รายงานวิจัยฉบับสมบูรณ์

โครงการ โพรเซสเซอร์ร่วมกำลังไฟต่ำสามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

โดย ผศ. ดร. วรรณรัช สันติอมรทัต และคณะ

กุมภาพันธ์ 2554

สัญญาเลขที่ MRG5080411

รายงานวิจัยฉบับสมบูรณ์

โครงการ โพรเซสเซอร์ร่วมกำลังไฟต่ำสามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

คณะผู้วิจัย	สังกัด
1. ผศ. ดร. วรรณรัช สันติอมรทัต	มหาวิทยาลัยสงขลานครินทร์
2. รศ. ดร. สินชัย กมลภิวงศ์	มหาวิทยาลัยสงขลานครินทร์

สนับสนุนโดย

สำนักงานกองทุนสนับสนุนการวิจัย

กิตติกรรมประกาศ

งานวิจัยนี้สำเร็จลุล่วงลงได้ ด้วยการสนับสนุนทุนวิจัย (ทุนพัฒนาศักยภาพในการทำงานวิจัยของอาจารย์รุ่นใหม่) จากสำนักงานคณะกรรมการอุดมศึกษาและสำนักงานกองทุนสนับสนุนการทำวิจัย ผู้เขียนขอขอบพระคุณ คณะผู้บริหารและเจ้าหน้าที่ของ สกว. ในทุกระดับชั้น ที่ได้ให้การสนับสนุนและช่วยเหลือในการทำงานวิจัยนี้

ผู้เขียนขอขอบคุณ รศ.ดร. สินชัย กมลภิวงศ์ที่ได้ออกหนังสือรับรองในการเป็นที่ปรึกษาให้กับโครงการวิจัยนี้

ผู้เขียนขอขอบพระคุณ อาจารย์ ผศ. อภิเนตร อูนากูล ภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ได้ให้แรงบันดาลและแนวคิดที่ช่วยเหลือในการทำงานหลายๆด้าน Dr. Jim Garside มหาวิทยาลัยแมนเชสเตอร์ ที่คอยให้คำปรึกษาแนวคิดผ่านทาง E-mail เสมอมา ขอขอบคุณกำลังใจจากคนที่ “บ้าน” ที่ช่วยให้ความท้อแท้หายไป แล้วสามารถเขียนงานได้สำเร็จ

สุดท้ายนี้ ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ครู อาจารย์ ที่ได้ให้ชีวิต เลี้ยงดู อบรม และ สั่งสอน จนผู้เขียนมีวันนี้ได้

บทคัดย่อ

รหัสโครงการ: MRG5080411

ชื่อโครงการ: โพรเซสเซอร์ร่วมกำลังไฟต่ำสามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

ชื่อนักวิจัย: ผศ. ดร. วรรณรัช สันติอมรทัต มหาวิทยาลัยสงขลานครินทร์

อีเมล์: wannarat@coe.psu.ac.th, suntiamorntut@gmail.com

ระยะเวลาโครงการ: 2 ปี

เนื่องจากหน่วยประมวลผลที่เป็นแบบ ASIC (application specific integrated circuit) สามารถตอบสนองความต้องการในเรื่องการทำให้พลังงานต่ำและสามารถให้ประสิทธิภาพการสูง แต่ราคาที่สูงของเทคโนโลยีแบบ ASIC ทำให้ไม่เหมาะที่จะนำมาใช้งานสำหรับผลผลิตทางอิเล็กทรอนิกส์ระดับกลาง – เล็ก สำหรับเทคโนโลยีแบบที่สามารถโปรแกรมได้ (programmable device) ที่อาจจะไม่ได้เปรียบในเชิงของการใช้พลังงาน แต่สามารถช่วยเพิ่มประสิทธิภาพและความยืดหยุ่นให้กับผู้ใช้งาน ในงานวิจัยนี้จึงใช้โครงสร้างแบบผสมระหว่าง ASIC ในบล็อคถือว่าเป็น element หนึ่ง โดยผู้ใช้สามารถควบคุมและโปรแกรมได้ ซึ่งมีโครงสร้างของ datapath ที่สามารถปรับเปลี่ยนได้ตามงานที่ได้รับ เราเรียกว่า dynamically reconfigurable datapath (DRD)

การปรับเปลี่ยนโครงสร้างภายในเช่นนี้ช่วยให้งานแต่ละชนิดสามารถใช้ทรัพยากรที่มีอยู่ได้อย่างคุ้มค่า ส่งผลให้เกิดการใช้พลังงานได้อย่างเหมาะสมที่สุด งานวิจัยนี้นำเสนอโพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนได้แบบไดนามิกที่ออกแบบด้วยเทคนิค clock-gating และนำเสนอแนวคิดของซอฟต์แวร์ช่วยออกแบบในการวางอุปกรณ์และการเชื่อมต่อที่ทำงานได้รวดเร็ว เพื่อลด overhead ในระหว่างการทำงาน

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 ไมโครวินาทีและสำหรับผลการทดสอบแนวคิดการวาง task แบบไดนามิกบน FPGA ของซอฟต์แวร์ช่วยออกแบบ พบว่าใช้อัลกอริทึมแบบ first fit และ best fit ใช้ความเร็ว และจำนวนของ task ที่วางลงบน FPGA ไม่ได้ไม่แตกต่างกัน ดังนั้นจึงเลือกใช้อัลกอริทึมแบบ first fit เนื่องจากมีจำนวนของพื้นที่ว่างน้อยกว่า

คำหลัก: dynamically reconfigurable datapath, run-time placement algorithm, FPGA

Abstract

Project Code: MRG5080411

Project Title: Low Power Coprocessor using Dynamically Reconfiguration

Investigator: Asst. Dr. Wannarat Suntiamorntut Prince of Songkla University

E-mail Address: wannarat@coe.psu.ac.th, suntiamorntut@gmail.com

Project Period: 2 years

ASIC (application specific integrated circuit) processor dissipates a low power consumption and is able to perform at a high speed. Unfortunately, ASIC production costs every high expense. Therefore, it is not suitable for a prototype or a small volume product. A programmable device becomes a good choice. Reconfigurable hardware will give a better performance and flexibility. This research work proposes Dynamically Reconfigurable Datapath (DRD) which is to reconfigure a part of FPGA while the processor of FPGA is still working.

This reconfigurable datapath can increase resource utilization and sharing when image or video processing applications are applied. Thus it also increases the energy efficiency. This dynamically reconfigurable co-processor employs clock-gating technique. We also introduce the concept of computer aided software for dynamic reconfigurable design which is able to reduce an overhead of place and route algorithm.

This dynamic reconfigurable co-processor can be programmed and used up to four parallel functional units (FUs). The design was implemented on FPGA XC3SD1800. The simulation result shows that the co-processor can run at 59 MHz and dissipated 34.64 mW which can complete FIR 20-tap within 0.339 us. The preliminary result of task placement suggests that both first fit and best fit algorithms give the similar speed and the number of task that cannot be placed. Therefore, we choose first fit algorithm in our EDA tool because of the lower list of the empty space.

Keywords: dynamically reconfigurable datapath, run-time placement algorithm, FPGA

สารบัญ

หน้าสรุปโครงการ (Executive Summary)

ทุนพัฒนาศักยภาพในการทำงานวิจัยของอาจารย์รุ่นใหม่

---

1. ชื่อโครงการ โพรเซสเซอร์ร่วมกำลังไฟต่ำสามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

Low Power Coprocessor using Dynamically Reconfiguration

2. ชื่อหัวหน้าโครงการ หน่วยงานที่สังกัด ที่อยู่ หมายเลขโทรศัพท์ โทรสาร และ e-mail

ชื่อ น.ส. วรรณรัช สันติอมรทัต

(Ms. Wannarat Suntiamorntut)

คุณวุฒิ ปริญญาเอก

ตำแหน่ง ผู้ช่วยศาสตราจารย์

สถานที่ทำงาน ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยสงขลานครินทร์ อ.หาดใหญ่ จ.สงขลา 90112

โทรศัพท์ 074-287354 หรือ 074-287075

โทรสาร 074-287076

e-mail: wannarat@coe.psu.ac.th, wsuntiamorntut@gmail.com

3. ปัญหาที่ทำการวิจัย และความสำคัญของปัญหา

ปัจจุบันระบบสมองกลฝังตัวได้ถูกนำมาใช้งานกันอย่างแพร่หลายเนื่องมีขนาดเล็ก สะดวกต่อการใช้งานและช่วยลดการใช้พลังงาน ระบบสมองกลฝังตัวได้เริ่มพัฒนาจากระดับเล็กโดยการใช้งานไมโครคอนโทรลเลอร์ขนาด 8 บิตควบคุมอุปกรณ์อิเล็กทรอนิกส์ จากนั้นได้มีการพัฒนาอย่างต่อเนื่องเรื่อยมาจนกระทั่งมีการนำโพรเซสเซอร์ขนาดใหญ่ 16 – 32 บิต เข้ามาใช้งานโดยเฉพาะอย่างยิ่งกับงานที่จะต้องใช้การประมวลผลอย่างหนักเช่น โทรศัพท์มือถือรุ่นที่ 3 (3G) คอมพิวเตอร์พกพาขนาดเล็ก เครื่องเล่นเกมส์ Play Station (PS) เป็นต้น จะพบได้ว่าระบบสมองกลฝังตัวในปัจจุบันต้องรับภาระงานหนักมากขึ้น โดยเฉพาะอย่างยิ่งในการทำงานด้าน Multimedia applications หรือ Recognition applications เป็นต้น ซึ่งงานทางด้านนี้เกี่ยวข้องกับการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing) ที่จะต้องใช้หน่วยประมวลผล (Functional Unit; FU) ทำการคำนวณอยู่ตลอดเวลา ดังนั้นจำเป็นที่ระบบจะต้องการความรวดเร็วในการทำงาน โครงสร้างของโพรเซสเซอร์ทั่วไปจึงไม่เหมาะสำหรับการใช้งานประเภทนี้ นอกจากนี้แล้วระบบสมองกลฝังตัวจะมีขนาดเล็กและสามารถใช้ไฟจากแรงจ่ายแบตเตอร์รี่เท่านั้น ส่งผลให้งานวิจัยที่มุ่งเน้นการออกแบบหน่วยประมวลผลให้มีประสิทธิภาพสูงและใช้พลังงานต่ำเป็นที่น่าสนใจและท้าทาย

เนื่องจากหน่วยประมวลผลที่เป็นแบบ ASIC (application specific integrated circuit) สามารถตอบสนองความต้องการในเรื่องการทำให้พลังงานต่ำและสามารถให้ประสิทธิภาพการสูง แต่ราคาที่สูงของเทคโนโลยีแบบ ASIC ทำให้ไม่เหมาะที่จะนำมาใช้งานสำหรับผลผลิตทาง

อิเล็กทรอนิกส์ระดับกลาง – เล็ก สำหรับเทคโนโลยีแบบที่สามารถโปรแกรมได้ (programmable device) ที่อาจจะไม่ได้เปรียบในเชิงของการใช้พลังงานและประสิทธิภาพในการทำงาน แต่ถือว่าช่วยเพิ่มความยืดหยุ่นให้กับนักออกแบบและผู้ใช้งาน รวมทั้งช่วยหย่นระยะเวลาของการออกแบบและพัฒนาได้ ฉะนั้นในงานวิจัยนี้จึงใช้โครงสร้างแบบผสมระหว่าง ASIC ในบล็อคถือว่าเป็น element หนึ่ง โดยให้การต่อเชื่อมถึงกัน (Interconnection) นั้นสามารถถูกควบคุมและโปรแกรมได้โดยผู้ใช้งานถือได้ว่าเป็นสถาปัตยกรรมแบบ hybrid โดยจะสามารถสร้างโพรเซสเซอร์อย่างง่ายขนาดเล็กขึ้นมาได้จากการประกอบกันของ element ต่างๆ ให้เป็น datapath โดยโครงสร้างของ datapath นี้จะสามารถปรับเปลี่ยนได้ตามงานที่ได้รับ เราเรียกว่า dynamically reconfigurable datapath (DRD)

การปรับเปลี่ยนโครงสร้างภายในเช่นนี้ช่วยให้งานแต่ละชนิดสามารถใช้ทรัพยากรที่มีอยู่ได้อย่างคุ้มค่า ส่งผลให้เกิดการใช้พลังงานได้อย่างเหมาะสมที่สุด แต่เนื่องจากการปรับเปลี่ยนและก่อตัวเป็นโครงสร้าง datapath นั้นจะต้องใช้เวลา overhead ที่เกิดขึ้นจนเป็นปัญหาที่น่าสนใจคือ *จะต้องปรับเปลี่ยนโครงสร้างแบบ dynamically ในลักษณะใดจึงจะช่วยให้ประสิทธิภาพและการใช้พลังงานโดยรวมของระบบดีขึ้น* ดังนั้นหัวข้อสำคัญในการศึกษาค้นคว้าและทำวิจัยคือ

- ลด overhead ที่เกิดจากระยะเวลาในการ form โครงสร้างการทำงาน เพื่อให้เป็นระบบที่สามารถ reconfigurable แบบ dynamically อย่างแท้จริง
- เปรียบเทียบประสิทธิภาพและพลังงานระหว่างโครงสร้างที่ปรับเปลี่ยนได้อย่างเร็วรวดกับโครงสร้างที่ปรับเปลี่ยนได้ภายในระยะเวลาที่กำหนดแต่จะให้มีจำนวนครั้งในการปรับเปลี่ยนน้อยที่สุด
- เพื่อให้ได้ประสิทธิภาพอย่างแท้จริง ดังนั้นการปรับเปลี่ยนโครงสร้างควรเป็น datapath ในระดับของ Functional Unit (FU) ขึ้นไป หรือรวมถึงโครงสร้างภาย Functional Unit ด้วย
- โครงสร้างในการเชื่อมต่อที่มีประสิทธิภาพและพลังงานต่ำ ที่รวมถึง routing และ switching ซึ่งที่มีอยู่จะเป็นแบบใช้สัญญาณนาฬิกา synchronize ถึงกัน แต่ในงานวิจัยนี้มุ่งเน้นแบบไม่ใช้สัญญาณนาฬิกา (Asynchronous)
- ลักษณะการวาง hardwired unit ในที่นี้ควรจะเป็นระดับ FU หรือ element สำหรับการประมวลผลทั่วไป เพื่อให้ประหยัดพื้นที่และการเชื่อมต่อสายสัญญาณระหว่างกัน
- โมเดลที่ช่วยวิเคราะห์การออกแบบและกำหนดรูปแบบการปรับเปลี่ยนโครงสร้างที่ให้ได้ประสิทธิภาพตามที่ต้องการ
- นำเสนอ Design flow ที่ช่วยให้การออกแบบและพัฒนาสะดวกและมีประสิทธิภาพมายิ่งขึ้น

4. วัตถุประสงค์

4.1 เพื่อศึกษา วิจัย พัฒนาการปรับเปลี่ยนโครงสร้างของ datapath ให้ได้ประสิทธิภาพและประหยัดพลังงานมากที่สุด

4.2 เพื่อลด overhead (เวลา, พื้นที่, ขนาดของ configuration bit, พลังงาน) ที่เกิดจากการปรับเปลี่ยนโครงสร้างแบบ dynamically

4.3 เพื่อสร้างโพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนโครงสร้างแบบ dynamically ที่มีประสิทธิภาพและประหยัดพลังงานมากที่สุดได้

4.4 เพื่อพัฒนา design flow ในการ map โค้ดโปรแกรมที่ออกแบบโดยโปรแกรมเมอร์ลงบนโพรเซสเซอร์ที่ได้ออกแบบ

5. ระเบียบวิธีวิจัย

5.1 ศึกษาโครงสร้างทั่วไปของสถาปัตยกรรมแบบ reconfigurable

5.2 ศึกษาการปรับเปลี่ยนโครงสร้างแบบ dynamically

5.3 ศึกษาและวิเคราะห์ความละเอียดของโครงสร้างที่จะทำการปรับเปลี่ยนใหม่ (ความละเอียดของการ Coarse-grain)

5.4 ออกแบบและพัฒนาวิธีการปรับเปลี่ยนโครงสร้างแบบ dynamically ที่ใช้เวลาน้อย

5.5 ออกแบบและพัฒนาวิธีการ Place และ Route ที่เหมาะกับโครงสร้างที่สามารถปรับเปลี่ยนได้แบบ dynamically

5.6 สร้างโมเดลขึ้นเพื่อทดสอบสถาปัตยกรรมที่ออกแบบขึ้นใหม่

5.7 พัฒนาโพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างได้แบบ dynamically บนเทคโนโลยี FPGA

5.8 ทดสอบและเก็บผล

5.9 นำเสนอเครื่องมือช่วยในการออกแบบ การ place & route

5.10 วิเคราะห์และสรุปทำงานรายงาน

ตารางที่ 1 แผนการดำเนินการตลอดโครงการวิจัย

ขั้นตอน	วิธีการ	ระยะเวลา	วัตถุประสงค์ข้อที่
1	ทำการศึกษาโครงสร้างสถาปัตยกรรมภายในของ FPGA เนื่องจากเป็นอุปกรณ์ที่สามารถโปรแกรมเพื่อปรับเปลี่ยนโครงสร้างได้ โดยมีแหล่งข้อมูลจากหนังสือและเอกสารรายงานการวิจัย	1 เดือน	5.1
2	ศึกษาขั้นตอน กระบวนการ ของการปรับเปลี่ยนโครงสร้างเพื่อนำมาวิเคราะห์ถึงข้อจำกัดของการทำแบบ dynamically (สามารถหาข้อมูลได้จากบทความทางวิจัย)	1 เดือน	5.1
3	ศึกษา วิเคราะห์โครงสร้างแบบ coarse-grain และทำการหาความหยาบของการทำ coarse-grain โดยการสร้างโมเดลภาษาซีเพื่อทดสอบ	1 เดือน	5.1
4	ออกแบบและพัฒนาการปรับเปลี่ยนโครงสร้างแบบ dynamically ให้เกิด overhead (เวลา, พื้นที่, การใช้พลังงาน) น้อย โดยการสร้างโมเดลที่สามารถคำนวณหา overhead แล้วนำวิธีการหรือเทคนิคที่ได้คิดค้นมาทดสอบและวัดผล	3 เดือน	5.1, 5.2
5	พัฒนาการ place และ route โดยการสร้างวงจรที่ได้จากการออกแบบขึ้นเพื่อทดสอบ โดยจะทำการวัดผลจากการใช้โปรแกรมจำลองการทำงานเพื่อวัดหาค่า power, delay ที่เกิดขึ้น	2 เดือน	5.2, 5.3
6	สร้างโมเดลของโพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างได้ด้วยภาษาซีแล้วทำการวัดผลทดสอบขั้นต้นด้วยโมเดลที่ใช้คำนวณวัดค่า overhead จากขั้นตอนที่ 4	4 เดือน	5.3
7	ทำการพัฒนาโพรเซสเซอร์ที่ได้ผ่านการทดสอบแล้วลงบนเทคโนโลยี FPGA โดยจะทำการวัดผลเฉพาะในการส่วนที่ทำงานหลังจากการโปรแกรมโครงสร้างแล้ว โดยระยะเวลาและ overhead ของการโปรแกรมเพื่อปรับเปลี่ยนโครงสร้างจะใช้จากโมเดลในขั้นตอนที่ 4 ดังนั้นจึงจะได้ประสิทธิภาพโดยรวมทั้งหมดของทั้งระบบ	4 เดือน	5.3
8	ทำการทดสอบ เก็บและวิเคราะห์ผล	2 เดือน	5.3
9	พัฒนาเครื่องช่วยในการ mapping และใช้งานโดยให้โปรแกรมเมอร์สามารถถ่ายโปรแกรมในระดับภาษาซีลงบนระบบได้	3 เดือน	5.4
10	จัดทำรายงาน	3 เดือน	-

6. ผลงาน

ผลงานการตีพิมพ์ในที่ประชุมวิชาการ (ที่เกี่ยวข้องกับงานวิจัยนี้)

1. Sasatorn somwatee, Wannarat Suntiamorntut, survey of finding empty space algorithm for partial reconfigurable FPGAs, Proceedings *of International Joint Conference on Computer Science and Software Engineering* (JCSSE2009), pp. 489-491, 2009.
2. Sasatorn somwatee, Wannarat Suntiamorntut, performance evaluation of the efficient algorithms for free resources management on the FPGA, Proceedings *of International Joint Conference on Computer Science and Software Engineering* (JCSSE2009), pp. 303-305, 2009.
3. W. Suntiamorntut, C. Vongchumyen, Design Techniques for Energy Efficient Multiplier, Ladkrabang Journal, June, 2007
4. W. Suntiamorntut, L.E.M. Brackenbury, Jim Garside Design and Implementation of an Energy Efficient, Parallel, Asynchronous DSP, ITC-CSCC07, July, pp. xxx, 2007.

บทที่ 1
บทนำ

1.1 ชื่อโครงการวิจัย

ภาษาไทย : โพรเซสเซอร์ร่วมกำลังไฟต่ำสามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

ภาษาอังกฤษ : Low Power Coprocessor using Dynamically Reconfiguration

1.2 สาขาวิชาที่ทำการวิจัย : วิศวกรรมไฟฟ้า เกี่ยวกับการออกแบบวงจรรวมดิจิตอล การออกแบบโพรเซสเซอร์ การออกแบบวงจรพลังงานต่ำ

1.3 ความสำคัญและที่มาของปัญหาที่ทำการวิจัย

ปัจจุบันระบบสมองกลฝังตัวได้ถูกนำมาใช้งานกันอย่างแพร่หลายเนื่องมีขนาดเล็ก สะดวกต่อการใช้งานและช่วยลดการใช้พลังงาน ระบบสมองกลฝังตัวได้เริ่มพัฒนาจากระดับเล็กโดยการใช้งานไมโครคอนโทรลเลอร์ขนาด 8 บิตควบคุมอุปกรณ์อิเล็กทรอนิกส์ จากนั้นได้มีการพัฒนาอย่างต่อเนื่องเรื่อยมาจนกระทั่งมีการนำโพรเซสเซอร์ขนาดใหญ่ 16 – 32 บิต เข้ามาใช้งานโดยเฉพาะอย่างยิ่งกับงานที่จะต้องใช้การประมวลผลอย่างหนักเช่น โทรศัพท์มือถือรุ่นที่ 3 (3G) คอมพิวเตอร์พกพาขนาดเล็ก เครื่องเล่นเกมส์ Play Station (PS) เป็นต้น จะพบได้ว่าระบบสมองกลฝังตัวในปัจจุบันต้องรับภาระงานหนักมากขึ้น โดยเฉพาะอย่างยิ่งในการทำงานด้าน Multimedia applications หรือ Recognition applications เป็นต้น ซึ่งงานทางด้านนี้เกี่ยวข้องกับการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing) ที่จะต้องใช้หน่วยประมวลผล (Functional Unit; FU) ทำการคำนวณอยู่ตลอดเวลา ดังนั้นจำเป็นที่ระบบจะต้องการความรวดเร็วในการทำงาน โครงสร้างของโพรเซสเซอร์ทั่วไปจึงไม่เหมาะสำหรับการใช้งานประเภทนี้ นอกจากนี้แล้วระบบสมองกลฝังตัวจะมีขนาดเล็กและสามารถใช้ไฟจากแรงจ่ายแบตเตอร์รี่เท่านั้น ส่งผลให้งานวิจัยที่มุ่งเน้นการออกแบบหน่วยประมวลผลให้มีประสิทธิภาพสูงและใช้พลังงานต่ำเป็นที่น่าสนใจและท้าทาย

เนื่องจากหน่วยประมวลผลที่เป็นแบบ ASIC (application specific integrated circuit) สามารถตอบสนองความต้องการในเรื่องการทำให้พลังงานต่ำและสามารถให้ประสิทธิภาพการสูง แต่ราคาที่สูงของเทคโนโลยีแบบ ASIC ทำให้ไม่เหมาะที่จะนำมาใช้งานสำหรับผลผลิตทางอิเล็กทรอนิกส์ระดับกลาง – เล็ก สำหรับเทคโนโลยีแบบที่สามารถโปรแกรมได้ (programmable device) ที่อาจจะไม่ได้เปรียบในเชิงของการใช้พลังงานและประสิทธิภาพในการทำงาน แต่ถือว่าช่วยเพิ่มความยืดหยุ่นให้กับนักออกแบบและผู้ใช้งาน รวมทั้งช่วยหย่นระยะเวลาของการออกแบบและพัฒนาได้ ฉะนั้นในงานวิจัยนี้จึงใช้โครงสร้างแบบผสมระหว่าง ASIC ในบล็อคถือว่าเป็น element หนึ่ง โดยให้การต่อเชื่อมถึงกัน (Interconnection) นั้นสามารถถูกควบคุมและโปรแกรมได้โดยผู้ใช้งานถือได้ว่าเป็นสถาปัตยกรรมแบบ hybrid โดยจะสามารถสร้างโพรเซสเซอร์อย่างง่ายขนาด

เล็กขึ้นมาได้จากการประกอบกันของ element ต่างๆ ให้เป็น datapath โดยโครงสร้างของ datapath นี้จะสามารถปรับเปลี่ยนได้ตามงานที่ได้รับ เราเรียกว่า dynamically reconfigurable datapath (DRD)

การปรับเปลี่ยนโครงสร้างภายในเช่นนี้ช่วยให้งานแต่ละชนิดสามารถใช้ทรัพยากรที่มีอยู่ได้อย่างคุ้มค่า ส่งผลให้เกิดการใช้พลังงานได้อย่างเหมาะสมที่สุด แต่เนื่องจากการปรับเปลี่ยนและก่อตัวเป็นโครงสร้าง datapath นั้นจะต้องใช้เวลา overhead ที่เกิดขึ้นจนเป็นปัญหาที่น่าสนใจคือ *จะต้องปรับเปลี่ยนโครงสร้างแบบ dynamically ในลักษณะใดจึงจะช่วยให้ประสิทธิภาพและการใช้พลังงานโดยรวมของระบบดีขึ้น* ดังนั้นหัวข้อสำคัญในการศึกษาค้นคว้าและทำวิจัยในโครงการนี้คือ

- ลด overhead ที่เกิดจากระยะเวลาในการ form โครงสร้างการทำงาน เพื่อให้เป็นระบบที่สามารถ reconfigurable แบบ dynamically อย่างแท้จริง
- เปรียบเทียบประสิทธิภาพและพลังงานระหว่างโครงสร้างที่ปรับเปลี่ยนได้อย่างเร็วรวดกับโครงสร้างที่ปรับเปลี่ยนได้ภายในระยะเวลาที่กำหนดแต่จะให้มีจำนวนครั้งในการปรับเปลี่ยนน้อยที่สุด
- เพื่อให้ได้ประสิทธิภาพอย่างแท้จริง ดังนั้นการปรับเปลี่ยนโครงสร้างควรเป็น datapath ในระดับของ Functional Unit (FU) ขึ้นไป หรือรวมถึงโครงสร้างภาย Functional Unit ด้วย
- โครงสร้างในการเชื่อมต่อที่มีประสิทธิภาพและพลังงานต่ำ ที่รวมถึง routing และ switching ซึ่งที่มีอยู่จะเป็นแบบใช้สัญญาณนาฬิกา synchronize ถึงกัน แต่ในงานวิจัยนี้มุ่งเน้นแบบไม่ใช้สัญญาณนาฬิกา (Asynchronous)
- ลักษณะการวาง hardwired unit ในที่นี้ควรจะเป็นระดับ FU หรือ element สำหรับการประมวลผลทั่วไป เพื่อให้ประหยัดพื้นที่และการเชื่อมต่อสายสัญญาณระหว่างกัน
- โมเดลที่ช่วยวิเคราะห์การออกแบบและกำหนดรูปแบบการปรับเปลี่ยนโครงสร้างที่ให้ได้ประสิทธิภาพตามที่ต้องการ
- นำเสนอ Design flow ที่ช่วยให้การออกแบบและพัฒนาสะดวกและมีประสิทธิภาพมายิ่งขึ้น

1.4 วัตถุประสงค์ของโครงการวิจัย

1.4.1 เพื่อศึกษา วิจัย พัฒนาการปรับเปลี่ยนโครงสร้างของ datapath ให้ได้ประสิทธิภาพและประหยัดพลังงานมากที่สุด

1.4.2 เพื่อลด overhead (เวลา, พื้นที่, ขนาดของ configuration bit, พลังงาน) ที่เกิดจากการปรับเปลี่ยนโครงสร้างแบบ dynamically

1.4.3 เพื่อสร้างโพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนโครงสร้างแบบ dynamically ที่มีประสิทธิภาพและประหยัดพลังงานมากที่สุดได้

1.4.4 เพื่อพัฒนา design flow ในการ map โค้ดโปรแกรมที่ออกแบบโดยโปรแกรมเมอร์ลงบนโพรเซสเซอร์ที่ได้ออกแบบ

1.5 ผลงานวิจัยที่เกี่ยวข้องและเอกสารอ้างอิง

โครงสร้างที่สามารถโปรแกรมได้ (programmable architecture) จะถูกนำมาใช้งานเพื่อสร้างสถาปัตยกรรมที่สามารถปรับเปลี่ยนโครงสร้าง (reconfigurable) การทำงานได้ตามการใช้งาน มีงานวิจัยจำนวนมากเกี่ยวกับโพรเซสเซอร์ที่สามารถโปรแกรมได้สร้างอยู่บน reconfigurable logic โดยมีจุดประสงค์เพื่อเพิ่มความเร็วในการทำงานสำหรับประยุกต์ใช้งานเฉพาะด้าน โดยที่โพรเซสเซอร์ที่เข้ามาช่วยเหล่านี้สามารถแบ่งออกได้เป็น 2 ประเภทคือ

- fine-grain ตัวอย่างเช่น GARP[2], NAPA[3], Chimaera[4] และ PRISC[5]
- coarse-grain ตัวอย่างเช่น Pleiades[6], PipeRench[7], Chameleon[8], RAPID[9] และ MorphoSys[10]

จากการศึกษาในงานวิจัยข้างต้นพบว่ารูปแบบ coarse-grain มีข้อดีเหนือกว่าแบบ fine-grain ในแง่ของความเร็วในการปรับเปลี่ยนโครงสร้าง (reconfigurable) ใช้ขนาดของ configuration bit ที่จะไปโปรแกรม FPGA น้อย และสามารถทำงานได้ด้วยสัญญาณนาฬิกาความถี่สูง ดังนั้น coarse-grain จึงเหมาะกับงานที่มีการใช้ข้อมูลจำนวนมากอย่างเช่น multimedia application และ communication ส่วน fine-grain เหมาะสำหรับงานที่ต้องการคำนวณในระดับบิต ซึ่งในงานวิจัยนี้ใช้แบบ coarse-grain แต่จะทำการศึกษาและวิเคราะห์ว่าการทำงานแบบ coarse-grain ในระดับ FU หรือ block element (multiplier, adder) ที่จะให้การใช้พลังงานอย่างมีประสิทธิภาพ

เมื่อมีการอนุญาตให้ปรับเปลี่ยนโครงสร้างบนอุปกรณ์ที่สามารถโปรแกรมได้ทำให้ส่วนของการ routing เชื่อมต่อแต่ละหน่วยถึงกันให้มีประสิทธิภาพมีงานวิจัย [11, 12] ได้เสนออัลกอริทึมในการ routing แต่งานวิจัยจำนวนหนึ่งที่มุ่งเน้นในถึงการโครงสร้างของการเชื่อมต่ออาทิเช่นในงานวิจัยใช้การเชื่อมต่อแบบบัสหลายทาง[13] และใช้การเชื่อมต่อเป็นระบบเครือข่ายบนชิป [14,15,16] ซึ่งในงานวิจัยนี้จะใช้แนวทางของการเชื่อมต่อเป็นเครือข่ายเนื่อง overhead ที่เกิดจากบัสจะเพิ่มขึ้นตามจำนวนของ element ที่ใช้ ในงานวิจัย [17] ชี้ให้เห็นว่าสามารถนำการเชื่อมต่อแบบเครือข่ายและใช้การส่งผ่าน packet ลงใน Network on Chip (NOC) [15]

เมื่อได้โพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างการเชื่อมต่อได้แล้ว การที่จะให้ผู้ใช้สามารถทำให้โพรเซสเซอร์ได้อย่างมีประสิทธิภาพ ฉะนั้นการพัฒนาเครื่องมือช่วยในการ map การแปลโปรแกรมภาษาระดับสูงอย่างเช่นภาษาซี จึงต้องถูกนำมาพิจารณา ดังนั้นจึงต้องศึกษางานวิจัยที่เกี่ยวข้องกับการพัฒนา design flow ในอุปกรณ์ที่สามารถโปรแกรมได้[18,19,20] รวมถึงเทคนิคการจัดแบ่งงานระหว่างซอฟต์แวร์และฮาร์ดแวร์ [21,22]

1.6 ระเบียบวิธีวิจัย

ขั้นตอนการดำเนินงาน

1.6.1 ศึกษาโครงสร้างทั่วไปของสถาปัตยกรรมแบบ reconfigurable

1.6.2 ศึกษาการปรับเปลี่ยนโครงสร้างแบบ dynamically

1.6.3 ศึกษาและวิเคราะห์ความละเอียดของโครงสร้างที่จะทำการปรับเปลี่ยนใหม่ (ความละเอียดของการ Coarse-grain)

1.6.4 ออกแบบและพัฒนาวิธีการปรับเปลี่ยนโครงสร้างแบบ dynamically ที่ใช้เวลาน้อย

1.6.5 ออกแบบและพัฒนาวิธีการ Place และ Route ที่เหมาะกับโครงสร้างที่สามารถปรับเปลี่ยนได้แบบ dynamically

1.6.6 สร้างโมเดลขึ้นเพื่อทดสอบสถาปัตยกรรมที่ออกแบบขึ้นใหม่

1.6.7 พัฒนาโพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างได้แบบ dynamically บนเทคโนโลยี FPGA

1.6.8 ทดสอบและเก็บผล

1.6.9 พัฒนาเครื่องมือช่วยในการออกแบบ การ mapping และการแปลโปรแกรมจากภาษาซี

1.6.10 วิเคราะห์และสรุปทำงานรายงาน

1.7 ขอบเขตของโครงการวิจัย

1.7.1 สร้างโพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างแบบ dynamically ให้มีประสิทธิภาพสูงและใช้พลังงานต่ำ

1.7.2 โครงสร้างโพรเซสเซอร์ใหม่ที่สามารถปรับเปลี่ยนได้แบบ dynamically จะถูกทดสอบบน FPGA

1.7.3 ใช้วิธีการจำลองการทำงานเพื่อตรวจสอบและวัดผล

1.8 แผนการดำเนินงานวิจัยตลอดโครงการ

กิจกรรมตลอดโครงการแสดงดังตารางที่ 1 โดยมีภาพรวมดังนี้

ปีที่ 1 เดือนที่ 1-6: กิจกรรมตามขั้นตอนที่ 1 โดยมีผลลัพธ์ที่คาดว่าจะได้รับคือ โมเดลวงจรใหม่ที่สามารถสร้างระบบ DSP บน FPGAs ได้

ปีที่ 1 เดือนที่ 7-12: กิจกรรมตามขั้นตอนที่ 2 พัฒนาอัลกอริทึมสำหรับโมเดลวงจรที่พัฒนาได้

ปีที่ 2 เดือนที่ 1-6: กิจกรรมตามขั้นตอนที่ 3 พัฒนาซอฟแวร์ตามอัลกอริทึมที่พัฒนาได้

ปีที่ 2 เดือนที่ 7-12: กิจกรรมตามขั้นตอนที่ 4 ทดสอบและวิเคราะห์ประสิทธิภาพของอัลกอริทึม

ตารางที่ 1 แผนการดำเนินการตลอดโครงการวิจัย

ปีที่ 1			
ลำดับ	**กิจกรรม**	**เวลา**	**ผล**
1	ศึกษาโครงสร้างสถาปัตยกรรมแบบปรับเปลี่ยนโครงสร้างได้, ขั้นตอนกระบวนการ ของการปรับเปลี่ยนโครงสร้างแบบ dynamic และโครงสร้างแบบ coarse-grain และทำการหาความหยาบของการทำ coarse-grain	3 เดือน	White paper
2	โมเดลเพื่อช่วยคาดคะแน power, delay โครงสร้างที่ปรับเปลี่ยนได้	3 เดือน	Technical report
3	สร้าง functional unit ด้วยภาษา HDL เพื่อใช้ในการทดสอบ	2 เดือน	Soft core
4	พัฒนาวงจรของการ routing ให้ใช้พลังงานต่ำ	2 เดือน	International paper
5	โมเดลของโพรเซสเซอร์ที่สามารถปรับเปลี่ยนโครงสร้างได้	2 เดือน	-

1.9 ผลงานวิจัย

ผลงานการตีพิมพ์ในที่ประชุมวิชาการ

1. Sasatorn somwatee, Wannarat Suntiamorntut, survey of finding empty space algorithm for partial reconfigurable FPGAs, Proceedings *of International Joint Conference on Computer Science and Software Engineering* (JCSSE2009), pp. 489-491, 2009.
2. Sasatorn somwatee, Wannarat Suntiamorntut, performance evaluation of the efficient algorithms for free resources management on the FPGA, Proceedings *of International Joint Conference on Computer Science and Software Engineering* (JCSSE2009), pp. 303-305, 2009.
3. W. Suntiamorntut, C. Vongchumyen, Design Techniques for Energy Efficient Multiplier, Ladkrabang Journal, June, 2007
4. W. Suntiamorntut, Hamming Distance and Normalization Circuits, ICCCAS07, July, pp.1053-1056, 2007.
5. W. Suntiamorntut, L.E.M. Brackenbury, Jim Garside Design and Implementation of an Energy Efficient, Parallel, Asynchronous DSP, ITC-CSCC07, July, pp. xxx, 2007.

1.10 เอกสารอ้างอิง

1. W.Suntiamorntut, L.E.M. Brackenbury, Functional Unit for Low-Power DSP Architecture, *IEE System-On-Chip*, Cardiff, UK, 2-3 Sept. 2003.

2. HAUSER, J. R. ANDWAWRZYNEK, J., Garp: A MIPS processor with a reconfigurable coprocessor. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM '97). 24–33, 1997.

3. RUPP, C. R., LANDGUTH, M., GARVERICK, T., GOMERSALL, E., HOLT, H., ARNOLD, J. M., AND GOKHALE, M., The NAPA adaptive processing architecture. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.

4. HAUCK, S., FRY, T. W.,HOSLER, M. M., AND KAO, J. P., The Chimaera reconfigurable functional unit. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1997.

5. RAZDAN, R. AND SMITH, M., High-performance micro-architectures with hardware programmable functional units. In *Proceedings of the 27th Annual IEEE/ACM International Symposium on Microarchitecture*. 172–180, 1994.

6. WAN, M., ZHANG, H., GEORGE, V., BENES, M., ABNOUS, A., PRABHU, V., AND RABAEY, J., Design methodology of a low-energy reconfigurable single-chip DSP system. *J. VLSI Signal Process,* 2000.

7. GOLDSTEIN, S. C., SCHMIT, H., MOE, M., BUDIU, M., CADAMBI, S., TAYLOR, R., AND LAUFER, R., PipeRench: A coprocessor for streaming multimedia acceleration. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*. 28–39, 1999.

8. SALEFSKI, B. AND CAGLAR, L., Re-configurable computing in wireless. In *Proceedings of the 38th Design Automation Conference*, Las Vegas, Nevada, 178–183, 2001.

9. EBELING, C., CRONQUIST, D., AND FRANKLIN, P., RaPiD—Reconfigurable Pipelined Datapath. In *The 6th International Workshop on Field-Programmable Logic and Applications*, 1996.

10. H. Singh et al., MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Communication-Intensive Applications, in *IEEE Trans. on Computers*, vol. 49, no. 5, pp. 465-481, May 2000.

11. A. Ahmadinia et al., A New Approach for On-Line Placement on Reconfigurable Devices, Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS 04), IEEE CS Press, p. 134, 2004.

12. C. Steiger et al., Online Scheduling and Placement of Real-Time Tasks to Partially Reconfigurable Devices, Proc. 24th Int'l Real-Time Systems Symp. (RTSS 03), IEEE CS Press, pp. 224-235, 2003.

13. R. Vaidyanathan and J.L. Trahan, Dynamic Reconfiguration: Architectures and Algorithms, Plenum US, 2004.

14. C. Bobda et al., A Dynamic NoC Approach for Communication in Reconfigurable Devices, Proc. 14th Int'l Field-Programmable Logic Conf. (FPL 04), LNCS vol. 3203, Springer, pp. 1032-1036, 2004.

15. L. Benini and G. Micheli, Networks on Chips: A New SoC Paradigm, J. Computer, vol. 35, no. 1, Jan., pp. 70-78, 2001.

16. T. Marescaux et al., Networks on Chip as Hardware Components of an OS for Reconfigurable Systems, Proc. 13th Int'l Field-Programmable Logic Conf. (FPL 03), LNCS vol. 2778, Springer, pp. 595-605, 2003.

17. V. Baumgarte et al., Pact XPPA Self-Reconfigurable Data Processing Architecture, J. Supercomputing, vol. 26, no. 2, Sept., pp. 167-184, 2003.

18. J. Becker et al., Datapath and Compiler Integration of Coarse-grain Reconfigurable XPP-Arrays into Pipelined RISC Processor, in *Proc. of IFIP VLSI SoC*, pp. 288-293, 2003.

19. J. Villareal et al., Improving Software Performance with Configurable Logic, in *Design Automation for Embedded Systems*, Springer, vol. 7, pp. 325-339, 2002.

20. G. Stitt et al., Energy Savings and Speedups from Partitioning Critical Software Loops to Hardware in Embedded Systems, in *ACM TECS*, vol.3, no.1, pp. 218-232, Feb., 2004.

21. D. D. Gajski et al., SpecSyn: An environment supporting the specify-explore-refine paradigm for hardware/software system design, in *IEEE Trans. on VLSI Syst.*, vol. 6, no. 1, pp. 84–100, 1998.

22. J. Henkel, A low power hardware/software partitioning approach for core-based embedded systems, in *Proc. of the 36th ACM/IEEE DAC*, pp. 122–127, 1999.

บทที่ 2
การออกแบบโพรเซสเซอร์ร่วมกำลังไฟต่ำสามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก

2.1 บทนำ

เทคโนโลยีดิจิตอลอิเล็กทรอนิกส์ได้รับการพัฒนามาอย่างต่อเนื่องและยาวนาน โดยเฉพาะไมโครโพรเซสเซอร์ที่ได้พัฒนาขึ้นในทุกๆ 2 ปี จะมีจำนวนทรานซิสเตอร์เพิ่มมากขึ้นเกือบเท่าตัว เนื่องจากขนาดของทรานซิสเตอร์และราคาที่ถูกกว่าเท่าตัวในทุกๆปี ดังนั้นประสิทธิภาพของไมโครโพรเซสเซอร์จึงที่ได้รับการพัฒนาเพื่อเพิ่มศักยภาพในการประมวลผลมากขึ้นเรื่อยๆ 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

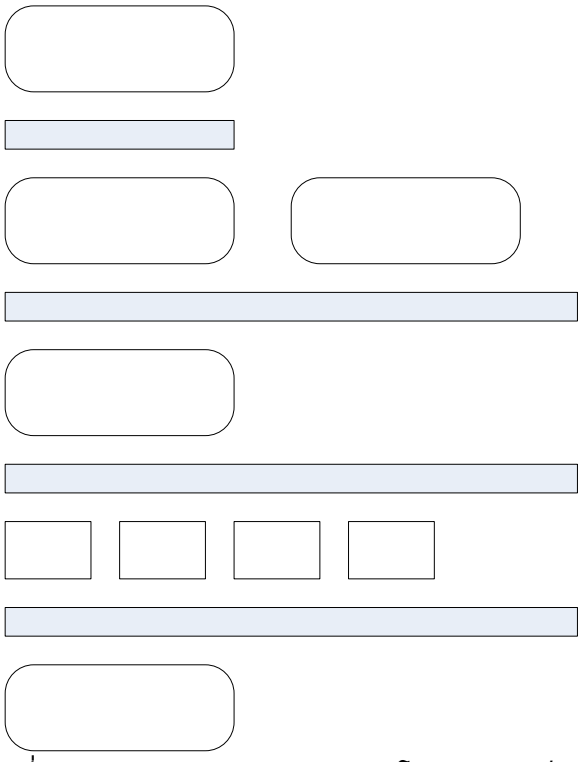
2. *กลุ่ม Configurable instruction* เป็นกลุ่มคำสั่งที่ใช้ใน FU เพื่อควบคุมอินพุตที่จะเข้าประมวลผลใน FU รูปแบบของคำสั่ง และการไหลของข้อมูลใน FU ซึ่ง FU ที่ได้ออกแบบมีคำสั่งหลักทั้งหมด 8 คำสั่งได้แก่ MPY MAC ADD SUB AND OR XOR และ MOV ซึ่งเป็นพื้นฐานของการประมวลผลงานทางด้านการคำนวณทางคณิตศาสตร์ โดยจะมี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

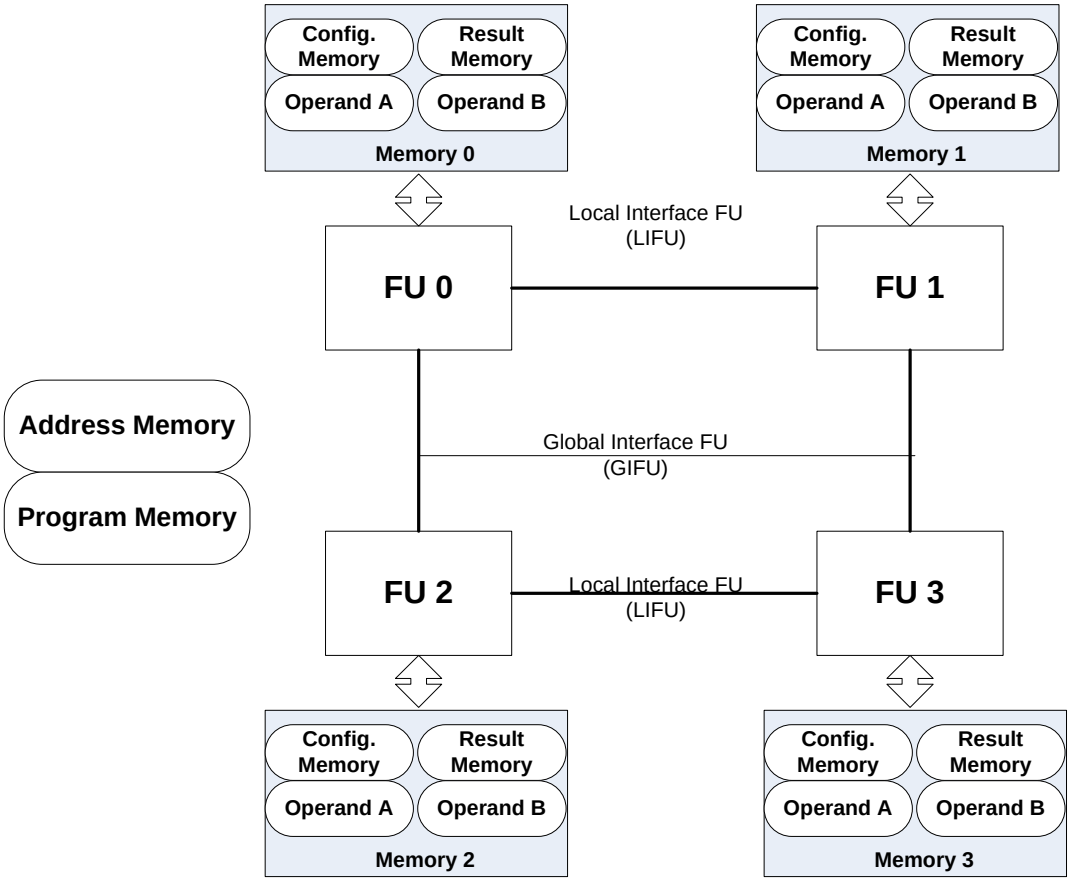
เมื่อได้ออกแบบชุดคำสั่งของโพรเซสเซอร์ร่วมโดยแบ่งออกเป็น 2 กลุ่ม คือกลุ่มของ top-level ซึ่งเป็นการควบคุมโครงสร้างใหญ่ของโพรเซสเซอร์หรือสามารถกล่าวได้ว่าเป็นการควบคุมกลุ่มของหน่วยประมวลผล FU ของโพรเซสเซอร์ ในการทดลองนี้กำหนดจำนวนกลุ่มของ FU เป็น 4 หน่วย สำหรับการใช้งานจริง จำนวนบิตในชุดคำสั่งที่เหลือสามารถปรับขยายได้ตามความต้องการว่าจะใช้ 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

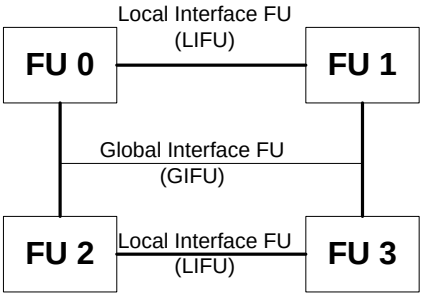
ขั้นตอนแรกเป็นการเก็บลำดับของคำสั่งที่จะใช้ในการทำงาน ซึ่งโพรเซสเซอร์หลักจะทำการเขียนโปรแกรมลงบนหน่วยความจำที่เรียกว่า *address memory นี้* ซึ่งลำดับของคำสั่งจะสอดคล้องกับคำสั่งที่เก็บใน *Program memory* และค่าที่จะใช้ในการประมวลซึ่งเก็บไว้ในหน่วยความจำ *Operand A และ Operand B* โดยที่โพรเซสเซอร์ร่วมจะทำการอ่านค่าคำสั่งและ 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



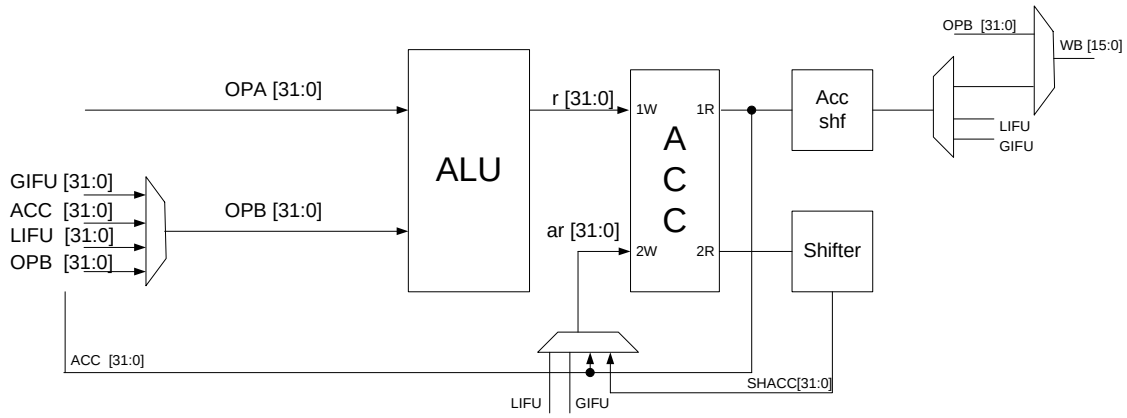
รูปที่ 2 โครงสร้างของโพรเซสเซอร์ร่วมที่ออกแบบ

โครงสร้างของโพรเซสเซอร์แสดงไว้ในรูปที่ 2 ประกอบด้วยหน่วยความจำหลักที่จะใช้งานร่วมกัน (หน่วยความจำ Address และ Program) ซึ่งในงานวิจัยนี้ใช้ SRAM ในวงจรเนื่องจากมีความเร็วสูงและราคาถูก โดยที่ข้อมูลที่ได้จากหน่วยความจำ program จะถูกส่งไปยังหน่วยความจำ configurable เพื่อจัดเตรียมการประมวลผลใน 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 บิต จากบัส 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

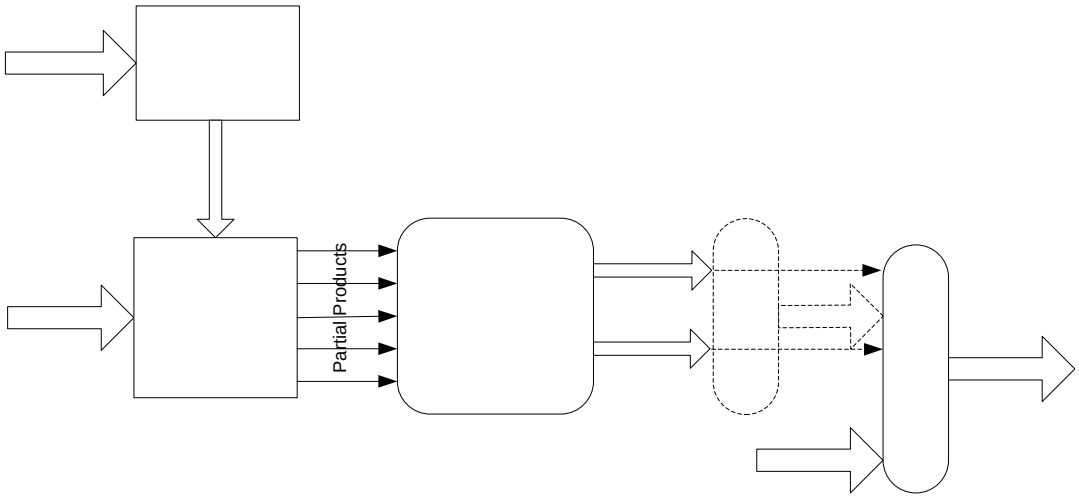
นอกจากนี้การออกแบบวงจรคำนวณในหน่วยประมวลผล FU ให้มีพลังงานต่ำจะช่วยให้ภาพรวมของการใช้พลังงานในการประมวลผลต่ำลงกว่าเดิม ในส่วนนี้นำเสนอเทคนิคการออกแบบวงจรพลังงานต่ำสำหรับใช้งานใน FU แต่การทดสอบจะใช้วิธีการนำจำนวนลอจิกเกตที่ใช้งานแทนการจำลองการทำงานจริง เนื่องจากบน FPGA ผู้ใช้ไม่สามารถควบคุมการโปรแกรมหรือการเลือกใช้ลอจิกเกตได้เหมือนกับการออกแบบวงจรแบบ ASIC

1) การลดรูปวงจรคำนวณคำสั่ง MAC

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

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

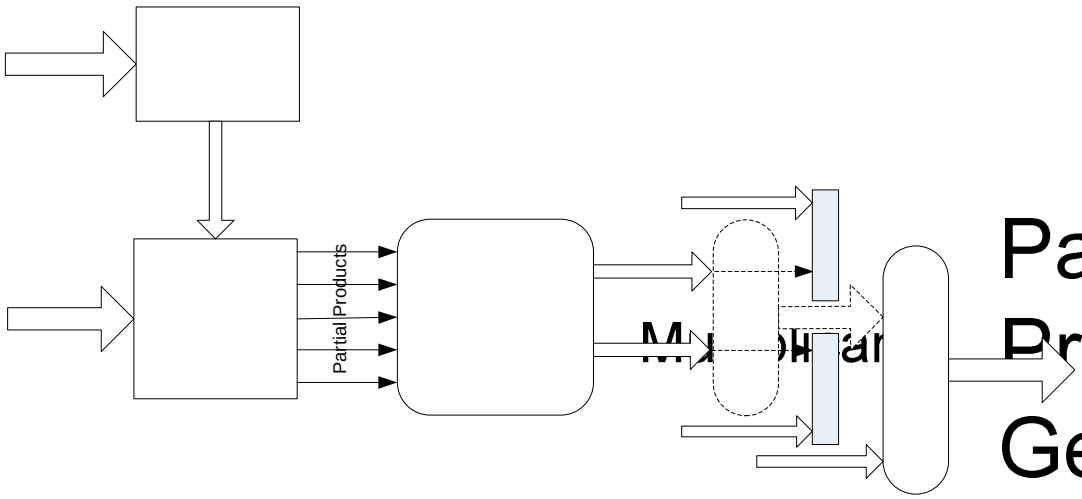
พบว่าพื้นฐานวงจรคูณคือการบวกค่า Partial Product (ผลคูณในแต่ละหลัก) โดยที่วงจรบวกที่นิยมใช้คือวงจรบวกแบบหลายอินพุตเช่น Wallace Tree หรือ Dadda Adder ซึ่งในท้ายที่สุดจะได้ผลลัพธ์ 2 ชุดคือ ผลรวมของ Partial Sum (PS) และ ผลรวมของบิตทด Partial Carry (PC) ในขั้นตอนสุดท้ายจึงใช้วงจรบวก 2 อินพุต ดังนั้นเพื่อเป็นการลดวงจรที่ซ้ำซ้อน 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

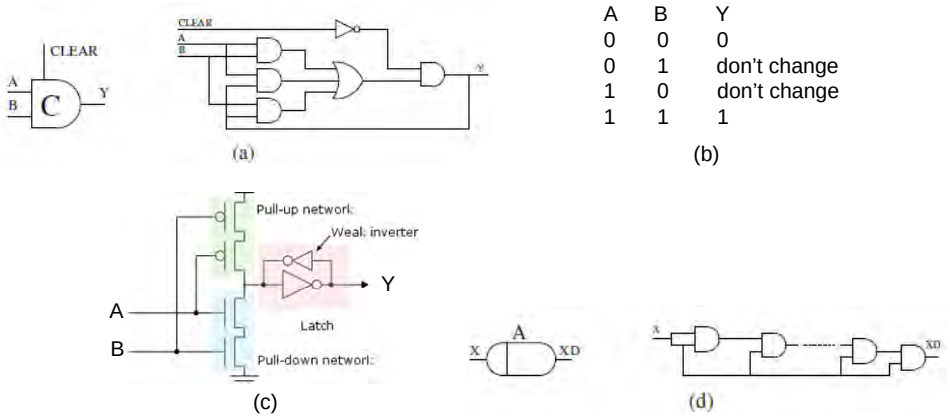


รูปที่ 6 การวงจรคำนวณ MAC MPY ADD และ SUB

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

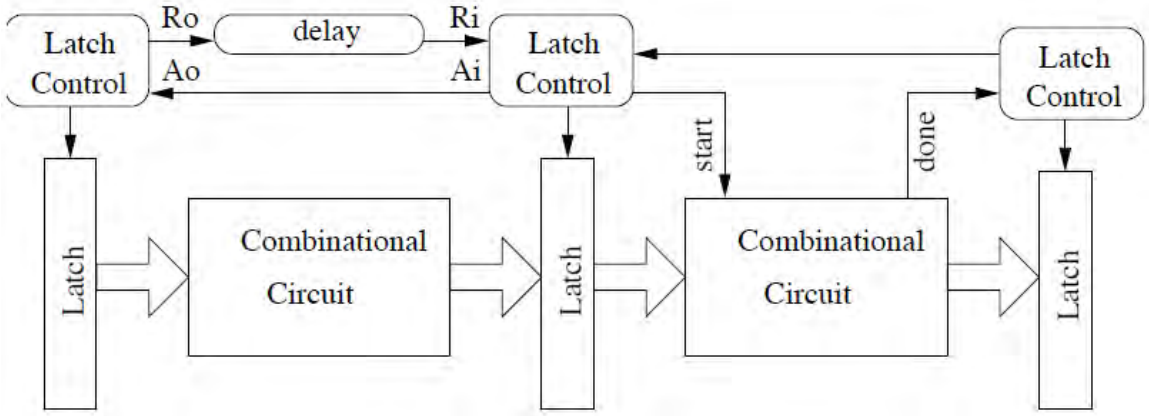


รูปที่ 7 ลอจิกของ C-element (a) ตารางค่าความจริงของ C-element (b) วงจรทรานซิสเตอร์ของ C-element (c) วงจรหน่วงเวลา (d)

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

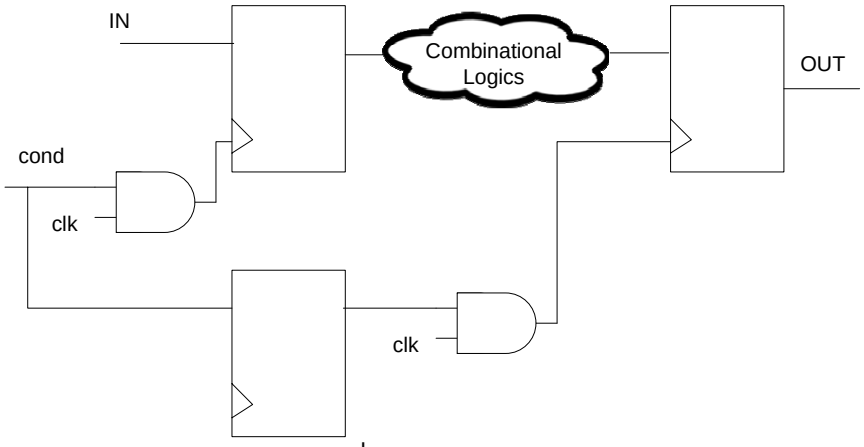
stage นี้ สัญญาณ R0 จะผ่านวงจรหน่วงเวลา หรือ delay ดังรูปที่ 7 (d) ซึ่งจะมีระยะเวลาเท่ากับการทำงานของวงจรใน stage นั้น จากนั้นสัญญาณ request (r1) ที่เกิดจากวงจรหน่วงเวลาจะส่งไปแจ้งตัวพักข้อมูล Latch ถัดไป Latch n+1 เพื่อแจ้งให้ทราบว่าการทำงานนั้นเสร็จสิ้นแล้ว วงจร Latch n+1 จะสร้างสัญญาณ enable เพื่อให้ทำการเก็บผลลัพธ์ในบัสเอาท์พุตทันที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

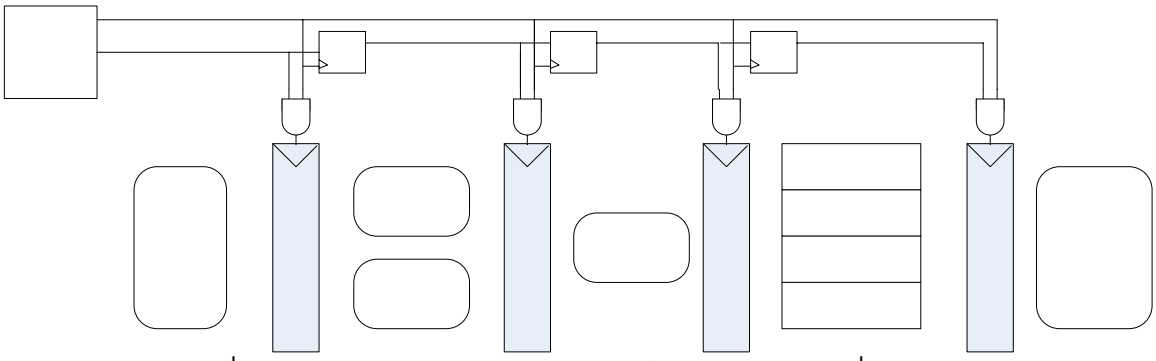


รูปที่ 8 โครงสร้างไมโครไปป์ไลน์

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

clk

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

บทที่ 3
ผลการศึกษาและทดลองซอฟต์แวร์ช่วยออกแบบ

3.1 บทนำ

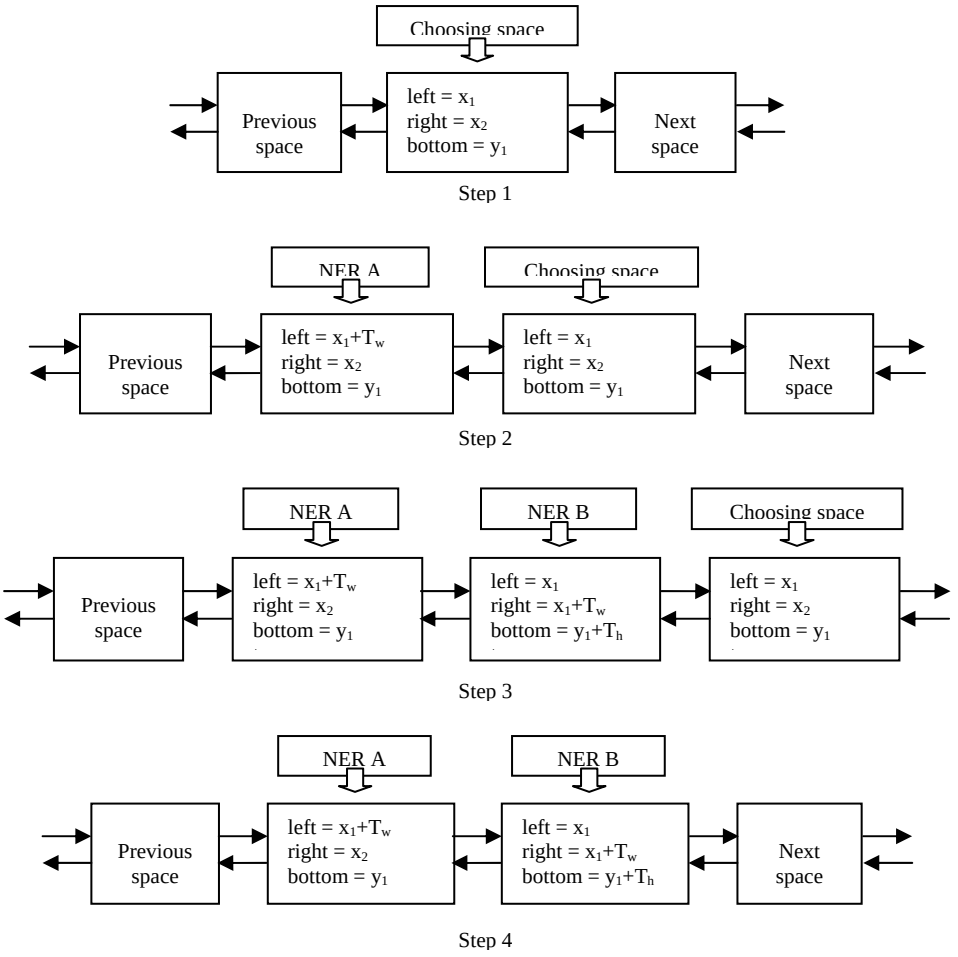
ทฤษฎีการโปรแกรมอุปกรณ์ที่สามารถโปรแกรมได้แบบ run-time หมายถึงให้สามารถโปรแกรมอุปกรณ์ฮาร์ดแวร์อย่างเช่น FPGA ให้สามารถปรับเปลี่ยนวงจรได้ในขณะที่กำลังทำงาน สามารถนำมาประยุกต์เป็นส่วนหนึ่งของการให้โพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนโครงสร้างแบบไดนามิก ดังนั้นผู้วิจัยจึงได้ทำการศึกษาแนวคิดของการสร้างซอฟต์แวร์ที่ช่วยออกแบบ ซึ่งหน้าที่หลักของซอฟต์แวร์ที่จะช่วยออกแบบหรือทำให้โพรเซสเซอร์ร่วมสามารถปรับเปลี่ยนโครงสร้างแบบไดนามิกได้นั้นจะต้องประกอบด้วย 1)การพิจารณาการวางหน่วยประมวลผล FU ให้เหมาะสมกับฮาร์ดแวร์ที่สามารถให้งานได้ทั้งหมด นั่นคือการ Placement 2) เนื่องจากในแต่ละ FU สามารถเชื่อมต่อถึงกันผ่านทางระบบบัส ดังนั้นซอฟต์แวร์ช่วยออกแบบจึงต้องคำนึงถึงการเชื่อมโยงแต่ละ FU หรือที่เรียกว่า Routing ในงานวิจัยนี้จะศึกษาเฉพาะในส่วนของอัลกอริทึมของการวางหรือ placement ก่อนในขั้นต้น

3.2 อัลกอริทึมการ Placement

เมื่อต้องการให้โพรเซสเซอร์ร่วมที่จะต้องปรับเปลี่ยนโครงสร้างแบบไดนามิกได้ทำงานได้ พบว่าฟังก์ชันการทำงานหรือคำสั่งสำหรับการประมวลผลในแต่ละงานประยุกต์จะไม่สามารถทราบได้ล่วงหน้า ดังนั้นลำดับของคำสั่งจะรู้ในขณะที่กำลังทำงานอยู่ตอนนั้น (run time) ซึ่งจะเรียกการจัดวางหน่วย FU ลงบนอุปกรณ์ที่สามารถโปรแกรมได้ในขณะที่กำลังทำงานอยู่นั้นว่า online placement ซึ่งทำให้จะต้องทำการ placement ให้รวดเร็วและมีประสิทธิภาพมากที่สุดเพื่อให้ไม่กระทบต่อการทำงานโปรแกรมนั้นๆที่กำลังดำเนินการอยู่ ซึ่งความรวดเร็วและประสิทธิภาพของการวางนั้นจะเกี่ยวข้องกับการหาที่ว่างที่จะสามารถวางหน่วย FU ซึ่งเรียกว่า finding empty space นอกจากนี้ถ้าที่ว่างบนอุปกรณ์ที่สามารถโปรแกรมได้นั้นไม่เพียงพอหรืออาจจะมีที่ว่างแต่กระจัดกระจายเป็นชิ้นเล็กชิ้นน้อย (fragmentation) ไม่เพียงพอที่จะให้หน่วย FU ทั้งหมดจัดวางลงไปได้ ทำให้ส่งผลต่อการจัดการพื้นที่บนอุปกรณ์ที่สามารถโปรแกรมได้ โดยเฉพาะอย่างยิ่งสำหรับระบบที่สามารถโปรแกรมได้แบบไดนามิกอย่างเช่นโพรเซสเซอร์ที่กำลังออกแบบ

แนวทางของการจัดการกับที่ว่าง (maintaining empty space) บนอุปกรณ์ที่สามารถโปรแกรมได้นั้นสามารถดำเนินการได้ 3 รูปแบบ โดยที่พื้นที่ของการออกแบบเป็นลักษณะของรูปทรงสี่เหลี่ยมพื้นผ้า ฉะนั้นการจะค้นหาพื้นที่ว่างจะสามารถทำได้ดังนี้ 1) การหาพื้นที่ว่างที่ไม่ซ้อนทับกัน 2) หาพื้นที่ว่างที่มากที่สุด 3) ค้นหาจุดที่มารวมกัน ปัญหาของการจัดการพื้นที่ว่างนั้นจัดว่าเป็นปัญหาสำหรับการคำนวณทางคณิตศาสตร์ของ geometry ซึ่งผู้วิจัยได้ทำการเสนอวิธีการค้นหาจัดการกับพื้นที่ว่างที่มีประสิทธิภาพดีขึ้นกว่าเดิมด้วยการใช้ link list ในการเก็บข้อมูลพื้นที่ว่าง

การดูแลรักษาและจัดการกับพื้นที่ว่างบน FPGA จะใช้ double link list ในตอนเริ่มต้นจะมีเพียงพื้นที่ว่างเพียง 1 ชิ้นที่มีขนาดใหญ่เท่ากับพื้นที่ที่สามารถโปรแกรมได้บน FPGA แบ่งเป็น 3 ขั้นตอนได้แก่ เพิ่มพื้นที่ว่างชิ้นใหม่ ปรับรายการพื้นที่ว่างใหม่และการรวมพื้นที่ว่าง เมื่อมี task ส่วนไหน ทำงานเสร็จสิ้นแล้วออกจาก FPGA พร้อมทั้งคืนพื้นที่ว่างให้ส่วนกลางจะเรียกว่าใช้ โพรเซส *adding new space* ซึ่งจะสร้างพื้นที่ว่างแบบไม่ทับซ้อนโดยที่มุมขอบด้านซ้ายและล่างจะว่างลงบนตำแหน่งของ task ที่เสร็จสิ้นแล้ว สำหรับการรวมพื้นที่ว่างเข้าด้วยกันจะเรียกใช้ AUTO_MERGE ขั้นตอนการจัดการสามารถสรุปได้ดังรูปที่ 11



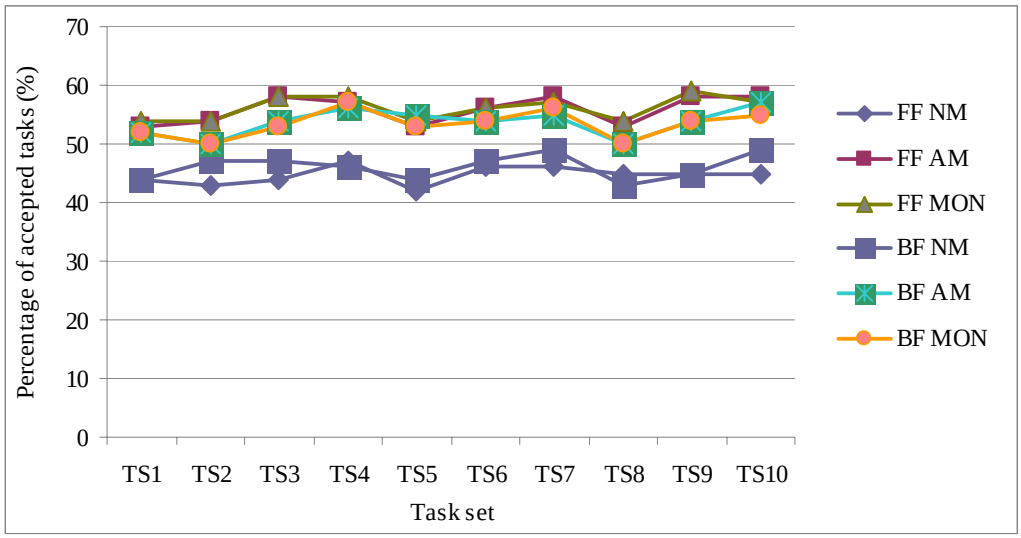
รูปที่ 11 ขั้นตอนของการวาง task

3.3 ผลการทดลองขั้นต้น

อัลกอริทึมที่ทำการทดสอบถูกสร้างขึ้นด้วยภาษาซี ทดสอบบนคอมพิวเตอร์ CPU Pentium IV 3.2 GHz หน่วยความจำ 1 GB ทดลองอัลกอริทึมการเลือกวางแบบ first fit และ best fit ถ้าหากไม่สามารถหาพื้นที่ว่างได้จะทำการ reject task นั้นทิ้งไป กำหนดให้เริ่มต้นจากมีลอจิกบล็อกของ FPGA ที่สามารถโปรแกรมได้ทั้งสิ้น 100 x 100 และชุดทดสอบจำนวน 10 ชุดกำหนดให้แต่ละชุดมี 1,000 task แบบสุ่ม มีขนาดในแต่ละ task อยู่ระหว่าง 2 – 20 หน่วย และมีเวลาในการทำงานอยู่

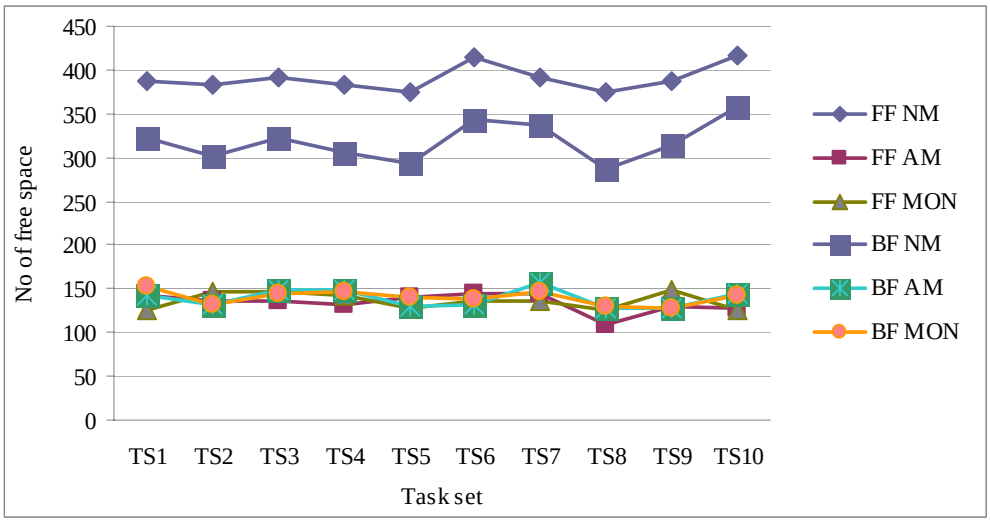
ระหว่าง 50 – 200 ต่อ task การรายงานผลจะเป็นร้อยละของงานที่สามารถหาที่ว่างได้ จำนวนของพื้นที่ว่าง และเวลาโดยเฉลี่ยของการทำงานหน่วยเป็น ns ซึ่งการทดลองจะแบ่งออกเป็น 6 กรณีเช่น

- FF NM: First Fit algorithm with No Merging space
- FF AM: First Fit algorithm with Automatic Merging space
- FF MON: First Fit algorithm with Merging if Only Need
- BF NM: Best Fit algorithm with No Merging space
- BF AM: Best Fit algorithm with Automatic Merging space
- BF MON: Best Fit algorithm with Merging if Only Need

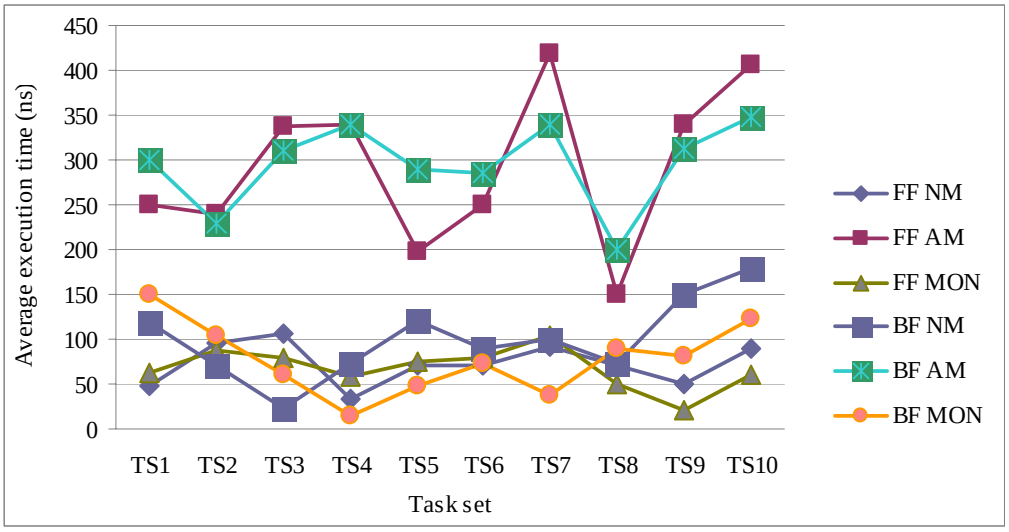


รูปที่ 12 เปอร์เซ็นต์ของงานที่สามารถหาพื้นที่ว่างได้

รูปที่ 12 แสดงร้อยละของงานที่สามารถหาพื้นที่ว่างได้ (accepted task) พบว่าอัลกอริทึมแบบ first fit ให้ผลที่ดีที่สุดในทุกกรณี ในขณะที่ first fit จะให้ผลของจำนวนพื้นที่ว่างเหลือมากกว่าอัลกอริทึมแบบอื่นๆ ดังรูปที่ 13 แต่เวลาเฉลี่ยของอัลกอริทึมแบบ best first จะใช้เวลาน้อยกว่าอัลกอริทึมอื่นๆดังรูปที่ 14



รูปที่ 13 จำนวนพื้นที่ว่างที่คงเหลือ



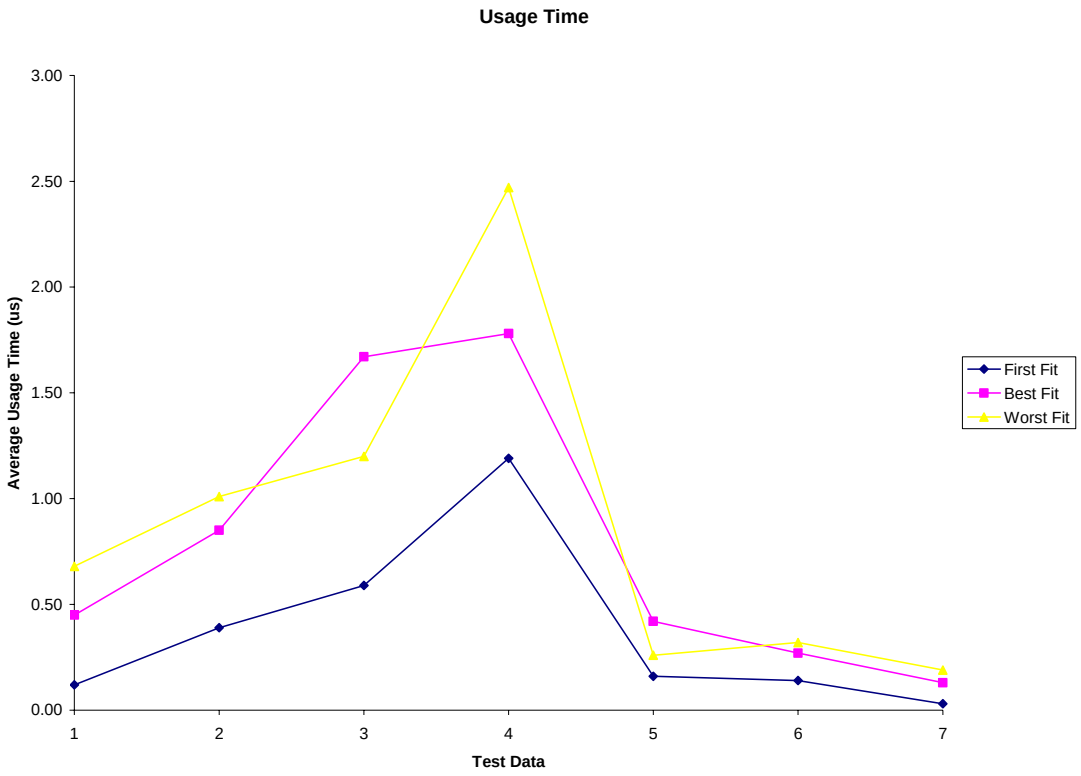
รูปที่ 14 เวลาโดยเฉลี่ยของแต่ละอัลกอริทึม

นอกจากนี้ผู้วิจัยยังได้เพิ่มเติมก็คือในส่วนของการเลือกพื้นที่ว่าง สำหรับวาง task ใหม่ที่เข้ามา โดยทำการเปรียบเทียบระหว่าง 3 algorithm คือ first fit, best fit และ worst fit ซึ่งทั้งสาม algorithm จะรับ input เป็น list ของ maximum empty rectangle และขนาดของ task ที่ต้องการจะวาง โดย first fit algorithm จะเลือกพื้นที่ว่างอันแรกใน list ที่สามารถวาง task ได้ สำหรับ best fit algorithm จะเลือกพื้นที่ว่างที่เมื่อวาง task ใหม่แล้วจะเหลือพื้นที่ที่ไม่ได้ใช้งานน้อยที่สุดและในทางตรงข้าม worst fit algorithm จะเลือกพื้นที่ว่างที่เมื่อวาง task ใหม่แล้วจะเหลือพื้นที่ที่ไม่ได้ใช้งานมากที่สุด ในกรณีที่ขนาดของพื้นที่ที่ไม่ได้ใช้งานมีขนาดเท่ากันใน best fit และ worst fit ก็จะเลือกพื้นที่อันแรกที่หาเจอ

โดยในการทดลองได้จำลองขนาดของ FPGA 1,000x1,000 และทำการสุ่มตัวอย่าง test data ขึ้นมาทั้งหมด 7 ชุด แต่ละชุดมีจำนวน task 10,000 tasks Test data ชุดที่ 1 – 4 มีขนาดของ task อยู่ระหว่าง 25x25 – 250x250 หรือคิดเป็น 1/40 – 1/4 ของพื้นที่บน FPGA โดย task life จะมีขนาดต่างกัน สำหรับ test data ชุดที่ 5-7 มีขนาดของ task อยู่ระหว่าง 25x25 – 500x500 หรือคิดเป็น 1/40 – 1/2 ของพื้นที่บน FPGA

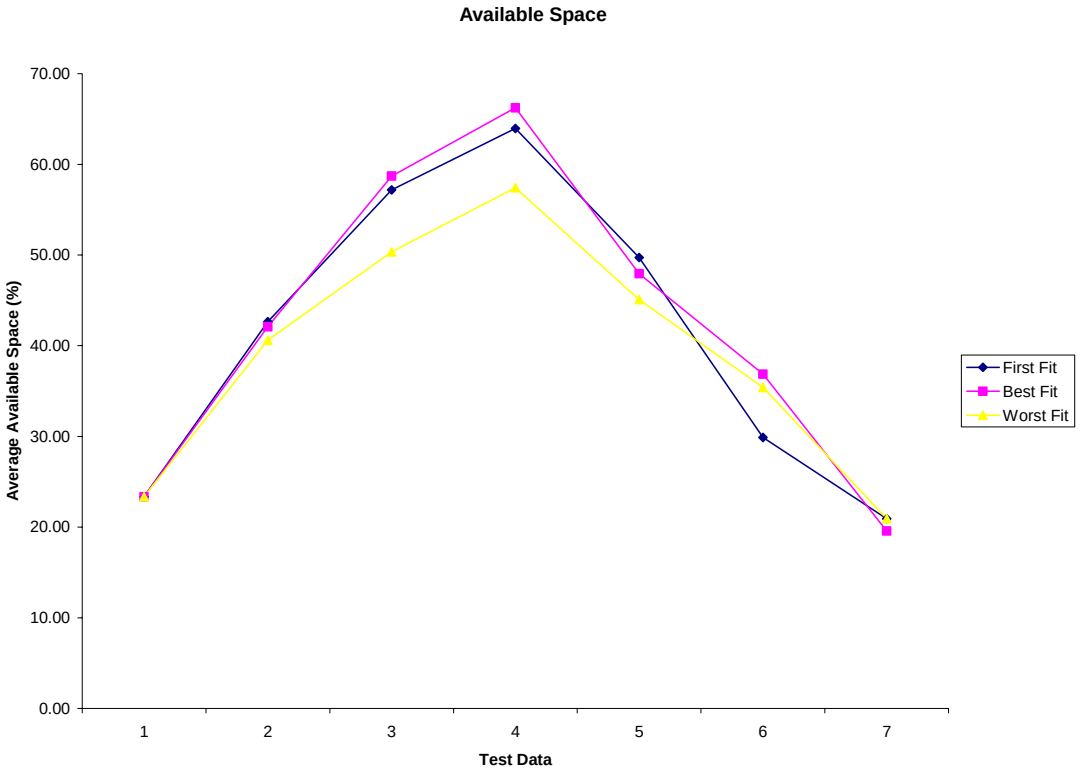
test data	board size	min task life	max task life	min task	max task	total task
1	1000x1000	10	250	25x25	250x250	10000
2	1000x1000	20	500	25x25	250x250	10000
3	1000x1000	40	1000	25x25	250x250	10000
4	1000x1000	80	2000	25x25	250x250	10000
5	1000x1000	10	250	25x25	500x500	10000
6	1000x1000	10	125	25x25	500x500	10000
7	1000x1000	10	50	25x25	500x500	10000

ซึ่งผลการทดลองที่ได้มา พบว่า เวลาเฉลี่ยที่ใช้ในการรัน first fit จะดีที่สุด รองลงมาคือ best fit และ worst fit ดังรูปที่ 15

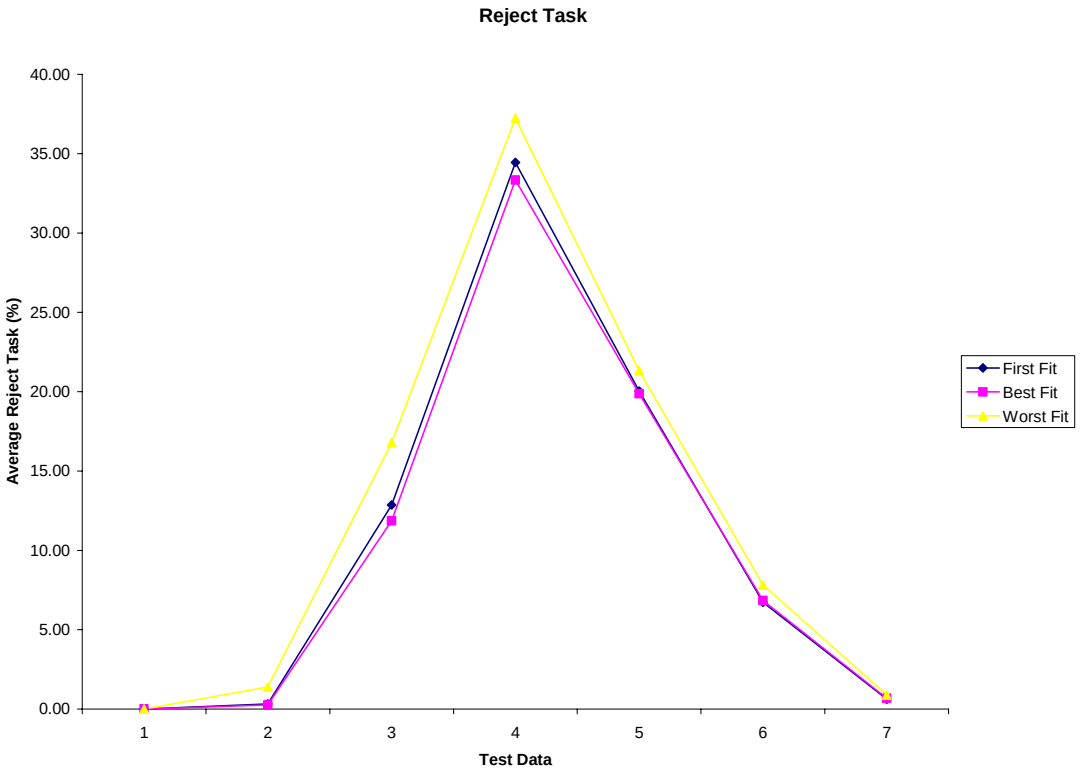


รูปที่ 15 เวลาโดยเฉลี่ยของแต่ละอัลกอริทึม

และเมื่อมาดูพื้นที่ว่างที่เหลืออยู่หลังจากทำการแทรก task ทั้ง 10,000 tasks แล้ว กลับพบว่า worst fit มีแนวโน้มที่ดีที่สุด ที่เป็นเช่นนี้เพราะว่าใน worst fit มีจำนวน task ที่ถูก rejected มากที่สุดนั่นเอง และเมื่อเปรียบเทียบระหว่าง first fit กับ best fit พบว่า มีขนาดพื้นที่ว่างและจำนวน task ที่ถูก rejected ไม่ได้แตกต่างกันมากนัก ดังรูปที่ 16 และ 17



รูปที่ 16 พื้นที่ที่เหลือโดยเฉลี่ยของแต่ละอัลกอริทึม



รูปที่ 17 อัตราการ rejected task ของแต่ละอัลกอริทึม

3.4 เอกสารอ้างอิง

[1] P. Healy and M. Creavin. An Optimal Algorithm for Rectangle Placement. In Technical Report UL-CSIS-97-1. Dept. of Computer Science and Information Systems, University of Limerick, Limerick, Ireland, 1997.

[2] M. Orlowski. A New Algorithm for the Largest Empty Rectangle. In Algorithmica, volume 5, pages 65–73, 1990.

[3] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast Template Placement for Reconfigurable Computing Systems. In IEEE Design and Test of computers Special Issue on Reconfigurable Computing, volume 17(1), pages 68–83, Jan.-Mar. 2000.

[4] M. Handa, and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement". In Proceedings of the 41st annual conference on design automation, San Diego, CA, June 2004, pp 960-965.

[5] J. Cui, Q. Deng, X. He, and Z. Gu, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs". In Proceedings of the conference on Design, automation and test in Europe, Nice, France, April 2007, pp 1-6.

[6] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resources management on the FPGA". In Proceedings of the conference on Design, automation and test in Europe, Munich, Germany, March 2008, pp 1095-1098.

[7] M. Gericota, G. Alves, M. Silva and J. Ferreira, “Runtime management of logic resources on reconfigurable systems”, In Proceedings of Design, Automation and Test in Europe, March, 2003.

[8] M. Handa, and R. Vemuri, “Area fragmentation in reconfigurable operating systems”, In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, CSREA Press, June, 2004.

บทที่ 4
สรุปผลการวิจัย

งานวิจัยนี้นำเสนอโครงสร้างสถาปัตยกรรมโพรเซสเซอร์ร่วมที่สามารถปรับเปลี่ยนโครงสร้างได้แบบไดนามิก โดยนำเสนอตั้งแต่การออกแบบชุดคำสั่ง สถาปัตยกรรม ดาต้าพาส โมเดลทดสอบการทำงานด้วยภาษาซี และตัวแปลภาษาระดับแอสแซมเบลอร์ โพรเซสเซอร์ร่วมที่ออกแบบสามารถควบคุมการทำงานผ่านทางโพรเซสเซอร์หลักโดยการจัดลำดับการเข้าทำงานและโปรแกรมเก็บไว้ในหน่วยความจำแบบ SRAM ที่ทำงานอย่างรวดเร็ว ในลักษณะของแนวคิดเดียวกับโครงสร้างการโปรแกรมบนเทคโนโลยี FPGA ซึ่งในการออกแบบขั้นต้นนี้จะมีหน่วยประมวลผล FU ให้สามารถเลือกใช้งานได้ไม่เกิน 4 ตัว ทั้งนี้ในการใช้งานจริงสามารถเพิ่มหรือขยายจำนวนของการประมวลผลแบบขนานนี้ได้ ซึ่งขั้นตอนการเตรียมข้อมูลโปรแกรมลงบนหน่วยความจำจะทำงานแบบไปป์ไลน์ด้วยเทคนิคของการทำ clock-gating เพื่อประหยัดพลังงานในช่วงเวลาที่ไม่ต้องการใช้งาน นอกจากนี้หน่วยประมวลผล FU ที่ได้ออกแบบสำหรับโพรเซสเซอร์ร่วมนี้สามารถประหยัดพลังงานหรือจำนวนลอจิกได้เนื่องจากมีการลดรูปวงจรและการใช้วงจรร่วมกันระหว่างคำสั่ง MAC MPY ADD และ SUB

ผลการจำลองการทำงานบนเทคโนโลยี FPGA ด้วยโปรแกรมช่วยออกแบบ Xilinx ISE บน XC3SD1800 ในกรณีที่ทำงานด้วย FU สูงสุด 4 ตัว พบว่าทำงานได้ด้วยความถี่ 59 MHz ใช้พลังงาน 34.64 mW ซึ่งสามารถประมวลผล FIR 20-tap เสร็จภายในเวลา 0.339 ไมโครวินาทีและสำหรับผลการทดสอบแนวคิดการวาง task แบบไดนามิกบน FPGA ของซอฟต์แวร์ช่วยออกแบบพบว่าใช้อัลกอริทึมแบบ first fit และ best fit ใช้ความเร็ว และจำนวนของ task ที่วางลงบน FPGA ไม่ได้ไม่แตกต่างกัน ดังนั้นจึงเลือกใช้อัลกอริทึมแบบ first fit เนื่องจากมีจำนวนของพื้นที่ว่างน้อยกว่า

ภาคผนวก

Design Techniques for Energy Efficient Multiplier

W. Suntiamorntut
Department of Computer Engineering,
Faculty of Engineering,
Prince of Songkla University, Hatyai, Songkhla,
90112 Thailand
wannarat@coe.psu.ac.th

C. Vongchumyen
Department of Computer Engineering,
Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang
Ladkrabang, Bangkok, 10520, Thailand
kvocharo@kmitl.ac.th

Abstract

Power consumption has become a critical concern in VLSI design, especially for portable applications. Multiplier is a fundamental operation which is executed in most clock cycle. Multiplier also has a large area, long latency and consumes a huge power. This paper presents the design technique considering at architecture, logic and circuits level such as optimization in the depth of Wallace tree addition, using modified Booth's algorithm to reduce the number of partial products, input swapping and pre-calculated signed-extension. The simulation results of the post-layout of multiplier implemented on 0.18 micron process technology show that the multiplier using our design techniques has the lowest energy consumption and also giving the best energy delay product.

Keyword: energy efficiency, multiplier, low-power design

1. Introduction

Multiplication is a basic arithmetic operation that is fundamental to digital signal processing. Whereas it consumes a huge power because the multiplier has been involved in most time slots during the execution of DSP algorithms such as the FIR filter, Discrete Fourier Transform (DFT), DCT or Linear Predictive Coding (LPC).

However, multiplier has a large area, long latency and consumes relative high power compared to other circuit components. Therefore, both low-power and low latency multiplier design have been an important part in energy efficient VLSI system design. The techniques can be applied at technology, physical, circuit and logic levels to achieve an energy efficiency. In this paper, we address at the structure, logic and circuits implementation of the multiplier. The main contribution of this work is to present the design techniques of power reduction for a parallel multiplier which has been used in CADRE-s digital signal processor proposed in [1].

The complex systems for mobile phones and portable applications are expected to support the requirements of modern features such as multimedia, high quality games and so on. The parallel or tree multiplier with a high performance carry save adder (CSA) tree has been justified for these applications[2-3]. Therefore, this research work focuses on using a coherent technique at architecture, logic and circuits level to implement a parallel tree multiplier.

The remain of this paper is organized as follows. In section2, the power awareness of CMOS circuits is discussed. The parallel multiplier architecture is described in section3 while the power saving techniques, input swapping, sign-extension and reduction in addition are presented in section4. The simulation and comparison of this multiplier has been analysed in section5. Finally, the conclusion is drawn in section6.

2. Power Awareness in CMOS

In digital CMOS circuit, there are three major sources of power dissipation as shown in the equation 1[1]:

$$P_{avg} = P_{switching} + P_{shortcircuit} + P_{leakage} \quad \ldots (1)$$
$$= \alpha_{0->1} \cdot C_L \cdot V^2_{DD} \cdot f_{clk} + I_{SC} \cdot V_{DD} + I_{leakage} \cdot V_{DD}$$

The switching or dynamic component of power consumption is the product of the load capacitance (C_L), clock frequency (f_{clk}), and activity factor, $\alpha_{0->1}$, is used to denote the average fraction of clock cycles in which a low-to-high transition occurs and the power supply V^2_{DD}. The second term is the power dissipated

due to the direct-path short circuit current, I_{SC}, which occurs during switching when both NMOS and PMOS transistor are active. The last term is the power caused by the leakage current, $I_{leakage}$, which arises from substrate injection and sub-threshold effects. The leakage current becomes a significant problem when the fabrication technology is scaled down or when there are significant periods of idle time.

Since the switching event in CMOS circuit dominates the most power, it is extremely important to reduce this source of power dissipation. Firstly, C_L could be minimized through the choice of logic style and logic topology. Secondly, the multiplier could ignore the power dissipates according to f_{clk} by using the asynchronous circuit design where it is a clock-less system[4]. Finally, reducing the supply voltage (V_{DD}) can yield more than an order of magnitude saving which is shown clearly from the equation.

In this paper, we are focusing only on parallel tree multiplier which the multiplication can be done in one clock-cycle. This leads to treat it as a combinational logic and asynchronous circuit technique is not concerned in this paper. Meanwhile, reducing a supply voltage will be ignored because there is a trade-off between performance and power when voltage has been scaling. This paper is therefore discussed only minimized load capacitance.

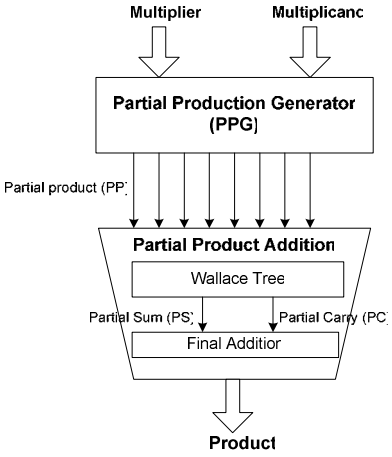


Figure 1 Parallel Multiplier Architecture

3. Parallel Multiplier Architecture

Multiplier composes of two main components: partial product generator (PPG) and the addition of those partial products (PPs) as shown in Figure 1. In order to reduce the number of PPs, modified Booth's algorithm has been used in this design, whilst the partial product addition employs the Wallace tree to produce the final partial sum (PS) and partial carry (PC). Finally, the result of multiplication is generated by the sum of PS and PC.

For a K x K multiplier, K partial products are produced and a K-input CSA tree is used to reduce them to two operands for the final addition. However, the number of partial products and logic depth of the CSA tree can be reduced when Booth's algorithm is applied. The modified Booth's algorithm[5] (radix-4 Booth's recoding) considers multiplier bits in pairs and treats the pair as a signed two's complement number. The higher radix leads to fewer partial products (PP) but requires more complex hardware and a longer time for encoding. With a radix-4 modified Booth's algorithm, the number of PPs can be reduced by half, whereas the PP generator has been built using pass-transistor multiplexer.

3.1 Partial Product Addition

In the partial product addition, several methods such as the ripple carry adder (RCA), the CSA, the CSA tree, the Wallace tree[6] and Dadda's strategy[7] are all candidates for the addition scheme in the parallel multiplier. Both Wallace's and Dadda's strategies are based on compression techniques which were first introduced by Weinberger[8]. The differences between Wallace and Dadda are that the Wallace tree tries to combine the partial product bits at the earliest opportunity, whilst Dadda's scheme combines them as late as possible and keeps the critical path (level) of the tree minimal. So Dadda's structure is simpler but has a wider CPA at the end compared to the Wallace tree. However, combining the partial product bits as soon as possible makes the Wallace tree scheme faster than Dadda.

Multi-operand addition schemes based on Wallace tree has been widely employed because of its performance. The 4-2 compressor is therefore popular in many digital multiplications. However, the compressor differs from Dadda's counter in that it is not necessary to have the pattern of M outputs drawn from 2^M inputs. An N:M compressor in essence is a variation of the Dadda counter that employs a separate path between compressor units in order to generate M final outputs using

$N>2^M$ input bits. 4-2 compressor is based on 5:3 counter using 3 bits to represent the 5 binary input bits resulting from the following equations 2-4:

$$PS = P1 \oplus P2 \oplus P3 \oplus P4 \oplus Cin \quad \ldots(2)$$

$$PC = Cin\,(P1 \oplus P2 \oplus P3 \oplus P4) + P3\,/(P1 \oplus P2 \oplus P3 \oplus P4) \quad \ldots(3)$$

$$Cout = P1(P2 \oplus P4) + P2\,/(P2 \oplus P4) \quad \ldots(4)$$

3.2 Final Addition

Four-bit carry-look-ahead tree unit is used to form the 40-bit carry-look-ahead tree by 10 blocks of these 4-bit units where the delay time is shorter than a conventional carry-look-ahead. The carry propagates from the bottom (C_0) bit to the top carry bit (C_4) of the 4-input unit within $3t_{mux}$ and used $12t_{mux}$ for the delay carry-chain of the 40-bit carry-look-ahead tree. Therefore the delay of a W-bit CLA-tree implemented by 4-bit unit is $((W/4)+2)t_{mux}$ as shown in Figure2 for an 8-bit adder where the multiplexer generating the sum is shown only for the LSB in each 4-bit unit.

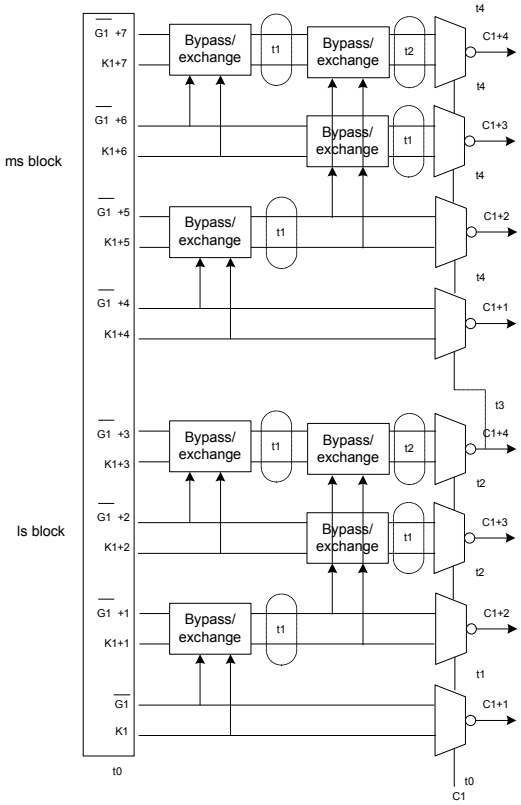


Figure 2 The structure of 4-bit CLA tree block

The adder as shown in Figure 2 has been used to form the final product by adding partial sum (PS and partial carry (PC).

4. Power Saving Techniques

4.1 Input Swapping

The switching activities of the functional blocks dominate a large portion of power consumption in a parallel multiplier where all activities inside each block depending on the data. One of two inputs data is sent to Radix-4 modified Booth's encoding block where the multiplier is decoded to generate the partial products. It can be viewed as a digit-set conversion with the 2 bits denoting the number 0 to 3 converted to the set of {-2,2}. Effectively two bits are treated as a signed two's complement number.

In the multiplication process, the control bits for the input with the smaller effective dynamic range should be used for the Booth's encoding to increase the chance of neighbouring partial products being '0'. This could result less activities.

However, this scheme has a trade-off according to an additional block as shown in Figure 3 to determine when the inputs should be swapped. It uses to detect the dynamic ranges of the input data. This block makes a decision whether the two input data paths should be exchanged or remain unchanged.

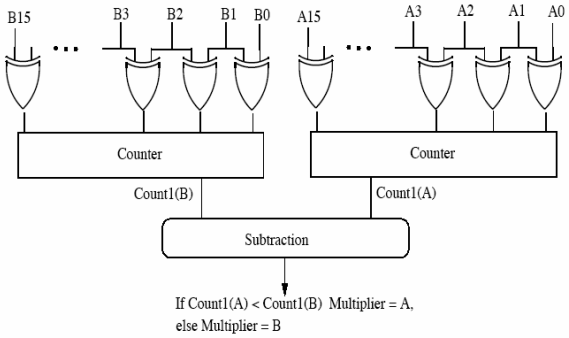


Figure 3 Additional block for input swapping

The simulation result of a 16-bit additional block shows that it takes about 1.4ns to generate the select signal for switching the inputs of the multiplier while the power dissipates at about 9mW (where the data is fetched every 1.5ns). Therefore, these performance and power consumption information of the input swapping technique has to be considered at the architectural level before adopting in the design.

4.2 Sign Extension

Most DSPs have a 16x16 multiplier located in the 40-bit datapath. So the sign-extension is

required to produce the 40-bit result. The eight PPs as shown in Figure 4 are the output of the PPG which produces eight 16 bit PPs (which is the same as the multiplicand length). However, the output is expected by a general DSP to be in 40-bit format. So each PP has to be sign-extended to 40 bits prior to their addition. If we represent logic 1-bit as a one cell, the area inside the red border shown in Figure4 is used to compute the sign-extension. The energy saving approach in the multiplier is to reduce the logic for the sign-extended bits by using a pre-calculated sign extension, as shown in Figure 5, over a group of 4 PPs. This pre-calculated number assumes that all partial products negative and thus have a sign bit of one. Thus we can save the large amount of the logic for the sign-extended calculation.

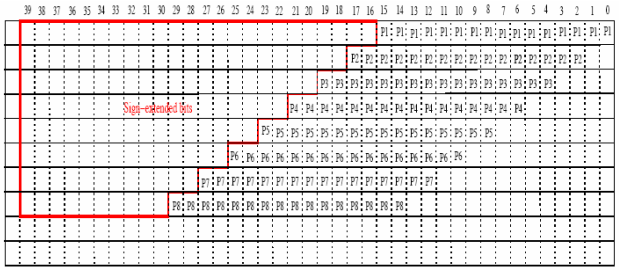


Figure 4 Structure of eight PPs addition

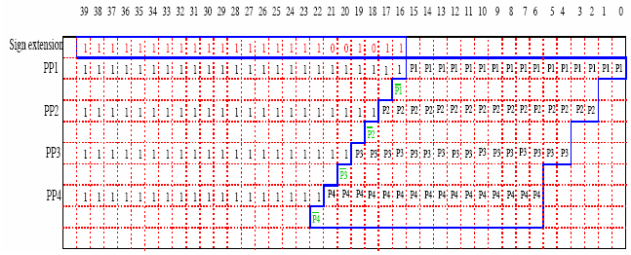
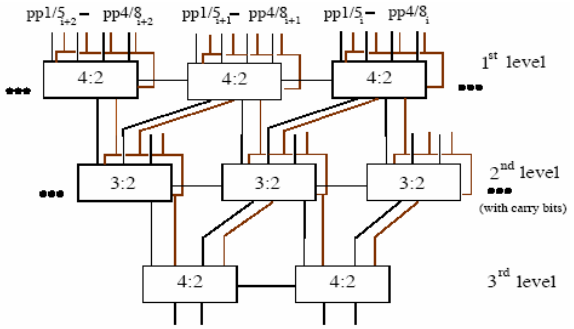


Figure 5 Pre-calculated sign extension

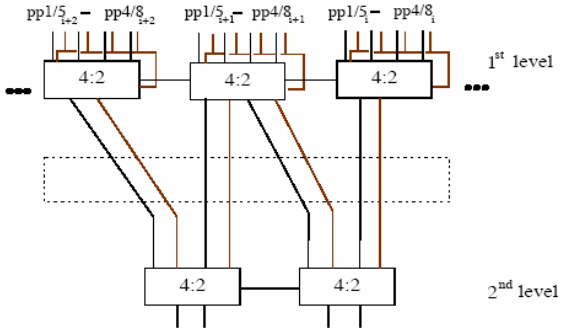
In Figure 5, the assumed sign extension for PP1 to PP3 are shown and when added yield the sign extension constant shown along the top line. If the real PP has a most significant bit (msb) equal to *one*, no adjustment is necessary. In contrast, when the real MSB is *zero,* adjustment is required equivalent to all the 'one' bits in the extension being flipped back to zero. This is achieved by adding a Px (x = 1 to 3) in the bit position shown; it should be noted that the '0's in the sign extension constant (0XFFFFCB) makes it less likely that the carry chain will extend across all bits in the extension constant. With this technique, the constant number required can be computed before the multiplication takes place in the hardware. This allows the addition hardware of the sign-bit part to be reduced.

4.3 Reduction in Wallace Tree

The Wallace tree has been applied in the addition of multiplier instead of the well-known Dadda's strategy as the partial products combining takes place as late as possible. Therefore, Wallace's method is faster than Dadda's. The addition of the PPs is usually done in 3 levels of compressors as shown in Figure 6(a). At the top level, two groups of compressors compress a group of 4 PPs to two outputs. In the second stage, two groups sum the four outputs from the top level in 3-2 compressors, with 4-2 compressors used at the final level. If the multiplier bits require a -a or -2a, then the multiplicand bits are inverted and +1 is input to the addition tree to form the two's complement input to the second stage. In this multiplier design, we propose to save the sign bits from Booth's encoding and add it up later in the final addition. Consequences, the second level compressors can be omitted as the sign bits are forwarded directly to the 4-input adder, as shown in Figure 6(b).



(a) Conventional Wallace Tree



(b) Optimal Wallace Tree

Figure 6 Wallace Tree Structure

A significant reduction in logic arises from losing the second level compressors and this

both improves performance and saves considerable circuitry as can be seen in Figure 6(b).

As a result of using this tree structure topology, the number of stages traversed by each input is approximately the same for all inputs. This leads to a balanced delay tree and results in less switching activity due to input skew.

5. Simulation Results

The design is implemented on a 0.18µm CMOS process having 6-metal layers and runs from 1.8V. The results presented here in this section are the results of simulating the layout using Nanosim eda tool and the test inputs are the random number. The multiplier can produce a PS and PC every 1.5ns in the worst case of random inputs and consumes an average power of only 10.8mW in the PPG & PP addition (excluded the input swapping and final addition). Then later, the completed multiplier (included input swapping) has been simulated to produce the final result using a data fetching rate at 200MHz. A comparison against other designs is difficult because of the differences in term of process technology and bit width.

However, selecting some other low energy designs that have been reported such as the 16x16 multiplier (32-b output) on a 0.09µm process technology running from 1.2V[9], the 4x4 wave pipeline on a 0.18µm process running from 1.8V[10], a 16x16 pass transistor multiplier (32-bit output) on a 0.8µm running from 3.3V[11] and a 16x16 multiplier (40-bit output) on a 0.13µm process running from 1.2V[12]. To get some idea of the relative merits of this multiplier, all results have been scaled to a 0.18µm geometry running from 1.8V. Whilst scaling does not take into account all effects, the results do give an indication of the energy efficiency of the different multipliers.

Table 1 shows the second lowest energy consumption (PDP) and energy delay product (EDP) of the multiplier described here (including the time to add and the adder energy) compared to others reported; it demonstrates a good compromise between performance and power for the 0.18µm. process used. The power delay product can be useful for comparisons in which absolute energy values are not known whilst the energy delay product is normally used when the circuit-speed is important for an energy efficiency comparison.

Types	Scaled Power to 0.18um @1.8V (mW)	Speed (MHz)	Energy (PDP) (pJ)	Energy-Delay Product (EDP) (pJxns)
This Multiplier	26.08	200	130.4	652.0
[9]	49.50	500	99.0	198.0
[10]	74.48	167	446.8	2,681.3
[11]	11.34	44	257.0	5,855.5
[12]	201.9	435	464.4	1,068.0

Table 1 Comparison of EDP of multiplier

6. Conclusion

In this paper, the design techniques for energy efficient multiplier have been described. As a result, this multiplier can operate with energy efficiency when running the simulation using the random number as the test inputs. It shows that the design techniques of power reduction at the architectural, logic and circuits level proposed in this paper can make a good contribution.

Acknowledgement

This work has been developed and used in Configurable Asynchronous DSP for Reduced Energy – successor (CADRE-s), APT group, School of Computer, University of Manchester, UK.

7. Reference

1. W. Suntiamorntut, "Energy Efficient Functional Unit for a Parallel Asynchronous DSP", *Ph.D. Thesis, University of Manchester,* 2005.
2. A. A. Katkar and J. E. Stine, "Modified booth truncated multipliers", *Proceedings of the 14th ACM Great Lakes Symposium on VLSI*, pp. 444-447, 2004.
3. A. P. Chadrakasan and R. Broderson, "Low Power CMOS Design", *John Wiley & Sons Inc.,* ISBN 0780334299, 1997.
4. J. Sparsø and S. Furber, "Principles of Asynchronous Circuit Design - A Systems Perspective", *Kluwer Academic Publishers*, 2001.
5. B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", *Oxford* University Press, 2002.
6. C.S. Wallace, "A Suggestion for Fast Multipliers", *IEEE Transactions Electronic Computer*, vol.EC-13, Feb., pp. 14-17, 1964.

7. L. Dadda and D. Ferrai, “Digital multipliers: A unified approach”, *Alta Frequenza,* vol. 37, Nov., pp.1079-1086, 1968.
8. A. Weinberger, “4:2 Carry-Save Adder module”, *IBM Technical Disclosure Bullentin*, vol.23, Jan., 1981.
9. B. R. Zeydel, V. G. Oklobdzija, S. Mathew, R. K. Krishnamurthy and S. Borkar, “A 90nm 1GHz 22mW 16x16-bit 2’s Complement Multiplier for Wireless Based-band”, *Symposium on VLSI Circuits Digest of Technical Papers*, pp.235-236, 2003.
10. J.B. Sulistyo and D. Sam Ha, “5GHz pipelined multiplier and MAC in 0.18um complementary static CMOS”, *ISCAS’03*, May, pp.117-120, 2003.
11. C.F. Law, S.S. Rofail and K.S. Yeo, “A low power 16x16-b parallel multiplier utilizing pass transistor logic”, *IEEE Journal of Solid-State Circuits*, Vol.34,
No.10, Oct., pp.1395-1399, 1999.
12. S. Agarwala et al, “A 600MHz VLIW DSP”, *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, vol.1, pp.56-444, 2002.

Design and Implementation of an Energy Efficient, Parallel, Asynchronous DSP

W.Suntiamorntut[1], L.E.M.Brackenbury[2] and Jim Garside[2]
[1] Department of Computer Engineering,
Prince of Songkla University
Hatyai, Songkhla, 90112, Thailand
[2] School of Computer Science, University of Manchester
Manchester, M13 9PL, UK
E-mail: wannarat@coe.psu.ac.th, {lbrackenbury, jdg}@cs.man.ac.uk

Abstract: Energy efficient computing in a DSP has become an important research issue in order to have a longer battery operating time to support the modern portable devices. The energy efficient functional unit has been designed and implemented for an in-house asynchornous DSP named, Configurable Asynchronous DSP for Reduced Energy (CADRE). CADRE-successor (CADRE-s) has been implemented as a full custom design and simulations are presented to successfully demonstrate the energy-efficiency of the FU. The results show that the FU designed can achieve an energy improvement by a factor of 5 in the multiply accumulator units and a factor of nearly 2 for the overall system compared with the original CADRE system. This demonstrates the importance that energy efficient logic, circuit and layout techniques contribute to a design.

1. Introduction

An asynchronous parallel architecture, named CADRE (Configurable Asynchronous DSP for Reduced Energy)[1] was designed in the School of Computer Science, University of Manchester and expected to use in portable applications. CADRE expanded instructions to give a flexible VLIW capability including a large register file, instruction buffer and four functional units. It has been design based on the sign magnitude number representation and asynchronous circuit design. Even though CADRE had an efficient architecture for DSP processors, it required a large amount of power as demonstrated in[1,2].

CADRE was implemented by exploiting four-way parallelism, as this appeared to be optimal for power reduction[3]. This was based on the premise area can be traded for increased speed because silicon area is rapidly becoming less expensive. As well-known, arithmetic operations, such as multiply, add and multiply-accumulate are frequently performed and consumed hungry power. From the previous work using random plus speed data, a large percentage of power was found to be dissipated in the Multiply Accumulator Units (MAC units) amounting to about 50% of the overall power consumption.

Therefore, the functional unit (FU) has been re-designed and re-implemented to demonstrate the energy saving improvement possible. To reduce the power consumption in the new FU, the major dominant components, the multiplier and adder, are designed and implemented with coherent low power techniques. The new four-way parallel asynchronous DSP which has been designed, implemented and tested is called CADRE-s (CADRE successor) and will be referred to as CADRE-s throughout in this paper.

This paper is organized as follows. In Section II we describe the CADRE-s Top-Level Architecture. Coherent low power techniques using in the new FU is illustrated in Section III. The implementation is shown in Section IV. Finally, Section V concludes this paper.

2. CADRE-s Top-level Architecture

The top-level architecture for CADRE-s has been designed to demonstrate the energy efficiency of the functional unit and the other advantageous features of CADRE-s such as four-way parallelism and configurable memories. In this top-level architecture, the 32x64 bit configuration memory has been attached to each FU and it stores the opcode. In contrast with the original CADRE, the operands of each FU in the current architecture are fetched directly from on-chip RAMs (OPA and OPB). The new system consists of four FUs connected together with a global bus named the Global Interface Functional Unit (GIFU), whilst each pair of FUs are connected locally via the bus named Local Interface Functional Unit (LIFU). The output data from each FU is stored in another on-chip RAM. Two RAMs are used for the top level control. One is the top-level 14-bit instruction and is stored in a program memory. The second one is the address RAM which effectively performs the function of a 16-bit program counter (PC).

CADRE-s employs a scan path structure for downloading instructions/data and uploading results as shown in Figure1. There are five separate serial scan paths in the design, one is for the address and program memories and the others used for the operand, configuration and result memories. This makes the system fully testable as internal states can be controlled and monitored.

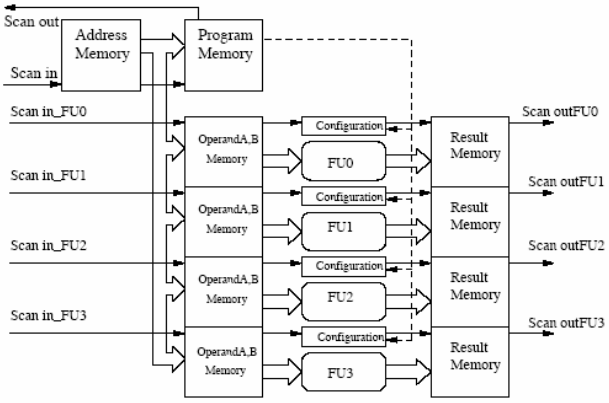


Figure1 CADRE-s Data Organization

2. 1 MIMD with VLIW encoding

In CADRE-s, each FU has its own operands, configurable and result memories. Hence, each FU can

execute independently. It differs from the FU that has been used in an asynchronous superscalar[4]. It has shared memories and registers. A special mechanism for the instruction dispatch unit is therefore required. Because CADRE-s contains four FUs which work independently, the instruction is relatively long. To minimize the length, the instruction has been divided into two parts. One part is stored in the configuration memory and the other part in the top level instruction memory. The configuration memory contains the FU opcode, input/output selection, and the shift condition whilst the top level instruction contains four enable signals plus four accumulator write enable control signals and a common 6-bit address for the configuration memory. Storing the 6-bit address of configuration memory not only reduces the number of instruction bits but the instruction can be compressed. This means that the instructions can be recalled when the same instructions but different data are required. Most DSP algorithms can take advantage of this feature to reduce the size of the program.

2. 2 Five Stage Asynchronous Pipeline

The CADRE-s pipelined processing unit is organized using self-timing techniques. An asynchronous pipeline operates at a variable rate determined by current conditions, unlike a single clock system, where the whole pipeline will be clocked at a rate determined by the worst-case delay in the slowest stage. Although, a multi-clocked system has been proposed in [5-6] to allow a variable rate operation, this method is limited by the number of the various clock cycles which are generated by the clock generator. In a clockless system, the next instruction can start as soon as the previous result is generated. Therefore, the self-timed system is able to operate at the average-case performance rather than worst-case.

3. Low Power Design Techniques

Two novel techniques have been employed in the implementation. The first is to build the functional units (FU) using pass transmission gate (PTG) logic; such logic promises significantly lower power than conventional CMOS whilst delivering high performance. The second unusual technique employed it that the whole system is clock-free. Clockless (or asynchronous) logic is used to eliminate clock generation, buffering and distribution – a major power user – at the system level. Each functional unit is responsible for its own timing; data is passed in and out using handshake signals. This means that a functional unit which is not in use dissipates almost no power. It also allows the implementation of 'unusual' DSP functions – such as Hamming distance calculation or signal clipping – in an energy efficient manner. If evaluation in one unit is slow the whole system will adapt on a cycle-by-cycle basis. The final advantage of asynchronous logic here is that it adapts automatically as the supply voltage is changed; a reduced input voltage gives slower processing but much greater energy efficiency.

4. Implementation

CADRE-s is implemented on a 0.18µm CMOS process having 6-metal layers and runs from 1.8V. The tests described seek to demonstrate the energy efficiency of the FU designed by the coherent low energy design techniques described in the previous section. The kernel benchmarks are translated into binary codes by the CADRE assembler. A Verilog module then rearranges this binary code into the format for serially shifting into CADRE-s. The CADRE-s die, shown in Figure2, measures 4.5 x 3.9mm and contains over 230,000 transistors plus 14 RAM memories of 2k x 16 bits plus a further 4 RAM memories of 64 x 32 bits. The results show that the FU designed can achieve an energy improvement by a factor of 5 in the multiply accumulator units and a factor of nearly 2 for the overall system compared with the original CADRE system.

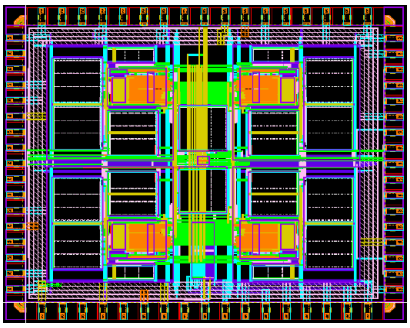


Figure 2 CADRE-s Die Photo

5. Conclusion

CARE-successor (CADRE-s) has been implemented as a full custom design and simulations are presented to successfully demonstrate the energy-efficiency of the FU. This demonstrates the importance that energy efficient logic, circuit and layout techniques contribute to a design.

6. Acknowledgement

This work was funded by EPSRC grant GR/S61270/01 and the authors are grateful for this support. The authors are also grateful to Dave Clark and Jeff Pepper for encouragement in this work.

References

[1] M.Lewis and L. Brackenbury, "CADRE: An Asynchronous Embedded DSP for Mobile Phone Applications", *Design Automation for Embedded Systems,* Vol.6, No.4, pp.451-475, 2002.

[2] M. J. G. Lewis, "Low Power Asynchronous Digital Signal Processing", *Doctor of Philosophy Thesis*, Faculty of Science and Engineering, University of Manchester, 2000.

[3] Anantha P. Chandrakasan, Robert W. Brodersen, "Minimizing Power Consumption in CMOS Circuits", Proceedings of the IEEE, vol.83, Apr., pp.498-523, 1995.

[4] D.K.Arvind and Robert D. Mullins, "A Fully Asynchronous Superscalar Architecture", *International Conference on Parallel Architecture and Compilation Techniques,* Oct., p.17-22, 1999.

[5] D. Peiliang, Y. Rilong, X. Hongbo and Y. Chengfang, "Multi-clock driven system: a novel VLSI architecture", *Proceedings 4th International Conference on ASIC,*Oct., pp.555-558, 2001.

[6] M. Singh and M. Theobald, "Generalized latency-insensitive systems for single-clock and multi-clock architectures", *Proceedings on Design, Automation and Test in Europe Conference and Exhibition,* vol.2, Feb., pp.1008-1013, 2004.

Survey of finding empty space algorithm for partial reconfigurable FPGAs

Sasithorn Somvathee
Dept. of computer engineering, faculty of engineering
Prince of Songkla University
Hatyai Songkhla Thailand 90112
ssasatorn@hotmail.com

W. Suntiamorntut
Dept. of computer engineering, faculty of engineering
Prince of Songkla University
Hatyai Songkhla Thailand 90112
wannarat@coe.psu.ac.th

Abstract—**Partial reconfigurable FPGAs can dynamically modified their logic and interconnect configuration at runtime without affecting each other. Finding the available empty space for incoming tasks at runtime is the most time consuming process in the online placement algorithm. The primary goal of such an algorithm is speed and memory usage. This paper presents the survey of three finding free resources algorithms by presenting their data structure and method. A comparison of time complexity for each algorithm is also presented here.**

Keywords: online placement, partially reconfigurable FPGAs, hardware multitasking

I. INTRODUCTION

The flexibility of reprogrammability in FPGA (Field Programmable Gate Arrays) is a great advantage of this device. We can make the use of their reprogrammability in one of two ways: Compile-Time Reconfiguration (CTR) or Run-Time Reconfiguration (RTR). The FPGA configuration can be changed during the operation execution when the RTR system is applied. The configuration can be fully or partially reprogrammed.

A FPGA consists of a rectangular grid of Configurable Logic Blocks (CLBs) and interconnection between the cells. FPGA allows multiple tasks to be executed in parallel by hardware. For such systems an application is divided into smaller tasks, called hardware tasks. When the system requests its execution each of them is placed on the FPGA. After a task finishes execution, the system deletes it and its area can be reclaimed and reused by other tasks.

The task placement may be subdivided into two categories: offline and online. In the offline placement, the configuration order and location of tasks are known when the application is compiled.

On the other hand, in the online placement, all tasks are known at runtime and can be placed anywhere on the chip. The configuration of hardware tasks on the FPGA must be done on the fly. So the system searches available resources for a new task on a case-by-case basis, which is time-consuming.

Because requested tasks are known only at runtime, the system's management must face with two problems: where to place a new arrival task for execution [1], [2], [3] and how to provide communication among the modules running on the FPGA [4], [5].

This paper presents three approaches to solving the problem of finding the free space for arrival tasks on the FPGAs. The first is Staircase algorithm [1], which works by first finding all the maximal staircases and the extracting the maximum empty rectangles from them. The second, enhanced Scan Line algorithm [2] uses 2D FPGA surface model with encoding information. MKE points are defined to utilize the scanning process while looking for the maximum free rectangles. The third, Flow Scan algorithm [3] scans the maximum empty rectangles from the task edges.

In section 2, we describe the detail of these three algorithms. Then, we show the comparison of time complexity in section 3. Finally, we conclude this paper in section 4.

II. EXISTING ALGORITHMS FOR FINDING EMPTY SPACE

The free FPGA space is usually recorded as a set of rectangles because most of the hardware tasks can be fitted in a rectangular shape. There are two types of rectangles: the non-overlapping rectangles and the maximum rectangles. In the non-overlapping, algorithm has a shorter execution time but more task rejection rate than the maximum rectangles.

In this section we describe the algorithm for finding all maximum empty rectangles by using the same sample as shown in figure 1. As an example, FPGA consists of 10x10 cells with two placed tasks running. Maximum Empty Rectangle (MER) is denoted by the tuple (x, y, w, h), where (x, y) is the coordinates of its lower left corner, and (w, h) is its width and height. There are 6 maximum empty rectangles in total: (1, 1, 10, 1), (1, 1, 1, 10), (5, 1, 6, 4), (5, 1, 1, 10), (1, 7, 5, 4), (1, 9, 10, 2).

A. *Staircase algorithm*

This algorithm uses a 2D array with X number of columns and Y number of rows. This array is called area matrix. Each cell in the array represents a CLB in the FPGA. Bottom left cell is addressed (1, 1) and top right cell is addressed (X, Y).

If a CLB is used by a task, the cell is called an occupied cell, which is represented by a negative number. Otherwise it is called an empty cell, which is represented by a positive

number. Figure 2 shows the area matrix of a FPGA with two tasks placed on it. A positive number gives number of contiguous empty cells above and including that cell in that column and a negative number gives the remaining width of the task.

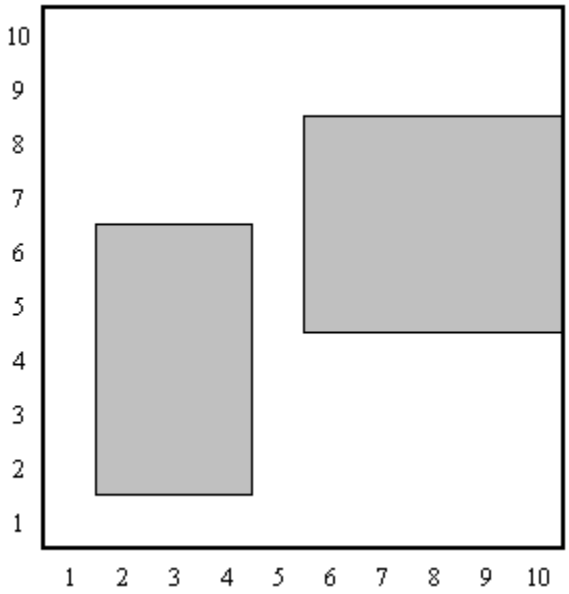


Figure 1. Example FPGA configuration with two placed tasks.

This algorithm scans the area matrix on a row-by-row basis from top to bottom and make staircase at the empty locations. We check only the maximal staircases for extracting maximum empty rectangles. A staircase is maximal if it cannot be extended down or to its right. Only the top horizontal boundary of already placed tasks and the bottom-most row will be scanned as shown in figure 2. In each scanned row, scanning is done from the left side to the right side. If the left boundary of a task is encountered, the occupied cells can be skipped. This gives considerable savings in runtime.

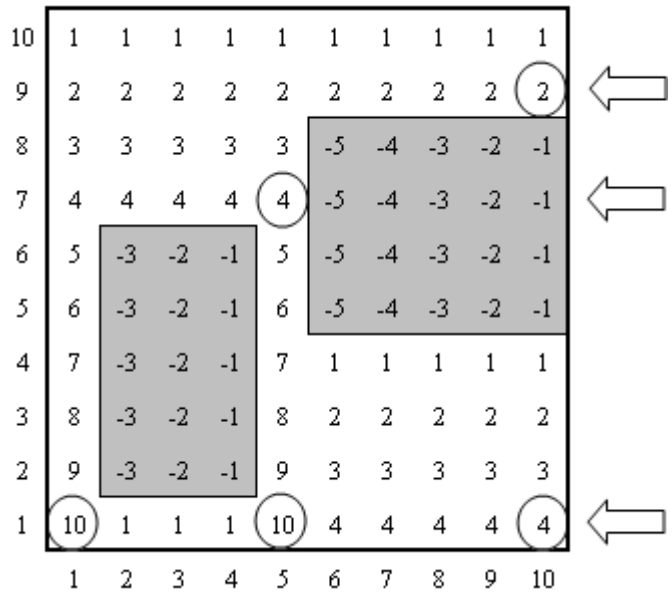


Figure 2. Modeling FPGA area matrix, the origin point of maximal staircase (circles) and the scanning row (block arrows)

In order to construct a staircase, there is only one stair in the staircase at the first leftmost positive entry of each scanned row. The height of that stair is given by positive integer stored at that location. A staircase at point (x+1, y) can be easily constructed from a staircase at point (x, y). Let (x, y') be the top right most point on the stair containing column x and (x+1, y'') be the coordinate of topmost empty cell on the column x+1. There are three possible cases to construct a staircase:

y" > y' A new stair is added to staircase and y" becomes top-left corner of the new stair

y" = y' Width of the rightmost stair is extended by one.

y" < y' All the stairs with height greater than that y" are deleted and the last one is replaced with new highest stair, which has height equal to y".

B. Enhanced Scan Line Algorithm

This algorithm also use 2D array but add one extra column to the right edge of the FPGA array, and assign value 0 to cells on that column. The occupied cell is represented by value 0 and the empty cell is represented by a positive number. A positive number in a cell gives number of contiguous empty cells at the left hand side and including that cell in that row.

Before scanning the free space with this algorithm, we have to find the Key Element, which is an empty cell with an occupied cell as its right hand neighbor, or an empty cell on the right edge of FPGA area. Next, we select the Scan Line. The Scan Line contains one or more Key Elements. Then we find the Valley Point to create a segment and choose the largest Key Element as the MKE.

The algorithm chooses the maximum value of MKEs in each column. Next, the scanning moves the tops and bottoms of multiple MKEs simultaneously. If they overlap with each other, then the subsequent steps of scanning these MKEs will be identical, and we only need to continue the scanning process for one of the MKEs to avoid any redundant scanning process.

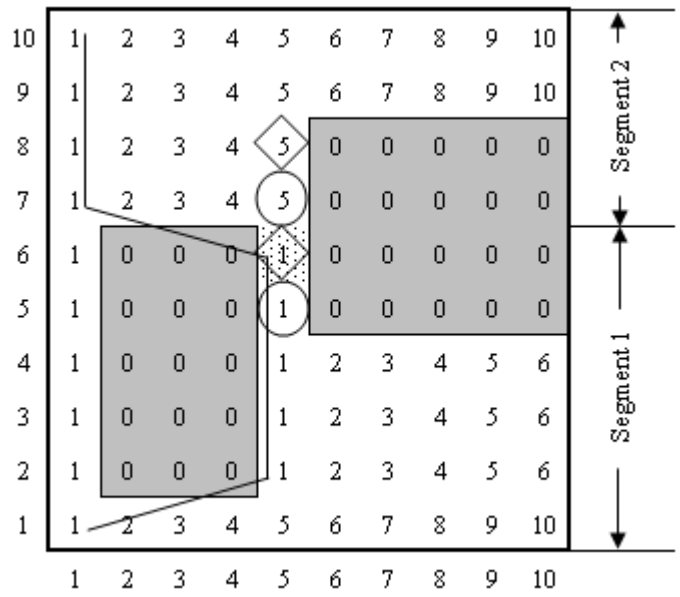
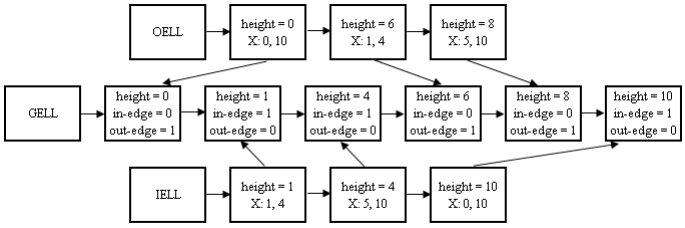


Figure 3. Modeling FPGA area matrix, MKEs (circles), Key Elements that are not maximal (diamond) and Valley Point (dotted cell) on column 5.

C. *Flow Scan Algorithm*

This algorithm uses linked list structure to store runtime information about the free space as in figure XXX. There are 4 different linked lists: general edge linked list (GELL), in-edge and out-edge linked lists (IELL and OELL), and rectangular well linked list (RWLL).



Because the linked lists is sorted when inserting or deleting task, the algorithm scans from the lowest height to the highest height of the GELL, which is the edge of FPGA. When the scanning flow reaches an out-edge, the out-edge processing is called. Only one new FRW is created. Its bottom has the same height as the out-edge.

When the scanning flow reaches an in-edge, the in-edge processing happens. The search for overlapped FRWs will start. If there is an overlapped FRW with an in-edge in the X direction, a maximum free rectangle is created by adding the height of the in-edge as a top line of the FRW and at most two new RWs can be created in case of the left side of FRW < the left side of in-edge or the right side of FRW > the right side of in-edge. No FRW will be generated if the height of the in-edge is the same as the top of FPGA.

III. COMPARISON OF TIME COMPLEXITY

The following sections provide comparisons of time of the previously described scanning algorithms.

An FPGA has x number of columns and y number of rows.

In the staircase algorithm, let n be the number of rows in FPGA which lie immediately above top boundary of some tasks. A staircase needs to be constructed at every column in a row. So, time complexity is O(xn). The worst case performance of staircase algorithm is O(xy).

For the scan line algorithm, it takes time O(y) to find all the MKEs of a Scan Line. If there are n scan lines, then the time complexity is O(ny). The worst case time complexity of scan line algorithm is O(xy), which is the same as the staircase algorithm.

On the other hand, only the task edges are processed. Let n be the number of placed task. The time complexity is O(n). The worst case for the flow scan algorithm is when all edges of n placed tasks are located on different height. This makes the worst case complexity of flow scan algorithm is O(y).

IV. CONCLUSION

In this article we provide descriptions of three algorithms for finding the maximum free rectangles on the partially reconfigurable FPGAs. We have presented a comparison of theses algorithms, highlighting their features and differences. Lastly since the use of online placement is important in the reconfigurable system, there are still many challenges that need to be met.

REFERENCES

[1] M. Handa, and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement". In *Proc. of the 41st annual conference on design automation*, San Diego, CA, June 2004, pp 960-965.

[2] J. Cui, Q. Deng, X. He, and Z. Gu, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs". In *Proc. of the conference on Design, automation and test in Europe*, Nice, France, April 2007, pp 1-6.

[3] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resources management on the FPGA". In *Proc. of the conference on Design, automation and test in Europe*, Munich, Germany, March 2008, pp 1095-1098.

[4] C. Bobda, A. Ahmadinia, "Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices". In *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 443-451, September/October, 2005.

[5] M. Tomono, M. Nakanishi, S. Yamashita, N. Nakajima, and K. Watanabe. "A new approach to online FPGA placement". In *40th annual conference on Information Sciences and Systems*, Princeton, USA, March 2006, pp 145--150.

Performance analysis of an efficient algorithm for free resources management on the FPGA

Sasathorn Somvathee
Department of computer engineering
Prince of Songkla University
Hatyai, Songkhla 90112 Thailand
ssasatorn@hotmail.com

W. Suntiamorntut
Department of computer engineering
Prince of Songkla University
Hatyai, Songkhla 90112 Thailand
wannarat@coe.psu.ac.th

Abstract— **The most time consuming of an online task placement is to find the available free space on the FPGAs. The flow Scan algorithm is recently purposed for finding a complete set of maximum empty rectangles. The performance of this algorithm compared with others is reported in this paper.**

Keywords: online placement, partially reconfigurable FPGAs, hardware multitasking

I. INTRODUCTION

Partially reconfigurable FPGA (Field Programmable Gate Array) allows their logic and interconnection between the cells modified at runtime without affecting each other. For such system an application is divided into smaller tasks called hardware tasks. Hardware tasks run concurrently on the FPGA. When a task finishes execution, the system removes it from the FPGA and its area can be relocated and reused by incoming tasks. Online partial reconfiguration allows the sequence of tasks to be performed unpredictably because the FPGA controller can make a decision online. Partially reconfigurable FPGA can also support the concept of dynamically dataflow reconfigurable coprocessor architecture.

In the offline placement, the sequence of request tasks is known in advance. In contrast with this method, the flow of tasks is known at runtime. We call this environment *online placement*. Various optimization algorithms have been applied to obtain good quality placements. The system needs to handle each task on a case-by-case basis. The placement time is an overhead on total execution time of the application. Finding and maintaining empty space on the FPGA is the most time consuming part of a placement algorithm. Thus a fast algorithm is required to efficiently manage free space for fast task placement.

There are three approaches for maintaining the empty space on the FPGA device: as a list of non-overlapping rectangles, a list of maximum empty rectangles, or a list of vertices, each with its pros and cons. Management a list of maximum empty rectangles is to fit more tasks on a given area than a list of non-overlapping rectangles. But it is often time consuming to maintain a complete set of maximum empty rectangles. In this paper, we implement an efficient algorithm for finding the complete set of maximum empty rectangles called Flow Scan algorithm [1].

The literature review of free space management in FPGA is explained in section 2. This paper details the flow scan algorithm in section 3. Then we discuss the problems found in the implement phase in section 4. In section 5, we present the simulation result and compare with the algorithm reported in [1]. Finally, we conclude this paper in section 6.

II. RELATED WORK

In the dynamic reconfigurable system, low area utilization [6,7] is the result of resource fragmentation in FPGA. An efficient algorithm to find empty space on the FPGA could help the defragment and task relocation. In this research area, we define MER as *Maximal Empty Rectangle* that means the empty rectangle that cannot be covered fully by any other empty rectangle.

Finding empty space is the basic fundamental problem in computational geometry. Jin Cui, et. al. in [8] proposed an efficient algorithm for finding the complete set of MERs in FPGA. Their works gave a better result compared to Scan Line Algorithm (SLA). However, in [9] showed that maintaining empty space as maximal rectangles leads to better utilization of resources. Therefore, Handa, et. al. in [1] reported a good quality results of their algorithms using maintaining maximal rectangles.

III. FLOW SCAN ALGORITHM

A. *Definitions[3]*

The lower Y coordinate of each placed task and top of FPGA are defined as *in-edge* while the *out-edge* is used for the higher Y coordinate and the bottom of the task. The direction of scan flow is from in-edge to out-edge.

Rectangular well (RW) is defined as temporally rectangles without top lines during the scanning process. *Formed rectangular well (FRW)* is any RW that can only be expanded upwards. Only FRW is recorded and temporal RW will be removed when there are several RWs with the same X coordinate crated during the scanning process. *Maximum free rectangular* is defined as rectangle that top, bottom, left and right edge cannot be expanded.

B. Data Structure

The flow scan algorithm uses linked list to store the runtime information. Figure 2 shows the linked lists, representing the situation as depicted in figure 1.

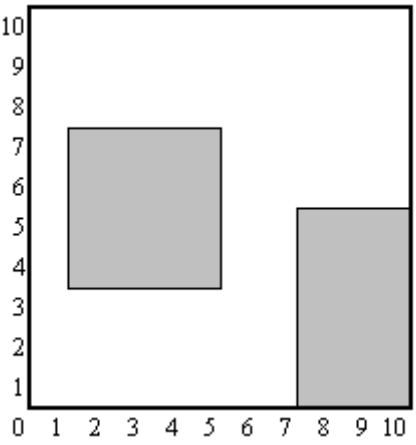


Figure 1. FPGA surface with two placed tasks.

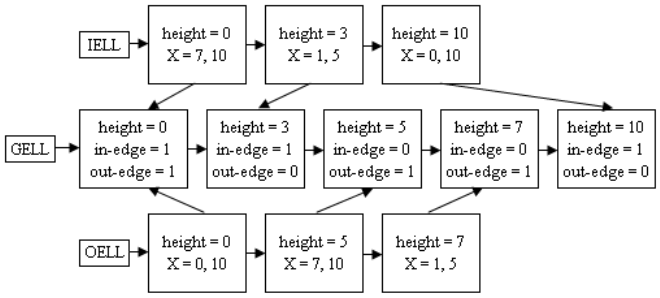


Figure 2. Linked lists.

For each placed task, its lower Y coordinate is defined as in-edge and its higher Y coordinate is defined as out-edge. The bottom and top of FPGA are defined as out-edge and in-edge by default respectively.

A rectangular well (RW) corresponds to the temporary rectangle without a top line, which created during the scanning process. For any RW that can be only expanded upwards is defined as formed rectangular well (FRW).

A maximum empty rectangle is a rectangle whose top, bottom, left and right cannot be expanded. In this paper, it is abbreviated as (left, right, bottom, top).

There are two scanning process in the flow scan algorithm: in-edge processing and out-edge processing.

The out-edge processing happens when the scanning flow leaves an out-edge. Only one new FRW is created. Its bottom has the same height of the out-edge.

When reaching an in-edge, the in-edge processing is called. The search for overlapped FRWs will start. If a FRW is overlapped with an in-edge in the X direction, a maximum empty rectangle is created by adding the height of the in-edge as the top line of new FRW. And, then, if the height of the in-edge is not the full height of FPGA, at most two new RWs can be created for the non-overlapping area within the FRW.

IV. Problems Found While Implementing

In this section, we discuss about the problems, which found at the implement phase.

The first problem is to find overlapped FRWs with an in-edge. This problem solved by comparing the X coordinates between FRW and in-edge. The overlapped FRW occurs if the right side of FRW is greater than the left side of in-edge or the left side of FRW is less than the right side of in-edge. Figure 3 shows the overlapped and non-overlapped FRWs.

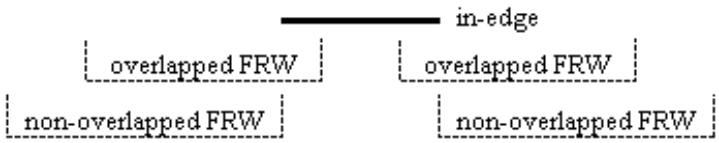


Figure 3. Example of overlapped and non-overlapped FRWs

Next problem is how to find the bottom of new FRW. In the out-edge processing, we can easily use the height of the edge as the bottom of new FRW. Meanwhile in the in-edge processing, new FRW is only created when it has overlapped FRWs. So we use the bottom of overlapped FRWs (dot cells) as the bottom of new FRW as shown in figure 4 (b).

Similar to the previous problem, what is the X coordinates of new FRW? In the in-edge processing, when it has overlapped FRWs and the non-overlapped part is on the left hand side, we use the left of overlapped FRW as the left of new FRW and the left of in-edge as the right of new FRW. If the non-overlapped part is on the left hand side, we use the right of in-edge as the left of new FRW and the left of overlapped FRW as the right of new FRW.

On the other hand, in the out-edge processing, the left and right of new FRW come from the leftmost of current FRW which the right of current FRW is equal to the left of out-edge and the rightmost of current FRW which the left of current FRW is equal to the right of out-edge as depicted in figure 4 (a).

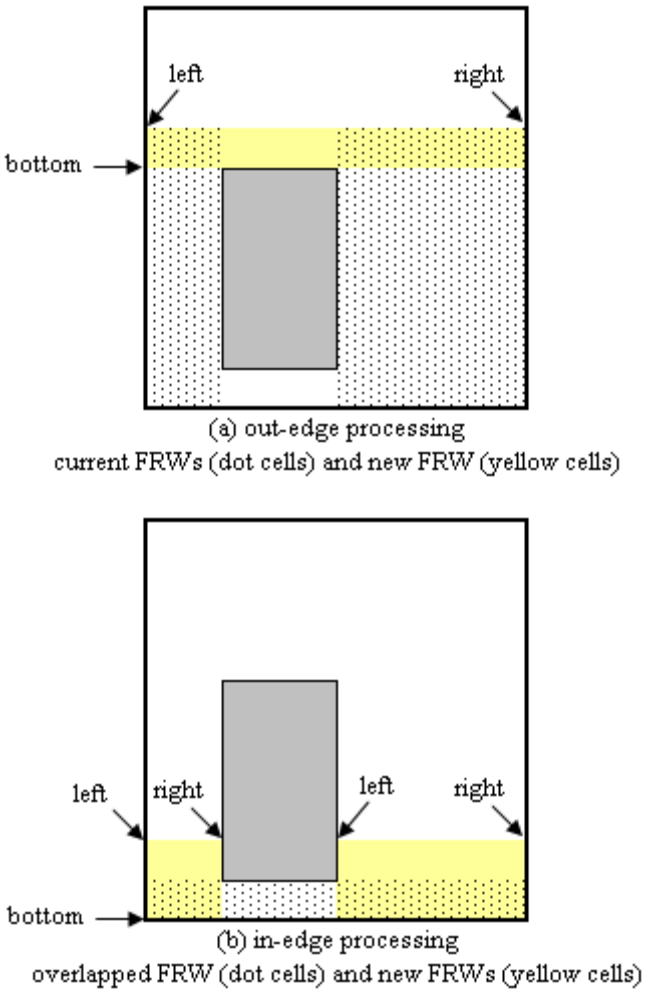


Figure 4. FPGA surface with placed task.

Finally, when the FRW is created, the important thing is to verify that the new one is not duplicated or overlapped with the current in lists. Also, when the maximum empty rectangle is recorded in the in-edge processing, we have to check that the bottom and the top is not the same height.

V. Simulation Result

The algorithm was implemented in C, and evaluated under Ubuntu 8.04 running on Intel Core 2 Duo 2.0 CPU 1.8 GHz with 2 GB main memory. We integrated with the simple online placement algorithm. The placement algorithm uses first fit policy to find the free space for incoming tasks from a complete set of maximum empty rectangles generated by the scanning process.

In order to compare with the original paper (1), we use the same simulation test. We start from 100x100 configurable logic units of FPGA. 1000 tasks were generated randomly by using the output of the previous scanning. Due to the simulation is not considered on the task rejection rate. One of the maximum empty rectangles is selected randomly and the size of the new task is randomly generated within the selected maximum empty rectangle. The arrival time is assigned between [5..25] time units. The task life time has 3 ranges: T_{250}, T_{500} and T_{1000}. For T_{250} the task life time is randomly chosen from the time interval [5..250]. For T_{500} the [251..500] is used and for T_{1000} the [501..1000] is used.

Because the CPU clock speed is too fast, then we use gprof program to calculate the amount of time spent in each routine. First, we compiled the algorithm with 'gcc -pg <source.c> -o <program>'. Next we ran the simulation and see the execution profile by using 'gprof <program>'. Figure 5 shows the result of gprof program.

```
File  Edit  View  Terminal  Tabs  Help
dekmaidee@dekmaidee-laptop:/media/disk/Code$ gprof fs_algo -b
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 50.00      0.01     0.01    16959     0.59     0.59  get_overlap_list
 50.00      0.02     0.01    15341     0.65     0.97  remove_task
  0.00      0.02     0.00    41715     0.00     0.00  create_frw
  0.00      0.02     0.00    24648     0.00     0.00  record_maximum_rectangle
  0.00      0.02     0.00    15960     0.00     0.00  out_edge_processing
  0.00      0.02     0.00     7376     0.00     1.36  in_edge_processing
  0.00      0.02     0.00     5000     0.00     0.00  random_int
  0.00      0.02     0.00     2002     0.00     0.00  insert_edge
  0.00      0.02     0.00     2002     0.00     0.00  insert_gell
  0.00      0.02     0.00     1982     0.00     0.00  remove_edge
  0.00      0.02     0.00     1967     0.00     5.08  fs_algorithm
  0.00      0.02     0.00     1000     0.00     0.00  create_task
  0.00      0.02     0.00     1000     0.00     0.00  find_position
  0.00      0.02     0.00     1000     0.00     5.08  insert_task
  0.00      0.02     0.00        1     0.00     0.00  initial
```

Figure 5. Screen shot of gprof program

The algorithm is executed every time when a new task arrives or one is removed. In our simulation with 10000 tasks, the algorithm is invoked approximately 19000 times. As shown in the figure 6, the average number of microseconds spent in this algorithm per call is presented.

In the flow scan algorithm, only the task edges are processed. The worst case is when all edges of n placed tasks located on different heights. The worst case time complexity of flow scan algorithm is O(2n).

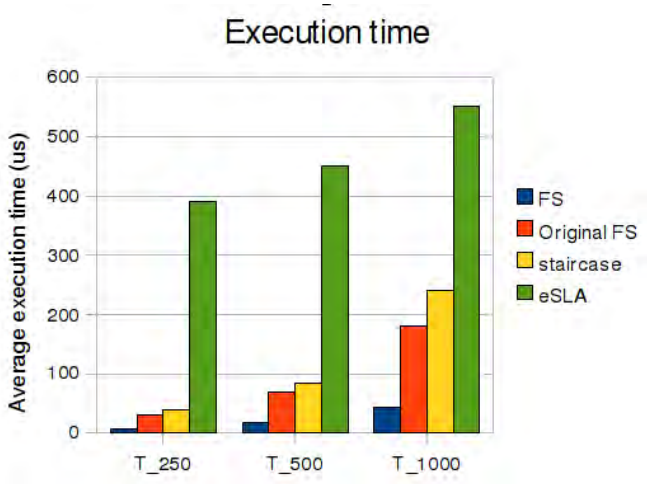


Figure 6. Execution time

VI. CONCLUSION

In this paper, we describe the scanning algorithm called flow scan algorithm. The problems and their solution are presented. Lastly we simulate the test to evaluate performance compare with the original paper.

REFERENCES

[1] M. Handa, and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement". In *Proc. of the 41st annual conference on design automation*, San Diego, CA, June 2004, pp 960-965.

[2] J. Cui, Q. Deng, X. He, and Z. Gu, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs". In *Proc. of the conference on Design, automation and test in Europe*, Nice, France, April 2007, pp 1-6.

[3] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resources management on the FPGA". In *Proc. of the conference on Design, automation and test in Europe*, Munich, Germany, March 2008, pp 1095-1098.

[4] C. Bobda, A. Ahmadinia, "Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices". In *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 443-451, September/October, 2005.

[5] M. Tomono, M. Nakanishi, S. Yamashita, N. Nakajima, and K. Watanabe. "A new approach to online FPGA placement". In *40th annual conference on Information Sciences and Systems*, Princeton, USA, March 2006, pp 145--150.

[6] M. Gericota, G. Alves, M. Silva and J. Ferreira, "Runtime management of logic resources on reconfigurable systems", *In Proceedings of Design, Automation and Test in Europe*, March, 2003.

[7] M. Handa, and R. Vemuri, "Area fragmentation in reconfigurable operating systems", *In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms,* CSREA Press, June, 2004.

[8] Jin Cui, Qingxu Deng, Xiuqiang He and Zonghua Gu, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs", *Design, Automation & Test in Europe Conference & Exhibition (DATE'07)*, April, 2007, pp.1-6.

[9] K. Bazargan, R. Kastner, and M. Sarrafzadeh. "Fast template placement for reconfigurable computing systems",