ละประเภท

ประเภทของเซลล์มาตรฐาน	อัตราการกินกำลังงาน (μW/MHz)
เซลล์ประเภทวงจรเชิงจัดหมู่	3245
เซลล์ประเภทวงจรเชิงลำดับ	2013
หน่วยความจำ	2798

เมื่อพิจารณาการใช้งานเซลล์แต่ละประเภท พบว่าเซลล์ประเภทวงจรเชิงจัดหมู่ร้อยละ 50 อยู่ภายในหน่วยคูณและสะสม ซึ่งจะทำงานเมื่อหน่วยประมวลผลดำเนินการคำสั่งในกลุ่มการคูณ

และสะสม เซลล์ประเภทวงจรเชิงลำดับจะทำงานเกือบตลอดเวลาเนื่องจากขาสัญญาณของเซลล์เชื่อมต่อกับสัญญาณนาฬิกาโดยตรง ส่วนหน่วยความจำจะทำงานเมื่อดำเนินการคำสั่งที่มีแอดเดรสชิงโมดที่ติดต่อกับหน่วยความจำ เช่น การอ่านและเขียนข้อมูลกับหน่วยความจำ จากการโปรแกรมบรรทัดฐานส่วนใหญ่พบว่า คำสั่งภายในลูปส่วนใหญ่เป็นคำสั่งในกลุ่มการคูณและสะสม และคำสั่งกลุ่มการโอนย้ายข้อมูล โดยข้อมูลที่ใช้ประมวลผลมาจากหน่วยความจำข้อมูลภายใน เมื่อพิจารณาคำสั่งภายในลูปของโปรแกรมบรรทัดฐานจะสามารถแบ่งสัดส่วนการใช้งานของเซลล์แต่ละประเภทโดยประมาณได้ดังตารางที่ 6.3

ตารางที่ 6.3 สัดส่วนการใช้งานของเซลล์แต่ละประเภท

ประเภทของเซลล์มาตรฐาน	สัดส่วนการใช้งาน
เซลล์ประเภทวงจรเชิงจัดหมู่	0.50
เซลล์ประเภทวงจรเชิงลำดับ	0.90
หน่วยความจำ	0.75

จากข้อมูลในตารางที่ 6.2 และ 6.3 สามารถนำมาคำนวณการกินกำลังงานเฉลี่ยของหน่วยประมวลผลอย่างหยาบด้วยการหาค่าเฉลี่ยแบบถ่วงน้ำหนักดังสมการ 6.2 ได้เท่ากับ 5.7 mW/MHz ทั้งนี้ค่าที่ได้จากการคำนวณจะมีค่ามากกว่าการกินกำลังงานเฉลี่ยที่ได้จากการวัดจริง

$$Power = (0.5 \times 3245) + (0.9 \times 2013) + (0.75 \times 2798) \approx 5.7 mW / Mhz \quad \text{.....} (6.2)$$

บทที่ 7

บทสรุป

7.1 สรุป

หน่วยประมวลผลสัญญาณดิจิทัล D-CHIP เป็นหน่วยประมวลผลสัญญาณดิจิทัลแบบทศ-นิยมคงที่ 16 บิต ที่มีตัวกรองเอฟไออาร์ภายในที่สามารถปรับความยาวได้ และอุปกรณ์บริวารที่จำเป็นอื่นๆ ได้แก่ ตัวตั้งเวลา วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S ดีเอ็มเอ และพอร์ตอินพุต-เอาต์พุต หน่วยประมวลผลมีโครงสร้างแบบไปป์ไลน์ 5 ขั้นตอนเพื่อให้วงจรภายในสามารถทำงานได้อย่างรวดเร็ว และมีสถาปัตยกรรมหน่วยความจำแบบฮาร์วาร์ด เนื่องจากงานกรรมวิธีสัญญาณดิจิทัลจำเป็นต้องประมวลผลข้อมูลจำนวนมากอย่างต่อเนื่อง ซึ่งข้อมูลส่วนใหญ่จะถูกเก็บอยู่ภายในหน่วยความจำ ชุดคำสั่งของ D-CHIP มีลักษณะเป็นชุดคำสั่งแบบริสค์ โดยมีการทำงานของคำสั่งไม่ซับซ้อน สามารถดำเนินการได้เร็ว และเนื่องจากหน่วยประมวลผลมีโครงสร้างแบบไปป์ไลน์ ทำให้สามารถประมวลผลคำสั่งส่วนใหญ่ได้ภายในหนึ่งวงรอบสัญญาณนาฬิกา

ตัวกรองเอฟไออาร์ที่ออกแบบมีโครงสร้างของหน่วยคูณและสะสมคล้ายกับหน่วยประมวลผลกลางทำให้สามารถทำงานได้สองลักษณะ การทำงานในลักษณะที่หนึ่ง ตัวกรองสามารถทำงานขนานอย่างอิสระจากการทำงานของหน่วยประมวลผลกลาง ส่วนการทำงานในลักษณะที่สอง ทรัพยากรภายในตัวกรอง ซึ่งได้แก่ หน่วยคูณและสะสม และหน่วยความจำ สามารถเรียกใช้โดยตรงได้จากหน่วยประมวลผลกลาง เมื่อหน่วยประมวลผลกลางทำงานร่วมกับตัวกรองจะสามารถคำนวณตัวกรองเอฟไออาร์แบบปรับตัวแบบกำลังสองน้อยที่สุดได้ภายใน 1.5N+26 วงรอบคำสั่ง

ตัวต้นแบบของหน่วยประมวลผลสัญญาณดิจิทัลถูกออกแบบและจำลองการทำงานด้วยภาษา VHDL แล้วจึงถูกนำมาสังเคราะห์บน เอฟพีจีเอ Spartan II ของบริษัท Xilinx บอร์ดต้นแบบประกอบไปด้วย บอร์ดเอฟพีจีเอ บอร์ดหน่วยความจำ บอร์ดของตัวเข้า-ถอดรหัสสัญญาณเสียง และบอร์ดโปรแกรมเอฟพีจีเอ การจำลองการทำงานของวงจรจะใช้การสร้างแบบจำลองของหน่วยความจำโปรแกรมที่บรรจุด้วยโปรแกรมที่ใช้ทดสอบ แล้วนำมาเชื่อมต่อกับแบบจำลองของหน่วยประมวลผล เพื่อนำมาจำลองการทำงานและตรวจสอบความถูกต้องร่วมกับเวคเตอร์ทดสอบที่เตรียมไว้ ส่วนการทดสอบด้วยตัวต้นแบบจะนำวงจรที่ถูกจำลองการทำงานมาสังเคราะห์ลงเอฟพีจี

เอ แล้วจึงป้อนสัญญาณทดสอบเพื่อวัดผลลัพธ์ของการประมวลผลด้วยเครื่องมือวัด เช่น ออสซิล-โลสโคป และเครื่องวิเคราะห์สเปกตรัม

การออกแบบลายวงจรรวมจะใช้ซอฟต์แวร์สำหรับการออกแบบวงจรรวมของบริษัท Mentor Graphic และ Cadence โดยขั้นตอนการออกแบบจะเริ่มต้นตั้งแต่การสังเคราะห์วงจรบนเทคโนโลยีวงจรรวม การวางและกำหนดเส้นทางลายวงจร การเปรียบเทียบระหว่างลายวงจรและแผนภาพ และการตรวจสอบความถูกต้องของวงจรตามลำดับ ลายวงจรรวมถูกออกแบบด้วยเซลล์มาตรฐานดิจิทัลบนเทคโนโลยีซีมอส 0.35 ไมครอนของบริษัท Austriamicrosystems จากผลการวิเคราะห์ลายวงจรรวมในหัวข้อที่ 6.3 วงจรรวมใช้พื้นที่ประมาณ 5.23 ตารางมิลลิเมตร (รวมหน่วยความจำ) ซึ่งสามารถประมาณความซับซ้อนของวงจรได้เทียบเท่า 34,000 เกต (ไม่รวมหน่วยความจำ) และกินกำลังไฟประมาณ 5.7 mW/MHz ที่แรงดันไฟเลี้ยง 3.3 โวลต์

7.2 ข้อเสนอแนะ

1. ปรับปรุงรหัสดำเนินการให้กว้างกว่า 16 บิต เพื่อให้หน่วยประมวลผลสามารถรองรับแอดเดรสซิงโมดได้หลากหลาย และสามารถอ้างตัวถูกดำเนินการได้มากขึ้น
2. ปรับปรุงโครงสร้างของวงจรคูณให้สามารถคำนวณผลลัพธ์ได้ภายในหนึ่งรอบสัญญาณนาฬิกา โดยไม่ใช้โครงสร้างของวงจรคูณแบบไปป์ไลน์ ซึ่งจะทำให้สมรรถนะโดยรวมของหน่วยประมวลผลดีขึ้น เนื่องจากเวลาที่ต้องรอผลลัพธ์จากการคูณน้อยลง
3. ขยายความกว้างของตัวสะสมให้มากกว่า 32 บิต เพื่อเพิ่มพิสัยพลวัตของการคำนวณให้มากขึ้น โดยต้องออกแบบวงจรในส่วนของตัวสะสมให้สามารถทำงานได้รวดเร็ว และไม่ทำให้หน่วยประมวลผลทำงานช้าลง
4. ออกแบบวงจรส่วนที่เป็นเส้นทางวิกฤติ ซึ่งได้แก่ วงจรคูณ ในระดับทรานซิสเตอร์เอง เพื่อทำให้วงจรมีเวลาล่าช้าน้อยกว่าการออกแบบด้วยเซลล์มาตรฐาน ซึ่งจะทำให้หน่วยประมวลผลสามารถทำงานได้ที่ความถี่สัญญาณนาฬิกาสูงขึ้น

รายการอ้างอิง

1. David J. Lilja. Trend in High-Performance Computer Architecture. Department of Electrical Engineering, University of Minesota, (April 1996)
2. Phil Lapsley,Jeff Bie ,Amit Shoham,Edward A. Lee. DSP Processor Fundamental. IEEE PRESS, 1997
3. Edward A. Lee. Programmable DSP Architectures:Part 1. IEEE ASSP Magazine (October 1988) : 4-19.
4. Edward A. Lee. Programmable DSP Architectures:Part 2. IEEE ASSP Magazine (October 1989) : 4-14.
5. Yamin Li and Wanming Chu. Aizup - A Pipelined Processor Design and Implementation on XILINX FPGA Chip. IEEE Symposium on FPGAs for Custom Computing Machines (17-19 April 1996) : 98-106.
6. Micheal Dolle, Satwinder Jhand, Walter Lehner, Otto Muller. A 32-b RISC/DSP Microprocessor with Reduced Complexity. IEEE Journal of Solid-State Circuit vol. 32, NO. 7 (July 1997)
7. Jean Joel, Vojin G. Oklobdzija. New Pipelined Architecture for DSP. IEEE Proceeding of ASILOPMAR-29 (1996)
8. Michael Gschwind, Valentina Salapura, and Dietmar Maurer. FPGA Prototyping of a RISC Processor Core for Embedded Applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems vol. 9, NO. 2 (April 2001) : 241-250
9. Xilinx Incorporation. The Programmable Logic Data Book 2000. United State of America, 2000.
10. คณิตพงศ์ เพ็งวัน. การออกแบบวงจรรวมของไมโครคอนโทรลเลอร์ขนาด 16 บิต สำหรับเครื่องรับโทรทัศน์. วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย, 2545.
11. Wayne Wolf . Modern VLSI Design. New Jersey, Prentice Hall PTR, 1998.
12. Philips Semiconductors. The I^2S Bus Specification. (June 5, 1996).
13. Alan, V.O., Ronald, W.S. and John, R.B. Discrete-Time Signal Processing. 2nd ed. New Jersey : Prentice-Hall, 1999.

14. Austriamicrosystems. 0.35 um CMOS C35 Digital Standard Cell Databook. (January 2003).

15. Austriamicrosystems. 0.35 um CMOS C35 Design Rules. (April 2003).

16. V. Carl Hamacher, Zvonko G. Vranesic, Safwat G. Zaky. Computer Organization. Singapore, Mc Graw Hill, 1996.

17. Charles H. Roth, Jr. Digital Systems Design Using VHDL. Boston, PWS Publishing Company, 1998.

18. Zainalabedin Navabi. VHDL. New York, Mc Graw Hill, 1998.

19. Johnny Pihl, Espen Sand. Arithmetic Module Generator for High Performance VLSI Designs. Available from: http://www.fysel.ntnu.no/modgen, Norwegian University of Science and Technology, 1998.

20. Texas Instruments Incorporation. TLC320AD77C 24-Bit 96 kHz Stereo Audio Codec Data Manual. Available from: http://www.ti.com, 1999.

21. Integrated Device Technology, Inc. IDT71V256SA Low Power 3.3V Fast SRAM 256K Data Sheet. (Feb, 2001).

22. Motorola Incorporation. DSP56300 Family Manual. Available from: http://ewww.motorola.com/, 2001.

23. Texas Instruments Incorporation. C54x DSP Benchmarks. Available from: http://www.ti.com.

24. Texas Instruments Incorporation. C62x DSP Benchmarks. Available from: http://www.ti.com.

25. M. Alidina, G. Burns, C. Holmqvist, E. Morgan, D. Rhodes, S. Simanapalli, M. Thierbach. DSP16000: A High Performance, Low Power Dual-MAC DSP Core for Communication Application. IEEE Custom Intregrated Circuit Conference (1998) : 119-122.

ภาคผนวก

ภาคผนวก ก

โปรแกรมบรรทัดฐานของ D-CHIP

```
------------------------------------------------------------
1. Real multiply
------------------------------------------------------------

                mulu    *x0,*y0
                nop
                nop
                nop
                nop
                stix    acch,*x1

                cycle = 6

------------------------------------------------------------
2. N Real multiply
------------------------------------------------------------

                mulu    *x0+,*y0+
                mulu    *x0+,*y0+

                srpt    #(N-4)/4
                erpt    loop_end

loop_start:     stix    acch,*x1+
                mulu    *x0+,*y0+
                stix    acch,*x1+
                mulu    *x0+,*y0+
                stix    acch,*x1+
                mulu    *x0+,*y0+
                stix    acch,*x1+
loop_end:       mulu    *x0+,*y0+

                mulu    *x0+,*y0+
                mulu    *x0+,*y0+
                stix    acch,*x1+
                nop
                stix    acch,*x1+
                stix    acch,*x1+
                stix    acch,*x1+


                cycle = 2N+4

------------------------------------------------------------
3. Real Update
------------------------------------------------------------

                ldiy    r0,*y0
                nop
                madd    *x1,*y1,rreg
                nop
                nop
                nop
                nop
                stix    acch,*x0

                cycle = 8



------------------------------------------------------------
4. N Real Update
------------------------------------------------------------

                ldiy    r0,*y0+
```

```
                ldiy    r0,*y0+
                madd    *x1+,*y1+,rreg

                srpt    #(N-2)/2
                erpt    loop_end

loop_start:     madd    *x1+,*y1+,rreg
                ldiy    r0,*y0+
                stix    acch,*x0+
                madd    *x1+,*y1+,rreg
                ldiy    r0,*y0+
loop_end:       stix    acch,*x0+

                madd    *x1+,*y1+,rreg
                nop
                stix    acch,*x0+
                nop
                nop
                stix    acch,*x0+

                cycle = 3N+6
```

---

5. Real Correlation

```
                muls    *x0+,*y0+

                srpt    #(N-1)
                erpt    loop_end

loop_end:       macs    *x0+,*y0+

                nop
                nop
                nop
                nop
                ldio    r0,acch

                cycle = N+8
```

---

6. Real Correlation with FIR H/W (type 1)

```
                ldiy    r0,*y
                ldio    r0,ldata
                stfs    rreg
                stiy    rreg,*y0+

                firstart

                muls    *x0+,*y0+

                srpt    #(N-2)/2
                erpt    loop_end
loop_end:       macs    *x0+,*y0+

                nop
                nop
                nop
                ldio    r1,facch
                ldio    r0,acch
                add     r0,r1

                cycle = 0.5N+10
```

---
7. Real Correlation with FIR H/W (type 2)

```
                ldiy    r0,*y
                ldio    r0,ldata
                stis    rreg,*y0
                stiy    rreg,*y0+

                dmuls   *x0+,*y0+

                srpt    #(N-2)/2
                erpt    loop_end
loop_end:       dmacs   *x0+,*y0+

                nop
                nop
                nop
                nop
                ldio    r0,acch
                ldio    r1,facch
                add     r0,r1

                cycle = 0.5N+10
```

---
8. Sum of Square

```
                ldix    r0,*x0+
                muls    r0,r0

                srpt    #N-1
                erpt    loop_end
loop_start:     ldix    r0,*x0+
loop_end:       macs    r0,r0

                nop
                nop
                nop
                nop
                ldio    r0,acch
                ldio    r1,accl

                cycle = 2N+11
```

---
9. N-Tap Real FIR

```
                ldio    r0,ldata
                nop
                stiy    rreg,*y0+

                muls    *x0+,*y0+

                srpt    #(N-1)
                erpt    loop_end
loop_end:       macs    *x0+,*y0+

                nop
                nop
                nop
                nop
                ldio    r0,acch

                cycle = N+11
```

---
10. N-Tap Real FIR with FIR H/W (Type 1)

```
                ldiy    r0,*y
                ldio    r0,ldata
                stfs    rreg
                stiy    rreg,*y0+

                firstart

                muls    *x0+,*y0+

                srpt    #(N-2)/2
                erpt    loop_end
loop_end:       macs    *x0+,*y0+

                nop
                nop
                nop
                ldio    r1,facch
                ldio    r0,acch
                add     r0,r1

                cycle = 0.5N+14
```

---
11. N-Tap Real FIR with FIR H/W (Type 2)

```
                ldiy    r0,*y
                ldio    r0,ldata
                stis    rreg,*y0
                stiy    rreg,*y0+

                dmuls   *x0+,*y0+

                srpt    #(N-2)/2
                erpt    loop_end
loop_end:       dmacs   *x0+,*y0+

                nop
                nop
                nop
                nop
                ldio    r0,acch
                ldio    r1,facch
                add     r0,r1

                cycle = 0.5N+14
```

---
12. Second order Real biquad IIR

```
                ldio    r0,ldata
                xor     r1,r1
                setf    r1,15

                muls    r0,r1
                macs    r0,r1

                msbs    *x0+,*y0+
                msbs    *x0-,*y0+
                macs    *x0+,*y0+
```

```
                macs    *x0,*y0

                ldix    r0,*x-
                stix    r0,*x+
                stix    acch,*x0
                nop
                stdx    acch,@result

                cycle = 14
```

---

13. N Cascaded Second order Real biquad IIR

```
                ldio    r0,ldata
                xor     r1,r1
                setf    r1,15

                muls    r0,r1
                macs    r0,r1

                srpt    #N-1
                erpt    loop_end
loop_start:     msbs    *x0+,*y0+
                ldix    r0,*x
                msbs    *x0-,*y0+
                macs    *x0+,*y0+
loop_end:       macs    *x0,*y0+

                nop
                stdx    acch,@result

                cycle = 7N+10
```

---

14. N tap LMS adaptive FIR

```
                ldio    r0,ldata
                nop
                stiy    rreg,*y0+

                muls    *x0+,*y0+

                srpt    #(N-1)
                erpt    loop_end1
loop_end1:      macs    *x0+,*y0+

                nop
                mov     r4,ref
                nop
                mov     r5,beta
                ldio    r0,acch

                sub     r0,r4                   ; caculate error
                muls    r0,r5                   ; error x beta

                nop
                nop
                nop
                nop
                ldio    r0,acch
                nop

; update coef
                madd    *x0+,*y0+,rreg
                madd    *x0+,*y0+,rreg
```

```
                srpt    #(N-4)/4
                erpt    loop_end2
loop_start2:    stix    acch,*x1+
                madd    *x0+,*y0+,rreg
                stix    acch,*x1+
                madd    *x0+,*y0+,rreg
                stix    acch,*x1+
                madd    *x0+,*y0+,rreg
                stix    acch,*x1+
loop_end2:      madd    *x0+,*y0+,rreg

                madd    *x0+,*y0+,rreg
                madd    *x0+,*y0+,rreg
                stix    acch,*x1+
                nop
                stix    acch,*x1+
                stix    acch,*x1+
                stix    acch,*x1+

                cycle = 3N+23
```

---

14. N tap LMS adaptive FIR with FIR H/W

```
                ldiy    r0,*y
                ldio    r0,ldata
                stis    rreg,*y0
                stiy    rreg,*y0+

                dmuls   *x0+,*y0+

                srpt    #(N-2)/2
                erpt    loop_end1
loop_end1:      dmacs   *x0+,*y0+

                nop
                mov     r4,ref
                nop
                mov     r5,beta
                ldio    r0,acch
                ldio    r1,facch
                add     r0,r1

                sub     r0,r4                   ; caculate error
                muls    r0,r5                   ; error x beta

                nop
                nop
                nop
                nop

                ldio    r0,acch
                nop
; update coef
                dmadd   *x0+,*y0+,rreg
                dmadd   *x0+,*y0+,rreg

                srpt    #(N-4)/8
                erpt    loop_end2

loop_start2:    dstix   acch,*x1+
                dmadd   *x0+,*y0+,rreg
                dstix   acch,*x1+
                dmadd   *x0+,*y0+,rreg
                dstix   acch,*x1+
                dmadd   *x0+,*y0+,rreg
                dstix   acch,*x1+
```

```
loop_end2:      dmadd *x0+,*y0+,rreg

                dmadd *x0+,*y0+,rreg
                dmadd *x0+,*y0+,rreg
                dstix   acch,*x1+
                nop
                dstix   acch,*x1+
                dstix   acch,*x1+
                dstix   acch,*x1+

                cycle = 1.5N+26
```

ภาคผนวก ข

บทความที่ได้รับการตีพิมพ์ใน
การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 26

การออกแบบวงจรรวมหน่วยประมวลผลสัญญาณดิจิทัลที่มีตัวกรองเอฟไอาร์

An IC Design of a DSP Processor with Built-in FIR Filter

รวิวร มะหะสิทธิ์ และวันเฉลิม โปรา

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ถนน พญาไท เขตปทุมวัน กรุงเทพมหานคร 10330 E-mail: m_ravivon@hotmail.com

บทคัดย่อ

บทความนี้นำเสนอการออกแบบหน่วยประมวลผลสัญญาณดิจิทัลแบบทศนิยมคงที่ 16 บิต ที่มีตัวกรองเอฟไอาร์ภายใน และอุปกรณ์บริวารอื่นๆ เช่น ตัวตั้งเวลา วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S ดีเอ็มเอ (DMA) และพอร์ตอินพุต-เอาต์พุต เป็นต้น โครงสร้างของหน่วยประมวลผลสัญญาณดิจิทัลมีโครงสร้างแบบไปป์ไลน์ 5 ขั้นตอน มีชุดคำสั่งแบบริสก์ (RISC) ตัวกรองเอฟไอาร์ที่ออกแบบสามารถทำงานได้สองลักษณะ โดยสามารถทำงานขนานอย่างอิสระจากการทำงานของหน่วยประมวลผล หรือทำงานเป็นหน่วยคูณและสะสมที่เรียกใช้โดยตรงได้จากหน่วยประมวลผลกลาง เมื่อหน่วยประมวลผลกลางทำงานร่วมกับตัวกรองจะสามารถคำนวณตัวกรองเอฟไอาร์แบบปรับตัวแบบกำลังสองน้อยที่สุดได้ภายใน 1.5N+26 วงรอบคำสั่ง หน่วยประมวลผลที่ออกแบบถูกนำมาจำลองการทำงานและสร้างตัวต้นแบบบนเอฟพีจีเอ แล้วจึงนำมาสร้างลายวงจรรวมบนเทคโนโลยีซีมอส 0.35 ไมครอน ผลการทดสอบคุณสมบัติของลายวงจรรวมก่อนนำไปเจือสาร ลายวงจรรวมใช้พื้นที่ประมาณ 6.23 ตารางมิลลิเมตร สามารถทำงานได้ที่ความถี่สูงสุด 90 MHz และกินกำลังไฟประมาณ 5.5 mW/MHz ที่แรงดันไฟเลี้ยง 3.3 โวลต์

Abstract

This paper presents a design which consists of a 16-b fixed point digital signal processor with a built-in FIR and some peripheral devices such as timer, I^2S interfacing circuit, DMA and I/O ports etc. The 5-stages pipelined architecture together with RISC based instruction set are employed. The filter can either operate in parallel mode with the processor or can be configured as a multiply and accumulate unit controlled mode by the processor. With the LMS adaptive FIR benchmark, the processor can finish the computation in 1.5N+26 cycles. All parts was simulated and implemented on FPGA. The chip is prepared to be fabricated in a 0.35-μm CMOS technology. The estimated chip area is about 6.23 mm^2 and its operation frequency is up to 90 MHz. The power consumption is estimated to be 5.5 mW/MHz at 3.3 V .

Keyword: DSP Processor, VLSI Design, Computer Architecture, FIR Filter.

1. คำนำ

ปัจจุบันกรรมวิธีสัญญาณดิจิทัลได้ถูกนำมาประยุกต์ใช้งานอย่างกว้างขวาง ตัวอย่างของการประยุกต์ใช้งานกรรมวิธีสัญญาณดิจิทัลได้แก่ การบีบอัดและการเข้ารหัสสัญญาณเสียง, การตรวจสอบลายเซ็นและการกำจัดเสียงสะท้อน (Echo Cancellation) เป็นต้น หน่วยประมวลผลสัญญาณดิจิทัล (Digital Signal Processor) ถือได้ว่าเป็นส่วนประกอบหลักในระบบประมวลสัญญาณดิจิทัลเหล่านี้ หน่วยประมวลผลแบบนี้สามารถประมวลผลการดำเนินการทางคณิตศาสตร์ได้อย่างรวดเร็ว และมีชุดคำสั่งที่สนับสนุนการประมวลผลในระบบที่มีสัญญาณเข้ามาในอัตราสม่ำเสมอและต่อเนื่อง

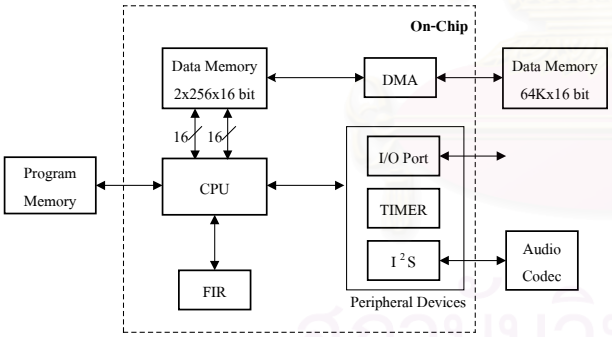
การประยุกต์ใช้งานหลายๆอย่าง เช่น การกำจัดเสียงสะท้อน ซึ่งต้องการการคำนวณส่วนย่อยๆหลายส่วน ได้แก่ การคำนวณในส่วนของตัวกรอง และการคำนวณเพื่อปรับค่าสัมประสิทธิ์ เป็นต้น หากเลือกใช้หน่วยประมวลผลสัญญาณดิจิทัลเพียงอย่างเดียวอาจจำเป็นต้องใช้หน่วยประมวลผลที่มีประสิทธิภาพสูงมาก แต่ทว่าหากเลือกใช้ฮาร์ดแวร์เฉพาะงานเพียงอย่างเดียวก็อาจจะทำให้ขาดความยืดหยุ่นในการปรับปรุงการทำงาน ทางเลือกอีกทางหนึ่งคือการเพิ่มฮาร์ดแวร์เฉพาะงาน (Application Specific Hardware) เพื่อช่วยหน่วยประมวลผลสัญญาณดิจิทัลทำงานจะช่วยลดปริมาณงานของหน่วยประมวลผลลง ทำให้หน่วยประมวลผลมีเวลาว่างสำหรับประมวลผลงานอื่น หรืออีกนัยหนึ่งหน่วยประมวลไม่จำเป็นต้องมีประสิทธิภาพสูงมากก็ได้

การเพิ่มฮาร์ดแวร์ที่ไม่ตรงกับงานจะทำให้เปลืองทรัพยากรฮาร์ดแวร์โดยเปล่าประโยชน์ แต่ทว่าโครงสร้างแบบตัวกรองเอฟไอาร์จัดได้ว่าเป็นองค์ประกอบหนึ่งที่ถูกนำมาใช้งานในระบบประมวลผลสัญญาณดิจิทัลส่วนใหญ่ ไม่ว่าจะเป็นงานกรองสัญญาณ การหาค่าสหสัมพันธ์ (Correlation) หรือการตรวจคุณลักษณะ (Identification) ดังนั้นการเพิ่มเติมฮาร์ดแวร์ตัวกรองเอฟไอาร์เพื่อทำงานร่วมกับหน่วยประมวลผล จะทำให้สามารถใช้งานฮาร์ดแวร์เฉพาะงานได้อย่างกว้างขวาง

บทความนี้นำเสนอการพัฒนาหน่วยประมวลสัญญาณดิจิทัลแบบใช้งานทั่วไป (General-Purpose Digital Signal Processor) ที่ประกอบด้วยฮาร์ดแวร์เฉพาะงานสำหรับการประมวลสัญญาณดิจิทัลได้แก่ ตัวกรองเอฟไอาร์ที่สามารถปรับความยาวและทำงานขนานกับหน่วยประมวลผลได้ นอกจากนี้หากใช้ในงานประมวลผลที่ไม่จำเป็นต้องใช้ตัวกรองดังกล่าว หน่วยประมวลผลกลางสามารถปรับโหมดการทำงานให้องค์ประกอบของตัวกรองซึ่งได้แก่ หน่วยความจำและวงจรทางคณิตศาสตร์กลายเป็นองค์ประกอบหนึ่งของหน่วยประมวลผลได้ ทำให้หน่วยประมวลผลสามารถทำงานได้ทั้งในโหมดการทำงานเฉพาะงานหรือการใช้งานทั่วไปได้ หน่วยประมวลผลที่ออกแบบสามารถประมวลผลสัญญาณที่มีแบนด์วิดธ์ในย่านเสียงได้

2 สถาปัตยกรรม

วงจรรวมที่ออกแบบประกอบด้วยองค์ประกอบหลัก ได้แก่ หน่วยประมวลผลกลาง (CPU) แบบ fixed point 16 บิต หน่วยความจำข้อมูล (Data Memory) ภายในขนาด 16 บิต 256 ตำแหน่งจำนวน 2 ชุด และตัวกรองเอฟไอาร์ (FIR) และอุปกรณ์บริวารอื่นๆ สำหรับการเชื่อมต่อกับอุปกรณ์ภายนอก ซึ่งได้แก่ ตัวตั้งเวลา (Timer) วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S (I^2S Interface Circuit) ดีเอ็มเอ (DMA) และพอร์ตอินพุต-เอาต์พุต (I/O Port) รายละเอียดของวงจรส่วนต่างๆ (ดูรูปที่ 1) จะอธิบายในหัวข้อย่อยถัดไป

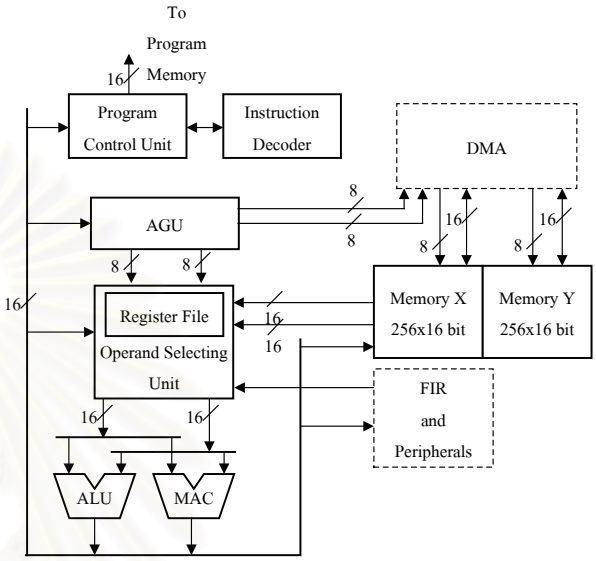


รูปที่ 1 สถาปัตยกรรมของหน่วยประมวลผลสัญญาณดิจิทัล

2.1 หน่วยประมวลผลกลาง

หน่วยประมวลผลกลางมีโครงสร้างแบบไปป์ไลน์ 5 ขั้นตอน และมีสถาปัตยกรรมของหน่วยความจำแบบฮาร์วาร์ดดัดแปลง [1] (Modified Harvard Architecture) ซึ่งประกอบด้วยหน่วยความจำโปรแกรม 1 ชุด และหน่วยความจำข้อมูลภายใน 2 ชุด ทำให้หน่วยประมวลผลจะสามารถประมวลผลการดำเนินการที่ใช้ข้อมูลจากหน่วยความจำข้อมูลทั้งสองชุดได้ภายในหนึ่งวงรอบคำสั่ง (ดูรูปที่ 2)

หน่วยประมวลผลกลางสามารถแบ่งเป็นวงจรย่อยได้ทั้งหมด 7 ส่วน ได้แก่ หน่วยควบคุมโปรแกรม (Program Control Unit) ตัวถอดรหัสคำสั่ง (Instruction Decoder) หน่วยเลือกตัวถูกดำเนินการและแฟ้มรีจิสเตอร์ (Operand Selector and Register File) หน่วยคำนวณและตรรกะ (Arithmetic and Logic Unit : ALU) หน่วยคูณและสะสม (Multiply and Accumulate Unit : MAC) และหน่วยกำเนิดตำแหน่งข้อมูล (Address Generation Unit : AGU)



รูปที่ 2 สถาปัตยกรรมของหน่วยประมวลผลกลาง

หน่วยควบคุมโปรแกรมมีหน้าที่จัดการและสร้างตำแหน่งของข้อมูลคำสั่งที่อยู่ภายในหน่วยความจำโปรแกรม การเปลี่ยนแปลงค่าของตัวนับโปรแกรมจะขึ้นอยู่กับคำสั่งในปัจจุบันหรือเมื่อเกิดการขัดจังหวะขึ้น นอกจากนี้ตัวนับโปรแกรมยังมีหน้าที่จัดการคำสั่งประเภทวนลูป ซึ่งจะทำให้โปรแกรมทำงานตามจำนวนรอบที่กำหนด

เมื่อคำสั่งถูกเฟทช์เข้ามาในหน่วยประมวลผลกลางจะเก็บค่าในรีจิสเตอร์และถอดรหัสที่ตัวถอดรหัสคำสั่ง หน้าที่ของตัวถอดรหัสคำสั่งคือการถอดรหัสคำสั่งและแยกฟิลด์ของรหัสดำเนินการให้อยู่ในรูปแบบที่วงจรภายในสามารถเข้าใจได้ เนื่องจากในแอดเดรสซิ่งโหมดแต่ละแบบจะมีฟิลด์ของรหัสดำเนินการซ้อนทับที่ตำแหน่งเดียวกัน

หน่วยกำเนิดตำแหน่งข้อมูลมีหน้าที่เก็บและปรับปรุงตัวชี้ข้อมูลของหน่วยความจำ โดยสามารถกำหนดการปรับปรุงของตัวชี้ข้อมูลได้ 3 รูปแบบ เช่น การเพิ่มตัวชี้ด้วยค่าดัชนี (Index) การลดตัวชี้ด้วยค่าดัชนี และการเปรียบเทียบตัวชี้กับค่าคงที่ เป็นต้น นอกจากนี้สามารถให้ตัวชี้ทำงานในลักษณะของ Circular Buffer ได้

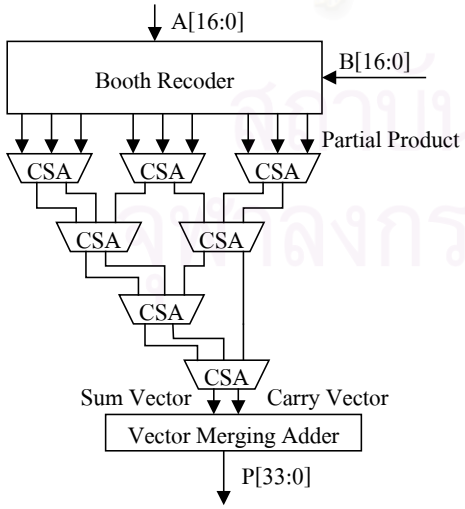
หน่วยเลือกตัวถูกดำเนินการมีหน้าที่เลือกประเภทของตัวถูกดำเนินการเพื่อส่งต่อไปยังหน่วยคำนวณอื่นๆ ตัวถูกดำเนินการแบ่งได้เป็น 6 ประเภท ได้แก่ ข้อมูลจากแฟ้มรีจิสเตอร์ ข้อมูลจากหน่วยความจำข้อมูล ข้อมูลจากหน่วยความจำโปรแกรม ข้อมูลตำแหน่งจากตัวชี้ข้อมูล ข้อมูลค่าคงที่สำหรับแอดเดรสซิ่งโหมดแบบใช้ทันที และข้อมูลจากรีจิสเตอร์ใช้งานพิเศษ (Special Function Register) นอกจากนี้หน่วยเลือกตัว

ถูกดำเนินการยังมีหน้าที่ตรวจสอบและแก้ไขอันตรายเชิงข้อมูล (Data Hazard) เนื่องมาจากการที่หน่วยประมวลผลมีโครงสร้างแบบไปป์ไลน์ด้วย

หน่วยคำนวณและตรรกะมีหน้าที่ดำเนินการคำสั่งทางคณิตศาสตร์ ซึ่งได้แก่ การบวกและเปรียบเทียบข้อมูล และคำสั่งทางตรรกะ นอกจากนี้คำสั่งประเภทการโหลดข้อมูลจะใช้หน่วยคำนวณและตรรกะเป็นทางผ่านในการย้ายข้อมูลจากหน่วยความจำไปยังแฟ้มรีจิสเตอร์ด้วย

หน่วยคูณและสะสมเป็นวงจรย่อยที่มีหน้าที่ดำเนินการคำสั่งที่เกี่ยวข้องกับการประมวลผลสัญญาณดิจิทัล ซึ่งมีความล่าช้ามากและถือเป็นเส้นทางวิกฤตของหน่วยประมวลผล ดั้งนั้นการออกแบบหน่วยคูณและสะสมให้สามารถทำงานได้เร็วจะส่งผลให้หน่วยประมวลผลทำงานได้เร็วยิ่งขึ้น องค์ประกอบหลักของหน่วยคูณและสะสม ได้แก่ วงจรคูณและวงจรบวก ในที่นี้การออกแบบจะเลือกใช้โครงสร้างของวงจรคูณชนิดมีเครื่องหมายแบบบู๊ธ (Booth Multiplier) และเลือกใช้โครงสร้างของวงจรบวกแบบเลือกตัวทด (Carry-Select Adder)

วงจรคูณแบบบู๊ธ (ดูรูปที่ 3) เป็นการประยุกต์ใช้อัลกอริทึมของบู๊ธ [2] (Booth Algorithm) ในการเข้ารหัสซ้ำ (Recoding) เพื่อลดจำนวนของผลคูณบางส่วน (Partial Product) ส่วนการรวมผลคูณบางส่วนที่ได้จากกการเข้ารหัสด้วยใช้อัลกอริทึมของบู๊ธจะใช้โครงสร้างแบบวอลเลซทรี (Walace Tree Structure) ซึ่งประกอบด้วยวงจรบวกแบบยกเว้นตัวทด (Carry-Save Adder : CSA) ต่อเรียงกัน แล้วจึงนำผลลัพท์จากวอลเลซทรี ซึ่งได้แก่ เวคเตอร์ผลรวม (Sum Vector) และเวคเตอร์ตัวทด (Carry Vector) มาเก็บไว้ในรีจิสเตอร์ก่อนที่จะนำมารวมกันด้วยวงจรบวกในขั้นตอนสุดท้ายอีกที การใช้รีจิสเตอร์มาแบ่งการทำงานของวงจรคูณให้มีโครงสร้างแบบไปป์ไลน์จะช่วยลดความล่าช้าของวงจร แต่ว่าจะทำให้ผลลัพท์ของวงจรคูณล่าช้าไปหนึ่งรอบสัญญาณนาฬิกา



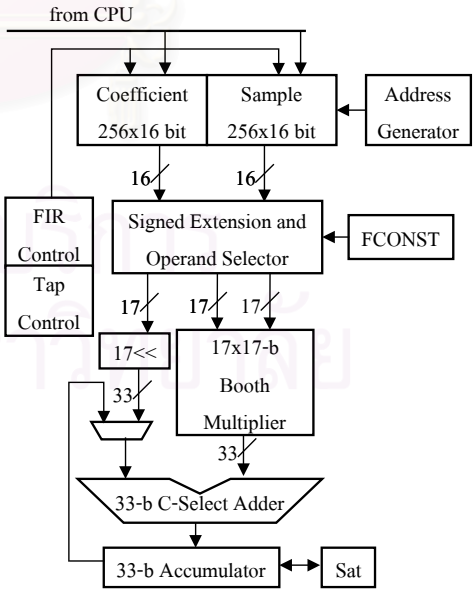
รูปที่ 3 วงจรคูณแบบบู๊ธ

2.2 ตัวกรองเอฟไอาร์

โครงสร้างของตัวกรองประกอบด้วย หน่วยคูณและสะสม หน่วยความจำขนาด 16 บิต 256 ตำแหน่งสองชุดสำหรับเก็บค่าสัมประสิทธิ์และตัวอย่าง ตัวสร้างตำแหน่งข้อมูลของหน่วยความจำ และวงจรควบคุมซึ่งทำหน้าที่ควบคุมลำดับการทำงานของตัวกรองหน่วยคูณและสะสมของตัวกรองจะประกอบไปด้วยวงจรคูณแบบบู๊ธและวงจรบวกแบบเลือกตัวทดเช่นเดียวกับหน่วยประมวลผลกลาง (ดูรูปที่ 4) นอกจากนี้ตัวสะสมจะประกอบด้วยวงจรป้องกันการล้นเกิน (Overflow) ซึ่งจะทำให้ค่าของตัวสะสมเกิดการอิ่มตัว (Saturation) เมื่อเกิดการล้นเกินขึ้น

ตัวกรองเอฟไอาร์ถูกออกแบบมาให้สามารถทำงานขนานกับหน่วยประมวลผลกลาง เมื่อตัวกรองคำนวณผลลัพธ์เสร็จจะส่งสัญญาณเพื่อไปขัดจังหวะการทำงานของหน่วยประมวลผลกลาง

ขณะที่ไม่ได้ใช้งานตัวกรอง หน่วยประมวลผลจะสามารถควบคุมองค์ประกอบภายในตัวกรอง เช่น หน่วยคูณและสะสม และหน่วยความจำให้ทำงานด้วยคำสั่งโปรแกรมของหน่วยประมวลผลโดยตรง การทำงานในกรณีนี้ หน่วยประมวลผลจะสามารถดำเนินการข้อมูลได้สองชุดด้วยคำสั่งเฉพาะเพียงคำสั่งเดียว (Single Instruction – Multiple Data) โดยจะสามารถประมวลผลข้อมูลที่อยู่ในหน่วยความจำของตัวกรองพร้อมกับข้อมูลที่อยู่ในหน่วยความจำของหน่วยประมวลผลโดยใช้ทรัพยากรวงจร ได้แก่ ตัวชี้ข้อมูล และวงจรควบคุมร่วมกัน ซึ่งการทำงานในรูปแบบนี้จะทำให้หน่วยประมวลผลและตัวกรองสามารถทำงานร่วมกันได้อย่างอย่างประสิทธิภาพ โดยที่ใช้ทรัพยากรฮาร์ดแวร์เพิ่มเติมเพียงเล็กน้อย



รูปที่ 4 โครงสร้างตัวกรองเอฟไอาร์

2.3 อุปกรณ์บริวาร

อุปกรณ์บริวารของหน่วยประมวลผลประกอบด้วย ตัวตั้งเวลา 16 บิต วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S ดีเอ็มเอ และพอร์ตอินพุต-เอาต์พุตตัวตั้งเวลามีหน้าที่กำหนดฐานเวลาของหน่วยประมวลผล วงจรเชื่อมต่ออุปกรณ์มาตรฐาน I^2S [3] มีหน้าที่เชื่อมต่อระหว่างตัวเข้า-ถอดรหัสสัญญาณเสียงแบบสเตอริโอ (Stereo Audio Codec) ซึ่งมีหน้าที่แปลงสัญญาณเสียงแอนะล็อกเป็นดิจิทัล สำหรับส่งให้หน่วยประมวลผล และรับข้อมูลดิจิทัลจากหน่วยประมวลผลเพื่อแปลงเป็นแอนะล็อกต่อไป ส่วนดีเอ็มเอมีหน้าที่โอนย้ายข้อมูลระหว่างหน่วยความจำภายในแลภายนอกอย่างอิสระจากหน่วยประมวลผล

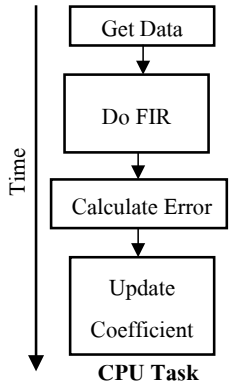
3. ชุดคำสั่ง

จุดมุ่งหมายของการออกแบบ คือ หน่วยประมวลผลจะต้องทำงานได้เร็วและมีความซับซ้อนน้อย เพื่อประหยัดทรัพยากรฮาร์ดแวร์ ดังนั้นชุดคำสั่งจึงถูกออกแบบด้วยแนวคิดของชุดคำสั่งแบบริสค์ (Reduced Instruction Set Computers : RISC) ซึ่งมีจำนวนคำสั่งน้อยและมีรูปแบบของรหัสดำเนินการที่เรียบง่าย ชุดคำสั่งที่ออกแบบประกอบด้วยคำสั่งทั้งหมด 46 คำสั่ง ซึ่งสามารถแบ่งออกเป็น 4 กลุ่ม ได้แก่ กลุ่มคณิตศาสตร์และตรรกะ กลุ่มโอนย้ายข้อมูล กลุ่มการคูณและสะสม และกลุ่มควบคุมโปรแกรม รหัสดำเนินการของคำสั่งมีความยาวคงที่ 16 บิต ซึ่งจัดว่าสั้นเมื่อเทียบกับหน่วยประมวลผลสัญญาณดิจิทัลที่จำหน่ายในท้องตลาด ข้อดีของรหัสดำเนินการที่มีความยาวน้อย คือ การออกแบบชุดคำสั่ง และการเขียนโปรแกรมสามารถทำได้ง่าย เนื่องจากมีแอดเดรสซึ่งโหมดน้อย และการเข้ารหัสคำสั่งไม่ซับซ้อนซึ่งส่งผลให้ฮาร์ดแวร์สำหรับการถอดรหัสคำสั่งไม่ซับซ้อน แต่ทว่ารหัสดำเนินการที่สั้นจะมีข้อเสีย คือคำสั่งสามารถอ้างตัวถูกดำเนินการและมีการดำเนินการย่อยได้น้อย ทำให้อาจต้องใช้จำนวนคำสั่งมาก

4. การใช้งานตัวกรอง

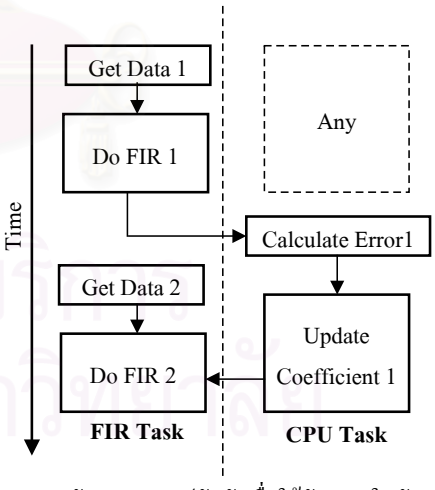
การใช้งานตัวกรองร่วมกับหน่วยประมวลผลกลางอาจพิจารณาได้ 2 ลักษณะได้แก่ การใช้งานตัวกรองเพื่อการกรองสัญญาณ และการใช้งานตัวกรองเป็นองค์ประกอบหนึ่งของหน่วยประมวลผลกลาง ในที่นี้จะยกตัวอย่างการใช้งานหน่วยประมวลผลเพื่อการกรองสัญญาณแบบปรับตัวด้วยระเบียบวิธีกำลังสองน้อยที่สุด (LMS)

การกรองสัญญาณแบบปรับตัวด้วยระเบียบวิธีกำลังสองน้อยที่สุดประกอบด้วยการดำเนินการหลัก ได้แก่ การคำนวณตัวกรอง การหาค่าความผิดพลาดของสัญญาณขาออกของตัวกรองเทียบกับสัญญาอ้างอิง และการปรับค่าสัมประสิทธิ์ของตัวกรอง ผังแสดงภาระงานของการกรองแบบปรับตัวกรณีที่ใช้งานเพียงหน่วยประมวลผลกลาง (ดูรูปที่ 5) แสดงให้เห็นว่าหน่วยประมวลผลมีหน้าที่ประมวลผลงานทุกส่วน

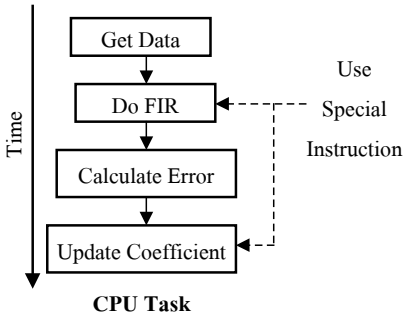


รูปที่ 5 ผังงานของการกรองสัญญาณแบบปรับตัวด้วยหน่วยประมวลผล

เมื่อใช้งานตัวกรองเอฟไอาร์ในลักษณะที่ 1 (ดูรูปที่ 6) จะทำให้ภาระงานของหน่วยประมวลผลกลางสำหรับการคำนวณการกรองแบบเอฟไอาร์ (Do FIR) ย้ายไปที่ฮาร์ดแวร์ตัวกรองแทน ซึ่งทำให้หน่วยประมวลผลมีเวลาเหลือสำหรับการประมวลผลงานอื่นหรือทำงานในลักษณะขนาน เมื่อใช้งานตัวกรองเอฟไอาร์ในลักษณะที่ 2 (ดูรูปที่ 7) การประยุกต์ใช้งานคำสั่งเฉพาะจะทำให้หน่วยประมวลผลจะสามารถคำนวณการกรองแบบเอฟไอาร์ได้ทีละสองแท็ปต่อหนึ่งคำสั่ง ซึ่งส่งผลให้จำนวนรอบคำสั่งในการคำนวณตัวกรองลดลงครึ่งหนึ่ง และในทำนองเดียวกันจะสามารถปรับปรุงค่าสัมประสิทธิ์ของตัวกรองได้พร้อมกันทีละสองชุดในหนึ่งคำสั่ง ซึ่งส่งผลให้เวลาในการปรับปรุงสัมประสิทธิ์ลดลงครึ่งหนึ่งเช่นเดียวกัน



รูปที่ 6 การกรองสัญญาณแบบปรับตัวเมื่อใช้ตัวกรองในลักษณะที่ 1



รูปที่ 7 การกรองสัญญาณแบบปรับตัวเมื่อใช้ตัวกรองในลักษณะที่ 2

5. การจำลองการทำงานและทดสอบด้วยตัวต้นแบบ

ตัวต้นแบบของหน่วยประมวลผลสัญญาณดิจิทัลถูกออกแบบและจำลองการทำงานด้วยภาษา VHDL แล้วจึงถูกนำมาสังเคราะห์บนเอฟพีจีเอ Spartan II ของบริษัท Xilinx [4] บอร์ดวงจรต้นแบบจะประกอบไปด้วย บอร์ดเอฟพีจีเอ บอร์ดหน่วยความจำ และบอร์ดของตัวเข้า-ถอดรหัสสัญญาณเสียง

การจำลองการทำงานของวงจรจะใช้การสร้างแบบจำลองของหน่วยความจำโปรแกรมที่บรรจุด้วยโปรแกรมที่ใช้ทดสอบ แล้วนำมาเชื่อมต่อกับแบบจำลองของหน่วยประมวลผล เพื่อนำมาจำลองการทำงานและตรวจสอบความถูกต้องร่วมกับเวคเตอร์ทดสอบ (Test Vector) ที่เตรียมไว้ ส่วนการทดสอบด้วยตัวต้นแบบจะนำวงจรที่ถูกจำลองการทำงานมาสังเคราะห์ลงเอฟพีจีเอ แล้วจึงป้อนสัญญาณทดสอบเพื่อวัดผลลัพท์ของการประมวลผลด้วยเครื่องมือวัด

6. สมรรถนะของหน่วยประมวลผล

ชุดคำสั่งของหน่วยประมวลผลถูกออกแบบให้รองรับการประมวลผลสัญญาณดิจิทัล โดยเฉพาะงานประเภทการกรองสัญญาณดิจิทัล การวัดสมรรถนะของหน่วยประมวลผลจะใช้การนับจำนวนรอบของการทำงานตามบรรทัดฐาน (Benchmark) ทั่วไปของงานกรรมวิธีสัญญาณดิจิทัล ผลของการวัดสมรรถนะแสดงดังตารางที่ 1

ตารางที่ 1 สมรรถนะของหน่วยประมวลผลที่ออกแบบ

การคำนวณ	จำนวนวงรอบคำสั่งที่ใช้
การคูณค่าจริง N จำนวน	2N+4
ตัวกรองเอฟไออาร์ยาว N แท็ป	0.5N+14
ตัวกรองไอไออาร์ไบควอดต่อเรียง N ชุด	7N+8
การปรับปรุงค่าจริง N จำนวน	1.5N+6
ตัวกรองปรับตัวแบบกำลังสองน้อยที่สุด	1.5N+26
ผลรวมกำลังสองของค่าจริง N จำนวน	2N+11

เมื่อพิจารณาสมรรถนะของหน่วยประมวลผลในแง่ความถี่สูงสุดที่วงจรสามารถทำงานได้ การพิจารณาจะแบ่งเป็นสองรูปแบบ ได้แก่ สมรรถนะของวงจรบนเทคโนโลยีเอฟพีจีเอ และสมรรถนะของวงจรบนเทคโนโลยีของวงจรรวม การเปรียบความถี่การทำงานสูงสุดของหน่วยประมวลผลแสดงดังตารางที่ 2

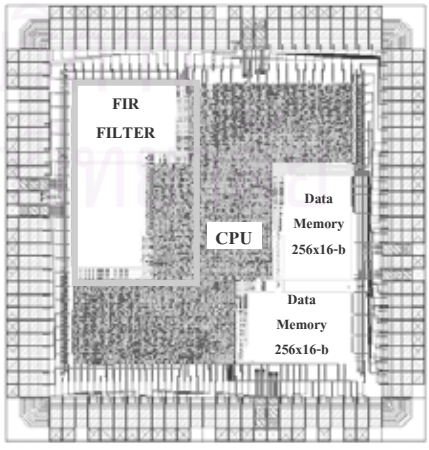
ตารางที่ 2 ความถี่การทำงานสูงสุดของหน่วยประมวลผล

เทคโนโลยี	ความถี่สูงสุด (Mhz)
เอฟพีจีเอ Spartan II ของ Xilinx	44
ซีมอส 0.8 ไมครอน ของ AMS	40
ซีมอส 0.5 ไมครอน ของ Alcatel	53
ซีมอส 0.35 ไมครอน ของ AMS	90

7. การออกแบบลายวงจรรวม

การออกแบบลายวงจรรวมจะใช้ซอฟต์แวร์สำหรับการออกแบบวงจรรวมของบริษัท Mentor Graphic และ Cadence โดยขั้นตอนการออกแบบ (Design Flow) จะเริ่มต้นตั้งแต่การสังเคราะห์วงจรบนเทคโนโลยีวงจรรวม การวางและกำหนดเส้นทางลายวงจร (Place and Route : P&R) การเปรียบเทียบระหว่างลายวงจรและแผนภาพ (Layout Versus Schematic : LVS) และการตรวจสอบความถูกต้องของวงจร (Design Rule Check : DRC) ตามลำดับ

ลายวงจรรวมถูกออกแบบด้วยเซลล์มาตรฐานดิจิทัลบนเทคโนโลยีซีมอส 0.35 ไมครอนของบริษัท Austriamicrosystems [5] ผลการทดสอบคุณสมบัติของลายวงจรรวมก่อนนำไปเจือสาร วงจรใช้พื้นที่ประมาณ 6.23 ตารางมิลลิเมตร (รวมหน่วยความจำ) ซึ่งสามารถประมาณความซับซ้อนของวงจรได้เทียบเท่า 30,000 เกต (ไม่รวมหน่วยความจำ) และกินกำลังไฟประมาณ 5.5 mW/MHz ที่แรงดันไฟเลี้ยง 3.3 โวลต์



รูปที่ 9 ลายวงจรรวมของหน่วยประมวลผลสัญญาณดิจิทัล

8. สรุป

บทความนี้ได้นำเสนอการออกแบบวงจรรวมของหน่วยประมวลผลสัญญาณดิจิทัลที่ประกอบด้วยฮาร์ดแวร์เฉพาะงานสำหรับการกรองสัญญาณแบบดิจิทัล ได้แก่วงจรตัวกรองเอฟไออาร์ การที่หน่วยประมวลผลทำงานร่วมกับตัวกรองทำให้เกิดความยืดหยุ่นในการใช้งานตัวกรองมากขึ้น หน่วยประมวลผลที่ออกแบบถูกนำมาจำลองการทำงาน และนำมาสร้างตัวต้นแบบเพื่อทดสอบความถูกต้อง แล้วจึงนำมาสร้างเป็นลายวงจรรวมซึ่งพร้อมที่จะส่งไปเจือสารต่อไป

9. กิตติกรรมประกาศ

ขอขอบคุณศูนย์พัฒนาธุรกิจออกแบบวงจรรวม (TIDI) ศูนย์อิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) ที่ความสนับสนุนด้านซอฟต์แวร์และเทคโนโลยีที่ใช้ในการออกแบบวงจรรวม

เอกสารอ้างอิง

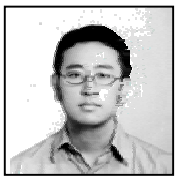
[1] Phil Lapsley, Jeff Bie, Amit Shoham, Edward A. Lee. “DSP Processor Fundamental”. IEEE PRESS,1997.

[2] V. Carl Hamacher, Zvonko G. Vranesic, Safwat G. Zaky. “Computer Organiztion”. Mc Graw Hill, 1996.

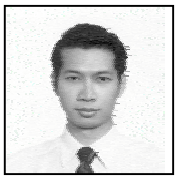
[3] Philips Semiconductors. “The I^2S Bus Specification”. June 1996.

[4] Xilinx Incorporation. ‘The Programmable Logic Data Book 2000’. United State of America, 2000.

[5] Austriamicrosystems. “0.35 um CMOS Digital Standard Cell Databook”, October 2001.



รวิวร มะหะสิทธิ์ สำเร็จการศึกษาวิศวกรรมศาสตร์บัณฑิต จากจุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. 2544 ปัจจุบันกำลังศึกษาต่อในระดับปริญญามหาบัณฑิต สาขาวิศวกรรมไฟฟ้า ที่สถาบันเดียวกัน งานวิจัยที่สนใจได้แก่ การออกแบบวงจรรวมเชิงเลข และการสร้างจริงการประมวลผลสัญญาณเชิงเลข เป็นต้น



วันเฉลิม โปรา สำเร็จการศึกษาระดับวิศวกรรมศาสตร์บัณฑิต และระดับปริญญามหาบัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. 2535 และ 2538 ตามลำดับ จากนั้นได้รับราชการที่ภาควิชาวิศวกรรมไฟฟ้า และได้รับทุนจากทบวงมหาวิทยาลัยเพื่อศึกษาต่อ และสำเร็จการศึกษาระดับปริญญาเอกที่ Imperial College, University of London ในปี 2543 งานวิจัยที่อยู่ในความสนใจได้แก่ การออกแบบ และประยุกต์ใช้งานวงจรเชิงเลข การประมวลผลสัญญาณเชิงเลข เป็นต้น

ประวัติผู้เขียนวิทยานิพนธ์

รวิวร มะหะสิทธิ์ เกิดวันที่ 26 ธันวาคม พ.ศ.2522 ที่กรุงเทพมหานคร สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยในปีการศึกษา 2543 และได้เข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิตสาขาวิศวกรรมไฟฟ้า แขนงวิชาวิศวกรรมไฟฟ้าอิเล็กทรอนิกส์เชิงเลข ที่คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2544 ในระหว่างการศึกษาได้รับรางวัลรองชนะเลิศการแข่งขันการประกวดการออกแบบวงจรรวมแห่งชาติครั้งที่ 2 (The National IC Design Contest 2001 : NIC2001) ประเภทดิจิทัล ระดับปริญญาตรี ซึ่งจัดขึ้นโดยศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) งานวิจัยที่สนใจได้แก่ การออกแบบวงจรรวมเชิงเลข และการสร้างจริง การประมวลผลสัญญาณเชิงเลข เป็นต้น