บทที่ 2
การออกแบบโพรเซสเซอร์ร่วมกำลังไฟต่ำสามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

2.1 บทนำ

เทคโนโลยีดิจิตอลอิเล็กทรอนิกส์ได้รับการพัฒนามาอย่างต่อเนื่องและยาวนาน โดยเฉพาะไมโครโพรเซสเซอร์ที่ได้พัฒนาขึ้นในทุก ๆ 2 ปี จะมีจำนวนทรานซิสเตอร์เพิ่มมากขึ้นเกือบเท่าตัว เนื่องจากขนาดของทรานซิสเตอร์และราคาที่ถูกกว่าเท่าตัวในทุก ๆปี ดังนั้นประสิทธิภาพของไมโครโพรเซสเซอร์จึงที่ได้รับการพัฒนาเพื่อเพิ่มศักยภาพในการประมวลผลมากขึ้นเรื่อย ๆ ไมโครโพรเซสเซอร์ในปัจจุบันจึงถูกออกแบบให้รองรับการทำงานแบบขนาน (รูปแบบหลายคำสั่ง) หรือโครงสร้างสถาปัตยกรรมแบบ out-of-order super scalar ซึ่งไมโครโพรเซสเซอร์แบบนี้จะมีข้อจำกัดของจำนวนคำสั่งที่จะสามารถทำงานแบบขนานกันได้ เนื่องจากอาจจะเกิดปัญหาความเชื่อมโยงของข้อมูลหรือที่เรียกว่า data dependency

การผลิตไมโครโพรเซสเซอร์แบบ ASIC ถึงแม้จะช่วยให้ขนาดของวงจรเล็ก ประหยัดพลังงาน และมีราคาถูกถ้ามีการผลิตเป็นจำนวนมาก เมื่อต้องการใช้งานผู้ใช้สามารถเขียนโปรแกรมขึ้นเพื่อประมวลผลงานที่ต้องการ ช่วยเพิ่มความยืดหยุ่นของการใช้งานดังนั้นเราจึงเรียกไมโครโพรเซสเซอร์แบบนี้ว่า General purpose processor (GPP) แต่รูปแบบสถาปัตยกรรมแบบนี้จะไม่สามารถเปลี่ยนแปลงได้แม้ว่าในบางช่วงของการประมวลผลอาจจะใช้งานวงจรเพียงส่วนเดียวของไมโครโพรเซสเซอร์ และอาจจะต้องมีการเปลี่ยนไมโครโพรเซสเซอร์หากพบว่าไม่สามารถรองรับการประมวลผลในบางงานประยุกต์ ทำให้วิศวกรจะต้องออกแบบบอร์ดวงจรใหม่ นอกจากนี้การมีโครงสร้างแบบคงที่ (fixed) จะมีข้อเสียในเรื่องของพลังงานที่ใช้ เนื่องจากแม้จะใช้งานเพียงบางส่วนก็จะต้องจ่ายพลังงานให้กับทั้งวงจร แม้ว่าในปัจจุบันจะมีแนวคิดของการออกแบบวงจรแบบไม่ใช่สัญญาณนาฬิกา (Asynchronous design หรือ clockless) แต่ก็ยังพบปัญหาที่วงจรจะอ่อนไหวต่อสภาพแวดล้อมเช่น อุณหภูมิ และวงจรยังทำได้งานช้า จึงทำให้ไม่สามารถนำออกมาใช้เชิงพาณิชย์ได้จริง

ดังนั้นงานวิจัยนี้จึงนำเสนอสถาปัตยกรรมของไมโครโพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก ทำหน้าที่ร่วมกับไมโครโพรเซสเซอร์หลัก เพื่อประมวลผลงานเฉพาะด้าน ซึ่งโครงสร้างนี้จะช่วยให้หน่วยประมวลผลไม่จำเป็นที่จะต้องเปลี่ยนฮาร์ดแวร์เมื่อต้องการความสามารถในการประมวลผลมากขึ้น ภายใต้กรอบแนวคิดของคุณลักษณะเด่นในโครงสร้างของเทคโนโลยี *Field Programmable Gate Array* (FPGA) ที่สามารถโปรแกรมวงจรได้ใหม่ เนื่องจากชุดของการควบคุมการเชื่อมต่อวงจรพื้นฐานเข้าด้วยกันจะเก็บไว้บนหน่วยความจำ จากนั้นจะทำการโปรแกรมการเชื่อมต่อบนโครงสร้างวงจรเลือกหรือ multiplexer เป็นสำคัญ ในงานวิจัยนี้ได้อาศัยหลักคิดของ FPGA มาประยุกต์สร้างโพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างแบบได

นามิกได้ดังกล่าว โดยเลือกที่จะใช้วงจรพื้นฐานในระดับของหน่วยประมวลผลหรือ Functional Unit แทนการสวิตช์เลือกการเชื่อมต่อวงจรในระดับเล็กเช่น Cell Logic Block (CLB) อย่างเช่นใน FPGA เนื่องจากโครงสร้างที่เล็กมากเกินไปจะทำให้เกิด overhead ในเรื่องของเวลาที่ใช้ในการ re-programmable หรือ reconfigurable ใหม่ ซึ่งทำให้ไม่เหมาะสมกับการใช้งานจริง

การทดสอบแนวคิดหรืออัลกอริทึมสำหรับการวาง (Place) และเชื่อมต่อ (Route) แต่ละเซลล์ (ในงานวิจัยนี้คือ FU) นั้นจะใช้การทดสอบบนโมเดลภาษาซี โดยแนวคิดที่ได้สามารถนำไปใช้ในการพัฒนาเครื่องมือช่วยการออกแบบหรือซอฟต์แวร์สำหรับการทำ dynamic reconfigurable ในอนาคตได้ สำหรับการพัฒนาและทดสอบโพรเซสเซอร์ร่วมที่ได้ออกแบบไว้นี้ จะเลือกทดสอบการทำงานบนเทคโนโลยี *Field Programmable Gate Array (FPGA)* ที่มีโครงสร้างเหมาะสมและเข้ากับสถาปัตยกรรมไมโครโพรเซสเซอร์แบบไดนามิก เนื่องจาก FPGA สามารถปรับเปลี่ยนการทำงานได้ (Reconfigurable) และ FPGA มีราคาถูก สะดวกและรวดเร็วต่อการทดสอบ

2.2 ขั้นตอนการออกแบบโพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

โพรเซสเซอร์ที่ออกแบบเป็นการพัฒนาต่อเนื่องจากงานในระดับปริญญาเอกของผู้วิจัยที่ออกแบบ DSP โดยทำการปรับลดรูปโครงสร้างหรือคำสั่งที่ไม่จำเป็นทิ้งไปเพื่อให้มีขนาดเล็ก ไม่ซับซ้อน และเหมาะสำหรับการประมวลผลทางคณิตศาสตร์

2.2.1 รูปแบบคำสั่ง

คำสั่งที่ได้ออกแบบสำหรับโพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก สามารถแบ่งออกเป็น 2 กลุ่มชนิดคำสั่งประกอบด้วย

1. *กลุ่ม Top-level instruction* มีหน้าที่ในการควบคุมหน่วยประมวลผล (Functional Unit, FU) ในโพรเซสเซอร์ ตัวอย่างของโพรเซสเซอร์ที่ออกแบบมีหน่วยประมวลผลได้สูงสุด 4 หน่วย นอกจากนี้ยังทำหน้าที่ในการควบคุมการเขียนผลลัพธ์ที่ได้กลับเข้าหน่วยประมวลผลทั้ง 4 หน่วยได้อย่างอิสระ โพรเซสเซอร์หลักสามารถใช้คำสั่ง top-level นี้ควบคุมลำดับการทำงานของโพรเซสเซอร์ร่วม ซึ่งกลุ่มคำสั่งจะถูกเก็บไว้ในหน่วยความจำที่สามารถเขียนโปรแกรมได้ใหม่ ทำให้การทำงานของโพรเซสเซอร์ร่วมนี้ปรับเปลี่ยนการทำงานตามสภาวะของการประมวลผลที่โพรเซสเซอร์หลักต้องการ คำสั่งชนิด top-level มีขนาด 14 บิต ดังแสดงรายละเอียดในตารางที่ 2

ตารางที่ 2 รายละเอียดคำสั่ง Top-level

บิตที่	ความหมาย
5-0	Config. Address เพื่อให้ควบคุมคำสั่งสำหรับประมวลผลใน FU
9-6	Enable FU3 – FU0
13-10	Accumulator write back FU3-FU0 enable

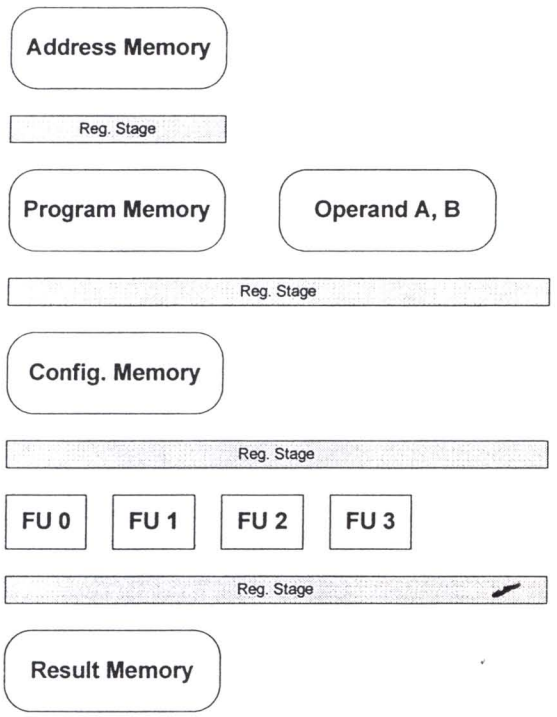
2. *กลุ่ม Configurable instruction* เป็นกลุ่มคำสั่งที่ใช้ใน FU เพื่อควบคุมอินพุตที่จะเข้าประมวลผลใน FU รูปแบบของคำสั่ง และการไหลของข้อมูลใน FU ซึ่ง FU ที่ได้ออกแบบมีคำสั่งหลักทั้งหมด 8 คำสั่งได้แก่ MPY MAC ADD SUB AND OR XOR และ MOV ซึ่งเป็นพื้นฐานของการประมวลผลงานทางด้านการคำนวณทางคณิตศาสตร์ โดยจะมี option ให้สามารถเลือกได้ว่าจะต้องการเลื่อนค่าหรือไม่ ดังแสดงรายละเอียดในตารางที่ 3

ตารางที่ 3 รายละเอียดคำสั่ง configuration memory

บิตที่	ความหมาย
1-0	Multiply operand control, 0 = OPA_low x OPB_low, 1 = OPA_high x OPB_low, 2 = OPA_low x OPB_high, 3 = OPA_high x OPB_high
3-2	Right input selection, 00 = GIFU, 01=ACC, 10:LIFU, 11:OPB
8-4	Opcode: 00 = MPY, 01 = MAC, 02 = ADD, 03 = SUB, 04 = AND, 05 = OR, 06 = XOR, 07 = MOV
13-9	Shift accumulator (shacc) shift distance
14	shacc direction (1 = left, 0 = right)
16-15	Reserved
18-17	Accumulator shift control, 00 = no shift, 01 = shift left, 10 = shift right, 11 = not used
21-19	000 = OPB, 001 = ACC[15:0], 010 = ACC[31:16], 011 = not used, 100 = GIFU, 101 = LIFU
23-22	Source of Accumulator write (ACCWR) 00 = GIFU, 01 = LIFU, 10 = ACC, 11 = SHACC
25-24	ACCWR DES (accumulator write destination) 00 = ACCA, 01 = ACCB, 10 = ACCC, 11 = ACCD
27-26	Arithmetic destination (1W) 00 = ACCA, 01 = ACCB, 10 = ACCC, 11 = ACCD
29-28	Accumulator source (1R) 00 = ACCA, 01 = ACCB, 10 = ACCC, 11 = ACCD
31-30	Shift accumulator source (2R) 00 = ACCA, 01 = ACCB, 10 = ACCC, 11 = ACCD

2.2.2 สถาปัตยกรรมของโพรเซสเซอร์ร่วม

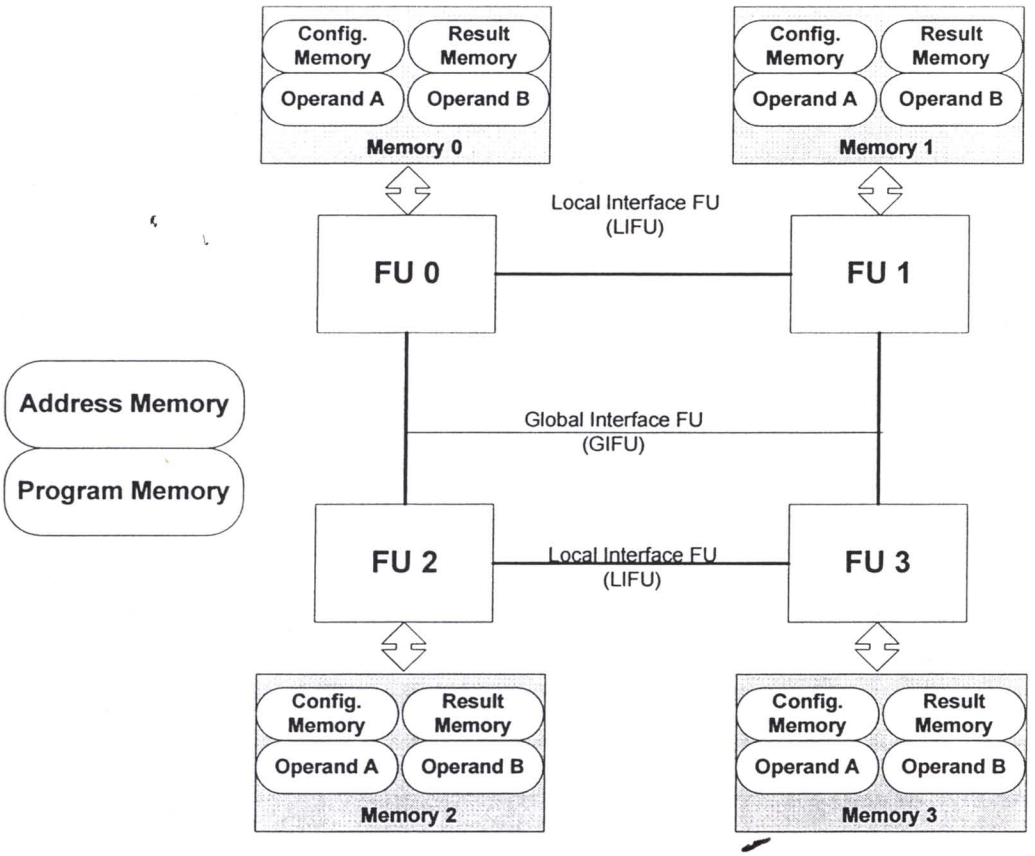
เมื่อได้ออกแบบชุดคำสั่งของโพรเซสเซอร์ร่วมโดยแบ่งออกเป็น 2 กลุ่ม คือกลุ่มของ top-level ซึ่งเป็นการควบคุมโครงสร้างใหญ่ของโพรเซสเซอร์หรือสามารถกล่าวได้ว่าเป็นการควบคุมกลุ่มของหน่วยประมวลผล FU ของโพรเซสเซอร์ ในการทดลองนี้กำหนดจำนวนกลุ่มของ FU เป็น 4 หน่วย สำหรับการใช้งานจริง จำนวนบิตในชุดคำสั่งที่เหลือสามารถปรับขยายได้ตามความต้องการว่าจะใช้ FU เป็นจำนวนกี่หน่วย ซึ่งในกลุ่มคำสั่ง top-level นี้จะถูกเก็บไว้ในหน่วยความจำแบบ SRAM เนื่องจากมีความเร็วในการทำงาน จะช่วยลด overhead ของการควบคุมโพรเซสเซอร์ร่วมจากโพรเซสเซอร์หลัก ทั้งนี้เรายังสามารถเรียกชุดคำสั่งนี้ว่าเป็น คำสั่งของการ reprogram การเชื่อมต่อโครงสร้างโพรเซสเซอร์ร่วม เนื่องจากจะมีกลุ่มบิตที่ใช้ในการสั่ง enable และ disable หน่วยประมวลผล FU ให้สามารถใช้งานจำนวน FU ที่เหมาะสมกับแต่ละโปรแกรมย่อยตามที่โพรเซสเซอร์หลักสั่งงาน โดยโครงสร้างขั้นตอนการทำงานของโพรเซสเซอร์ร่วมแบ่งออกได้เป็น 5 stage ดังรูปที่ 1



รูปที่ 1 การทำงานแต่ละ stage ของโพรเซสเซอร์ร่วม

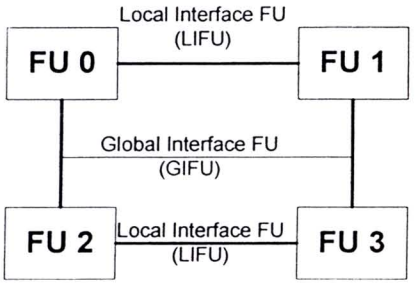
ขั้นตอนแรกเป็นการเก็บลำดับของคำสั่งที่จะใช้ในการทำงาน ซึ่งโพรเซสเซอร์หลักจะทำการเขียนโปรแกรมลงบนหน่วยความจำที่เรียกว่า *address memory* นี้ ซึ่งลำดับของคำสั่งจะสอดคล้องกับคำสั่งที่เก็บใน *Program memory* และค่าที่จะใช้ในการประมวลซึ่งเก็บไว้ในหน่วยความจำ *Operand A และ Operand B* โดยที่โพรเซสเซอร์ร่วมจะทำการอ่านค่าคำสั่งและ operand ไปใช้งานเป็นขั้นตอนที่สอง จากนั้นในขั้นตอนที่สามจะเป็นการโปรแกรมโครงสร้างในหน่วยประมวลผล FU ตามที่ถูกโปรแกรมเอาไว้แล้วด้วยโพรเซสเซอร์หลักซึ่งจะสอดคล้องกับคำสั่งและ operand ที่จะต้อง

ใช้ประมวลผล ขั้นตอนที่สี่เป็นการประมวลผลใน FU ในงานวิจัยนี้ใช้จำนวน 4 หน่วยเพื่อทดสอบโครงสร้างการทำงานของโพรเซสเซอร์ เมื่อใช้งาน FU จะถูกเปิดใช้งานให้สอดคล้องตามความจำเป็น (โดยทั่วไปจะเป็นหน้าที่ของคอมไพล์เลอร์ในการจัดสรรทรัพยากรให้สอดคล้องกับการใช้งาน) ตัวอย่างเช่น สำหรับการประมวลผลงานเกี่ยวกับ FIR filter จะเปิด FU ทั้ง 4 หน่วยทำงานแบบขนาน ในขณะที่การประมวลผลงานทั่วไปที่ต้องการความต่อเนื่องหรือมีความเป็น data dependency จำนวนมาก จะเปิดใช้ FU เพียง 1 หน่วยก็เหมาะสม เป็นต้น



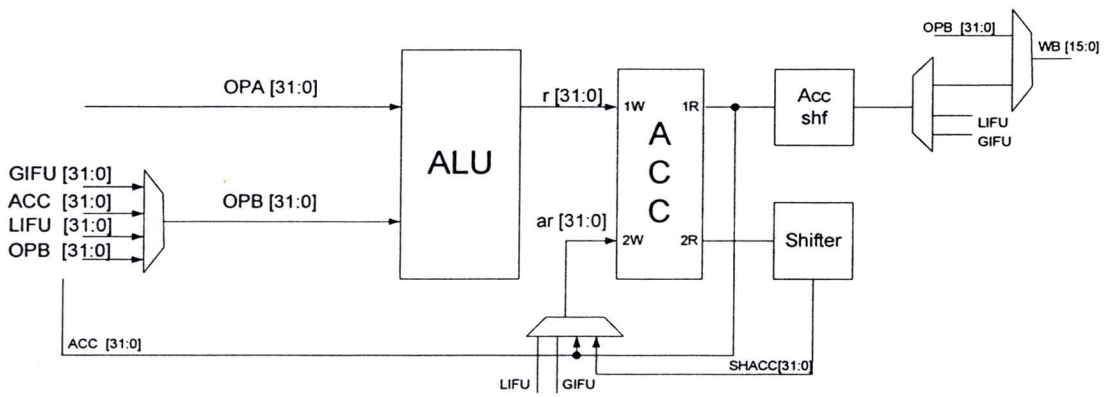
รูปที่ 2 โครงสร้างของโพรเซสเซอร์ร่วมที่ออกแบบ

โครงสร้างของโพรเซสเซอร์แสดงไว้ในรูปที่ 2 ประกอบด้วยหน่วยความจำหลักที่จะใช้งานร่วมกัน (หน่วยความจำ Address และ Program) ซึ่งในงานวิจัยนี้ใช้ SRAM ในวงจรเนื่องจากมีความเร็วสูงและราคาถูก โดยที่ข้อมูลที่ได้จากหน่วยความจำ program จะถูกส่งไปยังหน่วยความจำ configurable เพื่อจัดเตรียมการประมวลผล์ใน FU หน่วยความจำ configurable สำหรับแต่ละ FU สามารถถูกจัดเตรียมได้อิสระต่อกัน ตัวอย่างเช่น FU0 ถูกเตรียมเพื่อทำงานคำสั่ง MAC ในขณะที่ FU1 สามารถทำงานคำสั่ง ADD ซึ่งเป็นลักษณะเช่นเดียวกันกับหน่วยความจำ Operand A Operand B และหน่วยความจำ result



รูปที่ 3 การเชื่อมต่อระหว่างหน่วยประมวลผล FU

สำหรับแต่ละหน่วยประมวลผล FU สามารถแลกเปลี่ยนข้อมูลระหว่างกันผ่านโครงสร้างการเชื่อมต่อแบบบัส งานวิจัยนี้ออกแบบให้มีบัสเชื่อมต่อ 2 ชนิดคือ 1) Local Interface FU (LIFU) สำหรับการเชื่อมระหว่าง FU0 กับ FU1 และระหว่าง FU2 กับ FU3 2) Global Interface FU (GIFU) สำหรับการเชื่อมต่อระหว่าง FU0 FU1 FU2 และ FU3 ดังแสดงไว้ในรูปที่ 3



รูปที่ 4 ดาต้าพาสของหน่วยประมวลผล FU

เมื่อทำการออกแบบโครงสร้างสถาปัตยกรรมของโพรเซสเซอร์ร่วมเพื่อให้สามารถโปรแกรมการเชื่อมต่อได้ตามการใช้งานแล้ว จากนั้นเป็นการออกแบบดาต้าพาสภายในหน่วยประมวลผล FU ดังแสดงไว้ในรูปที่ 4 เพื่อให้สอดคล้องกับคำสั่งที่ออกแบบไว้ configuration memory ซึ่งโครงสร้างของหน่วยประมวลผลประกอบด้วย1) หน่วยคำนวณทางคณิตศาสตร์หรือเรียกว่า Arithmetic and Logic Unit (ALU) 2) รีจิสเตอร์ที่เก็บผลลัพธ์จากการคำนวณหรือเรียกว่า Accumulator Register (ACC) จำนวน 4 ตัวขนาดตัวละ 32 บิต 3) วงจรเลื่อนค่าหรือเรียกว่า Shifter 4) วงจรเลื่อนค่า ACC 1 บิต หรือเรียกว่า ACC Shf

1) ALU: วงจรส่วนของการคำนวณทางคณิตศาสตร์ประกอบด้วยคำสั่ง 8 รูปแบบได้แก่ คำสั่ง MPY MAC ADD SUB AND OR XOR MOV ซึ่งสามารถถอดรหัสจากชุดคำสั่งใน configuration memory บิตที่ 8-4 (configmem[8:4]) สามารถรับข้อมูลขนาด 32 บิตจากบัส OPA และ OPB ซึ่งบัส OPB สามารถมีทางเลือก (ใช้วงจร multiplexer) ใช้ข้อมูล 4 ทางได้แก่ ส่งมาจาก FU อื่นผ่านทาง GIFU หรือ มาจากการอ่านค่าใน ACC

หรือมาจาก FU ใกล้เคียงผ่านทาง LIFU หรือ เป็นค่าจากหน่วยความจำ operand B ผลลัพธ์ที่ได้จากการคำนวณขนาด 32 บิตจะถูกนำใส่บัส r[31:0] เพื่อส่งไปเก็บไว้ในรีจิสเตอร์ ACC (ACCA หรือ ACCB หรือ ACCC หรือ ACCD) ตัวใดตัวหนึ่งตามที่ต้องการ ซึ่งถูกกำหนดโดย configuration memory บิตที่ 27-26 (configmem[27:26])

2) ACC: รีจิสเตอร์เก็บผลลัพธ์จากการคำนวณในที่นี้ทำการสร้างไว้ทั้งหมด 4 ตัวได้แก่ A B C และ D แต่ละตัวมีขนาด 32 บิต โดยที่ ACC สามารถถูกอ่านค่าได้ 2 ทางคือ พอร์ต 1R เชื่อมต่อกับบัส ACC และ 2R เชื่อมต่อกับวงจรเลื่อนค่า shifter ขณะที่สามารถเขียนข้อมูลลงใน ACC ได้ 2 ทางคือพอร์ต 1W รับข้อมูลมาจาก ALU และ 2W เชื่อมต่อกับบัส ar[31:0] ที่สามารถรับค่ามาเขียนใน ACC ได้จากบัส ACC บัส SHACC จาก FU ตัวอื่นผ่านทาง GIFU และจาก FU ใกล้เคียงผ่านทาง LIFU โดยผู้ใช้สามารถกำหนดได้จากบิตที่ 23-22 ของ configuration memory (configmem[23:22])
3) ACCshf: เป็นวงจรในการเลื่อนค่าจาก ACC 1 บิตก่อนที่จะส่งไปเก็บในหน่วยความจำ result หรือส่งไปยัง FU อื่นๆ ผู้ใช้สามารถกำหนดทิศทางของการเลื่อนค่าได้จากบิตที่ 18-17 ของ configuration memory (configmem[18:17])
4) Shifter: เป็นวงจรที่ใช้ในการเลื่อนค่า ซึ่งรับอินพุตมาจาก ACC และส่งผลลัพธ์ที่ได้จากการเลื่อนออกทางบัส SHACC ขนาด 32 บิต ผู้ใช้สามารถกำหนดจำนวนของการเลื่อนค่าได้ทางบิตที่ 13-9 ของ configuration memory (configmem[13:9]) ซึ่งทำให้สามารถเลื่อนค่าได้สูงสุดถึง 32 ตำแหน่ง และสามารถกำหนดทิศทางของการเลื่อนได้จากบิตที่ 14 configuration memory (configmem[14])

2.2.3 ขั้นตอนการออกแบบโมเดลด้วยภาษาซี

เมื่อได้ออกแบบโครงสร้างการทำงาน ชุดคำสั่ง การเชื่อมต่อและดาต้าพาสของหน่วยประมวลผล FU ที่ต้องการแล้ว จึงได้ทำการออกแบบโมเดลของโพรเซสเซอร์ร่วมด้วยภาษาซีขึ้นเพื่อทดสอบการทำงานทั้งหมดก่อนที่จะสร้างวงจร โดยโมเดลนี้จะสามารถรับชุดคำสั่งจากไฟล์ที่ทำหน้าที่เสมือนเป็นหน่วยความจำ โดยคำสั่ง top-level จะถูกเก็บไว้ในไฟล์ p_mem และชุดคำสั่ง configuration memory จะอยู่ในไฟล์ programme ซึ่ง operand A และ operand B จะอยู่ในไฟล์ Data_X และ Data_Y ตามลำดับ

2.2.4 โครงสร้างของตัวแปลภาษา

ตัวแปลภาษาเครื่อง หรือการแปลภาษาระดับแอสแซมบลี้ไปเป็นภาษาเครื่องที่สามารถนำไปใช้ประมวลผลในโพรเซสเซอร์ได้เรียกว่า แอสแซมเบลอร์ (assembler) ถึงแม้ว่างานวิจัยนี้ไม่ได้มุ่งเน้นการสร้างตัวแปลภาษาโดยตรง แต่จะต้องทำการออกแบบและพัฒนาขึ้นเพื่อใช้ในการทดสอบโพรเซสเซอร์ที่ได้ออกแบบ แบ่งขั้นตอนของการแปลภาษาออกเป็น 1) ขั้นตอนการแปลจากระดับของภาษาแอสแซมบลี้แบบขนาน (parallel assembly) เป็นภาษาระดับกลาง (intermediate

code) 2) ขั้นตอนการแปลจาก intermediate code เป็นภาษาเครื่องตามรูปแบบของชุดคำสั่งที่ได้ออกแบบไว้ โดยผลลัพธ์ของตัวแปลภาษาจะถูกนำไปเป็นอินพุตของการทดสอบโพรเซสเซอร์บน simulation อีกครั้งหนึ่ง

2.3 เทคนิคการออกแบบ

2.3.1 เทคนิคการออกแบบ element พลังงานต่ำสำหรับ FU

แม้ว่าการออกแบบโพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนได้แบบไดนามิก เพื่อให้เหมาะกับสภาพการทำงานในแต่ละช่วงเวลา ทำให้สามารถประหยัดพลังงานได้มากกว่าการทำให้โพรเซสเซอร์ทำงานเต็มประสิทธิภาพอยู่ตลอดเวลา เช่นการให้โพรเซสเซอร์ active 4 หน่วยประมวลผลทำการประมวลผลการบวกเพียง 1 ค่า จะสิ้นเปลืองพลังงานมากกว่าการ active ให้มีหน่วยประมวลผลเพียงหน่วยเดียวทำการบวกอย่างง่ายๆ 1 ค่า

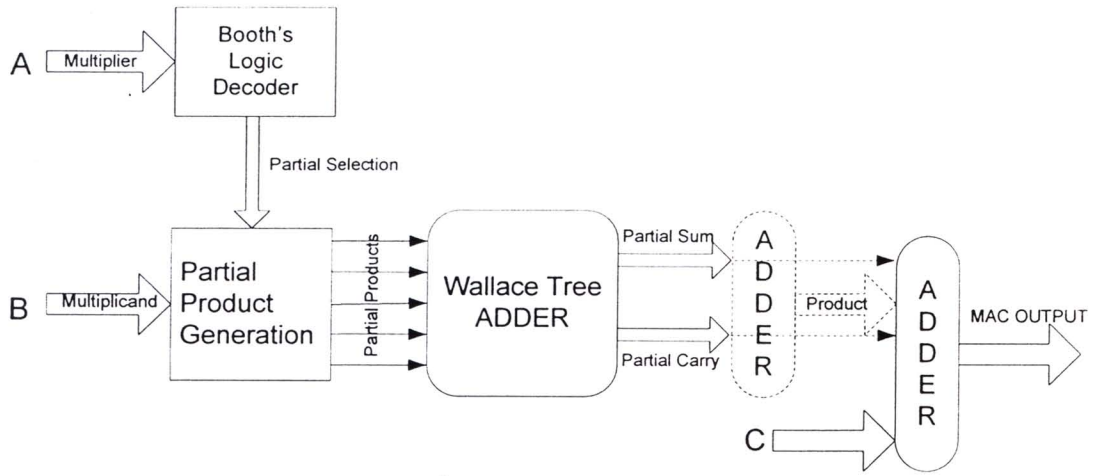
นอกจากนี้การออกแบบวงจรคำนวณในหน่วยประมวลผล FU ให้มีพลังงานต่ำจะช่วยให้ภาพรวมของการใช้พลังงานในการประมวลผลต่ำลงกว่าเดิม ในส่วนนี้นำเสนอเทคนิคการออกแบบวงจรพลังงานต่ำสำหรับใช้งานใน FU แต่การทดสอบจะใช้วิธีการนำจำนวนลอจิกเกตที่ใช้งานแทนการจำลองการทำงานจริง เนื่องจากบน FPGA ผู้ใช้ไม่สามารถควบคุมการโปรแกรมหรือการเลือกใช้ลอจิกเกตได้เหมือนกับการออกแบบวงจรแบบ ASIC

1) การลดรูปวงจรคำนวณคำสั่ง MAC

คำสั่ง MAC หรือเรียกเต็มๆว่า Multiply Accumulate (MAC) เกิดจากการลดรูปคำสั่งการคูณที่จำเป็นจะต้องทำการบวกตามหลังคำสั่งคูณทันที ดังนั้นฟังก์ชันการทำงานของคำสั่ง MAC คือ $Y = A * B + C$ ซึ่งการสร้างวงจรสำหรับคำนวณคำสั่ง MAC โดยทั่วไปจะใช้วงจรคูณกับวงจรบวกตามสมการที่กล่าวไว้ ซึ่งเมื่อพิจารณาแล้วพบว่าภายในวงจรคูณจะประกอบด้วยวงจรคูณบิตย่อยและวงจรบวก ดังสมการต่อไปนี้

$$Y = (PP_1 + PP_2 + PP_3 + \ldots + PPn) + C;\ PP = \text{partial product}$$
$$= \text{Partial Sum} + \text{Partial Carry} + C$$

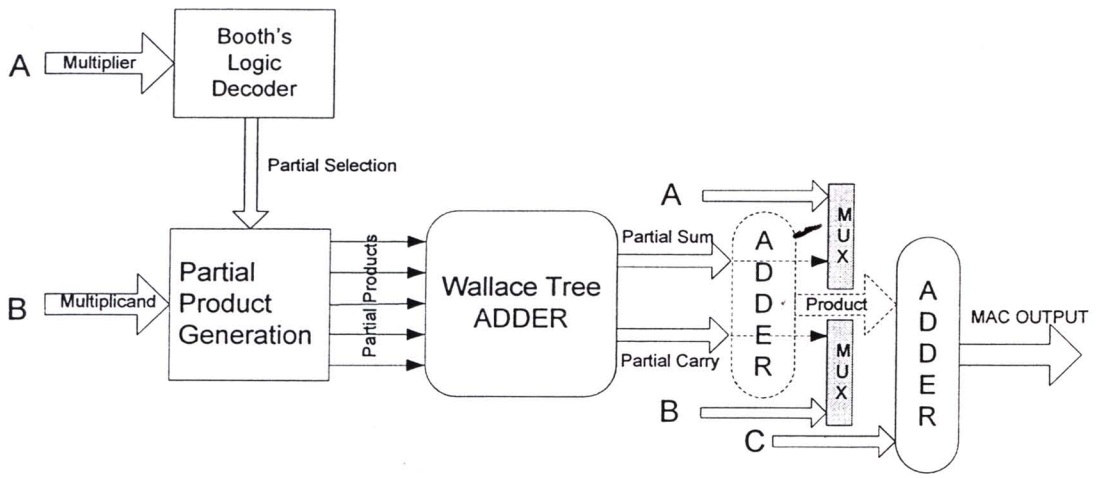
พบว่าพื้นฐานวงจรคูณคือการบวกค่า Partial Product (ผลคูณในแต่ละหลัก) โดยที่วงจรบวกที่นิยมใช้คือวงจรบวกแบบหลายอินพุตเช่น Wallace Tree หรือ Dadda Adder ซึ่งในท้ายที่สุดจะได้ผลลัพธ์ 2 ชุดคือ ผลรวมของ Partial Sum (PS) และ ผลรวมของบิตทด Partial Carry (PC) ในขั้นตอนสุดท้ายจึงใช้วงจรบวก 2 อินพุต ดังนั้นเพื่อเป็นการลดวงจรที่ซ้ำซ้อน จึงนำเสนอการบวกค่า C ที่ต้องการคำนวณตามคำสั่งของ MAC แทรกเข้าไปในวงจรบวกหลายอินพุต ทำให้สามารถลดรูปวงจรสำหรับการคำนวณ MAC ได้ดังรูปที่ 5 ซึ่งถ้าเป็นคำสั่ง MPY (Y = A*B) ก็จะกำหนดให้อินพุต C เป็นศูนย์ ดังนั้นวงจรในรูปที่ 5 สามารถลดรูปวงจรคำนวณ MAC และสามารถคำนวณการคูณปกติได้อีกด้วย



รูปที่ 5 การลดรูปวงจร MAC

2) การใช้วงจร MAC MPY ADD SUB ร่วมกัน

นอกจากนี้แล้วยังพบว่าคำสั่งที่สามารถใช้งานร่วมกับวงจร MAC และ MPY คือการคำนวณ ADD และ SUB โดยการเพิ่มวงจร multiplexer เข้าไปที่อินพุตทั้งสองตัวก่อนเข้าวงจร adder ชุดสุดท้าย ถ้าหากพบว่าเป็นคำสั่ง ADD หรือ SUB อินพุตจะเลือกเอาค่า A และ B เข้าวงจรบวกแล้วกำหนดให้ C เป็นศูนย์ จากนั้นถ้าพบว่าเป็นคำสั่ง SUB multiplexer ที่อินพุตจะทำการหาค่า 2's complement ของอินพุตก่อนเข้าสู่วงจรบวก ดังรูปที่ 6 ทำให้วงจรที่ได้มีจำนวนลอจิกน้อยกว่าวงจรของ ALU โดยทั่วไป

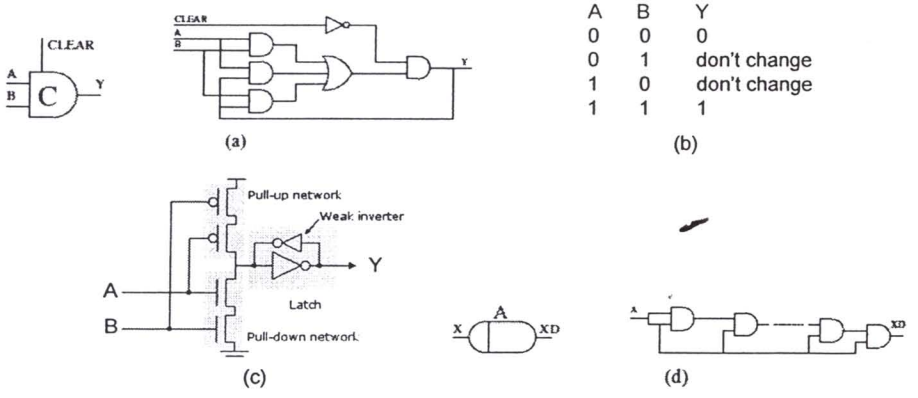


รูปที่ 6 การวงจรคำนวณ MAC MPY ADD และ SUB

2.3.2 การปรับวงจร asynchronous เป็น clock-gating สำหรับ FPGA

เนื่องจากผู้วิจัยมุ่งเน้นการออกแบบเพื่อให้ประหยัดพลังงาน วงจรส่วนใดที่ไม่ได้ใช้งานก็จะต้องไม่ทำงาน ซึ่งจะช่วยลดการสูญเสียพลังงานแบบไร้ประโยชน์ (Waste Energy) ดังนั้นหากเป็นการพัฒนาโพรเซสเซอร์ร่วมแบบ ASIC การนำวิธีการทำงานแบบ asynchronous หรือ clock-less จะสามารถทำงานในรูปแบบที่กระตุ้นวงจรเฉพาะที่จำเป็นให้ทำงานได้ แต่เนื่องจากประเทศไทยไม่ได้สนับสนุนสถาบันการศึกษาในการซื้อซอฟต์แวร์ช่วยออกแบบเช่น Cadence ที่ใช้กับการออกแบบวงจรแบบ ASIC ทำให้ไม่มีเครื่องมือในการออกแบบเหมือนเช่นขณะกำลังศึกษาในต่างประเทศ อีกทั้งราคาของการผลิตแบบ ASIC จะมีราคาสูง ทำให้ผู้วิจัยจึงได้ปรับเปลี่ยนรูปแบบโดยการนำเทคนิค clock-gating มาใช้แทน เพื่อให้สามารถทดสอบรูปแบบการทำงานแบบกระตุ้นให้วงจรทำงานเฉพาะที่จำเป็นเท่านั้น

โดยปกติการออกแบบวงจรแบบ asynchronous จะมีเกตพื้นฐานที่แตกต่างจากวงจรลอจิกทั่วไป เราเรียกเกตพื้นฐานสำหรับวงจรแบบ asynchronous ว่า Muller C-element โดยที่สามารถแทนการทำงานเป็นลอจิกทั่วไปได้ดังรูปที่ 7 (a) ซึ่งจะทำการหรือ fire เอาท์พุตเมื่ออินพุตทั้งสองมีค่าเหมือนกัน โดยที่ A, B = 0 จะได้เอาท์พุต 0 แต่ถ้า A,B = 1 จะได้เอาท์พุต 1 นอกนั้นจะต้องรักษาค่าเดิมไว้ดังรูปที่ 7 (b) แต่หากทำการสร้างวงจรสำหรับ ASIC โดยออกแบบทรานซิสเตอร์จะใช้ทรานซิสเตอร์เพียง 4 ตัวเท่านั้นดังรูปที่ 7 (c) ซึ่งน้อยกว่าลอจิกที่แสดงในรูปที่ 7 (a) หลายเท่าตัว ชี้ให้เห็นว่าการใช้งานแบบ asynchronous บน FPGA ทั่วไปไม่เหมาะสม ใช้สำหรับการทดสอบฟังก์ชันการทำงาน (functional test) เท่านั้น

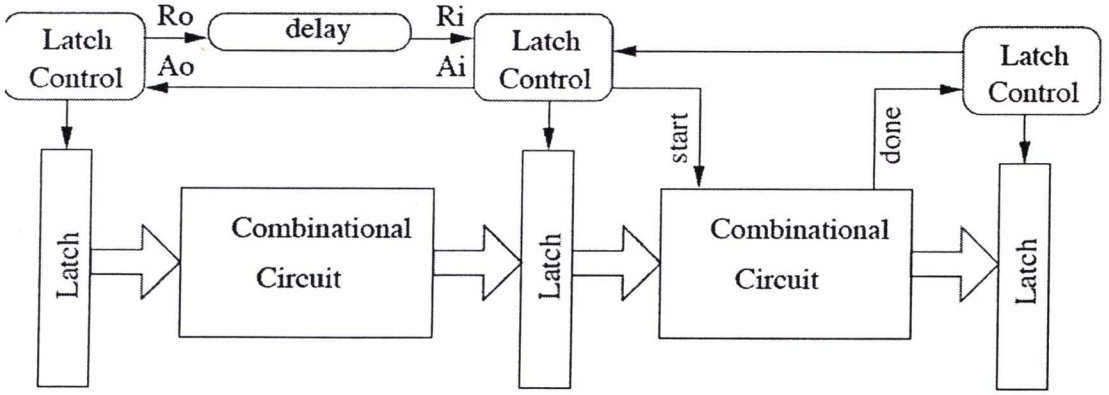


รูปที่ 7 ลอจิกของ C-element (a) ตารางค่าความจริงของ C-element (b) วงจรทรานซิสเตอร์ของ C-element (c) วงจรหน่วงเวลา (d)

การออกแบบวงจรไปป์ไลน์ทำงานในระดับการเชื่อมต่อระหว่างโพรเซสเซอร์ร่วมและโพรเซสเซอร์หลักตามที่ได้ออกแบบไว้แล้วนั้น วงจรที่ได้ออกแบบสำหรับการทำงานของไปป์ไลน์คือโครงสร้างแบบ bundled-data asynchronous ซึ่งตั้งอยู่บนพื้นฐานของไมโครไปป์ไลน์ มีการทำงานดังรูปที่ 8 ตัวพักข้อมูลชนิด Latch n จะส่งสัญญาณ request (R0) เมื่อพร้อมที่จะดำเนินงานใน

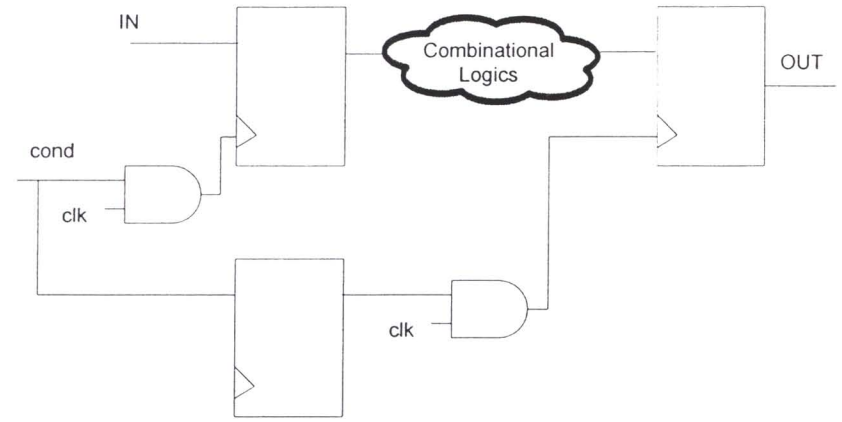
stage นี้ สัญญาณ R0 จะผ่านวงจรหน่วงเวลา หรือ delay ดังรูปที่ 7 (d) ซึ่งจะมีระยะเวลาเท่ากับการทำงานของวงจรใน stage นั้น จากนั้นสัญญาณ request (r1) ที่เกิดจากวงจรหน่วงเวลาจะส่งไปแจ้งตัวพักข้อมูล Latch ถัดไป Latch n+1 เพื่อแจ้งให้ทราบว่าการทำงานนั้นเสร็จสิ้นแล้ว วงจร Latch n+1 จะสร้างสัญญาณ enable เพื่อให้ทำการเก็บผลลัพธ์ในบัสเอาท์พุตทันที พร้อมทั้งสร้างสัญญาณแจ้งไปยัง Latch n ว่าได้รับข้อมูลแล้วผ่านทางสัญญาณ Acknowledge (Ai และ Ao)

พบว่าการทำงานเช่นนี้จำเป็นที่จะต้องมีตัวหน่วงเวลาที่มีระยะทำงานเท่านั้นวงจรการทำงานในช่วงนั้นๆ ทำให้จะต้องประเมินค่าหน่วงเวลาเป็นค่าที่มากที่สุดหรือเรียกว่า worse delay ส่งผลต่อความเร็วของระบบโดยรวม ดังนั้นจึงมีรูปแบบของการตรวจสอบระยะเวลาในการทำงานที่ให้สามารถใกล้เคียงกับการทำงานจริงคือการสร้างสัญญาณ start ส่งไป enable วงจรเมื่อทำงานเสร็จสิ้นจนได้ผลลัพธ์ก็จะสร้างสัญญาณ done โดยวงจรเช่นนี้จำเป็นที่จะต้องใช้งานลอจิกพื้นฐาน c-element เช่นกัน ดังนั้นการนำหลักการของวงจร asynchronous จึงไม่เหมาะและไม่สามารถดำเนินการได้จริงบน FPGA ที่มีอยู่ในปัจจุบันด้วยเหตุผลสรุปดังนี้ 1) จำเป็นที่จะต้องสร้างลอจิกพื้นฐาน c-element ขนาดใหญ่ 2) จะต้องเสียลอจิกสำหรับวงจรหน่วงเวลา 3) เนื่องจากไม่สามารถควบคุมการวาง (place) และเชื่อมต่อลอจิกได้ (route) ทำให้ยากต่อการควบคุมค่าหน่วงเวลาที่แม่นยำสำหรับการทำงานของวงจรได้ ซึ่งก่อให้เกิดความผิดพลาดของผลลัพธ์

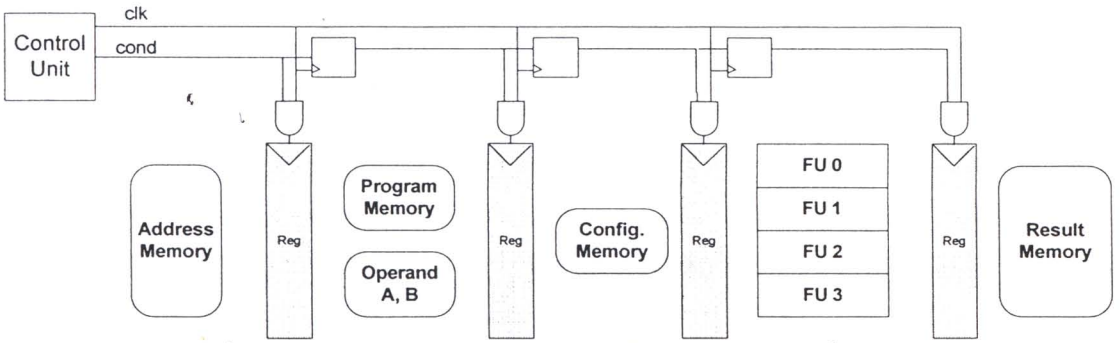


รูปที่ 8 โครงสร้างไมโครไปป์ไลน์

ในงานวิจัยนี้ได้ทำการปรับรูปแบบของโครงสร้างการทำงานแบบไมโครไปป์ไลน์เป็นแบบ clock-gating แทนเนื่องจากมีแนวคิดตรงกับสิ่งที่ทางผู้วิจัยต้องการที่จะไม่กระตุ้นวงจรให้ทำงานในส่วนที่ไม่ต้องการ ทำให้ช่วยลดพลังงานที่จะต้องสูญเสียไปโดยเปล่าประโยชน์ มีโครงการสร้างการทำงานเหมือนดังในรูปที่ 9 และรูปที่ 10 เมื่อพบว่าจะต้องกระตุ้นการทำงานในวงจรใดก็จะกำหนดให้สัญญาณ condition ที่ต่อเข้ากับลอจิกเกต AND คู่กับสัญญาณนาฬิกาเป็นลอจิก 1 ทำให้มีการส่งสัญญาณนาฬิกาไปยังวงจรส่วนนั้นๆเริ่มการทำงานทันที สำหรับส่วนใดที่ไม่ต้องการใช้งานก็สามารถให้ลอจิกเป็น 0 ข้อเสียของวงจรนี้คือการที่จะต้องมีวงจรถอดรหัสหรือ decoder เพิ่มขึ้นเพื่อสร้างสัญญาณ condition ให้สอดคล้องกับชุดคำสั่งที่จะประมวลผล



รูปที่ 9 clock gating



รูปที่ 10 การออกแบบ clock gating กับโพรเซสเซอร์ร่วมที่ออกแบบ

2.4 ผลการทดลอง

เมื่อได้ทำการออกแบบโครงสร้างแล้วจึงได้เริ่มสร้างโพรเซสเซอร์ร่วมด้วยภาษา Verilog เพื่อทดสอบบน FPGA และจำลองการทำงานด้วยโปรแกรม ISE Xilinx Simulation โดยการป้อนอินพุตแบบ random เพื่อทำงาน FIR filter มีค่า coefficient จำนวน 20 tap สามารถสรุปผลของทรัพยากรที่ใช้ไปดังต่อไปนี้

Slice Flip Flop	1,464
LUT 4-input	18,335
- Used as Logic	8,971
- Used as Route-thru	84
- Used for Dual port RAMs	9,024
- Used for 32x1 RAMs	256
Multiplier	4
Adder	4

*คิดเป็น 90% ของทรัพยากรทั้งหมดใน FPGA XC3SD1800

โพรเซสเซอร์ร่วมมีโครงสร้างที่สามารถโปรแกรมให้ใช้งานได้สูงสุด 4 FU (Functional Unit) ผลการจำลองการทำงานบนเทคโนโลยี FPGA XC3SD1800 ในกรณีที่ทำงานด้วย FU สูงสุด 4 ตัว พบว่าทำงานได้ด้วยความถี่ 59 MHz ใช้พลังงาน 34.64mW ซึ่งสามารถประมวลผล FIR 20-tap เสร็จภายในเวลา 0.339 ไมโครวินาที

2.5 สรุปผล

โพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิกที่ได้ออกแบบและพัฒนาขึ้นนี้สามารถทำงานได้ถูกต้อง โดยที่โพรเซสเซอร์หลักจะทำหน้าที่ควบคุมคำสั่งลำดับการทำงานและการจัดโครงสร้างให้กับโพรเซสเซอร์ร่วมได้ แต่เนื่องจากข้อจำกัดของบอร์ด FPGA ที่ใช้ทดสอบมีขนาดไม่ใหญ่เพียงพอที่จะสามารถสร้างทั้งระบบขึ้นทดสอบได้ ดังนั้นจึงใช้วิธีการจำลองการทำงาน ซึ่งหากต้องการทำให้เป็นระบบโดยสมบูรณ์ตั้งแต่ตัว compiler assembler และบอร์ดทดสอบทั้งโพรเซสเซอร์หลักและโพรเซสเซอร์ร่วมนั้นจะต้องดำเนินการวิจัยอีกหลายปี (คิดจากภาระงานของอาจารย์ในมหาวิทยาลัยที่มีการเรียนการสอนไม่ต่ำกว่าปีละ 70 Load Unit) ฉะนั้นโครงสร้างสถาปัตยกรรมที่ได้พัฒนาจึงเปรียบเสมือนแนวคิดและจำลองการทำงานเพื่อทดสอบแนวคิดเท่านั้น ผลที่ได้จากการทดสอบพบว่าสามารถทำงานด้วยพลังงานเพียง 34.64 mW และสามารถทำงาน Filter (FIR-20 tap) ได้อย่างรวดเร็วภายใน 0.34 ไมโครวินาทีเท่านั้น