การออกแบบการแปลงเวฟเล็ตโดยใช้อัลกอริทึมลิฟต์ติ้งบน FPGA

A DESIGN OF WAVELET TRANSFORM USING LIFTING ALGORITHM ON FPGA

ไกรฤกษ์ โง้วสุวรรณ
KRAIRUEK NGOWSUWAN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2549

ISBN 974-15-2458-7

การออกแบบการแปลงเวฟเล็ตโดยใช้อัลกอริทึมลิฟต์ติ้งบน FPGA

A DESIGN OF WAVELET TRANSFORM USING LIFTING ALGORITHM ON FPGA

ไกรฤกษ์ โง้วสุวรรณ

KRAIRUEK NGOWSUWAN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2549

ISBN 974-15-2458-7

A DESIGN OF WAVELET TRANSFORM USING LIFTING ALGORITHM ON FPGA

KRAIRUEK NGOWSUWAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2006

ISBN 974-15-2458-7

หัวข้อวิทยานิพนธ์	การออกแบบการแปลงเวฟเล็ตโดยใช้อัลกอริทึมลิฟติ้งบน FPGA
นักศึกษา	นาย ไกรฤกษ์ โง้วสุวรรณ
รหัสนักศึกษา	46061003
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2549
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ. ดร. อรฉัตร จิตต์โสภักตร์

บทคัดย่อ

งานวิจัยนี้จะนำเสนอการออกแบบการแปลงเวฟเล็ตโดยใช้สมการของ Daubachies4 (D4) แบบ Lifting บน FPGA ในการออกแบบสามารถจำแนกโครงสร้างของโมดูลแปลงเวฟเล็ตออกได้เป็นสองประเภท ซึ่งโครงสร้างทั้งสองจะถูกออกแบบในลักษณะของเลขจำนวนจริง และเลขจำนวนเต็ม ในส่วนการออกแบบโมดูลที่ใช้คำนวณเลขจำนวนจริงจะอ้างอิงตามมาตรฐาน IEEE 754 แบบ 32 บิต จากนั้นก็นำโมดูลที่ใช้ในการคำนวณต่างๆ มาทำการจัดเรียงตามสมการของ Daubachies4 (D4) โดยที่โครงสร้างแปลงเวฟเล็ตที่ได้ออกแบบนั้นจะใช้ความสามารถการประมวลผลแบบขนานของ FPGA เพื่อให้สามารถทำการคำนวณค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่ต่ำ และความถี่สูงได้พร้อมกันทำให้เวลาในการประมวลผลลดลง ในการเปรียบเทียบประสิทธิภาพของการแปลงเวฟเล็ตแต่ละแบบ Lifting ที่ได้ออกแบบในงานวิจัยนี้จะทำการเปรียบเทียบกับการแปลงเวฟเล็ตแบบใช้ Scaling Function และ Wavelet Function โดยการเปรียบเทียบจะพิจารณาจากค่า PSNR, Zero Moment และความเร็วที่โมดูลสามารถทำงานตลอดจนทรัพยากรที่ใช้ในการออกแบบ FPGA นอกจากนี้จะกล่าวถึงประสิทธิภาพ และความสามารถของแต่โครงสร้างในการวิเคราะห์ภาพ

Thesis Title	A Design of Wavelet Transform Using Lifting Algorithm on FPGA
Student	Mr. Krairuek Ngowsuwan
Student ID	46061003
Degree	Master of Engineering
Programme	Computer Engineering
Year	2006
Thesis Advisor	Asst. Prof. Dr.Orachat Chitsobhuk

ABSTRACT

This thesis presents designs of wavelet algorithm using lifting algorithm for Daubachies4 (D4) functions on FPGA. Two main structures of Daubachies4 (D4) using lifting technique are implemented in both integer and floating point format. A 32-bit floating point computational module used in the proposed architecture is designed based on IEEE754 standard. The computational modules are arranged according to each Daubechies4 structure. A parallel architecture can be designed and implemented on FPGA to improve compute coefficients for low pass and high pass filters simultaneously and thus can reduce processing time. The efficiency of the proposed structures of Daubachies4 (D4) using lifting technique is compared to that of using scaling function and wavelet function. Several criteria are used to compare the efficiency of both structures such as PSNR, zero moment, required resources, and processing speed. We compare the quantitative measurement as stated above and discuss the difference in quality and ability of each structure in image analysis.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้จะสำเร็จลุล่วงไปเสียมิได้ หากปราศจากความช่วยเหลือจากผู้มีพระคุณหลายท่าน ขอกราบขอบพระคุณบิดามารดาของข้าพเจ้าที่ได้ให้กำลังใจมาโดยตลอด

ขอกราบขอบคุณท่านอาจารย์ที่ปรึกษา ผศ.ดร.อรฉัตร จิตต์โสภักตร์ สำหรับวิชาความรู้ที่ได้มอบให้และการให้คำปรึกษา คอยให้แนวทางในการแก้ปัญหาตั้งแต่เริ่มต้นจนสามารถสำเร็จเป็นวิทยานิพนธ์นี้ได้

ขอบคุณเพื่อนๆ และรุ่นพี่ที่ได้คอยเป็นกำลังใจให้ความช่วยเหลือ ให้คำแนะนำและเป็นกำลังใจที่ดีตลอดมา

สุดท้ายขอกล่าวคำขอบคุณ บัณฑิตวิทยาลัยที่ให้ทุนสนับสนุนงานวิจัยในครั้งนี้

คุณความดีและประโยชน์ที่ได้จากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้บิดามารดาที่รักและเคารพ ตลอดจนผู้มีพระคุณทุกท่าน

ไกรฤกษ์ โง้วสุวรรณ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

ตารางที่ หน้า

สารบัญรูป

สารบัญรูป(ต่อ)

รูปที่ หน้า

สารบัญรูป(ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันนี้อุตสาหกรรมการออกแบบวงจรรวม ได้มีการพัฒนาไปอย่างรวดเร็ว สาเหตุมาจากความต้องการของผู้ใช้งานในปัจจุบันต้องการอุปกรณ์ที่มีขนาดเล็กแต่สามารถทำงานได้อย่างมีประสิทธิภาพ และในการพัฒนาวงจรรวมเบื้องต้นสามารถทำได้โดยการใช้ FPGA (Field Programmable Logic Array) ข้อดีของการพัฒนาวงจรรวมโดยใช้ FPGA คือ สามารถทำการพัฒนาได้รวดเร็วและสามารถแก้ไขการทำงานของวงจรรวมที่ได้ทำการออกแบบไปแล้วได้อย่างง่ายดาย

การออกแบบระบบที่ใช้ในการประมวลผลภาพ (Image Processing) ไม่ว่าจะเป็นการปรับปรุงคุณภาพของภาพ การบีบอัดขอมูลภาพ ฯลฯ เป็นการเน้นหนักไปทางด้านการพัฒนาซอฟแวร์เป็นหลัก อาจไม่เหมาะสำหรับการนำไปใช้งานจริง ดังนั้นในงานวิจัยนี้จึงได้ทำการวิจัยและพัฒนาการประมวลผลภาพโดยการใช้ FPGA เพื่อช่วยลดขนาดของระบบที่ใช้ในการประมวลผล

งานวิจัยนี้จะเลือกการแปลงเวฟเล็ต (Wavelet Transform) มาทำการออกแบบวงจรรวมโดยใช้ FPGA สาเหตุที่ทำการเลือกการแปลงเวฟเล็ตมาทำการออกแบบ เนื่องจากแปลงเวฟเล็ตเป็นวิธีการที่ให้ผลของการบีบอัดข้อมูลมีคุณภาพสูงกว่าการแปลงข้อมูลวิธีอื่นเมื่ออัตราการบีบอัดข้อมูลเท่ากัน นอกจากนี้ข้อมูลที่ได้จากการแปลงเวฟเล็ตยังให้ข้อมูลด้านความถี่ และตำแหน่ง ซึ่งเป็นประโยชน์ในการนำไปใช้ในการวิเคราะห์ภาพในงานต่างๆได้ดี

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1.2.1 ศึกษาวิธีการแปลงเวฟเล็ตเบื้องต้น เพื่อหาข้อดีและข้อเสียของแต่ละวิธีที่จะนำมาใช้ในการออกแบบ FPGA

1.2.2 ศึกษาการออกแบบวงจรรวมโดยใช้ FPGA ตลอดจนการนำวงจรที่ได้ทำการออกแบบไปประยุกต์ใช้งาน

1.2.3 ศึกษาและปรับปรุงวิธีการเดิมโดยอาศัยสมมติฐาน ความเข้าใจ และทฤษฎีอื่นประกอบการทดลอง

1.2.4 สามารถประเมินประสิทธิภาพ รวมทั้งสรุปวิเคราะห์ผลลัพธ์ที่ได้จากงานวิจัยนี้

1.2.5 เป็นแหล่งข้อมูลอ้างอิงสำหรับผู้สนใจงานวิจัยด้านนี้ต่อไป

1.3 สมมติฐานของการศึกษา

การแปลงเวฟเล็ตจะถูกเขียนอยู่ในรูปสมการที่ใช้ในการหาค่าสัมประสิทธิ์ความถี่ต่ำ (Approximation) และสมการที่ใช้ในการหาค่าสัมประสิทธิ์ความถี่สูง (Detail) การใช้ซอฟแวร์ประมวลผลจะเป็นการเขียนโปรแกรมที่ใช้การคำนวณเป็นหลักอาจทำให้การทำงานล่าช้า เพราะการทำงานของ CPU จะเป็นแบบลำดับขั้น (Sequential Machine) ถ้าหากว่านำฟังก์ชั่นดังกล่าวมาสร้างเป็นวงจรดิจิตอลโดยใช้ FPGA เพื่อให้สามารถทำการคำนวณค่าสัมประสิทธิ์ ก็น่าจะใช้เวลาในการประมวลผลน้อยลง เพราะ FPGA สามารถทำงานหลายอย่างพร้อมกันได้ (Parallel Processing) ดังนั้นในงานวิจัยนี้จะทำการศึกษาและออกแบบการแปลงเวฟเล็ตโดยใช้ FPGA

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย

เพื่อให้สามารถทำการออกแบบการแปลงเวฟเล็ตบน FPGA ได้นั้นจะต้องอาศัยหลักการและทฤษฎีดังต่อไปนี้

1. หลักการทำงานเบื้องต้นของ FPGA
2. ทฤษฎีการออกแบบจรดิจิตอลเบื้องต้นบน FPGA
3. ทฤษฎีการแปลงเวฟเล็ตเบื้องต้น
4. ทฤษฎีการคำนวณเลขจำนวนจริงตามมาตรฐาน IEEE754

1.5 ขอบเขตการวิจัย

งานวิจัยนี้จะทำการศึกษาและดำเนินการวิจัยเพื่อทำการสร้างการแปลงเวฟเล็ตโดยใช้ FPGA งานวิจัยนี้จะนำเสนอการออกแบบ FPGA เพื่อใช้แปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้สมการของ Lifting และทำการเปรียบเทียบกับการแปลงเวฟแบบ Daubechies4 (D4) โดยใช้สมการ Scaling Function และWavelet Function ในการตรวจสอบจะใช้ค่า PSNR (Peek Signal to Noise Ratio) และค่า Zero Moment และรายละเอียดของข้อมูลที่ได้หลังจากการแปลงเวฟเล็ตเป็นตัวเปรียบเทียบประสิทธิภาพการแปลงเวฟเล็ตแต่ละแบบ นอกจากนี้ยังทำการตรวจสอบความเร็วที่ใช้ในการประมวลผล และทรัพยากรที่ใช้ในการออกแบบ FPGA

1.6 ขั้นตอนของการศึกษา

1.6.1 กำหนดวัตถุประสงค์และขอบเขตของงานวิจัย ว่าต้องการนำเสนอหรือศึกษางานวิจัยในหัวข้อนี้มากน้อยแค่ไหนและอย่างไร

1.6.2 ศึกษาค้นคว้าทฤษฎีและงานวิจัยที่เกี่ยวข้องกับการแปลงเวฟเล็ตโดยใช้ FPGA แล้ววิเคราะห์ ข้อดี ข้อด้อย และประเด็นที่น่าสนใจของงานวิจัยอื่นๆ ที่สามารถนำมาปรับปรุงและประยุกต์ใช้ เพื่อก่อให้เกิดประโยชน์แก่งานวิจัยนี้ได้

1.6.3 ตั้งสมมติฐานของการศึกษาและกำหนดวางแนวความคิดของงานวิจัย โดยมีการอ้างอิงทฤษฎีหรือหลักการที่เกี่ยวข้องเพื่อที่จะบรรลุตามวัตถุประสงค์ที่ได้กำหนดไว้

1.6.4 เตรียมฐานข้อมูลภาพ เพื่อนำมาใช้ทดสอบงานวิจัย โดยฐานข้อมูลดังกล่าวนี้ประกอบไปด้วยภาพหลากหลายชนิดเพื่อที่จะสามารถนำมาใช้ตรวจสอบประสิทธิภาพของระบบที่วิจัยได้อย่างเหมาะสม

1.6.5 พัฒนาโปรแกรมที่ใช้ในการแปลงเวฟเล็ตเพื่อใช้ในการตรวจสอบอัลกอริทึมก่อนที่จะนำไปออกแบบฮาร์ดแวร์

1.6.6 ทำการออกแบบฮาร์ดแวร์จากอัลกอริทึมที่ได้ทำการทดสอบโดยซอฟแวร์ อุปกรณ์ที่ใช้ในการพัฒนาคือ FPGA โดยในขณะทำการทดลองนั้นจะเก็บข้อมูลและผลลัพธ์ของทุกๆ ขั้นตอนเอาไว้ เพื่อนำมาวิเคราะห์ปรับปรุงงานวิจัยต่อไป

1.6.7 นำผลลัพธ์จากการทดลองมาวิเคราะห์และประเมินงานวิจัย ทั้งในแง่ของคุณภาพและประสิทธิภาพ โดยเปรียบเทียบกับงานวิจัยอื่นๆ แล้วสรุปผลเพื่อนำเสนอผลงานวิจัย

1.7 เครื่องมือและอุปกรณ์ที่ใช้ในงานวิจัย

เครื่องมือและอุปกรณ์ที่ใช้ในงานวิจัย ได้แก่

1.7.1 เครื่องคอมพิวเตอร์ส่วนบุคคลใช้หน่วยประมวลผลกลาง Intel Pentium 4 ความเร็ว 3.0GHz หน่วยความจำ (RAM) 1G Byte จำนวน 1 เครื่อง

1.7.2 ระบบปฏิบัติการวินโดวส์ XP

1.7.3 Microsoft C#.Net

1.7.4 Xilinx ISE 6.2

1.7.5 บอร์ด FPGA ที่ใช้ชิพ Xilinx Spartan-3 (XCS1500-FG456) ที่สามารถทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์ผ่านทาง PCI บัส

1.8 โครงสร้างของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้แบ่งออกเป็น 6 บทแต่ละบทมีเนื้อหาดังต่อไปนี้

บทที่ 1 กล่าวถึง ความเป็นมาและความสำคัญของปัญหา จุดมุ่งหมายและวัตถุประสงค์ของการศึกษา สมมติฐานของการศึกษา รวมทั้งทฤษฎีหรือแนวคิดที่ใช้ในการศึกษา ขอบเขตของการศึกษา และขั้นตอนของการศึกษา

บทที่ 2 กล่าวถึง คุณสมบัติต่างๆ ของ FPGA เมื่อเปรียบเทียบกับการทำงานของ CPU และ DSP และงานวิจัยที่เกี่ยวข้องกับการนำเอา FPGA ไปประยุกต์ใช้ในการแปลงเวฟเล็ต

บทที่ 3 กล่าวถึง ความรู้พื้นฐานที่เกี่ยวข้องกับงานวิจัยนี้ ประกอบไปด้วยทฤษฎีการแปลงเวฟเล็ตเบื้องต้น และระบบการแทนเลขจำนวนจริง

บทที่ 4 กล่าวถึง การออกแบบโมดูลที่ใช้ในการคำนวณเลขจำนวนจริงเพื่อนำไปใช้ในการออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ต และการออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ต

บทที่ 5 กล่าวถึง ผลการทดลองและการวิเคราะห์ผลการทดลอง

บทที่ 6 กล่าวถึง ข้อเสนอแนะ และแนวทางในการพัฒนาต่อไปในอนาคต

บทที่ 2
งานวิจัยที่เกี่ยวข้องกับการแปลงเวฟเล็ตโดยใช้ FPGA

ในปัจจุบันการแปลงเวฟเล็ตได้เข้ามีบทบาทในด้านการประมวลผลสัญญาณ การประมวลผลภาพ รวมทั้งเป็นตัวจัดการกับข้อมูลเบื้องต้นก่อนที่จะทำการบีบอัดข้อมูล แต่ว่าการแปลงเวฟเล็ตส่วนใหญ่จะนิยมใช้ซอฟแวร์ในการประมวลผล ในงานวิจัยนี้จะเป็นการนำซอฟแวร์ในส่วนของฟังก์ชันที่ใช้ในการแปลงเวฟเล็ตมาแปลงให้อยู่ในรูปของฮาร์ดแวร์เพื่อให้มีขนาดเล็กและทำงานได้อย่างรวดเร็ว ในส่วนของอุปกรณ์ที่ใช้ในการออกแบบฮาร์ดแวร์ฟังก์ชันจะเลือกใช้ FPGA (Field Programmable Gate Array) ซึ่งเป็นอุปกรณ์ที่ใช้ออกแบบวงจรรวม

หลายคนที่รู้จักชิพวงจรรวม FPGA มักนำไปเปรียบเทียบกับไมโครคอนโทรลเลอร์ ที่มีราคาถูกกว่ากันมาก แต่ไม่ได้มองทางด้านความเร็วของการทำงาน ซึ่งไมโครคอนโทรลเลอร์จะทำงานได้ช้ากว่ามาก และจุดที่ควรพิจารณาอีกประการก็คือ ประโยชน์ทางด้านการออกแบบวงจรดิจิตอล ซึ่งชิพวงจรรวม FPGA สามารถกระทำได้ง่ายมาก เพราะเราสามารถเรียกใช้สัญลักษณ์ของลอจิกเกตจากโปรแกรม หรือการเขียนด้วยภาษาคอมพิวเตอร์แล้วแปลภาษาลงไปยังชิพวงจรรวม FPGA ทำให้วงจรดิจิตอลขนาดใหญ่ๆ ในอดีตที่ใช้ไอซี 10 – 30 ตัว ถูกย่อส่วนลงในไอซี FPGA ขนาด PLCC44 เพียงตัวเดียวทำให้ประหยัดงบประมาณในการจัดซื้อไอซีจำนวนมากๆ อีกทั้งใช้พื้นที่ของแผ่นพิมพ์วงจร (PCB) เพียงเล็กน้อย และประหยัดเวลาในการออกแบบ ส่งผลให้ต้นทุนการผลิตต่ำกว่าในอดีตมาก

2.1 FPGA (Field Programmable Gate Array)

FPGA ถูกประดิษฐ์ครั้งแรกในปี ค.ศ. 1985 เพื่อทำการขยายขีดความสามารถของการออกแบบวงจรดิจิตอลให้มากกว่า PLA/PLD (Programmable Logic Array / Programmable Logic Device) ในเรื่องของจำนวนความจุเกตและความเร็ว สำหรับรองรับการนำไปใช้งานกับระบบที่มีความซับซ้อน FPGA เป็นอุปกรณ์ฮาร์ดแวร์ที่สามารถโปรแกรมหรือเปลี่ยนลักษณะการทำงานได้เหมือนซอฟแวร์ทำให้มีความยืดหยุ่นในการนำไปใช้งานสูง จึงไม่แปลกใจเลยที่ FPGA ได้รับความนิยมนำไปใช้ในการออกแบบวงจรรวม ทางด้านผู้ผลิต FPGA จะมีสองค่ายใหญ่ด้วยกันคือ Altera และ Xilinx

การออกแบบวงจรดิจิตอลด้วย FPGA จะมีองค์ประกอบ 4 ส่วนหลักๆ ดังแสดงในรูปที่ 2.1

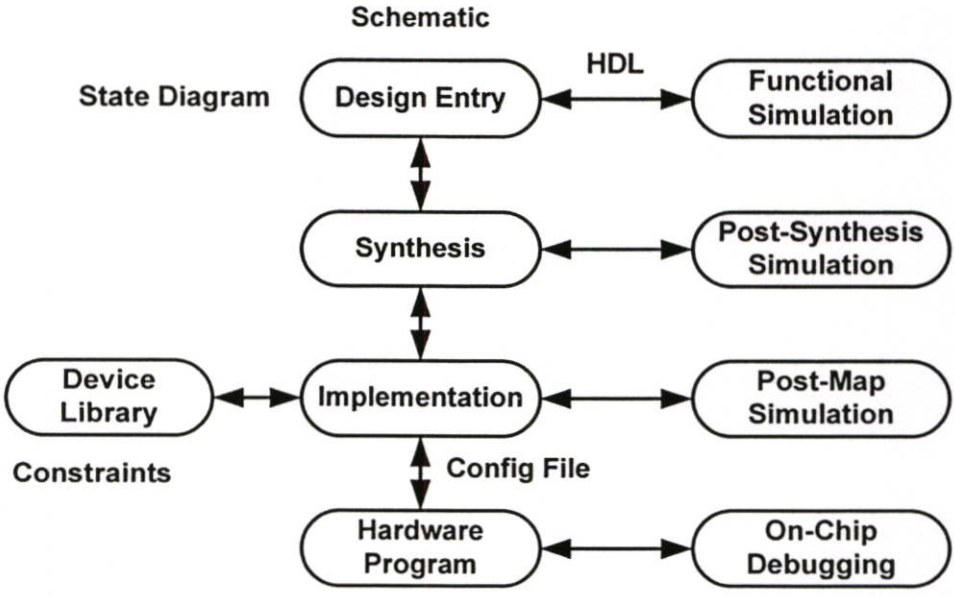
1. Design Entry เป็นขั้นตอนของการรับข้อมูลของการออกแบบเข้าไปในระบบ โดยทั่วไปสามารถทำได้หลายทางเช่น โดยใช้ Schematic Design Entry เพื่อดึงอุปกรณ์จากไลบารีของ FPGA หรือโดยใช้ภาษา HDL อย่างเช่น VHDL หรือ Verilog การออกแบบโดยใช้ภาษาระดับสูงจะมีข้อดี

คือ ไม่ขึ้นอยู่กับเทคโนโลยีหรือชิพที่จะนำมาโปรแกรม นอกจากนี้การออกแบบด้วยภาษาระดับสูง ผู้ออกแบบไม่จำเป็นต้องรู้ถึงรายละเอียด และการเชื่อมต่อภายใน ส่วนการทดสอบความถูกต้องจะเป็นในลักษณะการตรวจสอบระดับฟังก์ชันการทำงาน (Functional Simulation) โดยใช้ซอฟต์แวร์สำหรับจำลองการทำงาน

2. Synthesis ขั้นตอนนี้จะเกี่ยวกับการแปลงวงจรที่ได้ทำการออกแบบไปแล้วไม่ว่าจะเป็น HDL หรือ Schematic ให้เป็นลอจิก โดยมีขั้นตอนย่อยๆ คือการสังเคราะห์วงจร (Logic Synthesis) โดยปกติแล้วจะมีขั้นตอนการ Optimization ด้วยเพื่อให้วงจรที่ได้ใช้ทรัพยากรน้อยที่สุด หรือทำงานเร็วที่สุด ผลลัพธ์จากขั้นตอนนี้จะเป็น EDIF (Electronic Design Intermediate Form) ซึ่งเป็นไฟล์ที่อธิบายการเชื่อมต่อ (Netlist) การจำลองการทำงานในขั้นตอนนี้จะเป็นลักษณะของ Post-Synthesis Simulation

3. Implementation หลังจากได้ไฟล์ที่อธิบายการเชื่อมต่อมาแล้ว จะทำการแมป (Map) ให้เข้ากับเทคโนโลยีหรือตัวชิพที่ใช้งาน (Technology mapping) หลังจากนั้นก็จะทำการวางตำแหน่ง (Placement) ของลอจิกตามที่ลอจิกเซลล์ต่าง ๆ แล้วทำการเชื่อมต่อสายสัญญาณ (Routing) ก่อนที่จะได้ไฟล์สำหรับโปรแกรมลงชิป (Reconfiguration File) การจำลองการทำงานในระดับนี้คือ Post-Map Simulation จะสามารถดูดีเลย์ตามจุดต่าง ๆ ได้

4. Hardware Program การโปรแกรมอุปกรณ์หรือชิพ FPGA สามารถทำได้ทันทีและแก้ไขได้ตลอดเวลา โดยกลับไปทำขั้นตอนแรกใหม่ ก่อนที่จะทำการโปรแกรมใหม่ โดยไม่เสียค่าใช้จ่ายเพิ่มเติม



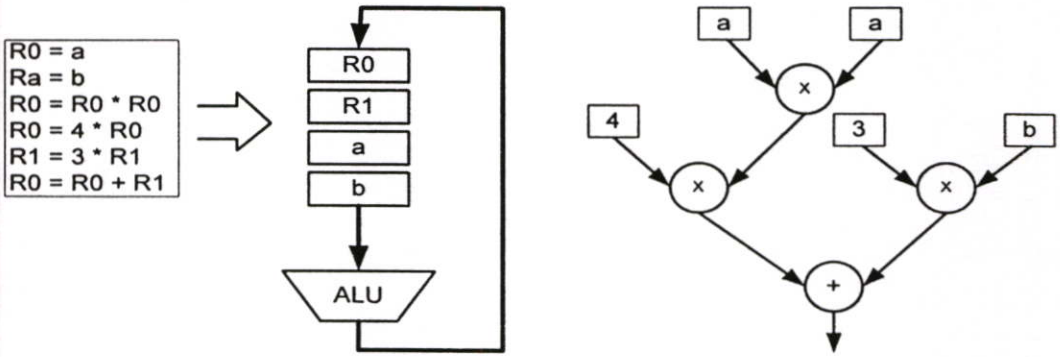
รูปที่ 2.1 ขั้นตอนการออกแบบบน FPGA

2.2 เปรียบเทียบการทำงานระหว่าง FPGA กับ CPU และ DSP

หัวข้อนี้จะอธิบายความแตกต่างและลักษณะการทำงานระหว่าง FPGA CPU และ DSP นอกจากนี้ยังจะกล่าวถึงข้อดีข้อเสียของ FPGA เมื่อนำมาใช้ในการออกแบบวงจรรวม

2.2.1 เปรียบเทียบการทำงานระหว่าง FPGA กับ CPU

FPGA เป็นอุปกรณ์ที่สามารถทำงานได้หลายๆ ฟังก์ชั่นพร้อมกัน หากเปรียบเทียบการทำงานกับ CPU ในปัจจุบันนี้สามารถทำงานได้โดยใช้สัญญาณนาฬิกาที่ระดับหลาย GHz และนอกจากนี้ยังสามารถทำการประมวลผลตามซอฟแวร์ได้อย่างรวดเร็ว แต่ในความเป็นจริงแล้วอุปกรณ์ที่ถูกออกแบบโดยฮาร์ดแวร์จะทำงานได้เร็วกว่าการใช้ซอฟแวร์ จากตัวอย่างเมื่อนำฟังก์ชั่นที่ใช้ในการคำนวณมาทำการออกแบบเป็นฮาร์ดแวร์ ดังแสดงในรูปที่ 2.2 FPGA จะใช้ความสามารถของการประมวลผลแบบขนานเพื่อช่วยลดเวลาที่ใช้ในการคำนวณลง



(a) การทำงานของซอฟแวร์ **(b)** การทำงานของฮาร์ดแวร์

รูปที่ 2.2 ลำดับการทำงานของซอฟแวร์เปรียบเทียบกับการทำงานของฮาร์ดแวร์

ลักษณะการทำงานดังแสดงในรูปที่ 2.2 (a) จะทำงานเป็นลำดับขั้นคือ จะทำการเก็บค่าไว้ในรีจิสเตอร์ก่อนจากนั้นจึงทำการคำนวณและทำเช่นนี้ไปเรื่อยๆ จนได้ค่าผลลัพธ์ ซึ่งจะแตกต่างจากในรูปที่ 2.2 (b) คือ เมื่อทำการใส่ข้อมูลอินพุทเข้าไปครบทุกจำนวนจะได้ผลลัพธ์ออกมาทันที

2.2.2 เปรียบเทียบการทำงานระหว่าง FPGA กับ DSP

DSP เป็นอุปกรณ์ที่มีลักษณะการทำงานคล้ายกับ CPU แต่ฟังก์ชันที่ถูกบรรจุใน DSP ส่วนใหญ่จะเป็นฟังก์ชันที่ใช้ในการประมวลผลสัญญาณที่เกี่ยวข้องกับการคำนวณ แต่ FPGA ก็สามารถทำงานในลักษณะเช่นเดียวกับ DSP ได้ แต่จะมีความซับซ้อนในการออกแบบมากกว่า การเปรียบเทียบคุณสมบัติต่างๆ ระหว่าง FPGA กับ DSP ดังแสดงในตารางที่ 2.1

ตารางที่ 2.1 แสดงการเปรียบเทียบคุณสมบัติต่างๆระหว่าง FPGA กับ DSP

Function	Industry's Faster DSP Processor Core	Xilinx Vertex-II Pro Platform
8x8 Multiply Accumulate (MAC)	4.8 Billion MAC/s fclk = 600 MHz	1 Trillion MAC/s fclk = 300 MHz
FIR Filter - 56 Tap, Liner Phase - 16 Bit Data/Coefficients	9.3 MSPS fclk = 600 MHz	300 MSPS fclk = 300 MHz
Complex FFT - 1024 Point, 16-Bit Data	10 µs fclk = 600 MHz	1 µs fclk = 150 MHZ
Reed-Solomon Decoding Throughput	4.1 Mbps fclk = 600 MHz	10 Gbps fclk = 85 MHz

2.2.3 ข้อดีข้อเสียของ FPGA

ข้อดีของ FPGA ประกอบไปด้วย

1) ความเร็วของไมโครคอนโทรลเลอร์เมื่อนำมาใช้งานโดยทั่วไปจะทำงานที่ความเร็วน้อยกว่า 50MHz แต่ชิพวงจรรวมสามารถทำงานได้สูงกว่า 200 MHz จึงเหมาะกับการ ใช้งานร่วมกับการประมวลสัญญาณดิจิตอล (Digital Signal Processing)
2) วงจรมีขนาดเล็ก การออกแบบวงจรดิจิตอลทั่วๆไปมีขนาดใหญ่ใช้พื้นที่มากกว่า เพราะต้องมีไอซีที่ใช้ประกอบเป็นวงจรเป็นจำนวนมาก แต่ชิพวงจรรวม FPGA สามารถออกแบบวงจรทั้งหมดโดยเรียกใช้สัญลักษณ์หรือการเขียนวงจรในรูปด้วยภาษาคอมพิวเตอร์ แล้วทำการโปรแกรมลงในชิพเพียงตัวเดียวซึ่งทำให้วงจรมีขนาดเล็กลง
3) การตรวจสอบความถูกต้องและการแก้ไขวงจรดิจิตอลในอดีต เมื่อต้องการตรวจสอบความถูกต้องของวงจรจำเป็นจะต้องใช้ลอจิกอะนาไลเซอร์ (Logic Analyzer) ที่มีราคาแพง และเมื่อเกิดความผิดพลาดขึ้นกับการทำงานของวงจรอาจจำเป็นต้องทำการออกแบบและสร้างวงจรขึ้นมาใหม่ทำให้ค่าใช้จ่ายสูงขึ้นด้วย แต่การตรวจสอบการทำงานของชิพวงจรรวม FPGA สามารถทำได้โดยใช้โปรแกรมจำลองการทำงานของวงจรก่อนที่จะทำการโปรแกรมลงชิพ และเมื่อโปรแกรมที่ใช้ทำงานผิดพลาดหรือต้องการแก้ไขบางส่วน ก็สามารถทำได้โดยง่ายเพียงแค่ทำการแก้ไขตัวโปรแกรมแล้วทำการโปรแกรมใหม่ จึงไม่จำเป็นที่จะต้องทำการแก้ไขในส่วนของแผ่นวงจรแต่อย่างใด

ข้อเสียของ FPGA ประกอบไปด้วย

1) ชิพวงจรรวม FPGA ไม่เหมาะแก่การนำไปใช้งานเป็นวงจรดิจิตอลขนาดเล็ก เพราะว่าจะเสียค่าใช้จ่ายสูงกว่าการใช้ไอซีสำเร็จรูป
2) ชิพวงจรรวม FPGA ไม่มีฟังก์ชันสำเร็จรูปที่ใช้ในการคำนวณเกี่ยวกับเลขจำนวนจริง ดังนั้นถ้าต้องการที่จะนำ FPGA ไปใช้ในการคำนวณเลขจำนวนจริงจำเป็นจะต้องทำการออกแบบฟังก์ชันดังกล่าวขึ้นมาเอง
3) ชิพวงจรรวม FPGA ไม่เหมาะที่จะนำไปออกแบบวงจรที่มีการทำงานเป็นลำดับขั้นทั้งหมด (Sequential Algorithms) เพราะ FPGA จะทำงานได้ช้ากว่า CPU หรือ DSP เนื่องจาก CPU และ DSP สามารถทำงานได้ที่สัญญาณนาฬิกาสูงระดับ GHz แต่ FPGA ที่ทำงานได้ในระดับ MHz

2.3 งานวิจัยที่ได้นำการแปลงเวฟเล็ตมาทำการสร้างบนฮาร์ดแวร์

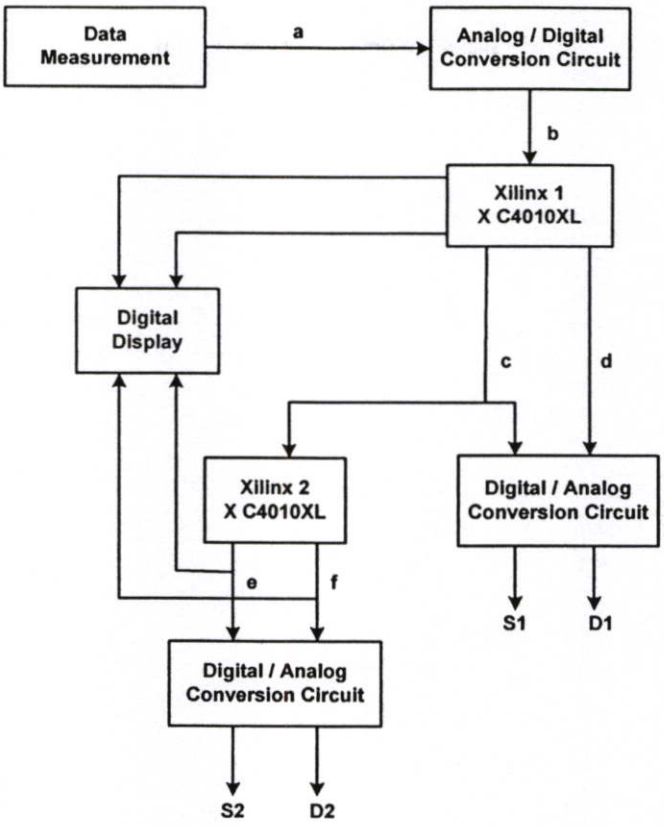
หัวข้อนี้จะนำเสนองานวิจัยที่เกี่ยวข้องกับการนำ FPGA มาใช้ในการออกแบบการแปลงเวฟเล็ตโดยใช้อัลกอริทึมแบบต่างๆ เพื่อนำมาใช้ในการพิจารณาการออกแบบโครงสร้างและการทำงานของโมดูลที่ใช้ในการแปลงเวฟเล็ต งานวิจัยนี้ได้นำเสนอต่อไป

2.3.1 งานวิจัยของ S.J. Huang, J.T. Huang, and T.M. [1]

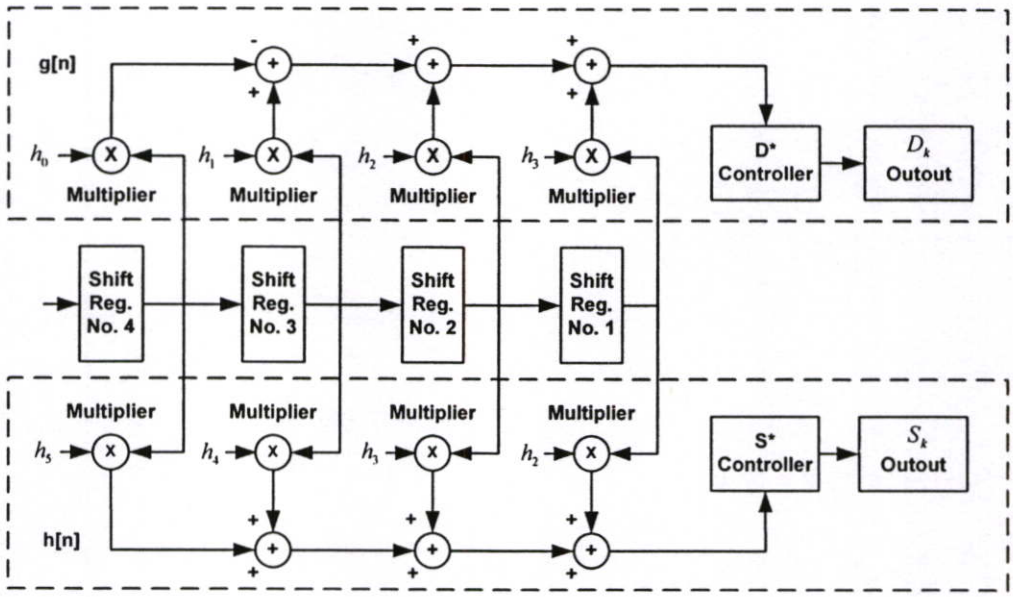
งานวิจัยนี้จะทำออกแบบการแปลงเวฟเล็ตแบบ Daubechies6 (D6) โดยใช้ FPGA เพื่อนำไปใช้ในการตรวจสอบความผิดเพี้ยนของสัญญาณไฟฟ้า วงจรที่ใช้ในการแปลงเวฟเล็ตจะใช้ FPGA เบอร์ XC4010XL ซึ่งเป็นไอซีที่ถูกพัฒนาโดยบริษัท Xilinx จำนวนสองตัว โครงสร้างภายในประกอบไปด้วยวงจรที่ใช้ในการคำนวณที่เป็นแบบเลขจำนวนเต็มหาค่าผลลัพธ์ที่จุด c และ d จากนั้นก็นำค่าผลลัพธ์ที่ได้ส่งต่อไปเพื่อคำนวณผลลัพธ์ที่จุด e และ f ดังแสดงในรูปที่ 2.3

ส่วนในรูปที่ 2.4 แสดงถึงโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ต จากโครงสร้างดังกล่าวประกอบไปด้วย ชิฟรีจิสเตอร์ (Shift Register) จำนวน 4 ชุด เพื่อใช้สำหรับการเก็บข้อมูลแล้วนำข้อมูลเหล่านี้ไปใช้ในการคำนวณหาค่าสัมประสิทธิ์ในการแปลงเวฟเล็ต ส่วนวงจรที่ใช้ในการคำนวณประกอบไปด้วยวงจรบวก วงจรลบ และวงจรคูณ

ลักษณะการทำงานของวงจรในรูปที่ 2.4 จะเหมือนกับการคำนวณแมทริกซ์ เพื่อหาค่าสัมประสิทธิ์การแปลงเวฟเล็ตดังแสดงในรูปที่ 2.5



รูปที่ 2.3 โครงสร้างของระบบที่งานวิจัยของ Shyh-Jier Huang



รูปที่ 2.4 โครงสร้างการแปลงเวฟเล็ต

$$\begin{bmatrix} S_1 \\ D_1 \\ S_2 \\ D_2 \\ S_3 \\ D_3 \\ S_4 \\ D_4 \\ S_5 \\ D_5 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & h_4 & h_5 & 0 & 0 & 0 & 0 \\ h_5 & -h_4 & h_3 & -h_2 & h_1 & -h_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & h_4 & h_5 & 0 & 0 \\ 0 & 0 & h_5 & -h_4 & h_3 & -h_2 & h_1 & -h_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 & h_4 & h_5 \\ 0 & 0 & 0 & 0 & h_5 & -h_4 & h_3 & -h_2 & h_1 & -h_0 \\ h_4 & h_5 & 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ h_1 & -h_0 & 0 & 0 & 0 & 0 & h_5 & -h_4 & h_3 & -h_2 \\ h_2 & h_3 & h_4 & h_5 & 0 & 0 & 0 & 0 & h_0 & h_1 \\ h_3 & -h_2 & h_1 & -h_0 & 0 & 0 & 0 & 0 & h_5 & -h_4 \end{bmatrix} \bullet \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \\ G_6 \\ G_7 \\ G_8 \\ G_9 \\ G_{10} \end{bmatrix}$$

(a) แสดงเมตริกซ์ที่ใช้ในการแปลงเวฟเล็ตที่งานวิจัยนี้นำเสนอ

$h_0 = 0.230377813308$
$h_1 = 0.806891509311$
$h_2 = 0.459877502118$
$h_3 = -0.135011020010$
$h_4 = -0.085441273882$
$h_5 = 0.035226291885$

(b) แสดงค่าสัมประสิทธิ์ต่างๆ ที่ใช้ในการแปลงเวฟเล็ต

รูปที่ 2.5 แสดงการคำนวณหาค่าสัมประสิทธิ์การแปลงเวฟเล็ตในรูปของแมทริกซ์

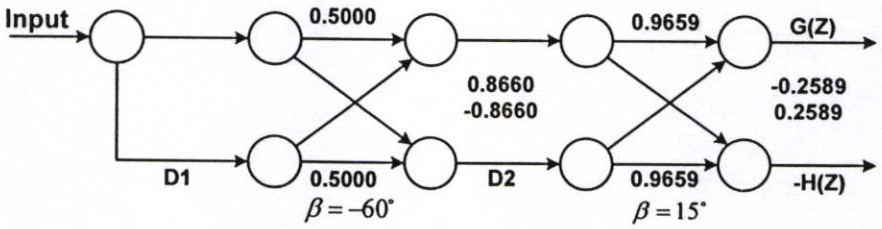
ทรัพยากรต่างๆของ FPGA ที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตของงานวิจัยนี้ดังแสดงในตารางที่ 2.2

ตารางที่ 2.2 แสดงทรัพยากรที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตใน

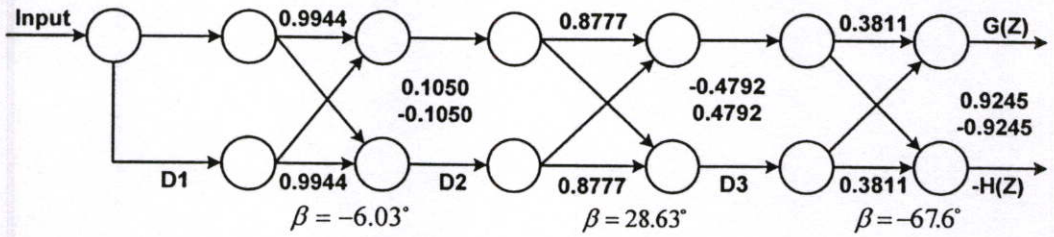
Xilinx 1			
CLB	IOB	Flip-Flops	Gate
400 (100%)	26 (40%)	75 (6.7%)	8184 (82%)
Xilinx 2			
CLB	IOB	Flip-Flops	Gate
400 (100%)	25 (48%)	73 (6.5%)	8160 (82%)

2.3.2 งานวิจัยของ W. Zhilu, and R. Guanghui [2]

งานวิจัยนี้เป็นการศึกษาเกี่ยวกับการแปลงเวฟเล็ตโดยใช้ FPGA และใช้การแปลงเวฟเล็ตแบบ Daubechies4 (D4) & Daubechies6 (D6) ในการออกแบบใช้หลักการของ Lattice-Filter-Bank โครงสร้างของการแปลงเวฟเล็ตแบบ Daubechies4 (D4) & Daubechies6 (D6) ดังแสดงในรูปที่ 2.6 และ 2.7



รูปที่ 2.6 โครงสร้างการแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้ Lattice Filter



รูปที่ 2.7 โครงสร้างการแปลงเวฟเล็ตแบบ Daubechies6 (D6) โดยใช้ Lattice Filter

จากรูปที่ 2.6 และรูปที่ 2.7 จะแสดงโครงสร้างสมบูรณ์ที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) & Daubechies6 (D6) ตามลำดับ โครงสร้างภายในประกอบไปด้วยวงจรบวก วงจรลบ และวงจรคูณ แต่เนื่องจากวงจรคูณจะมีความซับซ้อนและค่าหน่วงเวลาสูง ดังนั้นจึงทำการเปลี่ยนวงจรที่ใช้ในการคูณมาอยู่ในรูปของการบวก ลบ และการเลื่อนบิตข้อมูล เพื่อให้การออกแบบฮาร์ดแวร์ทำได้ง่าย และมีขนาดเล็กลง ตัวอย่างเช่น $0.5000 = 2^{-1}$, $0.8660 = 2^{0} - 2^{-3}$ $0.9659 = 2^{0} - 2^{-5}$, $0.2589 = 2^{-2} + 2^{-4}$ ซึ่งจะเห็นได้ว่าค่าสัมประสิทธิ์เหล่านี้จะถูกปรับให้อยู่ในรูปของสองยกกำลัง ทำให้สามารถทำการแปลงวงจรที่ใช้ในการคูณให้อยู่ในรูปของการเลื่อนบิตข้อมูล การบวก และการลบได้

งานวิจัยดังกล่าวจะทำการตรวจสอบการทำงานจากโปรแกรม MAX+PLUSII ซึ่งเป็นโปรแกรมจำลองการทำงาน และชิพ FPGA FLEX10K ของบริษัท ALTERA การตรวจสอบการทำงานข้อมูลอินพุตและเอาท์พุทจะถูกปรับให้อยู่ในรูปโครงสร้างข้อมูลแบบ 16 บิตที่ประกอบไปด้วย 3 ส่วนคือ ส่วนเครื่องหมาย 1 บิต ข้อมูลขนาด 7 บิต และส่วนทศนิยมขนาด 8 บิต ผลลัพธ์ที่ได้การคำนวณและจากการจำลองการทำงาน แสดงในตารางที่ 2.3 และ 2.4

ตารางที่ 2.3 แสดงผลลัพธ์ที่ได้จากการแปลงเวฟเล็ตแบบ Daubechies4 (D4)

ข้อมูลที่นำมาทดสอบ	4	8	16	32	32	16	8	4	2	1
ผลที่ได้จากการคำนวณ	1	7	15	30	44	39	20	8	4	2
ผลที่ได้จากการจำลองการทำงาน	2	7	15	30	44	38	19	7	3	1

ตารางที่ 2.4 แสดงผลลัพธ์ที่ได้จากการแปลงเวฟเล็ตแบบ Daubechies6 (D6)

ข้อมูลที่นำมาทดสอบ	1	2	3	4	5	6	7	8	9	10
ผลที่ได้จากการคำนวณ	0	1	3	4	6	7	8	10	11	13
ผลที่ได้จากการจำลองการทำงาน	0	1	3	4	6	7	9	10	12	10

บทที่ 3
ความรู้พื้นฐานในเรื่องที่เกี่ยวข้องกับงานวิจัย

ในบทนี้จะกล่าวถึงความรู้พื้นฐานต่างๆ ที่เกี่ยวข้องกับงานวิจัย ได้แก่ การแปลงเวฟเล็ต การทดสอบประสิทธิภาพของเวฟเล็ต วิธีการแปลงสมการที่ใช้ในการแปลงเวฟเล็ตจากเลขจำนวนจริงไปเป็นเลขจำนวนเต็ม และส่วนสุดท้ายจะกล่าวถึงระบบการแทนเลขจำนวนจริง

3.1 การแปลงเวฟเล็ต (Wavelet Transform)

การวิเคราะห์เวฟเล็ตได้ถูกคิดค้นและพัฒนาจากนักคณิตศาสตร์ในปี ค.ศ. 1980 [3] หลังจากนั้นได้ถูกนำมาใช้งานกันอย่างแพร่หลายในปี 1990 เพราะเวฟเล็ตสามารถที่ใช้ในการวิเคราะห์สัญญาณต่างๆ หรือแม้กระทั่งการวิเคราะห์ภาพหรือนำไปใช้ในการบีบอัดข้อมูล ข้อมูลที่ได้จากการแปลงเวฟเล็ตจะถูกจัดเรียงใหม่ โดยทำการแยกค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่สูงและความถี่ต่ำออกจากกัน ค่าสัมประสิทธิ์ที่ได้จากการแปลงเวฟเล็ตทั้งหมดจะมีค่าเท่ากับข้อมูลอินพุท ตัวอย่างการนำเวฟเล็ตไปประยุกต์ใช้งานเช่น การเก็บข้อมูลลายนิ้วมือของ FBI [4]

3.1.1 การเลือกเวฟเล็ตเพื่อนำไปใช้งาน

การเลือกเวฟเล็ตเพื่อนำไปใช้งานจะต้องพิจารณาจากหลายๆด้าน เพื่อให้เหมาะกับงานที่จะนำไปใช้ ไม่ว่าจะเป็นฟังก์ชั่นที่ใช้การแปลงไป (Decomposition) และแปลงกลับ (Reconstruction) ค่า PSNR หรือแม้แต่ค่า Zero Moment ถ้าหากพิจารณาจากฟังก์ชั่นที่ใช้ในการแปลงเวฟเล็ต จะต้องคำนึงถึงความซับซ้อนในการคำนวณ และอัตราการสูญเสียของข้อมูล ถ้าหากต้องการความถูกต้องของข้อมูลจะทำให้มีการคำนวณที่ซับซ้อน เนื่องจากค่าสัมประสิทธิ์ที่ได้จากการคำนวณจะอยู่ในรูปของจำนวนจริง (Floating Point) แต่หากต้องการลดความซับซ้อนในการคำนวณลง โดยยอมให้ข้อมูลเกิดการสูญเสียได้ ก็สามารถใช้การคำนวณหาค่าสัมประสิทธิ์แบบเลขจำนวนเต็มได้ (Integer Wavelet)

ค่า PSNR (Peek Signal to Noise Ratio) เป็นค่าที่ใช้ในขั้นตอนการวัดประสิทธิภาพของอัลกอริธึม จะดูจากคุณภาพของภาพที่ได้เป็นหลัก ค่าที่ PSNR หาได้จากสมการที่ (3.1)

$$PSNR = 20\log_{10}\left(\frac{A}{RMSE}\right) \tag{3.1}$$

โดยที่ ค่า RMSE หาได้จากสมการที่ (3.2)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(p(i) - p'(i)\right)^2} \tag{3.2}$$

เมื่อ A คือ จำนวน Gray level

N คือ จำนวน Pixel ในภาพ

$p(i)$ คือ ค่าจุดในภาพต้นฉบับ

$p'(i)$ คือ ค่าจุดในภาพที่ผ่านการเข้ารหัส

ค่า Zero Moment เป็นการนับจำนวนของข้อมูลที่มีค่าเป็นศูนย์ หรือข้อมูลที่ต่ำกว่าระดับที่ตั้งเอาไว้ (Threshold) ของข้อมูลทั้งหมดที่ได้จากการแปลงเวฟเล็ต (Wavelet Transform) ค่า Zero Moment จะใช้ในการวัดประสิทธิภาพของการบีบอัดข้อมูล ค่า Zero Moment หาได้จากสมการ (3.3)

$$Zero\ \ Moment = \sum_{x=1}^{M}\sum_{y=1}^{N} Z_{coef}(x,y) \tag{3.3}$$

โดยที่ Z_{coef} หาได้จากสมการที่ (3.4)

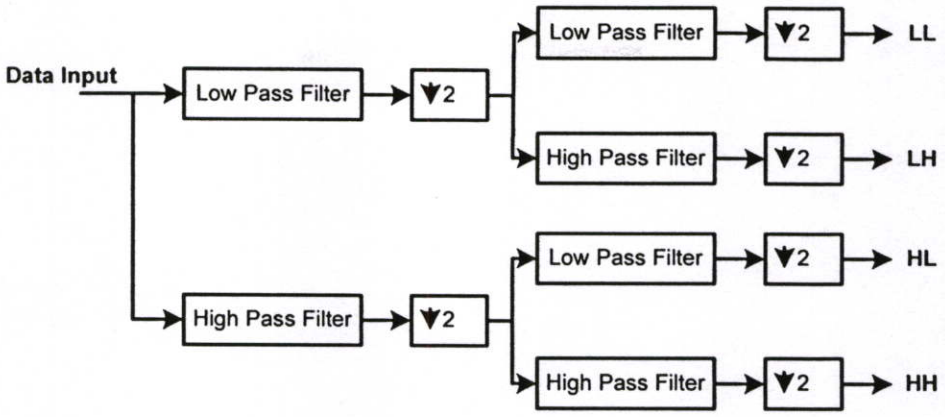
$$Z_{coef}(x,y) = \begin{cases} 1 & |w(x,y)| \le Threshold \\ 0 & Otherwise \end{cases} \tag{3.4}$$

เมื่อ $w(x,y)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงเวฟเล็ตที่ตำแหน่งต่างๆ

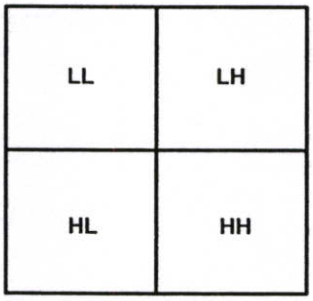
$Threshold$ คือ ค่าระดับของข้อมูลที่ตั้งเอาไว้ (ในที่นี้ใช้ 1.5)

3.1.2 การแปลงเวฟเล็ตกับข้อมูลภาพ

ข้อมูลภาพมีลักษณะเป็นข้อมูล 2 มิติ ดังนั้นการทำการแปลงเวฟเล็ตกับภาพจะเป็นการทำการแปลงเวฟเล็ตในทางแนวแกน x และแกน y สลับกันในแต่ละครั้ง สามารถเขียนเป็นแผนผังได้ดังรูปที่ 3.1



รูปที่ 3.1 แสดงแผนผังของการแปลงเวฟเล็ต



รูปที่ 3.2 แสดงลักษณะของการแบ่งแบนด์ย่อยของภาพ

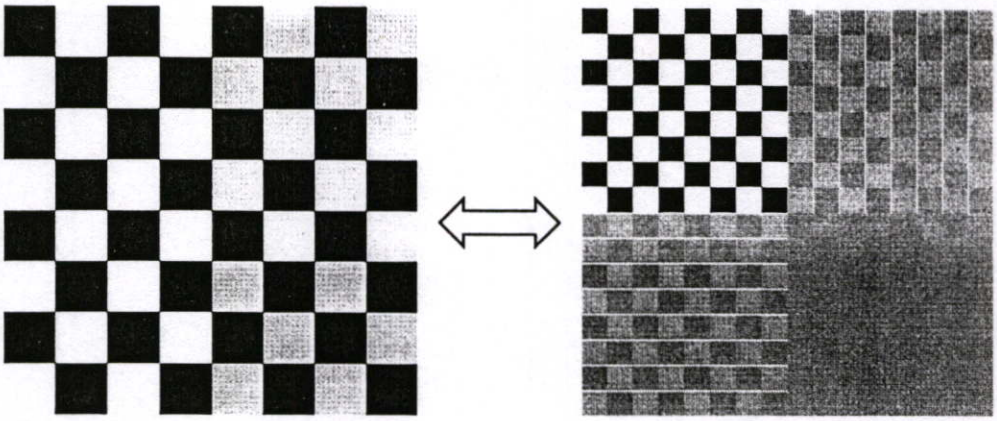
ลักษณะของข้อมูลที่ได้จากการแปลงเวฟเล็ตแบบ 2 มิติ จะถูกแบ่งออกเป็น 4 แบนด์ย่อย ภาพที่ได้จะมีลักษณะดังรูปที่ 3.2 โดยแต่ละแบนด์ย่อยจะมีคุณสมบัติดังนี้

LL : คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากการผ่านตัวกรองความถี่ต่ำสองครั้งและรายละเอียดของข้อมูลภาพส่วนใหญ่จะอยู่บริเวณนี้

LH : คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่ต่ำในแนวตั้งแล้วนำผลลัพธ์ที่ได้ไปผ่านตัวกรองความถี่สูงในแนวนอนส่วนนี้จะทำการเก็บข้อมูลในแนวตั้ง

HL : คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่สูงในแนวตั้งแล้วนำผลลัพธ์ที่ได้ไปผ่านตัวกรองความถี่ต่ำในแนวนอนส่วนนี้จะทำการเก็บข้อมูลในแนวนอน

HH : คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากการผ่านตัวกรองความถี่สูงสองครั้งในส่วนนี้จะเก็บข้อมูลในแนวทแยงมุม และส่วนนี้จะมีความสำคัญน้อยที่สุด



(a) (b)

รูปที่ 3.3 แสดงภาพตัวอย่างการแปลงเวฟเล็ต (a) ภาพต้นฉบับ (b) ภาพที่ได้จากแปลงเวฟเล็ต

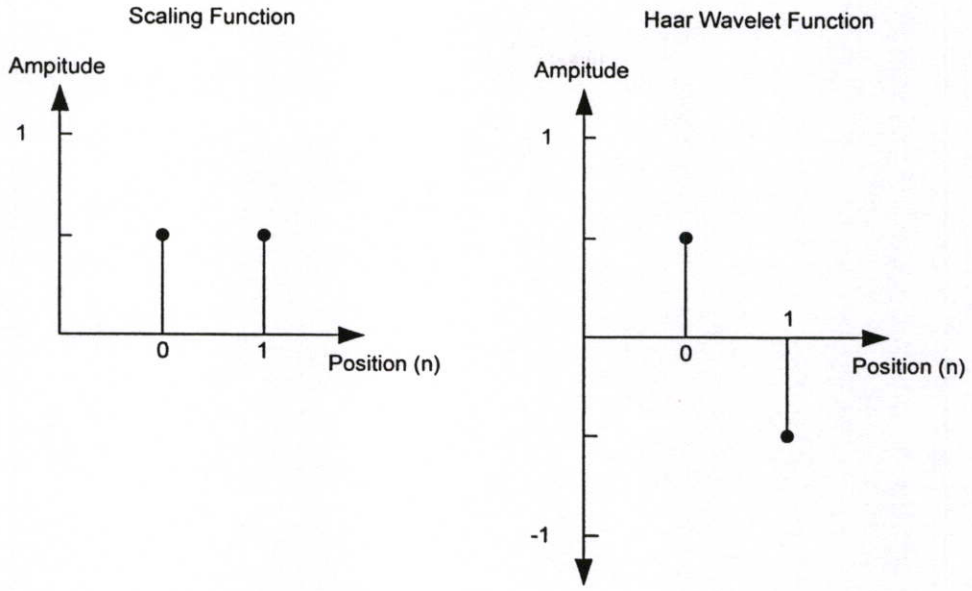
เพื่อให้ง่ายต่อการทำความเข้าใจ จึงยกตัวอย่างการแปลงเวฟเล็ตแบบ Haar [5] เพราะการแปลงเวฟเล็ตวิธีนี้ เป็นวิธีที่สามารถทำความเข้าใจได้ง่าย เพราะว่ามีการคำนวณที่ไม่ซับซ้อน

สมการที่ใช้ในการแปลงเวฟเล็ตโดยทั่วไปสามารถที่จะแบ่งออกเป็น 2 ส่วน คือ Scaling Function และ Wavelet Function ดังสมการที่ 3.5 และ 3.6 ตามลำดับ

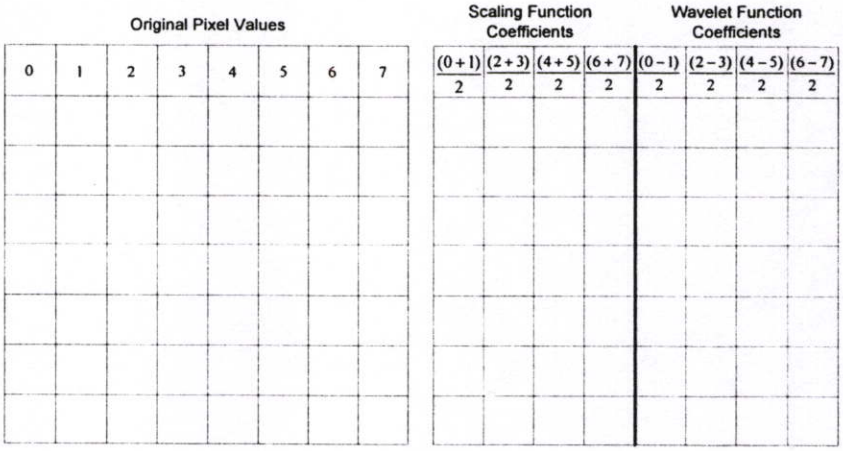
$$h_n = \begin{cases} \frac{1}{2} & n = 0,1 \\ 0 & Otherwise \end{cases} \tag{3.5}$$

$$g_n = \begin{cases} \frac{1}{2} & n = 0 \\ -\frac{1}{2} & n = 1 \\ 0 & Otherwise \end{cases} \tag{3.6}$$

เมื่อ h_n คือ Scaling Function และ g_n คือ Wavelet Function แบบ Haar ถ้านำสมการทั้งสองไปเขียนเป็นกราฟจะมีลักษณะดังแสดงในรูปที่ 3.4



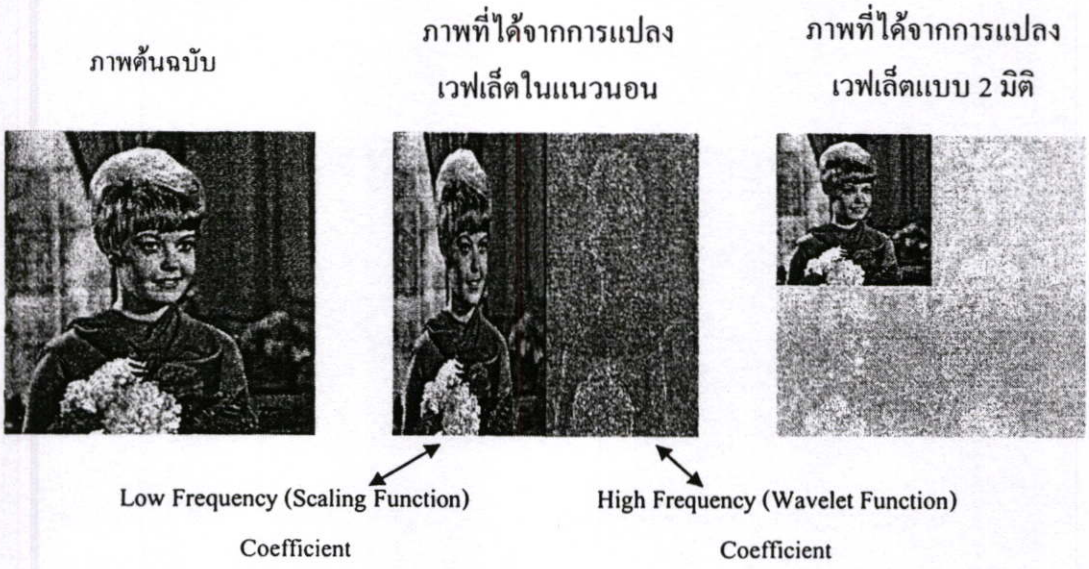
รูปที่ 3.4 แสดงรูปสัญญาณที่ใช้ในการแปลงเวฟเล็ตแบบ Haar



รูปที่ 3.5 แสดงการนำ Haar เวฟเล็ตไปประยุกต์ใช้งานกับข้อมูลที่เป็นภาพ

การประยุกต์ใช้ Haar เวฟเล็ตฟังก์ชันกับข้อมูลที่เป็นภาพ สามารถแสดงในรูปที่ 3.5 ซึ่งเป็นการแปลงเวฟเล็ตในแนวนอน (Horizontal) ค่าสัมประสิทธิ์ที่ได้จากการแปลงจะประกอบด้วย ค่าสัมประสิทธิ์ที่ได้จาก Scaling Function ซึ่งก็คือ ค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่ต่ำ (Low-Frequency Coefficient) และค่าสัมประสิทธิ์ของ Wavelet Function คือ ค่าสัมประสิทธิ์ที่ได้ตัวกรองความถี่สูง (High-Frequency Coefficient) การหาค่าสัมประสิทธิ์ของการแปลงเวฟเล็ตแบบ Haar ค่าที่ได้จาก Scaling Function คือ ค่าเฉลี่ยของผลรวมของสองพิกเซล ส่วนสัมประสิทธิ์ที่ได้จากของเวฟเล็ต ฟังก์ชัน จะเป็นการหาค่าเฉลี่ยของความแตกต่างของสองพิกเซล เมื่อทำการแปลงเวฟเล็ตไป

ประยุกต์ใช้กับข้อมูลภาพจะทำการแปลงเวฟเล็ตในแนวนอนและแนวตั้ง โดยแสดงผลลัพธ์ที่ได้ใน รูปที่ 3.6



รูปที่ 3.6 ภาพผลลัพธ์ที่ได้จากการแปลงเวฟเล็ตของข้อมูลภาพ

เมื่อทำความเข้าใจเกี่ยวกับการแปลงเวฟเล็ตโดยใช้ฟังก์ชัน Haar เป็นที่เรียบร้อยแล้วในหัวข้อถัดไปจะกล่าวถึงการแปลงเวฟเล็ตที่มีความซับซ้อนมากขึ้นคือ Daubechies4 (D4) ซึ่งเป็นวิธีที่นิยมใช้ในการบีบอัดข้อมูล หรือใช้ในการวิเคราะห์ภาพ

3.1.3 การแปลงเวฟเล็ตแบบ Daubechies4 (D4) [6]

การแปลงเวฟเล็ตแบบ Daubechies4 (D4) จะมีฟิวเตอร์ทั้งหมด 4 แท็ป (4 Tap Filters) ซึ่งประกอบไปด้วยค่า Scaling Function และ Wavelet Function จำนวน 4 ค่า และค่าสัมประสิทธิ์ของ Scaling Function ประกอบไปด้วยค่าตามสมการ 3.7 ถึง 3.10

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}} \tag{3.7}$$

$$h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}} \tag{3.8}$$

$$h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}} \tag{3.9}$$

$$h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}} \tag{3.10}$$

Wavelet Function จะประกอบไปด้วยค่าตามสมการ 3.11 ถึง 3.14

$$g_0 = h_3 \tag{3.11}$$

$$g_1 = -h_2 \tag{3.12}$$

$$g_2 = h_1 \tag{3.13}$$

$$g_3 = -h_0 \tag{3.14}$$

ในขั้นตอนการแปลงเวฟเล็ตจะใช้ Scaling Function และ Wavelet Function กับข้อมูลอินพุท ถ้าข้อมูลอินพุทมีจำนวนเท่ากับ N ค่า ต้องใช้ Scaling Function และ Wavelet Function ในการแปลงเวฟเล็ตจำนวน $N/2$ ครั้ง ค่าสัมประสิทธิ์ที่ได้จาก Scaling Function จะถูกเก็บไว้ในครึ่งแรก (ค่าเริ่มต้นถึง $N/2$) และค่าสัมประสิทธิ์ที่ได้จาก Wavelet Function จะถูกเก็บไว้ในครึ่งหลัง (จาก $N/2$ ถึง N)

ในการคำนวณหาค่าสัมประสิทธิ์ของ Scaling และ Wavelet Function แต่ละครั้งจะต้องการข้อมูลอินพุต 4 จำนวน ซึ่งสามารถเขียนเป็นสมการได้ดังต่อไปนี้

Daubechies4 Scaling Function

$$a[i] = h_o S[2i] + h_1 S[2i+1] + h_2 S[2i+2] + h_3 S[2i+3] \tag{3.15}$$

Daubechies4 Wavelet Function

$$c[i] = g_o S[2i] + g_1 S[2i+1] + g_2 S[2i+2] + g_3 S[2i+3] \tag{3.16}$$

ในการแปลงเวฟเล็ตของข้อมูลที่มีขนาดจำกัด จะทำการคำนวณได้จนถึงตำแหน่งที่ $N-4$ เพราะเมื่อทำการคำนวณต่อไปจะมีปัญหาเนื่องจากข้อมูลที่ใช้ในการคำนวณประกอบด้วย $S[N-2], S[N-1], S[N], S[N+1]$ แต่ $S[N]$ และ $S[N+1]$ หาค่าไม่ได้ (ทั้งสองตำแหน่งนี้เป็นตำแหน่งที่มีค่ามากกว่าอาร์เรย์) ปัญหาที่เกิดขึ้นนี้สามารถแสดงให้อยู่ในรูปของเมตริกซ์ (3.17) ได้ดังนี้

Daubechies D4 forward transform matrix for an 8 element signal

$$\begin{bmatrix} a_0 \\ c_0 \\ a_1 \\ c_1 \\ a_2 \\ c_2 \\ a_3 \\ c_3 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 \end{bmatrix} \bullet \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \end{bmatrix} \quad (3.17)$$

การแปลงข้อมูลกลับคืนมา(Inverse Transform) จะเกิดปัญหาคล้ายกับการแปลงเวฟเล็ต แต่ปัญหาดังกล่าวจะเกิดกับค่าเริ่มต้นสองค่าคือ ในการคำนวณค่าสัมประสิทธิ์ครั้งแรกจะใช้ค่าข้อมูล $a[-1], c[-1], a[0]$ และ $c[0]$ ดังแสดงในเมตริกซ์ (3.18) ดังต่อไปนี้

Daubechies D4 inverse transform matrix for an 8 element transform result

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \end{bmatrix} = \begin{bmatrix} h_2 & g_1 & h_0 & g_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_3 & g_3 & h_1 & g_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_2 & g_2 & h_0 & g_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_3 & g_3 & h_1 & g_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_2 & g_2 & h_0 & g_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_3 & g_3 & h_1 & g_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_2 & g_2 & h_0 & g_0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_3 & g_3 & h_1 & g_1 \end{bmatrix} \bullet \begin{bmatrix} a_0 \\ c_0 \\ a_1 \\ c_1 \\ a_2 \\ c_2 \\ a_3 \\ c_3 \end{bmatrix} \quad (3.18)$$

ปัญหาดังกล่าวนี้สามารถแก้ไขได้ 3 วิธี

1. แทนที่ข้อมูลที่หายไปด้วยค่ามูลตัวสุดท้ายหรือข้อมูลตัวก่อนหน้า
2. ทำการแทนข้อมูลที่หายไปในลักษณะของทรงกระบอก ตัวอย่างเช่น ข้อมูลลำดับที่ -1 ก็จะนำข้อมูลตัวที่ N มาใช้ในการคำนวณ
3. ทำการออกแบบ Scaling Function และ Wavelet Function พิเศษที่ใช้ในการคำนวณบริเวณขอบของชุดข้อมูล

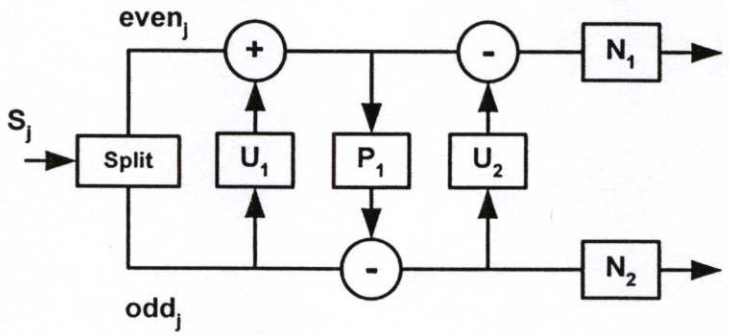
หมายเหตุ: การนำข้อมูลที่มีค่าเป็นศูนย์มาแทนที่ข้อมูลที่หายไปก็สามารถทำได้แต่ว่าอาจทำให้เกิดการสูญเสียมากกว่าวิธีอื่น

3.1.4 การแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธีการของ Lifting [6], [7]

การแปลงเวฟเล็ตแบบ Lifting ได้พัฒนาโดย Wim Sweldens [7] แนวคิดของการแปลงเวฟเล็ตแบบ Lifting มีประโยชน์เกี่ยวกับการจัดการหน่วยความจำเพราะว่าการแปลงเวฟเล็ตในลักษณะนี้จะไม่ต้องการหน่วยความจำสำหรับพักข้อมูล การแปลงเวฟเล็ตแบบ Daubechies4 (D4) ที่จะนำเสนอต่อไปจะมีลักษณะการทำงานดังแสดงในบล็อกไดอะแกรมรูปที่ 3.7 และขบวนการแปลงข้อมูลกลับคืนมา จะใช้การกลับสถานะของขบวนการแปลงเวฟเล็ต ทำได้โดยการสลับเครื่องหมาย รายละเอียดในส่วนนี้จะได้กล่าวถึงในหัวข้อถัดไป

3.1.4.1 การแปลงเวฟเล็ต (Forward Transform)

การแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้วิธีการของ Lifting จะประกอบไปด้วยขั้นตอนการ Update และ Predict ในขั้นตอนของการแปลงเวฟเล็ตจะต้องมีการปรับค่าผลลัพธ์ (Normalization) ของการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธีการของ Lifting ดังที่ได้แสดงดังรูปที่ 3.7



รูปที่ 3.7 การแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธีการของ Lifting

ขั้นตอน Split จะทำหน้าที่แยกข้อมูลอินพุทที่เข้ามา ลำดับที่เป็นเลขคู่ (odd) จะถูกเก็บไว้ในอาร์เรย์ส่วนแรก (S_0 ถึง S_{half-1}) และลำดับที่เป็นเลขคี่ (even) จะถูกเก็บไว้ในอาร์เรย์ครึ่งหลัง (S_{half} ถึง S_{N-1}) จากสมการที่ใช้ในการแปลงเวฟเล็ต (3.19) ถึง (3.25) โดยที่ $S[half+n]$คือ ลำดับที่เป็นเลขคี่ ส่วน $S[n]$หมายถึง ลำดับที่เป็นเลขคู่ ถึงแม้ว่าในไดอะแกรมรูปที่ 3.7 จะใช้ฟังก์ชันในการปรับค่า 2 ฟังก์ชัน แต่ทั้งสองฟังก์ชันจะเป็นส่วนกลับซึ่งกันและกัน

ฟังก์ชั่นที่ใช้ในการแปลงคือ

Update 1 (U1):

For n = 0 to half - 1

$$S[n] = S[n] + \sqrt{3}S[half + n] \tag{3.19}$$

Predict (P):

$$S[half] = s[half] - \frac{\sqrt{3}}{4}S[0] - \frac{\sqrt{3}-2}{4}S[half - 1] \tag{3.20}$$

For n = 1 to half -1

$$S[half + n] = s[half + n] - \frac{\sqrt{3}}{4}S[n] - \frac{\sqrt{3}-2}{4}S[n-1] \tag{3.21}$$

Update 2 (U2):

For n = 0 to half-2

$$S[n] = S[n] - S[half + n + 1] \tag{3.22}$$

$$S[half - 1] = S[half - 1] - S[half] \tag{3.23}$$

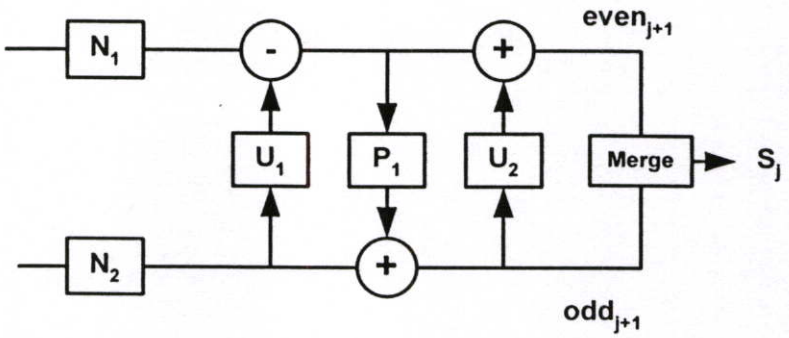
Normalize (N_1 และ N_2):

$$S[n] = \left[\frac{\sqrt{3}-1}{\sqrt{2}}\right]S[n] \tag{3.24}$$

$$S[n + half] = \left[\frac{\sqrt{3}+1}{\sqrt{2}}\right]S[n + half] \quad \text{หรือ} \quad \frac{S[n + half]}{\left[\frac{\sqrt{3}-1}{\sqrt{2}}\right]} \tag{3.25}$$

3.1.4.2 การแปลงกลับเวฟเล็ตแบบ Lifting (Inverse Transform)

ขบวนการในการแปลงข้อมูลกลับคืนมา จะมีลักษณะเดียวกับการแปลงเวฟเล็ต แต่จะทำการกลับสถานะของเครื่องหมายจากบวกเป็น ลบ และจากลบเป็นบวก ดังแสดงในรูปที่ 3.8



รูปที่ 3.8 แสดงการแปลงกลับของ Daubechies4 (D4) ด้วยวิธีการของ Lifting

หลังจากนั้นจะทำการรวมข้อมูลเข้าด้วยกัน (Merge) ระหว่างข้อมูลที่ได้จากลำดับที่เป็นคู่และลำดับที่เป็นคี่

ฟังก์ชันที่ใช้ในการแปลงกลับแบบ Lifting ดังแสดงในสมการ (3.26) ถึง (3.32)

Update 1':

For n = 0 to half - 1

$$S[n] = S[n] - \sqrt{3}S[half + n] \tag{3.26}$$

Predict':

$$S[half] = s[half] + \frac{\sqrt{3}}{4}S[0] + \frac{\sqrt{3}-2}{4}S[half - 1] \tag{3.27}$$

For n = 1 to half -1

$$S[half + n] = s[half + n] + \frac{\sqrt{3}}{4}S[n] + \frac{\sqrt{3}-2}{4}S[n-1] \tag{3.28}$$

Update 2':

For n = 0 to half-2

$$S[n] = S[n] + S[half + n + 1] \tag{3.29}$$

$$S[half - 1] = S[half - 1] + S[half] \tag{3.30}$$

Normalize':

$$S[n]=\left[\frac{\sqrt{3}+1}{\sqrt{2}}\right]S[n] \text{ หรือ } \frac{S[n]}{\left[\frac{\sqrt{3}+1}{\sqrt{2}}\right]} \tag{3.31}$$

$$S[n+half]=\left[\frac{\sqrt{3}-1}{\sqrt{2}}\right]S[n+half] \tag{3.32}$$

การแปลงเวฟเล็ตที่นำเสนอข้างต้นจะมีรูปแบบการคำนวณในลักษณะของเลขจำนวนจริงทั้งหมดทำให้สามารถกู้ข้อมูลกลับคืนมาได้เหมือนข้อมูลต้นฉบับ แต่ถ้าหากต้องการลดความซับซ้อนในการคำนวณลง และยอมให้เกิดการสูญเสียของข้อมูลได้ก็สามารถใช้การแปลงเวฟเล็ตแบบเลขจำนวนเต็มได้ (Integer Wavelet Transform) ในหัวข้อต่อไปจะอธิบายถึงการสร้างการแปลงเวฟเล็ตแบบเลขจำนวนเต็ม

3.1.5 การแปลงเวฟเล็ตแบบเลขจำนวนเต็ม (Integer Wavelet Transform) [8]

สมการที่ใช้ในการแปลงเวฟเล็ตทุกสมการ สามารถเขียนให้อยู่ในรูปของการคำนวณแบบเลขจำนวนเต็มได้ (Integer Wavelet Transform) โดยทำการปัดเศษ (Round-Off) ค่าสัมประสิทธิ์ที่ได้จากการคำนวณแต่ละขั้นตอนตามสมการ ก่อนที่จะทำการบวกหรือลบค่าสัมประสิทธิ์ต่างๆ เข้าด้วยกัน ดังแสดงสมการ (3.33) และ (3.34) เป็นสมการตัวอย่างที่ถูกแปลงให้อยู่ในรูปจำนวนเต็ม

$$d_{1,l}^{(i)} = d_{1,l}^{(i-1)} - \left\lfloor \sum_k p_k^i s_{1,l-k}^{(i-1)} + 1/2 \right\rfloor \tag{3.33}$$

$$s_{1,l}^{(i)} = s_{1,l}^{(i-1)} - \left\lfloor \sum_k u_k^i d_{1,l-k}^{(i-1)} + 1/2 \right\rfloor \tag{3.34}$$

ผลลัพธ์ที่ได้จากสมการทั้งสองนี้จะอยู่ในรูปของเลขจำนวนเต็ม และในการแปลงข้อมูลกลับคืนมาสามารถทำได้โดยกลับสมการที่ใช้ในการแปลงเวฟเล็ต เพื่อให้ง่ายต่อการทำความเข้าใจและการนำไปใช้งาน สามารถเขียนสมการที่ใช้ในการแปลงเวฟเล็ตแบบเลขจำนวนเต็มและแปลงกลับแบบเลขจำนวนเต็มให้อยู่ในรูปของ Pseudo-Code ดังนี้

Pseudo-Code ที่ใช้ในการแปลงเวฟเล็ต

$$
\begin{aligned}
& s_{1,l}^{(0)} := s_{0,2l} \\
& d_{1,l}^{(0)} := s_{0,2l+1} \\
& for \quad i = 1:(1):M \\
& \qquad \forall l \ : d_{1,l}^{(i)} := d_{1,l}^{(i-1)} - \left\lfloor \sum_k p_k^{(i)} s_{1,l-k}^{(i-1)} + 1/2 \right\rfloor \\
& \qquad \forall l \ : s_{1,l}^{(i)} := s_{1,l}^{(i-1)} - \left\lfloor \sum_k u_k^{(i)} d_{1,l-k}^{(i-1)} + 1/2 \right\rfloor \\
& end
\end{aligned}
$$

ส่วนของ Pseudo-Code ที่ใช้ในการแปลงกลับคือ

$$
\begin{aligned}
& for \quad i = M:(-1):1 \\
& \qquad \forall l \ : s_{1,l}^{(i-1)} := s_{1,l}^{(i)} + \left\lfloor \sum_k u_k^{(i)} d_{1,l-k}^{(i)} + 1/2 \right\rfloor \\
& \qquad \forall l \ : d_{1,l}^{(i-1)} := d_{1,l}^{(i)} + \left\lfloor \sum_k p_k^{(i)} s_{1,l-k}^{(i-1)} + 1/2 \right\rfloor \\
& end \\
& s_{0,2l+1} := d_{1,l}^{(0)} \\
& s_{0,2l} := s_{1,l}^{(0)}
\end{aligned}
$$

3.1.5.1 Integer Daubechies4 (D4) Matrix [9]

การแปลงค่าสัมประสิทธิ์ที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ให้เป็นการแปลงเวฟเล็ตแบบเลขจำนวนเต็ม ทำได้โดยการปรับค่าสัมประสิทธิ์ที่ใช้ในการ Scaling และ Wavelet Function ใหม่สามารถแสดงได้ดังนี้

ค่าสัมประสิทธิ์ Scaling Function ดังแสดงในสมการที่ (3.35) ถึง (3.38)

$$h_0 = -8 \tag{3.35}$$

$$h_1 = 14 \tag{3.36}$$

$$h_2 = 54 \tag{3.37}$$

$$h_3 = 31 \tag{3.38}$$

ค่าสัมประสิทธิ์ Wavelet Function ดังแสดงในสมการที่ (3.39) ถึง (3.40)

$$g_0 = h_3 \tag{3.39}$$

$$g_1 = -h_2 \tag{3.40}$$

$$g_2 = h_1 \tag{3.41}$$

$$g_3 = -h_0 \tag{3.42}$$

จะเห็นได้ว่าค่าสัมประสิทธิ์ทุกตัวที่ใช้ในการคำนวณจะเป็นเลขจำนวนเต็มทั้งหมดทำให้สามารถทำการคำนวณได้ง่าย แต่ว่าค่าที่ได้จากการคำนวณจะต้องทำการปรับค่าโดยจะถูกหารด้วย 2^6 นั้นก็คือ ทำการเลื่อนบิตข้อมูลของค่าสัมประสิทธิ์ที่ได้การแปลงเวฟเล็ตไปทางขวาจำนวน 6 ครั้ง

3.1.5.2 Integer Daubechies4 (D4) Lifting [10]

จากฟังก์ชั่นที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) Lifting สมการที่ (3.19) ถึง (3.25) ให้ทำการปัดค่าผลลัพธ์ที่ได้จากส่วนที่มีการดำเนินการเป็นเลขเป็นทศนิยมเพื่อให้เป็นเลขจำนวนเต็มก่อนที่จะทำการบวกหรือลบกับค่าอื่นๆ ต่อไป ดังแสดงในสมการที่ (3.43) ถึง (3.49)

Update 1 (U1):

For n = 0 to half - 1

$$S[n] = S[n] + \left\lfloor \sqrt{3} S[half + n] \right\rfloor \tag{3.43}$$

Predict (P):

$$S[half] = s[half] - \left\lfloor \frac{\sqrt{3}}{4} S[0] \right\rfloor - \left\lfloor \frac{\sqrt{3}-2}{4} S[half - 1] \right\rfloor \tag{3.44}$$

For n = 1 to half -1

$$S[half + n] = s[half + n] - \left\lfloor \frac{\sqrt{3}}{4} S[n] \right\rfloor - \left\lfloor \frac{\sqrt{3}-2}{4} S[n-1] \right\rfloor \tag{3.45}$$

Update 2 (U2):

For n = 0 to half-2

$$S[n] = S[n] - S[half + n + 1] \tag{3.46}$$

$$S[half - 1] = S[half - 1] - S[half] \tag{3.47}$$

Normalize (N1 และ N2):

$$S[n] = \left\lfloor \left[\frac{\sqrt{3} - 1}{\sqrt{2}} \right] S[n] \right\rfloor \tag{3.48}$$

$$S[n + half] = \left\lfloor \left[\frac{\sqrt{3} + 1}{\sqrt{2}} \right] S[n + half] \right\rfloor \tag{3.49}$$

3.2 รูปแบบการแทนเลขจำนวนจริง [11]

รูปแบบของเลขจำนวนเต็ม (Fix-Point Number or Integer) เช่น ทูคอมพลีเมนต์ สามารถที่จะนำไปใช้แทนเลขจำนวนตัวเลขทั้งจำนวนบวกและลบใดๆ ที่มีตั้งแต่ "0" จนถึงค่าจุดศูนย์กลางได้ แต่รูปแบบการแทนข้อมูลแบบนี้ไม่สามารถที่จะทำการแทนข้อมูลเป็นทศนิยมได้

สำหรับการแทนเลขจำนวนจริงของเลขฐานสิบจะใช้รูปแบบการแทนตัวเลขทางคณิตศาสตร์เข้ามาแก้ไขปัญหา เช่น ตัวเลข 976,000,000,000,000 สามารถเขียนให้อยู่ในรูป 9.76×10^{14} และ 0.0000000000000976 ด้วย 9.76×10^{-14} เป็นต้น สิ่งที่ได้ทำไปนั้นคือ การเลื่อนจุดทศนิยมไปยังตำแหน่งที่เหมาะสมและเขียนให้อยู่ในรูปเลขยกกำลังฐานสิบ ซึ่งจะทำให้เกิดความคล่องตัวในการจัดเก็บจำนวนที่มีขนาดใหญ่มากหรือเล็กมากได้อย่างสะดวกสบายด้วยตัวเลขเพียงไม่กี่ตัว และแนวทางนี้สามารถนำไปประยุกต์ใช้กับเลขฐานสองได้ สามารถเขียนเป็นสมการได้ดังนี้

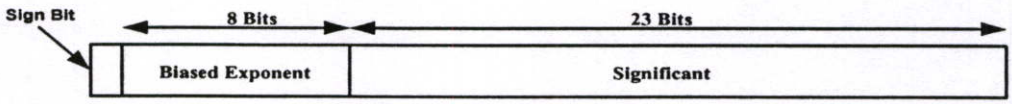
$$\pm SxB^{\pm E} \tag{3.50}$$

จำนวนใดๆ จะถูกแยกเก็บไว้สามส่วน ประกอบไปด้วย

1. ส่วนที่ใช้ในการเก็บเครื่องหมาย (Sign Bit)
2. ส่วนยกกำลัง E (Exponent)
3. ส่วนเลขทศนิยม S (Significant or Mantissa)

ส่วนสัญลักษณ์ B แทนค่าคงที่ของเลขฐานใดๆ จึงไม่จำเป็นที่จะต้องทำการจัดเก็บ

แนวคิดพื้นฐานที่นำมาใช้นี้ สามารถอธิบายโดยยกตัวอย่างได้ดังนี้ รูปที่ 3.9 แสดงโครงสร้างข้อมูลสำหรับเลขจำนวนจริงขนาด 32 บิต บิตทางซ้ายสุดใช้เป็นบิตเครื่องหมาย (0 คือเลขจำนวนบวก 1 เป็นจำนวนลบ) ส่วนถัดมาคือ ค่าของส่วนยกกำลังจะถูกเก็บในพื้นที่ขนาด 8 บิต รูปแบบลักษณะนี้เรียกว่า Biased Representation และค่าคงที่จำนวนหนึ่งที่เรียกว่า Bias จะถูกนำมาลบออกจากเขตข้อมูลนี้ เพื่อให้ได้ค่ายกกำลัง (Exponent Value) ที่แท้จริง โดยทั่วไป Bias มีค่าเป็น $2^{k-1} - 1$ โดยที่ k คือค่าจำนวนบิตในส่วนของ Binary Exponent ในที่นี้ เขตข้อมูลมีขนาด 8 บิตมีค่าอยู่ระหว่าง 0 ถึง 255 ทำให้ Bias มีค่าเป็น 127 แล้วจะทำให้ค่ายกกำลังที่แท้จริงมีค่าอยู่ระหว่าง -127 ถึง128 โดยเลขฐานที่ใช้ในที่นี้คือ เลขฐานสอง ส่วนสุดท้ายของโครงสร้างข้อมูลคือ Significant ที่เป็นตัวเลขขนาด 23 บิต หรืออาจเรียกว่า Mantissa เป็นส่วนที่บอกตัวเลขทศนิยมเช่น 1.638125×2^{20} มี Exponent เป็น 20 และ Mantissa เป็น 1.638125 เป็นต้น



(a) รูปแบบ

$0.11010001x2^{10100}$ = 0 10010011 10100010000000000000000

$-0.11010001x2^{10100}$ = 1 10010011 10100010000000000000000

$0.11010001x2^{-10100}$ = 0 01101011 10100010000000000000000

$-0.11010001x2^{-10100}$ = 1 01101011 10100010000000000000000

(b) ตัวอย่าง

รูปที่ 3.9 รูปแบบโครงสร้างข้อมูลโดยทั่วไปของเลขจำนวนจริงขนาด 32 บิต

เพื่อช่วยให้การทำงานเกี่ยวกับเลขจำนวนจริงง่ายขึ้น มีความจำเป็นที่จะต้องทำการ Normalize เลขจำนวนจริงก่อนที่จะนำมาใช้งาน กล่าวคือ บิตที่มีค่าสูงสุดของเลข Significant จะต้องเป็น "1" เสมอ และตามที่ได้กล่าวไว้แล้วคือจะมีเลขเพียงบิตเดียวที่หน้า (หรืออยู่ทางซ้าย) จุดฐาน ดังแสดงในตัวอย่างเลขจำนวนจริงที่ Normalize แล้วเช่น

$$\pm 1.bbb...bx2 \pm E \tag{3.51}$$

โดยที่ b คือบิต "0" หรือ "1" และ E คือ Exponent เนื่องจากบิตที่มีค่าสูงสุดของ Significant จะต้องเป็น "1" เสมอจึงไม่จำเป็นจะต้องเก็บข้อมูลส่วนนี้ไว้ในพื้นที่เก็บข้อมูล ทำให้พื้นที่ขนาด 23 บิต สามารถเก็บข้อมูลในส่วน Significant ขนาด 24 บิตได้ซึ่งมีค่าอยู่ระหว่าง 0 ถึง

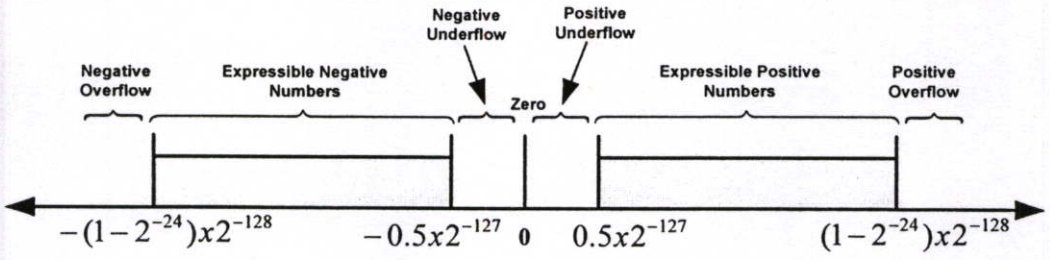
23 บิต และมีค่าอยู่ระหว่าง 1.000000 ถึง 1.999999 เลขจำนวนจริงใดๆ สามารถ Normalize ได้โดยการเลื่อนจุดฐานไปทางซ้ายแล้วปรับค่า Exponent ให้สอดคล้องกัน

จากรูปที่ 3.9(b) แสดงตัวอย่างการแทนเลขจำนวนจริงที่ทำการเก็บข้อมูลตามที่กล่าวมาแล้วในข้างต้น ซึ่งมีคุณสมบัติดังนี้คือ

1. เครื่องหมายจะทำการเก็บไว้ในบิตแรก
2. บิตแรกของ Significant หรือ Mantissa จะต้องเป็น "1" เสมอ แต่จะไม่ได้นำมาเก็บไว้กับตัวเลข
3. ค่า 127 จะถูกนำไปบวกเข้ากับ Exponent ก่อนที่จะนำไปเก็บตัวเลข
4. เลขฐานในที่นี้คือ ฐานสอง

รูปที่ 3.10 แสดงขอบเขตของตัวเลขจริงที่สามารถทำการเก็บในโครงสร้างขนาด 32 บิต โดยใช้โครงสร้างข้อมูลในรูปที่ 3.9 จะทำให้สามารถเก็บตัวเลขที่มีคุณสมบัติดังต่อไปนี้

- เลขจำนวนลบระหว่าง $-(1 - 2^{-24}) \times 2^{128}$ ถึง -0.5×2^{-127}
- เลขจำนวนบวกระหว่าง 0.5×2^{-127} ถึง $(1 - 2^{-24}) \times 2^{128}$



รูปที่ 3.10 แสดงขอบเขตของตัวเลขจริงที่สามารถทำการเก็บในโครงสร้างขนาด 32 บิต

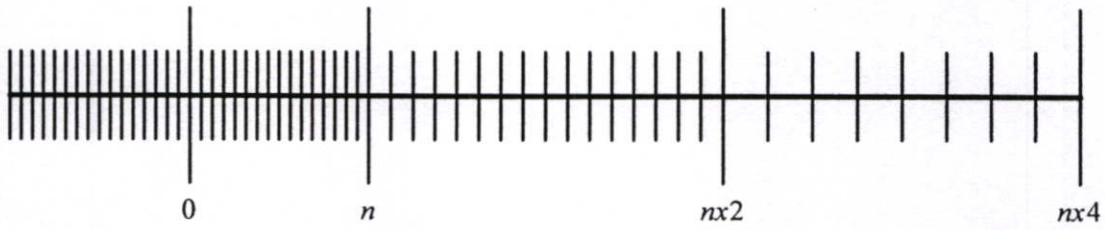
อย่างไรก็ตามจะมีตัวเลขอยู่ 5 จำนวนที่ไม่สามารถทำการเขียนอยู่ในรูปแบบนี้ได้ คือ

- เลขจำนวนลบที่มีค่าน้อยกว่า $-(1 - 2^{-24}) \times 2^{128}$ เรียกว่า Negative Overflow
- จำนวนลบที่มีค่ามากกว่า -2^{-127} เรียกว่า Negative Underflow
- เลขจำนวนศูนย์
- เลขจำนวนบวกที่มีค่าน้อยกว่า 2^{-127} เรียกว่า Positive Underflow
- เลขจำนวนบวกที่มีค่าน้อยกว่า $(2 - 2^{-23}) \times 2^{128}$ เรียกว่า Positive Overflow

จากรูปแบบที่ใช้ในการเก็บเลขจำนวนจริงที่ได้นำเสนอไปนั้นไม่สามารถใช้แทนเลขจำนวนศูนย์ได้ อย่างไรก็ตามรูปแบบที่ใช้ในการแทนเลขจำนวนจริงจะมีรูปแบบที่ใช้ในการแทนเลขจำนวนศูนย์เป็นการเฉพาะ การเกิด Overflow เกิดขึ้นเมื่อผลของการคำนวณมีค่ามากกว่าที่จะทำการเก็บในส่วนของ Exponent ซึ่งจะมีค่ามากกว่า 128 (เช่น $2^{120} \times 2^{100} = 2^{220}$) ส่วนการเกิด Underflow จะเกิดขึ้นเมื่อส่วน Magnitude มีค่าน้อยเกินไป (เช่น $2^{-120} \times 2^{-100} = 2^{-220}$) และการเกิด Underflow จะมีผลเสียน้อยกว่า เนื่องจากจะมีค่าน้อยมากสามารถแทนด้วยจำนวนศูนย์ได้

ข้อควรทราบคือ รูปแบบการแทนเลขจำนวนจริง ไม่ได้เป็นการเพิ่มจำนวนตัวเลขที่ระบบสามารถใช้แทนได้ กล่าวคือ จำนวนตัวเลขจำนวนจริงที่ระบบเลขจำนวนจริงจะสามารถแทนค่าได้ยังคงอยู่ที่ 2^{32} จำนวนเท่ากับรูปแบบการแทนเลขจำนวนเต็ม แต่สิ่งที่ทำได้เป็นเพียงการกระจายจำนวนตัวเลขออกเป็นสองกลุ่มคือ จำนวนบวกและจำนวนลบ

นอกจากนี้ ตัวเลขในรูปแบบเลขจำนวนจริง ยังใช้เนื้อที่ในการแสดงจำนวนตัวเลขไม่เท่ากันเหมือนเลขจำนวนเต็ม กล่าวคือ จำนวนตัวเลขจะมีค่าใกล้เคียงกันมากเมื่อมีค่าเข้าใกล้ศูนย์ และจะมีค่าห่างออกไปเมื่อมีค่ามากขึ้นดังที่แสดงในรูปที่ 3.11 ทำให้เกิดข้อเสียเมื่อนำมาใช้ในการคำนวณ นั่นคือการคำนวณจำนวนมากจะให้ค่าผลลัพธ์ที่ผิดพลาดไปจากความเป็นจริง เพราะถูกปัดเศษ (Rounded) ให้เป็นตัวเลขที่มีค่าใกล้เคียงแทนค่าจริง



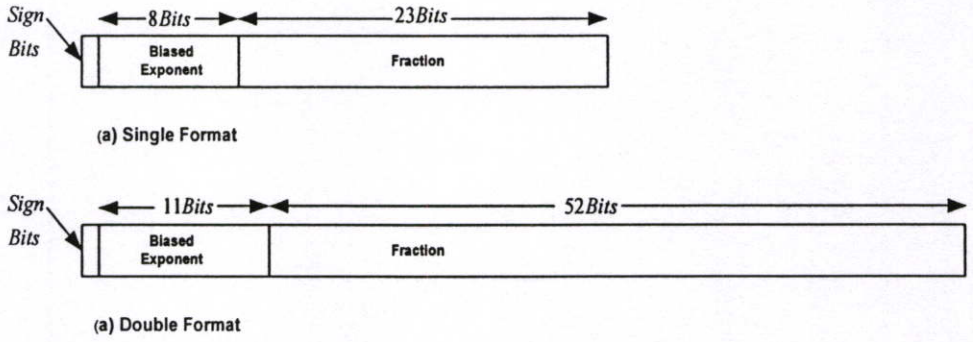
รูปที่ 3.11 ความหนาแน่น (Density) ของเลขจำนวนจริง

จากรูปแบบที่แสดงในรูปที่ 3.11 แสดงให้เห็นถึงการถ่วงดุลระหว่างขอบเขตของเลขจำนวนจริง (Range) กับจำนวนความเที่ยงตรง (Accuracy) ซึ่งแสดงการแบ่งส่วน Exponent ออกเป็น 8 บิต และส่วน Significant หรือ Mantissa ออกเป็น 23 บิต ถ้าทำการเพิ่มจำนวนบิตในส่วนของ Exponent จะช่วยในการขยายขอบเขตของจำนวนตัวเลขให้กว้างมากขึ้น แต่ว่าจะลดช่วงความหนาแน่นของตัวเลขลงไป ซึ่งก็คือการลดความเที่ยงตรงให้น้อยลง ในระบบเครื่องคอมพิวเตอร์ทั่วไปจะนำเสนอการเก็บเลขจำนวนจริงออกเป็นสองแบบ คือ Single-Precision จะใช้ 32 บิต และ Double-Precision ใช้ 64 บิต

3.3 มาตรฐาน IEEE สำหรับรูปแบบการแทนเลขจำนวนจริง

รูปแบบที่ใช้แทนเลขจำนวนจริง (Floating-Point Representation) ถูกกำหนดโดยองค์กร IEEE เลขรหัส 754 [12] มาตรฐานนี้ได้รับการพัฒนาขึ้นมาเพื่ออำนวยความสะดวกในการเคลื่อนย้ายโปรแกรมจากโปรเซสเซอร์ตัวหนึ่งไปใช้งานกับโปรเซสเซอร์อีกตัวหนึ่ง และเพื่อให้ส่งเสริมให้มีการพัฒนาโปรแกรมเชิงเลขที่มีความซับซ้อนมากยิ่งขึ้น มาตรฐานนี้ได้รับการนำเสนอไปใช้งานอย่างกว้างขวาง และกลายเป็นมาตรฐานสำหรับใช้งานกับโปรเซสเซอร์ส่วนใหญ่จนถึงปัจจุบัน

มาตรฐาน IEEE ได้กำหนดรูปแบบการเก็บเลขจำนวนจริงออกเป็นสองแบบคือ 32 บิต และ 64 บิต โดยจะมี Exponent ขนาด 8 บิต และ 11 บิต ตามลำดับดังแสดงใน รูปที่ 3.12 และเลขฐานที่ใช้คือ เลขฐานสอง นอกจากนี้ยังได้กำหนดมาตรฐานของส่วนขยาย (Extended Format) ไว้สองแบบ (Single และ Double) โดยให้ผู้ใช้สามารถเป็นผู้กำหนดรายละเอียดในการสร้างได้เอง ส่วนขยายได้กล่าวถึงบิตเพิ่มเติมในส่วนของ Exponent (เรียกว่า Extended Range) และในส่วนของ Significant (เรียกว่า Exponent Precision) ส่วนเพิ่มเติมนี้มีวัตถุประสงค์เพื่อเตรียมไว้ใช้ในระหว่างการคำนวณ ด้วยความเที่ยงตรงที่เพิ่มขึ้น เนื่องจากการลดการปัดเศษในระหว่างการคำนวณให้น้อยลง และด้วยขอบเขตการแสดงตัวเลขที่เพิ่มขึ้นจะช่วยลดโอกาสการเกิด Overflow ในระหว่างการคำนวณ ซึ่งจะให้ผลลัพธ์อยู่ภายในขอบเขตปกติได้ สิ่งที่ได้รับนอกเหนือจากนี้คือ ส่วนขยายของรูปแบบ Single-Precision จะช่วยให้รับผลดี บางส่วนที่ปกติจะได้รับจากการคำนวณแบบ Double-Precision โดยไม่ต้องเสียเวลาในการคำนวณเพิ่มขึ้นเหมือนการคำนวณแบบ Double-Precision ตารางที่ 3.1 แสดงข้อสรุปของคุณลักษณะของรูปแบบทั้ง 4 แบบ



รูปที่ 3.12 รูปแบบการเก็บเลขจำนวนจริงตามมาตรฐาน IEEE 754

ตารางที่ 3.1 แสดงค่า Parameter ของมาตรฐาน IEEE 754

Format				
Parameter	Single	Single Extended	Double	Double Extended
Word width (bits)	32	≥43	64	≥79
Exponent width (bits)	8	≥11	11	≥15
Exponent bias	127	Unspecified	1023	Unspecified
Maximum exponent	127	≥1023	1023	≥16383
Minimum exponent	-126	≥1022	-1022	≥16382
Number range (base 10)	10^{-38}, 10^{+38}	Unspecified	10^{-308}, 10^{+308}	Unspecified
Significant width (bits)	23	≥31	52	≥63
Number of exponents	254	Unspecified	2048	Unspecified
Number of fractions	2^{23}	Unspecified	2^{52}	Unspecified
Number of values	1.98×2^{31}	Unspecified	1.99×2^{63}	Unspecified

รูปแบบการแทนข้อมูลเลขจำนวนจริงตามมาตรฐาน IEEE ข้อมูลต่างๆ ที่ถูกจัดอยู่ในรูปมาตรฐาน IEEE754 ไม่ได้มีความหมายเหมือนกันทั้งหมด ทั้งนี้อาจถูกใช้แทนจำนวนตัวเลขที่มีความพิเศษดังต่อไปนี้

- สำหรับส่วน Exponent ค่าที่อยู่ในช่วง "1" ถึง "254" สำหรับรูปแบบ Single-Precision และค่า "1" ถึง "2046" สำหรับรูปแบบ Double-Precision ใช้แทนตัวเลขที่ไม่ใช่ศูนย์ที่ถูก Normalize แล้ว ทำให้ขอบเขตของตัวเลขอยู่ในช่วง "-126" ถึง "127" สำหรับรูปแบบ Single-Precision และ "-1022" ถึง "+1023" สำหรับรูปแบบ Double-Precision และ Significant ที่ถูก Normalize จะทำการตัดข้อมูล 1 บิตที่อยู่ทางซ้ายสุดทิ้ง ทั้ง Single-Precision และ Double-Precision
- ส่วนที่เป็น Exponent ที่มีค่าเป็น "0" เมื่อรวมกับส่วน Fraction ที่มีค่าเป็น "0" สามารถใช้แทน Positive Zero เมื่อมีบิตเครื่องหมายเป็นบวก และเป็น Negative Zero เมื่อมีบิตเครื่องหมายเป็นลบ
- ส่วนที่เป็น Exponent ที่มีค่าเป็น "1" ทั้งหมดเมื่อรวมกับส่วน Fraction ที่มีค่าเป็น "0" สามารถใช้แทน Positive Infinity และ Negative Infinity แล้วแต่บิตเครื่องหมายรูปแบบที่มีค่าอนันต์นี้เป็นการเพิ่มทางเลือกให้แก่ผู้ใช้สามารถทำการประมวลผลต่อไปได้
- ส่วนที่เป็น Exponent ที่มีค่าเป็น "0" เมื่อรวมกับ Fraction ที่มีค่าไม่ใช่ "0" ใช้แทนตัวเลข Denormalized ในกรณีที่บิตอยู่ทางซ้ายของจุดฐาน คือ "0" ทำให้ค่าจริงของ Exponent อยู่

ระหว่าง -126 หรือ -1022 โดยที่บิตเครื่องหมายใช้บอกให้ทราบว่าเป็นเลขจำนวนบวกหรือลบ

- ส่วนที่เป็น Exponent ที่มีค่าเป็น "1" ทั้งหมด เมื่อรวมกับส่วน Fraction ที่มีค่าไม่ใช้ "0" สามารถแทน NaN ที่ย่อมาจาก "Not a Number" ซึ่งใช้สำหรับการบอกประเภทของ Exception ชนิดต่างๆ

3.4 การคำนวณทางคณิตศาสตร์ของเลขจำนวนจริง

ตารางที่ 3.2 แสดงข้อสรุปของการทำงานพื้นฐานสำหรับการคำนวณทางคณิตศาสตร์ของเลขจำนวนจริง (Floating Point Arithmetic) สำหรับการบวกลบจำเป็นจะต้องแน่ใจว่าเลขทั้งสองที่นำมาทำการบวกและลบกันนั้นมีค่า Exponent เท่ากัน จึงอาจจะต้องอาศัยการเลื่อนจุดฐานของตัวดำเนินการตัวใดตัวหนึ่ง ในขณะที่การคูณและการหารนั้นสามารถที่จะทำได้ทันทีโดยไม่ต้องมีการปรับค่าเลขฐาน

ตารางที่ 3.2 แสดงการคำนวณเลขจำนวนจริง

Floating Point Number	Arithmetic Operations
$X = X_s \times B^{X_E}$	$\left.\begin{array}{l} X+Y=(X_s \times B^{X_E-Y_E}+Y_S)\times B^{Y_E} \\ X+Y=(X_s \times B^{X_E-Y_E}-Y_S)\times B^{Y_E} \end{array}\right\} X_E \le Y_E$
$Y = Y_s \times B^{Y_E}$	$X \times Y = (X_S \times Y_S) \times B^{X_E+Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E-Y_E}$

การทำงานกับเลขจำนวนจริงอาจจะทำให้เกิดสิ่งใดสิ่งหนึ่งดังต่อไปนี้

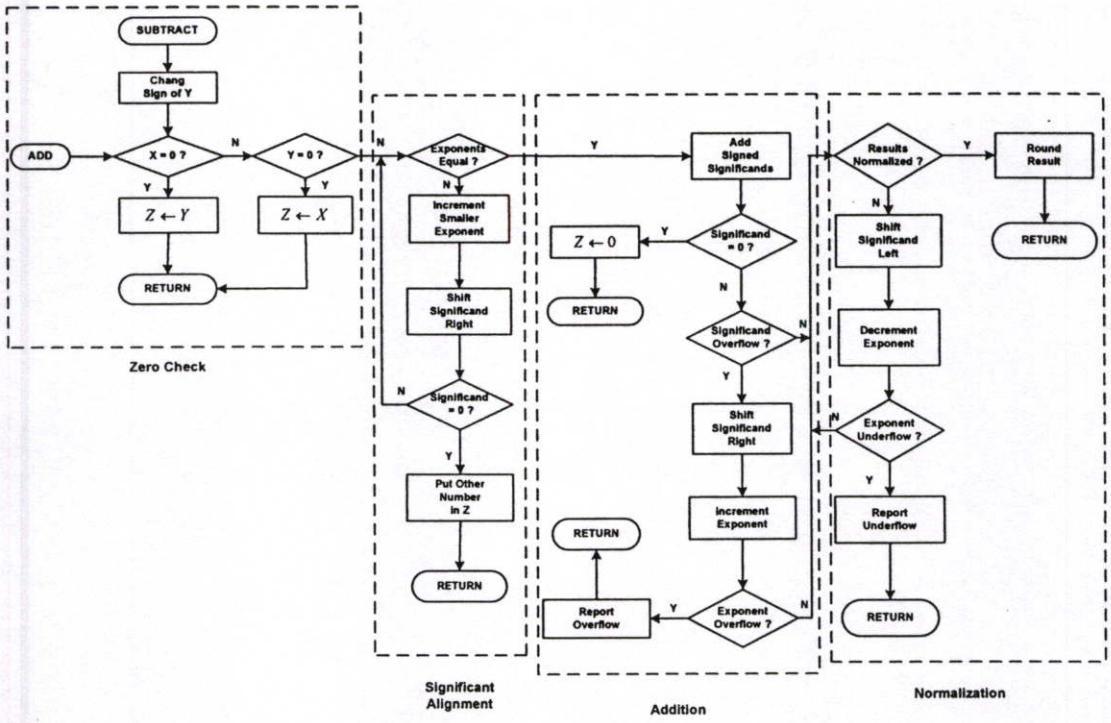
- Exponent Overflow : ส่วน Exponent เป็นจำนวนบวกมีค่าเกินกว่าค่าสูงสุดที่เป็นไปได้ นั่นคือ ได้ผลลัพธ์เป็นจำนวนที่มีค่าใหญ่เกินไป ในบางระบบได้กำหนดค่าเมื่อเกิด Overflow ให้เป็น $+\infty$ หรือ $-\infty$
- Exponent Underflow : ส่วน Exponent ที่เป็นจำนวนลบมีค่าต่ำกว่าค่าต่ำสุดที่เป็นไปได้ นั่นคือได้ผลลัพธ์เป็นเลขจำนวนที่มีค่าน้อยมากเกินไป จึงมักจะแทนผลลัพธ์ด้วยค่า "0"
- Significant Underflow : ในขณะที่ทำการเลื่อนบิตเพื่อทำการบวกหรือลบนั้น ตัวเลขอาจถูกเลื่อนออกไปจนเกินขอบขวาสุดของส่วน Significant ทำให้ต้องมีการปัดเศษตัวเลข (Rounding) ซึ่งจะกล่าวถึงต่อไป

- Significant Overflow : การนำส่วน Significant สองจำนวนที่มีเครื่องหมายเหมือนกันมาทำการบวกเข้าด้วยกัน อาจทำให้ได้ผลลัพธ์ที่มีค่าเกินกว่าที่จะทำการเก็บไว้ได้ ซึ่งอาจแก้ไขได้โดยการเลื่อนบิตข้อมูลดังที่จะกล่าวถึงต่อไป

3.4.1 การบวกและการลบเลขจำนวนจริง

ในการคำนวณทางคณิตศาสตร์ของเลขจำนวนจริง การบวกและการลบจะมีความซับซ้อนมากกว่าการคูณหรือการหาร ทั้งนี้เนื่องมาจากที่ต้องทำการเลื่อนบิตข้อมูลในส่วนทศนิยมเพื่อให้ตัวเลขสมดุลกัน (Word Alignment) ขั้นตอนในการบวกและการลบแบ่งออกเป็น 4 ขั้นตอนดังนี้

1. ทำการตรวจสอบว่าเป็น “0” หรือไม่
2. จัดลำดับของส่วนที่เป็นทศนิยมให้ตรงกัน
3. ทำการบวกหรือลบข้อมูลในส่วนทศนิยม
4. ทำการปรับค่าผลลัพธ์ให้เหมาะสม



รูปที่ 3.13 แผนผังการบวกและการลบเลขจำนวนจริง

รูปที่ 3.13 แสดงแผนผังการบวกและการลบเลขที่เกิดขึ้นโดยทั่วไป ซึ่งแสดงให้เห็นการทำงานแต่ละขั้นตอนของฟังก์ชั่นหลักที่จำเป็นต้องใช้ในการบวกและลบตัวเลข ในที่นี้ได้ใช้รูปแบบการแทนเลขจำนวนจริงที่แสดงไว้ในรูปที่ 3.12 สำหรับการบวกและการลบ ตัวถูกดำเนินการ

(Operand) ทั้งสองตัวต้องถูกอ่านเข้ามาเก็บไว้ในรีจิสเตอร์ ซึ่งจะถูกเรียกใช้โดยหน่วยเอแอลยู (ALU) ถ้ารูปแบบเลขจำนวนจริงที่นำมาใช้มีการใช้บิตเพิ่มพิเศษ (Implicit Significant Bit) ก็จะต้องทำการรวมบิตข้อมูลนี้เข้าไปในรีจิสเตอร์ด้วยก่อนที่จะทำการบวกหรือลบ

การทำงานในขั้นตอนที่ 1: Zero Check เนื่องจากการบวกและลบมีข้อแตกต่างกันเพียงบิตเครื่องหมายที่ไม่เหมือนกัน กระบวนการเริ่มต้นจากการเปลี่ยนเครื่องหมายของตัวลบในกรณีที่เป็นการลบ ขั้นตอนต่อไปถ้าตัวดำเนินการเป็น "0" ก็นำค่าตัวตั้งไปเป็นค่าผลลัพธ์ของการคำนวณได้ทันที

การทำงานในขั้นตอนที่ 2: Significant Alignment ขั้นตอนต่อไปคือ การจัดการให้ตัวเลขทั้งสองมีส่วน Exponent ที่เท่ากัน พิจารณาการบวกเลขฐานสอบสองจำนวนเข้าด้วยกัน ซึ่งจำเป็นจะต้องทำการจัดการส่วน Exponent ก่อนที่จะนำมาบวกกัน

$$\left(123\times10^{0}\right)+\left(456\times10^{-2}\right) \tag{3.53}$$

จากเลขข้างบนเห็นได้ชัดว่าไม่สามารถที่จะนำส่วน Significant ของเลขทั้งสองมาทำการบวกกันได้ทันที ดังนั้นจะต้องทำการปรับค่าให้เหมาะสมเสียก่อน นั่นก็คือเลข "4" ของจำนวนทางขวาจะต้องถูกจัดให้ตรงกับเลข "3" ของจำนวนทางซ้ายมือ ซึ่งจะทำให้ Exponent ของเลขทั้งสองจำนวนมีค่าเท่ากันเป็น

$$\left(123\times10^{0}\right)+\left(456\times10^{-2}\right)=\left(123\times10^{0}\right)+\left(4.56\times10^{0}\right)=127.56\times10^{0} \tag{3.54}$$

การจัดตำแหน่งของตัวเลข สามารถทำได้โดยการเลื่อนบิตข้อมูลของจำนวนที่มีค่าน้อยกว่าไปทางขวาหรือการเลื่อนบิตข้อมูลของจำนวนที่น้อยกว่าไปทางซ้าย เนื่องจากการเลื่อนบิตข้อมูลวิธีใดก็ตาม จะทำให้เกิดการสูญเสียไปบางส่วน ถ้าทำการเลื่อนบิตข้อมูลของตัวเลขที่มีค่าน้อย ตัวเลขที่สูญเสียไปก็จะมีค่าน้อยไปด้วย การจัดตำแหน่งจึงทำได้โดยการเลื่อนตำแหน่งของ Significant ไปทางขวา 1 ตำแหน่ง และทำการเพิ่มค่าของ Exponent ขึ้นไปอีก 1 ทำวนไปจนกว่าค่าของ Exponent ของเลขทั้งสองจะมีค่าเท่ากัน ถ้ากระบวนการนี้ทำให้ส่วนของ Significant มี ค่าเป็น "0" แล้วตัวเลขที่เหลืออีกจำนวนหนึ่งจะกลายเป็นผลลัพธ์ ดังนั้นถ้าเลขทั้งสองจำนวนมีส่วน Exponent ที่แตกต่างกันมากแล้วกระบวนการนี้จะมีการสูญเสียที่เกิดจากการเลื่อนบิตข้อมูลน้อยกว่า

การทำงานในขั้นตอนที่ 3: Addition ขั้นตอนต่อไปก็จะนำส่วน Significant ของเลขทั้งสองจำนวนมาทำการบวกเข้าด้วยกัน โดยจะต้องทำการพิจารณาบิตเครื่องหมายด้วย เนื่องจากเครื่องหมายอาจมีค่าต่างกัน ซึ่งอาจทำให้ค่าผลลัพธ์มีค่าเป็น "0" ก็ได้นอกจากนี้อาจเป็นไปได้ว่าส่วนของ Significant อาจเกิดการ Overflow ได้ ในกรณีนี้จะต้องทำการเลื่อนบิต Significant ไปทางขวา และทำการเพิ่มค่าให้แก่ Exponent และเพิ่มค่าให้แก่ Exponent ซึ่งก็อาจทำให้ส่วน

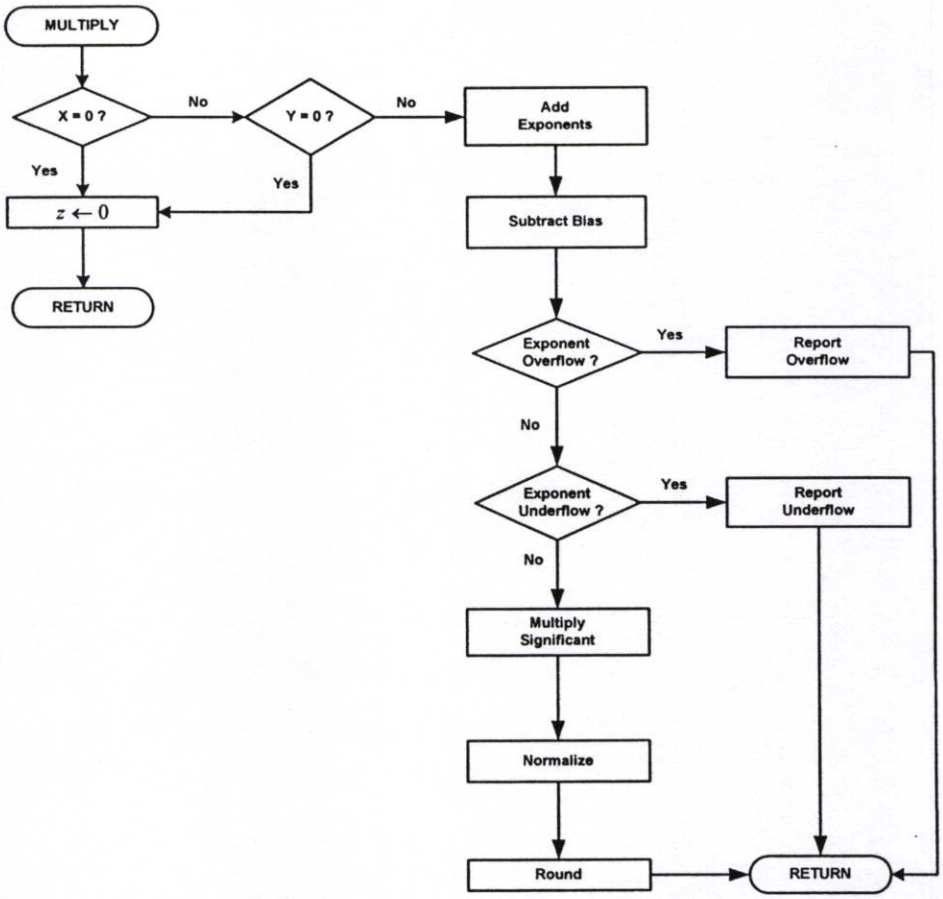
Exponent เกิดการ Overflow ได้ ในกรณีนี้การทำงานจะหยุดลง และจะต้องรายงานผลความล้มเหลวที่เกิดขึ้น

การทำงานขั้นตอนที่ 4: Normalization ขั้นตอนสุดท้ายจะต้องทำการ Normalize ผลลัพธ์ที่ได้ ซึ่งประกอบไปด้วยการเลื่อนบิต Significant ไปทางซ้ายจนกระทั่งบิตข้อมูลมีค่าสูงสุดเป็นค่าผลลัพธ์ที่ไม่ใช่ "0" การเลื่อนบิตแต่ละครั้งจะทำให้เกิดการลดค่าของ Exponent ลงไปซึ่งอาจทำให้เกิด Exponent Underflow ท้ายสุดตัวเลขอาจมีการปัดเศษเกิดขึ้น และรายงานผลลัพธ์ออกไป ขั้นตอนในการปัดเศษจะกล่าวถึงในภายหลัง

3.4.2 การคูณเลขจำนวนจริง

การคูณเลขจำนวนจริง เป็นกระบวนการที่ง่ายกว่าการบวกและลบ รูปที่ 3.14 แสดงกระบวนการคูณเลข ถ้าตัวดำเนินการตัวใดตัวหนึ่งมีค่าเป็น "0" ก็จะทำให้ผลลัพธ์มีค่าเป็น "0" ขั้นต่อไปคือ การนำส่วน Exponent มาบวกเข้าด้วยกัน ถ้าส่วน Exponent มีการจัดเก็บแบบ Biased Form ผลบวกของ Exponent ทั้งสองมีค่ามากกว่าสองเท่าของค่า Bias ดังนั้นจึงต้องนำค่า Bias มาทำการลบออกจากผลบวกที่ได้ ผลลัพธ์ยังอาจเป็น Overflow หรือ Underflow ก็ได้ซึ่งจะทำให้กระบวนการคูณสิ้นสุดและรายงานผลที่เกิดขึ้น

ถ้าส่วนของ Exponent อยู่ภายในขอบเขตที่เป็นไปได้ ก็จะต้องนำส่วน Significant มาทำการคูณกัน ซึ่งจะต้องคำนึงถึงบิตเครื่องหมายด้วย กระบวนการคูณเกิดขึ้นในทำนองเดียวกับการคูณเลขจำนวนเต็ม ผลคูณที่ได้จะทำให้จำนวนบิตข้อมูลเพิ่มเป็นสองเท่า บิตที่เกินมาจะถูกปัดเศษทิ้งไป หลังจากได้ผลลัพธ์แล้วก็ทำการ Normalize และทำการปัดเศษแบบเดียวกับที่ทำในการบวกและการลบ



รูปที่ 3.14 แสดงแผนผังการคูณจำนวนจริง

3.4.3 การปัดเศษ

รายละเอียดอีกประการหนึ่ง ที่ทำให้เกิดความแตกต่างต่อความเที่ยงตรงของการคำนวณผลลัพธ์คือ การปัดเศษ (Rounding) ผลลัพธ์ในส่วนของการทำงานของ Significant มักจะถูกนำมาเก็บไว้ในรีจิสเตอร์ที่มีความยาวมากกว่าเสมอ เมื่อผลลัพธ์ถูกส่งกลับเข้าไปอยู่ในรูปแบบของเลขจำนวนจริง บิตที่เกินมาจะถูกตัดทิ้ง

วิธีการปัดเศษได้รับการคิดค้นขึ้นมาด้วยกันหลายวิธี IEEE จึงได้ทำการกำหนดมาตรฐานไว้ 4 วิธีคือ

1. ปัดเศษไปยังค่าที่ใกล้เคียง ผลลัพธ์จะถูกปัดให้มีค่าเท่ากับตัวเลขที่ใกล้เคียงที่สุด
2. ปัดเศษไปทางจำนวนบวกอนันต์ ผลลัพธ์จะถูกปัดให้มีค่าไปในทิศทางบวกอนันต์
3. ปัดเศษไปยังทิศทางลบอนันต์ ผลลัพธ์ที่ได้จะถูกปัดให้มีค่าไปในทิศทางของจำนวนลบอนันต์
4. ปัดเศษไปทางจำนวนศูนย์ ผลลัพธ์จะถูกปัดให้มีค่าไปในทิศทางใกล้จำนวนศูนย์

3.4.3.1 การปัดเศษไปยังค่าที่ใกล้เคียง (Round to Nearest)

เป็นวิธีที่นิยมนำมาใช้งานและถูกกำหนดให้เป็นวิธีการมาตรฐานซึ่งกำหนดไว้ว่า ให้คืนค่าที่สามารถแสดงได้ด้วยรูปแบบตัวเลขที่เหมาะสม ที่มีค่าใกล้เคียงที่สุดกับตัวเลขที่ไม่สามารถนำมาแสดงผลได้ ถ้ามีเลขสองจำนวนที่มีค่าใกล้เคียงกันเท่ากัน ก็ให้ใช้จำนวนบิตที่มีค่าน้อยที่สุดเป็น "0"

ถ้าบิตที่เกินจาก 23 บิต ที่สามารถใช้แสดงแทนตัวเลขนั้นเป็น 10010 แสดงว่าค่าของบิตพิเศษนี้มีค่าเกินกว่าครึ่งหนึ่งของบิตสุดท้ายที่สามารถแสดงได้ก็ให้บวก "1" เข้ากับบิตสุดท้ายนั้นและปัดเป็นเลขจำนวนที่สามารถแสดงค่าได้ซึ่งมีผลเท่ากับการปัดเศษขึ้น (Round Up) แต่ถ้าบิตพิเศษเป็น 01111 ซึ่งมีค่าน้อยกว่าครึ่งหนึ่งของบิตสุดท้าย ก็ให้ตัดบิตพิเศษที่เกินมานั้นทิ้งไปทั้งหมดซึ่งมีผลเท่ากับเป็นการปัดเศษลง

มาตรฐานของ IEEE ยังได้กล่าวถึงกรณีพิเศษ เมื่อบิตพิเศษมีค่าเป็น 10000 ในกรณีนี้ค่าของบิตพิเศษมีค่าเป็นครึ่งหนึ่งของบิตสุดท้ายพอดี หนทางเลือกที่ง่ายทางหนึ่งคือ การปัดเศษทิ้งไปทั้งหมด อย่างไรก็ตามวิธีการนี้ทำให้เกิดความผิดพลาดสะสมเข้าไปในการคำนวณหนทางที่ไม่ให้เกิดความความผิดพลาดสะสมทางหนึ่งคือ วิธีการปัดเศษที่มีทั้งการปัดเศษขึ้นและปัดลงผสมกันไปซึ่งก็ช่วยลดความผิดพลาดสะสมลงได้ แต่ก็ให้ผลการคำนวณที่ไม่คงที่ IEEE จึงกำหนดมาตรฐานบังคับใช้ให้มีผลลัพธ์คงที่โดย ถ้าผลการคำนวณให้ค่าบิตพิเศษที่อยู่ระหว่างกลางพอดีปัดเศษขึ้น ถ้าบิตที่สามารถแสดงผลได้ (บิตที่ 23) มีค่าเป็น "1" และให้ปัดเศษทิ้งถ้าบิตสุดท้ายมีค่าเป็น "0"

3.4.3.2 การปัดเศษไปทางบวกหรือลบอนันต์ (Rounding to Plus or Minus Infinity)

มีประโยชน์ในการสร้างเทคนิคที่เรียกว่า Interval Arithmetic เทคนิคนี้เป็นวิธีการที่มีประสิทธิภาพที่ช่วยในการตรวจสอบ และควบคุมความผิดพลาดในการคำนวณเลขจำนวนจริงด้วยการสร้างค่าสองค่าสำหรับการคำนวณแต่ละครั้ง ซึ่งหมายถึงค่าสูงและค่าต่ำของช่วงตัวเลข (Interval) ที่มีผลลัพธ์ที่แท้จริงอยู่ ความกว้างของตัวเลขซึ่งก็คือ ค่าความแตกต่างระหว่างค่าสูงและค่าต่ำ จะเป็นตัวชี้ให้เห็นความเที่ยงตรงของผลลัพธ์ที่ได้ถ้าช่วงดังกล่าวเป็นค่าที่ไม่สามารถแสดงให้เห็นได้ ก็จะใช้วิธีปัดเศษลงสำหรับค่าสูง และปัดเศษขึ้นสำหรับค่าต่ำ แม้ว่าขอบเขตของช่วงดังกล่าวอาจมีค่าเปลี่ยนแปลงไปตามวิธีสร้างกระบวนการคำนวณ อัลกอริธึมจำนวนมากได้รับการออกแบบมาเพื่อทำให้ช่วงดังกล่าวมีขนาดแคบมาก ถ้าช่วงระหว่างค่าสูงและค่าต่ำมีค่าแคบมากแล้วผลลัพธ์ที่ได้ก็มีค่าเที่ยงตรงมาก มิฉะนั้นอย่างน้อยก็จะทราบได้ว่าต้องทำการวิเคราะห์เพิ่มเติมในด้านอื่นต่อไป

3.4.3.3 การปัดเศษไปทางจำนวนศูนย์ (Round Toward Zero)

เป็นเทคนิคการปัดเศษทิ้ง นั้นก็คือ การตัดค่าบิตพิเศษทิ้งทั้งหมด วิธีนี้ง่ายที่สุดแต่ผลลัพธ์ที่ได้จะมีค่าน้อยหรือเท่ากับค่าก่อนปัดเศษทิ้งเสมอ ซึ่งเป็นการสร้างความเสียหายเนื่องจากความไม่เที่ยงตรงของตัวเลขมากกว่าวิธีอื่นที่ได้กล่าวถึงไปแล้ว

บทที่ 4
การออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ตโดยใช้ FPGA

ในบทนี้จะกล่าวถึงการออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ต โดยใช้สมการของ Daubechies4 (D4) เพื่อให้สามารถทำการประมวลผลได้เหมือนกับเครื่องคอมพิวเตอร์ จำเป็นต้องทำการออกแบบโมดูลสำหรับใช้ในการคำนวณเลขจำนวนจริง ในส่วนของการออกแบบโมดูลที่ใช้ในการคำนวณเลขจำนวนจริงนี้จะอ้างอิงตามมาตรฐาน IEEE 754 จากนั้นก็นำหน่วยประมวลผลดังกล่าวมาจัดเรียงตามสมการ Daubechies4 (D4) เพื่อสร้างโมดูลที่ใช้ในการแปลงเวฟเล็ต และในส่วนสุดท้ายจะเป็นการนำเสนอการออกแบบการแปลงเวฟเล็ตแบบเลขจำนวนเต็มโดยใช้ FPGA โดยทำการเปลี่ยนโมดูลที่ใช้ในการคำนวณเลขจำนวนจริงไปเป็นโมดูลที่ใช้คำนวณเลขจำนวนเต็ม เพื่อที่จะทำการลดทรัพยากรที่ใช้ใน FPGA ลง

4.1 การออกแบบโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง

การออกแบบโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริงจะอ้างอิงตามมาตรฐาน IEEE 754 โมดูลนี้เป็นโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริงโดยใช้รูปแบบ 32 บิต ในความเป็นจริงการบวกและลบเลขจำนวนจริงจะมีกระบวนการที่ซับซ้อนมากกว่าการบวกและลบเลขจำนวนเต็ม เพราะว่าการบวกและลบเลขจำนวนจริงจะกระทำในลักษณะของเลขทศนิยม เมื่อพิจารณาการออกแบบโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริงในรูปของสมการทางคณิตศาสตร์สามารถแสดงเป็นสมการดังต่อไปนี้

ให้อินพุทของโมดูลประกอบไปด้วย $V1$ และ $V2$ ตามลำดับ

$$V1 = -1^{s1}2^{(c1-127)}(1.f1) \tag{4.1}$$

$$V2 = -1^{s2}2^{(c2-127)}(1.f2) \tag{4.2}$$

จากสมการทั้งสองจะประกอบไปด้วยตัวแปรต่างๆ คือ $s1, s2$ เป็นตัวแปรที่ใช้ในการเก็บบิตเครื่องหมาย $c1, c2$ จะเก็บค่าในส่วนยกกำลัง และ $f1, f2$ จะเก็บค่าในส่วนทศนิยม ซึ่งตัวแปลต่างๆ นี้จะถูกแปลงให้อยู่ในรูปของเลขฐานสอง

จากนั้นทำการเปรียบเทียบขนาดของสมการ $V1$ กับ $V2$ ถ้าขนาดของ $V2$ มากกว่าขนาดของ $V1$ ให้ทำการสลับค่าตามเงื่อนไขด้านล่าง

$$
\begin{aligned}
&if|V2| \geq |V1| \\
&\quad V1 \Leftarrow V2 \\
&\quad V2 \Leftarrow V1 \\
&else \\
&\quad V1 \Leftarrow V1 \\
&\quad V2 \Leftarrow V2
\end{aligned}
$$

หลังจากทำการตรวจสอบขนาด $V1$ กับ $V2$ เรียบร้อยแล้วก็จะทำการแก้สมการหาค่าผลลัพธ์ของการบวกและลบเลขจำนวนจริงตามสมการที่ (4.3) ถึง (4.8) ดังแสดงดังต่อไปนี้

$$V = V1 \pm V2 \tag{4.3}$$

$$V = -1^{s1}\left\{2^{c1-127}(1.f1) \pm 2^{c2-127}(1.f2)\right\} \tag{4.4}$$

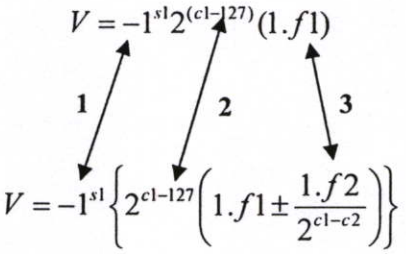
$$c2 = c1 - (c1 - c2) \tag{4.5}$$

$$V = -1^{s1}\left\{2^{c1-127}(1.f1) \pm 2^{c1-(c1-c2)-127}(1.f2)\right\} \tag{4.6}$$

$$V = -1^{s1}\left\{2^{c1-127}\left[(1.f1) \pm 2^{-(c1-c2)}(1.f2)\right]\right\} \tag{4.7}$$

$$V = -1^{s1}\left\{2^{c1-127}\left(1.f1 \pm \frac{1.f2}{2^{c1-c2}}\right)\right\} \tag{4.8}$$

ผลลัพธ์จากการแก้สมการในสมการที่ (4.8) จะถูกนำไปใช้ในการออกแบบโมดูลสำหรับใช้ในการบวกและลบเลขจำนวนจริงต่อไป หากทำการพิจารณาสมการที่ได้เพื่อนำมาวิเคราะห์การทำงานสามารถแสดงให้อยู่ในรูปที่ 4.1 ดังนี้



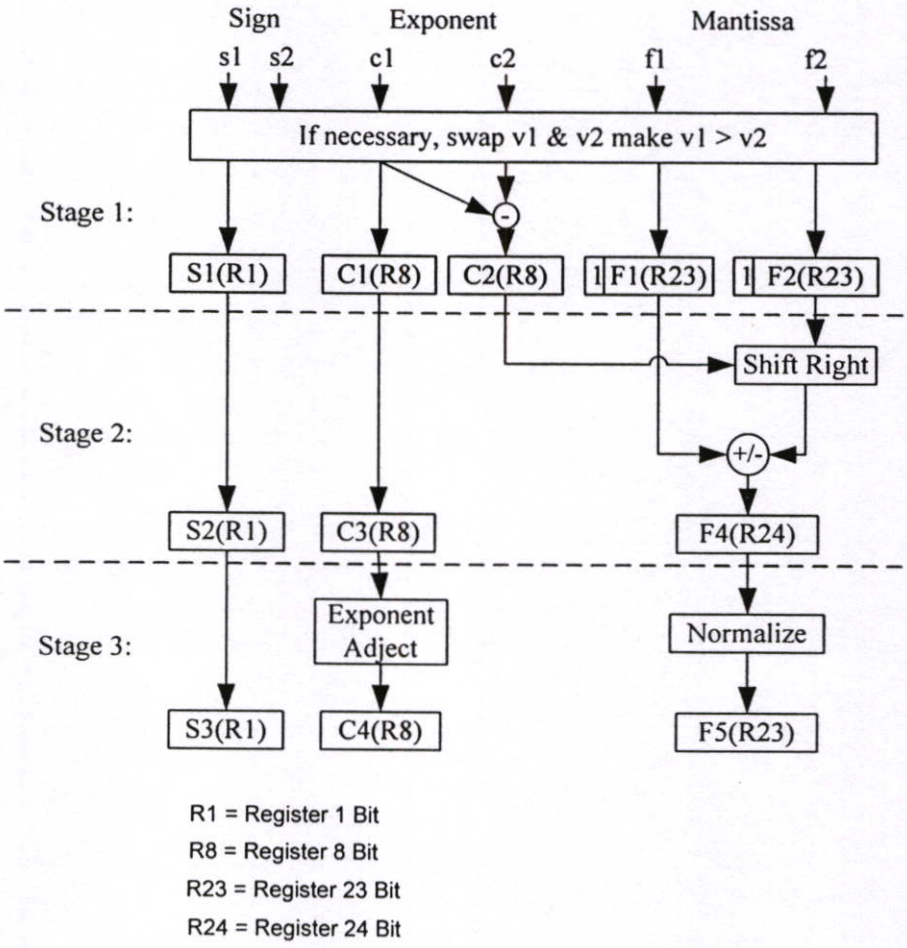
รูปที่ 4.1 แสดงการเปรียบเทียบสมการที่ใช้ในการบวกและลบเลขจำนวนจริงกับสมการมาตรฐาน

จากรูปที่ 4.1จะเห็นได้ว่าสมการที่ใช้ในการบวกและลบเลขจำนวนจริงประกอบไปด้วย 3 ส่วนคือ

1. ส่วนที่ใช้ในการเก็บบิตเครื่องหมาย (Sign Field)

2. ส่วนที่ใช้เก็บข้อมูลของเลขส่วนยกกำลัง (Exponent Field) ส่วนนี้ได้มาจากส่วนยกกำลังของ $V1$
3. ส่วนที่ใช้ในการเก็บข้อมูลของเลขทศนิยม (Mantissa Field) ส่วนนี้ได้มาจากการเลื่อนบิตข้อมูลของส่วนทศนิยมของ $V2$ ($f2$)ไปทางขวา (Shift Right) เป็นจำนวนครั้งเท่ากับผลต่างของ $c1$ และ $c2$ จากนั้นนำผลลัพธ์ที่ได้จากการเลื่อนบิตข้อมูลดังกล่าวทำบวกหรือลบกับส่วนทศนิยมของ $V1$($f1$)

การออกแบบโมดูลบวกและลบเลขจำนวนจริงโดยใช้ FPGA ในระดับ RTL (Register Transfer Level) จะอ้างอิงตามสมการ (4.8) และโครงสร้างดังแสดงในรูปที่ 4.2



รูปที่ 4.2 แสดงโครงสร้างภายในของโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง

จากรูปที่ 4.2 โครงสร้างของโมดูลจะประกอบไปด้วยส่วนอินพุตคือ ส่วนเครื่องหมายที่ประกอบไปด้วย $s1$ และ $s2$ อินพุตในส่วนยกกำลังที่ประกอบไปด้วย $c1$ และ $c2$ และส่วนสุดท้ายคือ ส่วนเลขทศนิยมที่ประกอบไปด้วย $f1$ และ $f2$ นอกจากนี้ภายในโครงสร้างของโมดูลก็จะมี

รีจิสเตอร์ต่างๆ เพื่อใช้สำหรับเก็บข้อมูลที่ใช้ในการดำเนินการแต่ละขั้นตอนจนได้ค่าผลลัพธ์โดยที่ $S1$ ถึง $S3$ เป็นรีจิสเตอร์ขนาด 1 บิตทำหน้าที่เก็บบิตเครื่องหมาย $C1$ ถึง $C4$ เป็นรีจิสเตอร์ขนาด 8 บิต ทำหน้าที่เก็บส่วนยกกำลัง $F1$, $F2$ และ $F5$ เป็นรีจิสเตอร์ขนาด 23 บิตทำหน้าเก็บข้อมูลที่ได้จากส่วนทศนิยม $F4$ เป็นรีจิสเตอร์ขนาด 24 บิตใช่เก็บผลลัพธ์ที่ได้จากการบวกหรือลบกันของ $f1$ และ $f2$ จากโครงสร้างของโมดูลการบวกและลบเลขจำนวนจริงสามารถแบ่งการทำงานออกเป็น 3 ขั้นตอนหลักๆ ดังต่อไปนี้

ขั้นตอนที่ 1 มีการทำงานดังนี้คือ

1. ให้ทำการพิจารณาค่าอินพุตทั้งสองตัว ถ้าขนาดของ $V1$ มีค่าน้อยกว่าขนาดของ $V2$ ให้ทำการสลับค่า $V1$ และ $V2$ การตรวจสอบค่าขนาดของจำนวนทั้งสองจะทำการตรวจสอบค่าในส่วนยกกำลัง(Exponent) และส่วนที่เป็นทศนิยม (Mantissa)
2. จากนั้นให้นำ $c1$ และ $c2$ ซึ่งส่วนยกกำลังของ (Exponent) ของอินพุตทั้งสองมาลบกัน ผลลัพธ์เก็บไว้ที่รีจิสเตอร์ $C2$ เพื่อคำนวณค่าที่ใช้ในการเลื่อนตำแหน่งของ $1.F2$

ขั้นตอนที่ 2 มีการทำงานดังนี้คือ

1. ทำการเลื่อนบิต $1.F2$ ไปทางขวา
2. ในกรณีที่ $s1$ เท่ากับ $s2$ ให้ทำการบวก $1.F1$ และ $1.F2$ เข้าด้วย แต่ถ้า $s1$ ไม่เท่ากับ $s2$ ให้ทำการลบ $1.F1$ ด้วย $1.F2$ ผลลัพธ์ที่ได้เก็บไว้ที่ $F4$
3. ทำการเก็บบิตเครื่องหมายที่ได้จาก $V1$ ซึ่งบิตเครื่องหมายนี้จะเป็นบิตเครื่องหมายของผลลัพธ์

ขั้นตอนที่ 3 มีการทำงานดังนี้คือ

1. ทำการปรับค่า $F4$ โดยการเลื่อนบิตข้อมูลไปทางซ้ายจนกระทั่งบิตทางด้านซ้ายสุดเป็นมีค่าเป็นหนึ่ง
2. ทำการปรับค่าในส่วนยกกำลัง (Exponent) โดยการนำค่าส่วนยกกำลัง (Exponent) มาลบกับจำนวนที่ใช้ในการเลื่อนบิต $F4$

4.2 การออกแบบโมดูลที่ใช้ในการคูณจำนวนจริง

การออกแบบโมดูลที่ใช้ในการคูณเลขจำนวนจริง จะมีความซับซ้อนน้อยกว่าการออกแบบโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง เพราะลักษณะการเก็บข้อมูลตัวเลขตามมาตรฐาน IEEE 754 จะถูกเก็บไว้ในลักษณะของเลขยกกำลังทำให้การหาผลลัพธ์ของการคูณทำได้โดยง่าย สามารถแสดงเป็นสมการทางคณิตศาสตร์ได้ดังนี้

ให้อินพุทของประกอบไปด้วย $V1$ และ $V2$ ตามลำดับ

$$V1 = -1^{s1}2^{(c1-127)}(1.f1) \tag{4.4}$$

$$V2 = -1^{s2}2^{(c2-127)}(1.f2) \tag{4.5}$$

จากนั้นก็นำ $V1$ และ $V2$ มาคูณเข้าด้วยกัน เพื่อหาสมการที่ใช้หาค่าผลลัพธ์ในการคูณ

$$V = V1 \times V2 \tag{4.6}$$

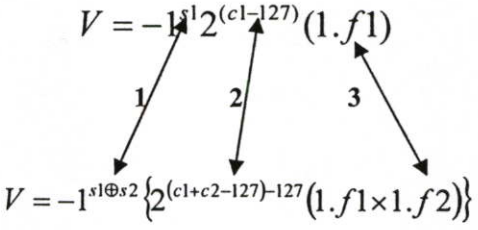
$$V = \left(V1 = -1^{s1}2^{(c1-127)}(1.f1)\right) \times \left(V1 = -1^{s2}2^{(c2-127)}(1.f2)\right) \tag{4.7}$$

ทำการแก้สมการที่ (4.6) ก็จะได้สมการที่ใช้ในการออกแบบโมดูลที่ใช้ในการคูณเลขจำนวนจริงดังสมการที่ (4.9)

$$V = -1^{s1 \oplus s2}\left\{2^{c1-127} \times 2^{c2-127}(1.f1 \times 1.f2)\right\} \tag{4.8}$$

$$V = -1^{s1 \oplus s2}\left\{2^{(c1+c2-127)-127}(1.f1 \times 1.f2)\right\} \tag{4.9}$$

จากสมการที่ (4.9) เป็นสมการที่ใช้หาค่าผลลัพธ์ของการคูณเลขจำนวนจริงและถูกนำไปใช้ออกแบบโมดูลคูณเลขจำนวนจริงต่อไป หากทำการพิจารณารูปที่ 4.3 สามารถทำการแบ่งสมการคูณเลขจำนวนจริงออกได้เป็น 3 ส่วนคือ

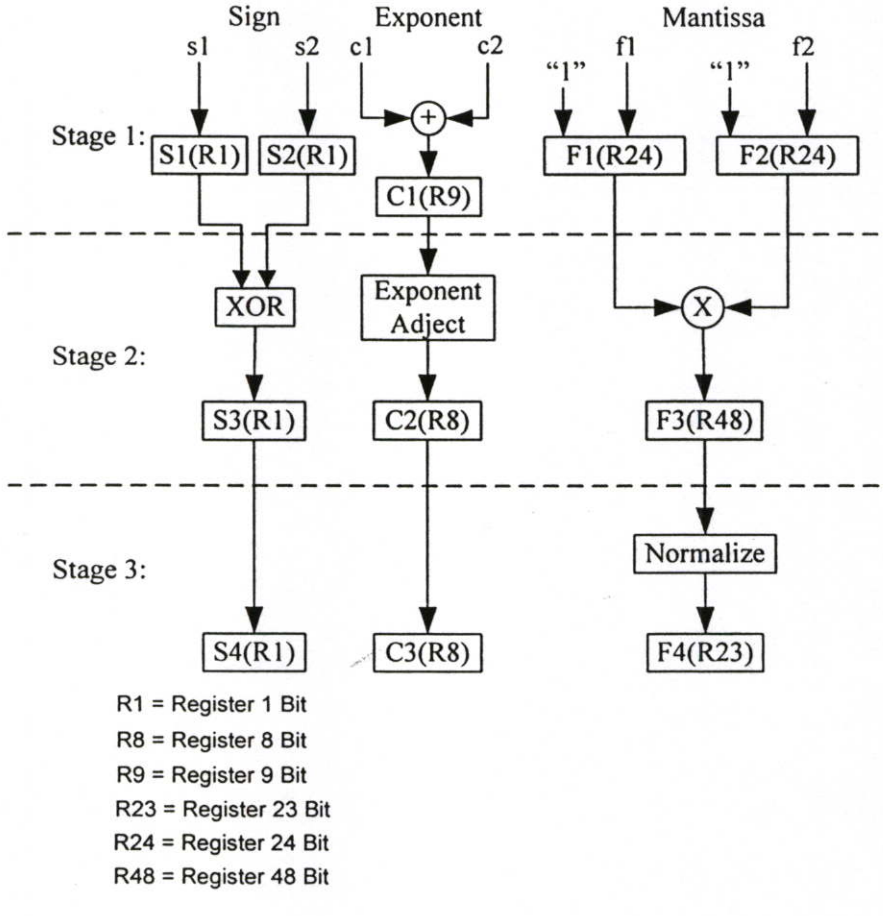


รูปที่ 4.3 แสดงการเปรียบเทียบสมการที่ใช้คูณเลขจำนวนจริงกับสมการมาตรฐาน

1. ส่วนที่ใช้ในการเก็บบิตเครื่องหมายหาได้จากการนำ $s1$ และ $s2$ มา Exclusive OR กัน
2. ส่วนที่ใช้เก็บข้อมูลของเลขส่วนยกกำลัง (Exponent Field) ส่วนนี้ได้มาจากการนำส่วนยกกำลังของ $V1(c1)$ และส่วนยกกำลังของ $V2(c2)$ มาทำการบวกเข้าด้วยกันจากนั้นก็นำ 127 มาลบออกจากผลลัพธ์ที่ได้

3. ส่วนที่ใช้ในการเก็บข้อมูลของเลขทศนิยม (Mantissa Field) ส่วนนี้ได้มาจากการนำส่วนทศนิยมของ $V1(1.f1)$ คูณกับส่วนที่เป็นทศนิยมของ $V2(1.f2)$

จากนั้นก็นำสมการที่ (4.9) มาทำการออกแบบโมดูลที่ใช้ในการคูณเลขจำนวนจริงลักษณะโครงสร้างภายในของโมดูลที่ใช้ในการคูณเลขจำนวนจริงดังแสดงในรูปที่ 4.4



รูปที่ 4.4 แสดงโครงสร้างภายในของโมดูลที่ใช้คูณเลขจำนวนจริง

จากรูปที่ 4.4 โครงสร้างของโมดูลจะประกอบไปด้วยส่วนอินพุตคือ ส่วนเครื่องหมายประกอบไปด้วย $s1$ และ $s2$ อินพุตส่วนยกกำลังประกอบไปด้วย $c1$ และ $c2$ และส่วนสุดท้ายคือส่วนเลขทศนิยมที่ประกอบไปด้วย $f1$ และ $f2$ เช่นดียวกับรูปที่ 4.1 ภายในโครงสร้างของโมดูลจะประกอบไปด้วยรีจิสเตอร์ $S1$ ถึง $S4$ เป็นรีจิสเตอร์ขนาด 1 บิตใช้สำหรับเก็บเครื่องหมาย $C1$ เป็นรีจิสเตอร์ขนาด 9 บิตใช้สำหรับเก็บผลลัพธ์ที่ได้จากการนำ $c1$ บวกกับ $c2$ และนอกจากนี้ $C2, C3$ เป็นรีจิสเตอร์ขนาด 8 บิตใช้เก็บข้อมูลของเลขยกกำลัง $F1$ และ $F2$ เป็นรีจิสเตอร์ขนาด 24

บิตใช้เก็บข้อมูลของส่วนเลขทศนิยมของ $f1$ และ $f2$ ตามลำดับ $F3$ เป็นรีจิสเตอร์ขนาด 48 บิตใช้สำหรับเก็บค่าผลลัพธ์จากการนำ $f1$ คูณกับ $f2$ แล้วนำผลลัพธ์ที่ได้จากการคูณมาทำการปรับค่าเพื่อเก็บไว้ใน $F4$ ซึ่งเป็นรีจิสเตอร์ขนาด 23 บิต จากโครงสร้างของโมดูลการคูณเลขจำนวนจริงสามารถแบ่งการทำงานออกเป็น 3 ขั้นตอนหลักๆ ดังต่อไปนี้

ขั้นตอนที่ 1 มีการทำงานดังนี้คือ

1. ทำการบวกค่าในส่วนยกกำลัง (Exponent) แล้วนำค่าผลลัพธ์ที่ได้ไปเก็บที่ไว้ใน $C1$ ซึ่งเป็นรีจิสเตอร์ขนาด 9 บิต
2. ทำการเพิ่ม 1's เข้าไปที่ตำแหน่งซ้ายสุดของ $f1$ และ $f2$ แล้วเก็บไว้ใน $F1$ และ $F2$ ซึ่งเป็นรีจิสเตอร์ขนาด 24 บิต
3. ทำการเก็บค่าส่วนที่เป็นบิตเครื่องหมายไว้ในรีจิสเตอร์ $S1$ และ $S2$

ขั้นตอนที่ 2 มีการทำงานดังนี้คือ

1. ใช้วิธีการคูณเลขแบบจำนวนเต็มโดยที่อินพุตคือ $1.f1$ และ $1.f2$ ผลลัพธ์ที่ได้จากการคูณนั้นให้นำไปเก็บไว้ในรีจิสเตอร์ $F3$ ซึ่งเป็นรีจิสเตอร์ขนาด 48 บิต
2. ทำการปรับค่าส่วนของการยกกำลัง (Exponent) โดยการลบด้วยค่าไบอัส (Bias)
3. ทำการเปรียบเทียบ $S1$ และ $S2$ ซึ่งเก็บบิตเครื่องหมาย ถ้าเหมือนกันจะเป็นบวกและเครื่องหมายต่างกันเป็นลบ ผลลัพธ์ที่ได้เก็บไว้ในรีจิสเตอร์ $S3$

ขั้นตอนที่ 3 มีการทำงานดังนี้คือ

1. ทำการปรับค่าผลลัพธ์ในส่วนที่เป็นทศนิยมให้เหมาะสมกับรีจิสเตอร์ $F4$
2. ผลลัพธ์ที่ได้คือ ส่วนที่เป็นเครื่องหมายจะถูกเก็บไว้ที่รีจิสเตอร์ $S4$ ส่วนที่เป็นเลขยกกำลังจะถูกเก็บไว้ที่รีจิสเตอร์ $C3$ และส่วนที่เป็นทศนิยมจะถูกเก็บไว้ในรีจิสเตอร์ $F4$

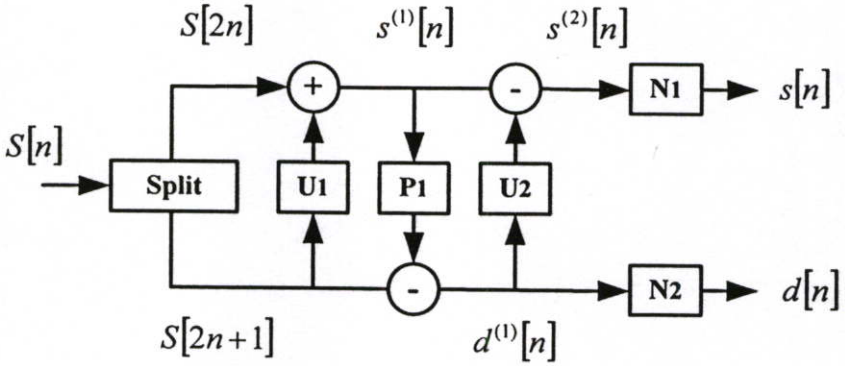
ในการนำโมดูลที่ใช้ในการบวกลบและคูณเลขจำนวนจริงไปใช้ในการออกแบบการแปลงเวฟเล็ตจะกล่าวถึงในภายหลัง

4.3 การออกแบบโมดูลแปลงเวฟเล็ตแบบเลขจำนวนจริง

ในหัวข้อนี้จะนำเสนอการออกแบบโมดูลแปลงเวฟเล็ตแบบเลขจำนวนจริง ในการออกแบบจะนำโมดูลที่ใช้ในการคำนวณเลขจำนวนจริงมาประยุกต์ใช้งานโดยนำโมดูลที่ใช้ในการคำนวณจริงมาจัดเรียงตามรูปสมการที่ใช้ในการแปลงเวฟเล็ต การออกแบบโมดูลแปลงเวฟเล็ตที่งานวิจัยนี้จะนำเสนอจะมีดังต่อไปนี้

4.3.1 การออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ตโดยใช้สมการ Daubechies4 (D4) ด้วยวิธี Lifting

การทำงานของ Daubechies4 (D4) ด้วยวิธี Lifting ดังแสดงในรูปที่ 4.5 หลักการทำงานเริ่มตั้งแต่ Split จะทำหน้าที่ในการแยกข้อมูลออกเป็นลำดับที่ $S[2n]$ และลำดับที่เป็น $S[2n+1]$ จากนั้นก็จะนำข้อมูลเหล่านี้มาผ่านการคำนวณทั้งหมด 4 ขั้นตอนดังแสดงในรูปที่ 4.5 ซึ่งจะมีรายละเอียดการทำงานดังต่อไปนี้



รูปที่ 4.5 แสดงการทำงานของ Daubechies4 (D4) วิธี Lifting

1. ทำการ Update Data ครั้งที่1 (U1) ด้วยสมการ (4.10)

$$s^{(1)}[n] = S[2n] + \sqrt{3}S[2n+1] \tag{4.10}$$

2. ทำการ Predict (P) ด้วยสมการ (4.11)

$$d^{(1)}[n] = S[2n+1] - \frac{1}{4}\sqrt{3}s^{(1)}[n] - \frac{\sqrt{3}-2}{4}s^{(1)}[n-1] \tag{4.11}$$

3. ทำการ Update Data ครั้งที่2 (U2) ด้วยสมการ (4.12)

$$s^{(2)} = s^{(1)}[n] - d^{(1)}[n+1] \tag{4.12}$$

4. ทำการ Normalize (N1, N2) ด้วยสมการ (4.13) และ (4.14)

$$s[n] = \frac{\sqrt{3}-1}{\sqrt{2}}s^{(2)}[n] \tag{4.13}$$

$$d[n]=\frac{\sqrt{3}+1}{\sqrt{2}}d^{(1)}[n] \tag{4.14}$$

ผลลัพธ์ที่ได้จากสมการที่ (4.13) และ (4.14) จะได้ค่าสัมประสิทธิ์ในการแปลงเวฟเล็ต ซึ่งประกอบไปด้วยค่าสัมประสิทธิ์จากตัวกรองความถี่ต่ำ (Approximation) และค่าสัมประสิทธิ์จากตัวกรองความถี่สูง (Detail) ตามลำดับ

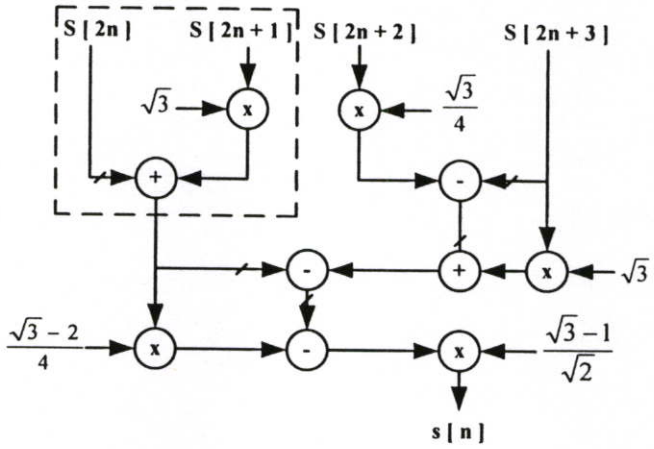
4.3.2 การจัดรูปสมการ Daubechies4 (D4) ด้วยวิธี Lifting เพื่อให้เหมาะสมกับโครงสร้างของ FPGA

การออกแบบโมดูลแปลงเวฟเล็ตโดยใช้สมการ Daubechies4 (D4) ด้วยวิธี Lifting พิจารณาสมการที่ (4.10) ถึง (4.14) จะประกอบไปด้วย 4 ขั้นตอนคือ Update Data ครั้งที่1 (U1) Predict (P) Update Data ครั้งที่2 (U2) Normalize (N1, N2) จากสมการดังกล่าวจำเป็นที่ต้องทำการจัดรูปให้เหมาะสมก่อนที่จะนำไปใช้ในการออก FPGA ซึ่งจะช่วยลดความซับซ้อนในการออกแบบ โดยการรวมสมการทั้งหมดเข้าด้วยกันเพื่อให้เหลือสมการเพียง 2 สมการ ดังแสดงในสมการ (4.15) เป็นสมการหาค่าสัมประสิทธิ์จากตัวกรองความถี่ต่ำ (Approximation) และสมการ (4.16) เป็นสมการใช้หาค่าสัมประสิทธิ์จากตัวกรองความถี่สูง (Detail) ตามลำดับ

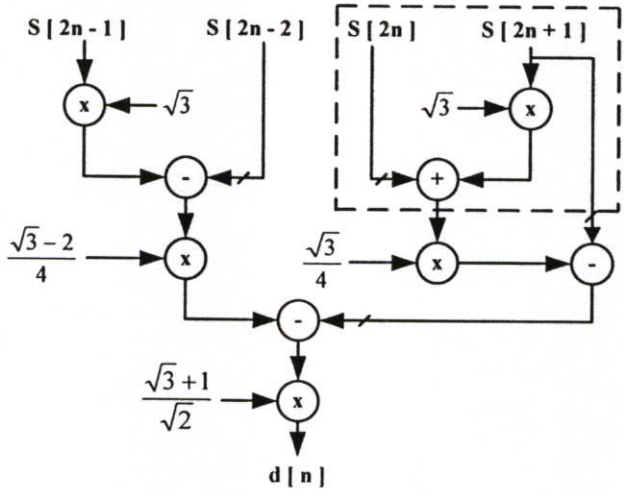
$$s[n]=\frac{\sqrt{3}-1}{\sqrt{2}}\left\{\left(S[2n]+\sqrt{3}S[2n+1]\right)-\begin{pmatrix} S[2n+3]-\frac{\sqrt{3}}{4}\left(S[2m+2]+\sqrt{3}S2n+3\right) \\ -\frac{\sqrt{3}-2}{4}\left(S[2n]+\sqrt{3}S[2n+1]\right)\end{pmatrix}\right\} \tag{4.15}$$

$$d[n]=\frac{\sqrt{3}+1}{\sqrt{2}}\left\{S[2n+1]-\frac{\sqrt{3}}{4}\left(S[2n]+\sqrt{3}[2n+1]\right)-\frac{\sqrt{3}-2}{4}\left(S[2n-2]+\sqrt{3}S[2n-1]\right)\right\} \tag{4.16}$$

จากนั้นก็นำสมการทั้งสองมาทำการออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ต ในการออกแบบจะนำโมดูลที่ใช้ในการบวก ลบ และคูณ เลขจำนวนจริงมาทำการจัดเรียงตามสมการที่ (4.15) และ (4.16) ผลลัพธ์ที่ได้ดังแสดงในรูปที่ 4.6 (a) และ 4.6 (b) ตามลำดับ



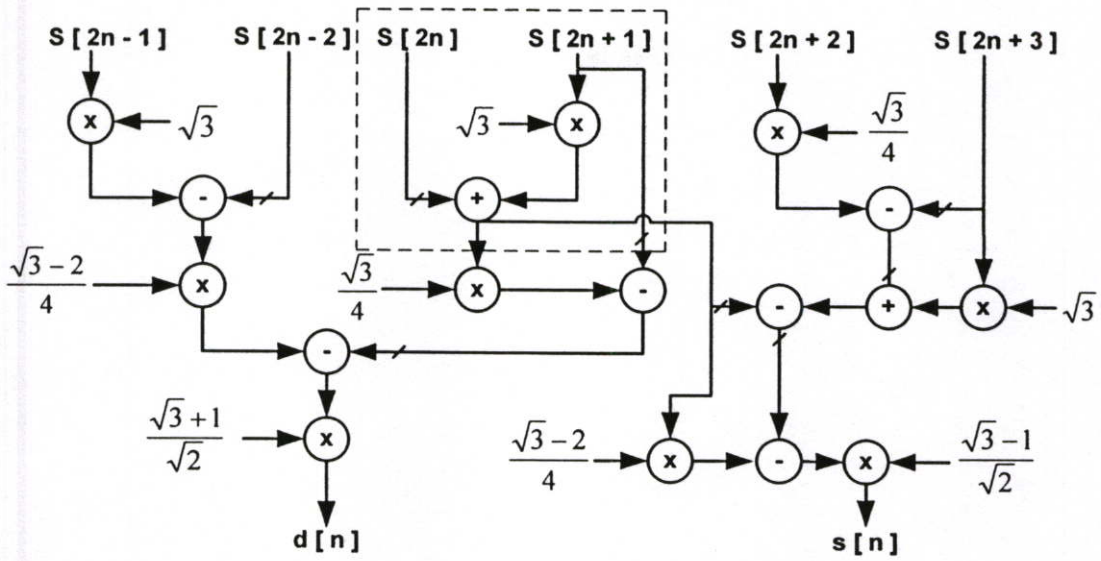
(a)



(b)

รูปที่ 4.6 แสดงโครงสร้างของโมดูลที่ใช้ในการหาค่าสัมประสิทธิ์
(a) จากตัวกรองความถี่ต่ำ (Approximation)
(b) จากตัวกรองความถี่สูง (Detail)

จากรูปที่ 4.6 (a) และ 4.6 (b) สังเกตได้ว่ามีส่วนที่ซ้อนทับกันอยู่ ดังแสดงในกรอบของรูปที่ 4.6 ทำให้เราสามารถที่จะนำสองส่วนนี้รวมเข้าด้วยกันเพื่อจะเป็นการประหยัดทรัพยากรของ FPGA ซึ่งได้ผลลัพธ์ดังรูปที่ 4.7



รูปที่ 4.7 แสดงโครงสร้างภายในของโมดูลแปลงเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting เมื่อนำส่วนกรองความถี่ต่ำ (Approximation) รวมกับส่วนกรองวคามถี่สูง (Detail)

ค่าคงที่ในสมการที่ (4.15) และ (4.16) จะถูกแปลงให้อยู่ในรูปของ IEEE 754 ดังแสดงในตารางที่ 4.1 เพื่อสะดวกแก่การนำไปใช้ในการออกแบบ FPGA ต่อไป

ตารางที่ 4.1 แสดงค่าคงที่ของโมดูลแปลงเวฟเล็ตในรูปแบบ IEEE 754 แบบ 32 บิต

ค่าคงที่	รูปแบบ IEEE 754
$\frac{\sqrt{3}-2}{4}$	1_01111011_00010010011000010100011
$\frac{\sqrt{3}}{4}$	0_01111101_10111011011001111010111
$\sqrt{3}$	0_01111111_10111011011001111010111
$\frac{\sqrt{3}-1}{\sqrt{2}}$	0_01111101_01110110110011110101111
$\frac{\sqrt{3}+1}{\sqrt{2}}$	0_01111111_01011101101100111101100

4.3.3 การออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ตโดยใช้ Scaling Function และ Wavelet Function แบบ Daubechies4 (D4)

การแปลงเวฟเล็ตวิธีนี้จะเป็นการนำค่าสัมประสิทธิ์ของเวฟเล็ตมาทำคอนโวลูชันกับข้อมูลอินพุตเพื่อหาค่าสัมประสิทธิ์ของผลลัพธ์จากการแปลงเวฟเล็ต ค่าสัมประสิทธิ์ที่ใช้ในการแปลงเวฟเล็ตประกอบด้วยค่าสัมประสิทธิ์จาก Scaling Function และ Wavelet Function ซึ่งค่าสัมประสิทธิ์ดังกล่าวที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) มีดังต่อไปนี้

ค่าสัมประสิทธิ์ของ Scaling Function ประกอบไปด้วย

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}} \tag{4.17}$$

$$h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}} \tag{4.18}$$

$$h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}} \tag{4.19}$$

$$h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}} \tag{4.20}$$

ค่าสัมประสิทธิ์ของ Wavelet Function ประกอบไปด้วย

$$g_0 = h_3 \tag{4.21}$$

$$g_1 = -h_2 \tag{4.22}$$

$$g_2 = h_1 \tag{4.23}$$

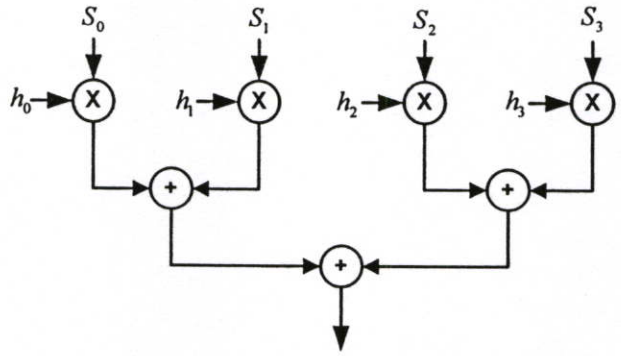
$$g_3 = -h_0 \tag{4.24}$$

เพื่อให้ง่ายต่อการทำความเข้าใจจึงนำค่าสัมประสิทธิ์ ที่ใช้ในการคำนวณมาเขียนให้อยู่ในรูปของแมทริกซ์ดังแสดงในรูปที่ 4.8

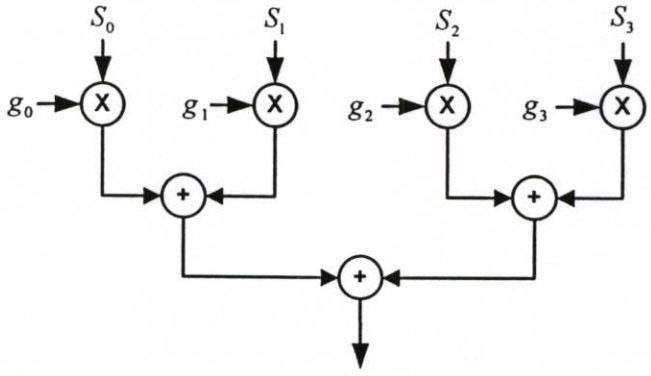
$$\begin{bmatrix} a_0 \\ c_0 \\ a_1 \\ c_1 \\ a_2 \\ c_2 \\ a_3 \\ c_3 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 \end{bmatrix} \bullet \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \end{bmatrix}$$

รูปที่ 4.8 แสดงแมทริกซ์ที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4)

จากนั้นนำโมดูลที่ใช้ในการคำนวณเลขจำนวนจริงที่ได้ทำการออกแบบไว้ในข้างต้นมาจัดเรียงดังรูปที่ 4.9 (a) และ 4.9 (b) เพื่อให้สามารถทำการคำนวณหาค่าสัมประสิทธิ์ของการแปลงเวฟเล็ตต่อไป และค่าสัมประสิทธิ์ต่างๆ ที่ใช้ในการแปลงเวฟเล็ตที่ถูกจัดให้อยู่ในรูป IEEE 754 ดังแสดงในตารางที่ 4.2



(a) โครงสร้างภายในของส่วนกรองความถี่ต่ำ



(b) โครงสร้างภายในของส่วนกรองความถี่สูง

รูปที่ 4.9 แสดงโครงสร้างภายในของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function

ตารางที่ 4.2 แสดงค่าคงที่ในโมดูลแปลงเวฟเล็ตในรูปแบบ IEEE 754 แบบ 32 บิต

ค่าคงที่	รูปแบบ IEEE 754
$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}$	0_01111101_11101110100011011101010
$h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}$	0_01111110_10101100010010111110-1111
$h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}$	0_01111100_11001011000010111111011
$h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}$	1_01111100_00001001000001111101101
$g_0 = \frac{1-\sqrt{3}}{4\sqrt{2}}$	1_01111100_00001001000001111101101
$g_1 = -\frac{3-\sqrt{3}}{4\sqrt{2}}$	1_01111100_11001011000010111111011
$g_3 = -\frac{1+\sqrt{3}}{4\sqrt{2}}$	1_01111101_11101110100011011101010

4.4 การออกแบบการแปลงเวฟเล็ตแบบเลขจำนวนเต็ม

การออกแบบการแปลงเวฟเล็ตแบบเลขจำนวนเต็มจะมีโครงสร้างคล้ายคลึงกับโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบเลขจำนวนจริง สิ่งที่แตกต่างอย่างเห็นได้ชัดก็คือ จะทำการเปลี่ยนโมดูลที่ใช้ในการคำนวณเลขจำนวนจริง ไปเป็นโมดูลที่ใช้ในการคำนวณเลขจำนวนเต็ม และนอกจากนี้ยังทำการปรับค่าสัมประสิทธิ์ที่ใช้ในการคำนวณให้อยู่ในรูปของเลขจำนวนเต็มเพื่อลดความซับซ้อนในการออกแบบ ในหัวข้อต่อไปจะแสดงการออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบเลขจำนวนเต็มที่ปรับปรุงมาจากการแปลงเวฟเล็ตแบบเลขจำนวนจริงที่นำเสนอไปในข้างต้น

4.4.1 การออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบเลขจำนวนเต็มโดยปรับปรุงจากสมการ Daubechies4 (D4) ด้วยวิธี Lifting

จากวิธีการแปลงสมการแปลงเวฟเล็ตแบบเลขจำนวนจริง เป็นการแปลงเวฟเล็ตแบบเลขจำนวนเต็มที่ได้นำเสนอไปในบทที่ 3 หัวข้อที่ 3.1.5 เมื่อนำมาใช้กับสมการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting มาทำการปรับปรุงเพื่อให้อยู่ในรูปการคำนวณแบบเลขจำนวนเต็มดังแสดงในสมการที่ (4.25) ถึง (4.29)

- ทำการ Update Data ครั้งที่1 (U1) ด้วยสมการ (4.25)

$$s^{(1)}[n] = S[2n] + \lfloor \sqrt{3} S[2n+1] \rfloor \tag{4.25}$$

- ทำการ Predict (P) ด้วยสมการ (4.26)

$$d^{(1)}[n] = S[2n+1] - \left\lfloor \frac{1}{4}\sqrt{3} s^{(1)}[n] \right\rfloor - \left\lfloor \frac{\sqrt{3}-2}{4} s^{(1)}[n-1] \right\rfloor \tag{4.26}$$

- ทำการ Update Data ครั้งที่2 (U2) ด้วยสมการ (4.27)

$$s^{(2)} = s^{(1)}[n] - d^{(1)}[n+1] \tag{4.27}$$

- ทำการ Normalize (N1, N2) ด้วยสมการ (4.28) ถึง (4.29)

$$s[n] = \left\lfloor \frac{\sqrt{3}-1}{\sqrt{2}} s^{(2)}[n] \right\rfloor \tag{4.28}$$

$$d[n] = \left\lfloor \frac{\sqrt{3}+1}{\sqrt{2}} d^{(1)}[n] \right\rfloor \tag{4.29}$$

โดยที่สัญญาลักษณ์ $\lfloor \ \rfloor$ จะหมายถึงการการตัดส่วนทศนิยมทิ้ง

4.4.2 การออกแบบค่าคงที่ในการคำนวณ Daubechies4 (D4) ด้วยวิธี Lifting ให้เหมาะสมกับโครงสร้างของ FPGA

เมื่อปรับสมการที่ใช้ในการแปลงเวฟเล็ตให้เป็นแบบเลขจำนวนเต็มดังแสดงในสมการที่ (4.25) ถึง (4.29) จะสังเกตเห็นว่ามีการคำนวณในลักษณะของเลขจำนวนจริงก่อน จากนั้นจึงทำการปัดค่าผลลัพธ์ที่ได้ให้อยู่ในรูปจำนวนเต็ม ทำให้ไม่เหมาะสำหรับนำไปใช้ในการออกแบบ FPGA เพราะว่าจะมีความซับซ้อนมาก ดังนั้นเพื่อให้เหมาะแก่การนำไปออกแบบ FPGA จึงต้องทำการปรับปรุงค่าคงที่ต่างๆ ที่ใช้ในสมการให้อยู่ในรูปของเลขจำนวนเต็มดังแสดงในตารางที่ 4.3 เพื่อให้เกิดความเหมาะสมในการนำไปใช้ในการออกแบบ FPGA

ตารางที่ 4.3 แสดงการปรับปรุงค่าคงที่สำหรับการคำนวณ

ค่าคงที่	ค่าที่ทำการปรับปรุงแล้ว
$\frac{\sqrt{3}-2}{4}$	$\frac{-2}{32}$
$\frac{\sqrt{3}}{4}$	$\frac{14}{32}$
$\sqrt{3}$	$\frac{55}{32}$
$\frac{\sqrt{3}-1}{4}$	$\frac{17}{32}$
$\frac{\sqrt{3}+1}{4}$	$\frac{62}{32}$

จากตารางที่ 4.3 จะเห็นได้ว่าค่าคงที่ค่าใหม่ทุกค่าจะถูกหารด้วย 32 หรือ 2^5 ซึ่งจะมีค่าเท่ากับการเลื่อนบิตข้อมูลไปทางขวาจำนวน 5 ครั้ง ดังนั้นจะได้รูปสมการที่ใช้ในการแปลงเวฟเล็ตแบบจำนวนเต็มที่ทำการปรับปรุงค่าคงที่ใหม่ดังแสดงในสมการที่ (4.30) ถึง (4.34)

- ทำการ Update Data ครั้งที่1 (U1) ด้วยสมการ (4.30)

$$s^{(1)}[n] = S[2n] + \left(S[2n+1] \times 14 >> 5\right) \tag{4.30}$$

- ทำการ Predict (P) ด้วยสมการ (4.31)

$$d^{(1)}[n] = S[2n+1] - \left\{\left(s^{(1)}[n] \times 14\right) >> 5\right\} - \left\{\left(s^{(1)}[n-1] \times -2\right) >> 5\right\} \tag{4.31}$$

- ทำการ Update Data ครั้งที่2 (U2) ด้วยสมการ (4.32)

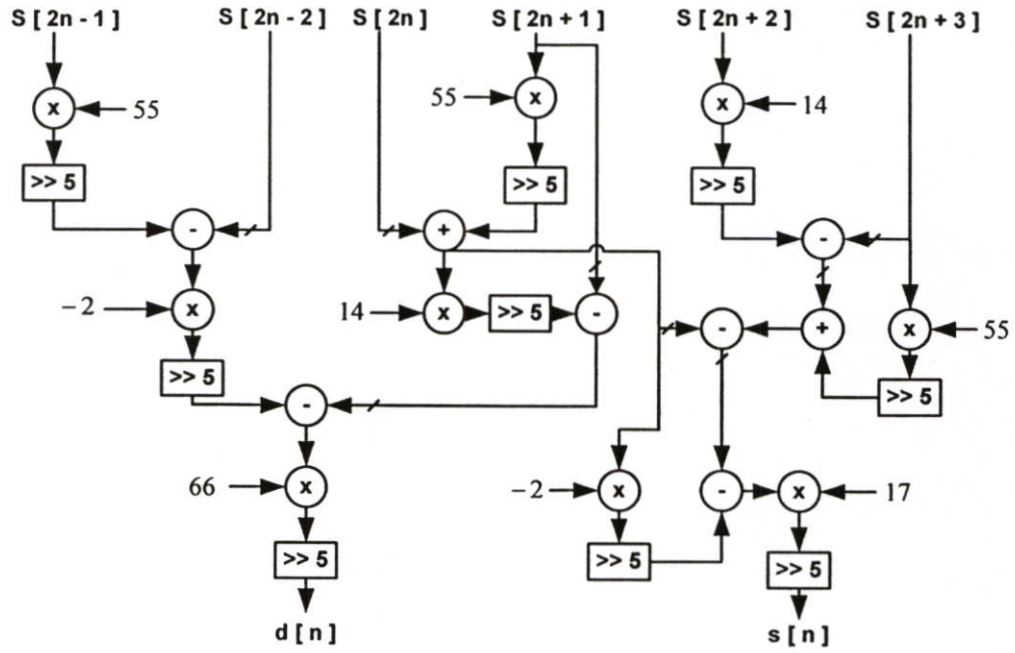
$$s^{(2)} = s^{(1)}[n] - d^{(1)}[n+1] \tag{4.32}$$

- ทำการ Normalize (N1, N2) ด้วยสมการที่ (4.33) และ (4.34)

$$s[n] = \left(s^{(2)}[n] \times 17\right) >> 5 \tag{4.33}$$

$$d[n] = \left(d^{(1)}[n] \times 62\right) >> 5 \tag{4.34}$$

จากนั้นนำโมดูลที่ใช้ในการแปลงเวฟเล็ตในรูปที่ 4.7 มาทำการปรับปรุงใหม่เพื่อให้เป็นโมดูลที่การแปลงเวฟเล็ตแบบเลขจำนวนเต็ม ดังแสดงในรูปที่ 4.10



รูปที่ 4.10 แสดงโครงสร้างของโมดูลแปลงเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนเต็ม

4.4.3 การออกแบบโมดูลที่ใช้แปลงเวฟเล็ตแบบเลขจำนวนเต็มโดยใช้ Scaling Function และ Wavelet Function แบบ Daubechies4 (D4)

การออกแบบโมดูลที่ใช้แปลงเวฟเล็ตแบบเลขจำนวนเต็มโดยใช้ค่าสัมประสิทธิ์จาก Scaling Function และ Wavelet Function แบบ Daubechies4 (D4) จะมีค่าสัมประสิทธิ์ที่ใช้ในการคำนวณมีดังต่อไปนี้

ค่าสัมประสิทธิ์ Scaling Function แบบเลขจำนวนเต็ม

$$h_0 = -8 \tag{4.35}$$

$$h_1 = 14 \tag{4.36}$$

$$h_2 = 54 \tag{4.37}$$

$$h_3 = 31 \tag{4.38}$$

ค่าสัมประสิทธิ์ Wavelet Function แบบเลขจำนวนเต็ม

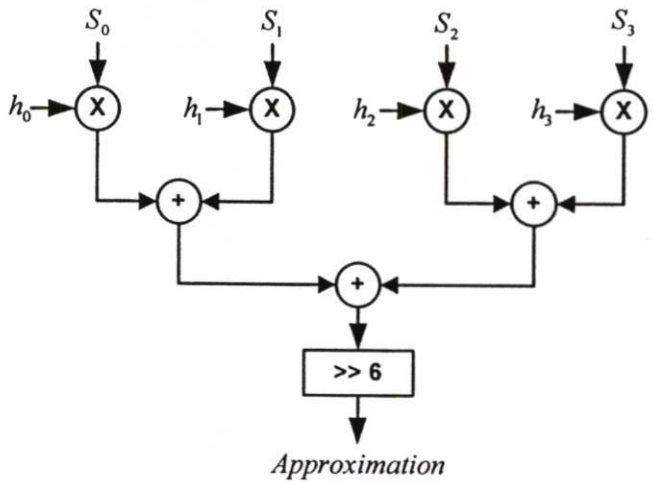
$$g_0 = h_3 \tag{4.39}$$

$$g_1 = -h_2 \tag{4.40}$$

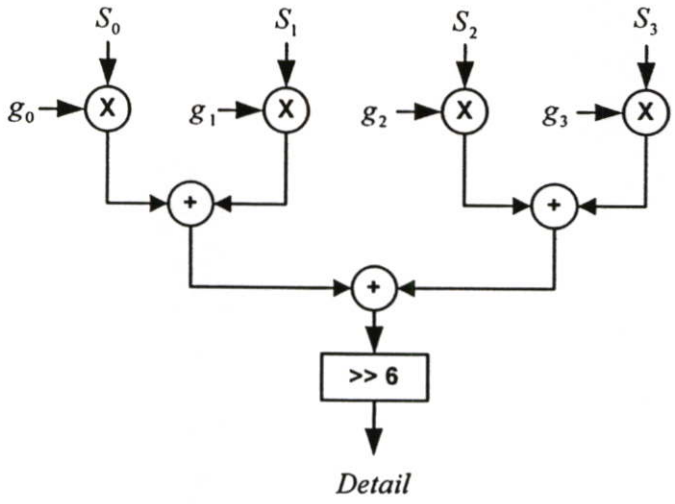
$$g_2 = h_1 \tag{4.41}$$

$$g_3 = -h_0 \tag{4.42}$$

หลังจากการคำนวณต้องทำการปรับค่าผลลัพธ์ที่ได้โดยหารด้วย 64 หรือ 2^6 นั้นก็คือ การเลื่อนบิตข้อมูลไปทางขวาจำนวน 6 ครั้ง โครงสร้างของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบเลขจำนวนเต็มดังแสดงในรูปที่ 4.9



(a) โครงสร้างภายในของส่วนกรองความถี่ต่ำ



(b) โครงสร้างภายในของส่วนกรองความถี่สูง

รูปที่ 4.11 แสดงโครงสร้างภายในของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบเลขจำนวนเต็มโดยใช้ Scaling Function และ Wavelet Function ของ Daubechies4 (D4)

บทที่ 5

ผลการทดลอง

ในบทนี้จะนำเสนอผลการทดลองที่ได้จากการทดสอบการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ที่ใช้สมการแบบ Lifting เปรียบเทียบกับการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) การเปรียบเทียบคุณสมบัติของการแปลงเวฟเล็ตจะพิจารณาจากค่า PSNR (Peek Signal to Noise Ratio) และ Zero Moment ซึ่งค่า PSNR เป็นค่าที่แสดงถึงประสิทธิภาพของการแปลงเวฟเล็ต ส่วน Zero Moment เป็นค่าที่บอกถึงความสามารถในการบีบอัดข้อมูล และเปรียบเทียบทรัพยากรที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตแต่ละแบบบน FPGA รวมถึงการพิจารณาความถี่สูงสุดของสัญญาณนาฬิกาที่โมดูลแปลงเวฟเล็ตแต่ละแบบสามารถทำงานได้ ในการทดสอบจะใช้ภาพแบบ Gray Scale 256 ระดับจำนวน 41 รูป

การทดสอบโมดูลที่ใช้ในการแปลงเวฟเล็ตจะใช้บอร์ด FPGA ที่มีชิพ Xc3s200 และ Xc3s1500 ของบริษัท Xilinx รวมทั้งโปรแกรม Modelsim ซึ่งใช้สำหรับจำลองการทำงาน และตรวจสอบความถูกต้องของโมดูล ในการทดสอบจะเริ่มจากการทดสอบโมดูลคำนวณเลขจำนวนจริงและเปรียบเทียบผลลัพธ์ที่ได้จากการคำนวณกับเครื่องคอมพิวเตอร์ เพื่อตรวจสอบความถูกต้องของผลลัพธ์ที่ได้ จากนั้นก็จะทำการทดสอบโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) แบบ Lifting และ Daubechies4 (D4) แบบ Matrix (Scaling Function และ Wavelet Function) ทั้งแบบเลขจำนวนจริงและเลขจำนวนเต็ม

5.1 การทดสอบโมดูลที่ใช้ในการคำนวณเลขจำนวนจริง

ในหัวข้อต่อไปจะกล่าวถึงผลการทดลองที่ได้จากการทดสอบโมดูลที่ใช้ในการคำนวณเลขจำนวนจริงที่ได้ทำการออกแบบเพื่อใช้ในโมดูลแปลงเวฟเล็ต ในการเปรียบเทียบผลลัพธ์จะเปรียบเทียบกับผลการคำนวณที่ได้จากเครื่องคอมพิวเตอร์ เพื่อตรวจสอบความถูกต้อง

5.1.1 แสดงตัวอย่างผลการทดลองที่ได้จากการทดสอบโมดูลบวกและลบเลขจำนวนจริง

หัวข้อนี้จะแสดงตัวอย่างผลการคำนวณที่ได้จากโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง การทดลองจะประกอบไปด้วยผลที่ได้จากการจำลองการทำงาน (Simulation) และทดลองใช้งานจริง เปรียบเทียบผลการทดลองที่ได้กับเครื่องคอมพิวเตอร์เพื่อทดสอบความถูกต้อง

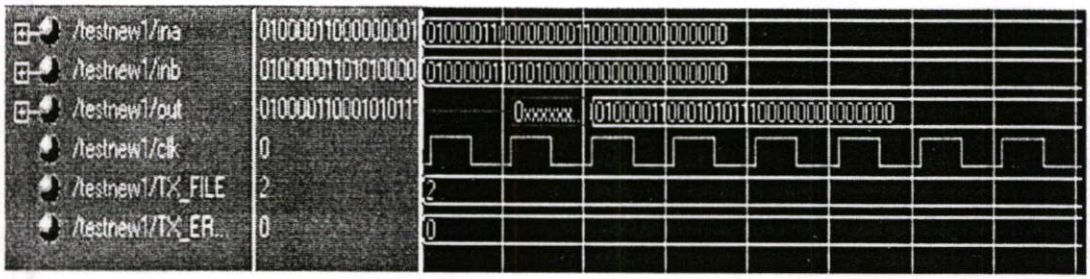
ina = 128.75 $(0\_10000110\_00000001100000000000000)_{\text{IEEE754 32 bit}}$

inb = 21 $(0\_10000011\_01010000000000000000000)_{\text{IEEE754 32 bit}}$

out = ina + inb

out = (128.75 + 21) = 149.75 $(0\_10000011\_01010000000000000000000)_{FPGA}$

$(0\_10000011\_01010000000000000000000)_{PC}$

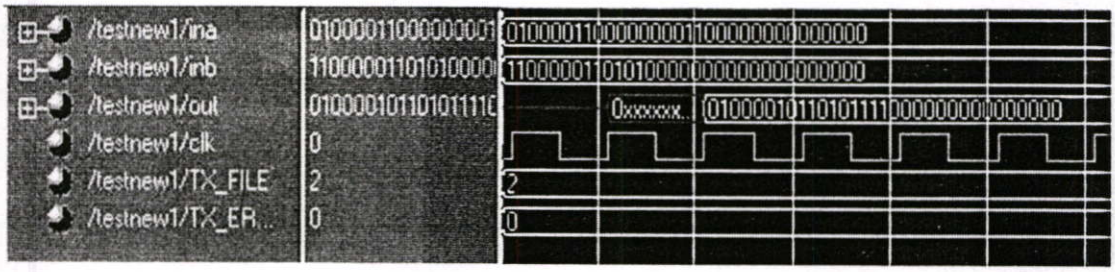


รูปที่ 5.1 แสดงผลลัพธ์ที่ได้จากการจำลองการทำงาน 128.75 + 21

out = ina - inb

out = (128.75 - 21) = 107.75 $(0\_10000101\_10101111000000000000000)_{FPGA}$

$(0\_10000101\_10101111000000000000000)_{PC}$



รูปที่ 5.2 แสดงผลลัพธ์ที่ได้จากการจำลองการทำงาน 128.75 - 21

5.1.2 แสดงตัวอย่างผลการทดลองที่ได้จากการทดสอบโมดูล

ข้อนี้จะแสดงตัวอย่างผลการคำนวณที่ได้จากโมดูลที่ใช้คูณเลขจำนวนจริง การทดลองจะประกอบไปด้วยผลที่ได้จากการจำลองการทำงาน (Simulation) และทดลองใช้งานจริง เปรียบเทียบผลการทดลองที่ได้กับเครื่องคอมพิวเตอร์เพื่อทดสอบความถูกต้อง

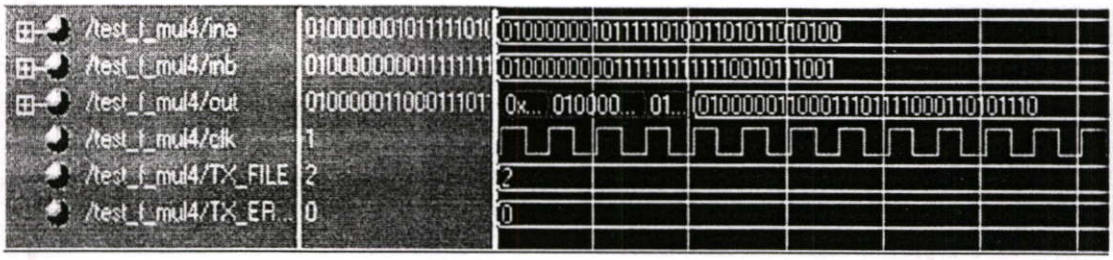
ina = 5.9564 $(0\_10000001\_01111101001101011010100)_{IEEE754\ 32\ Bit\ Format}$

inb = 2.9998 $(0\_10000000\_01111111111110010111001)_{IEEE754\ 32\ Bit\ Format}$

out = ina * inb

out = 17.86800872 (0_10000011_00011101111000110101110) $_{FPGA}$

(0_10000011_00011101111000110101110) $_{PC}$



รูปที่ 5.3 แสดงผลลัพธ์ที่ได้จากการจำลองการทำงาน 5.9564 * 2.9998

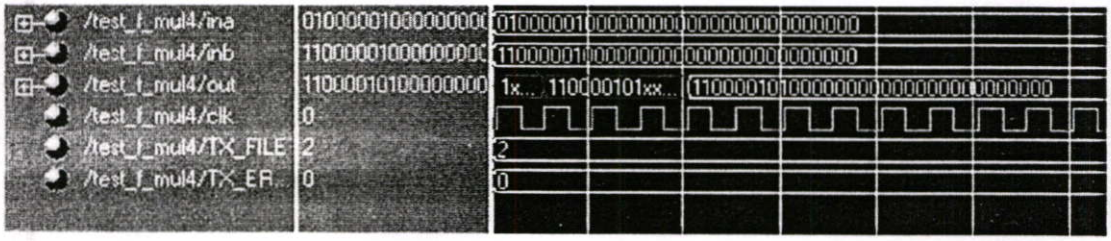
ina = 8 (0_10000010_00000000000000000000000) $_{IEEE754\ 32\ Bit\ Format}$

inb = -8 (1_10000010_00000000000000000000000) $_{IEEE754\ 32\ Bit\ Format}$

out = ina * inb

out = -64 (1_10000101_00000000000000000000000) $_{FPGA}$

(1_10000101_00000000000000000000000) $_{PC}$



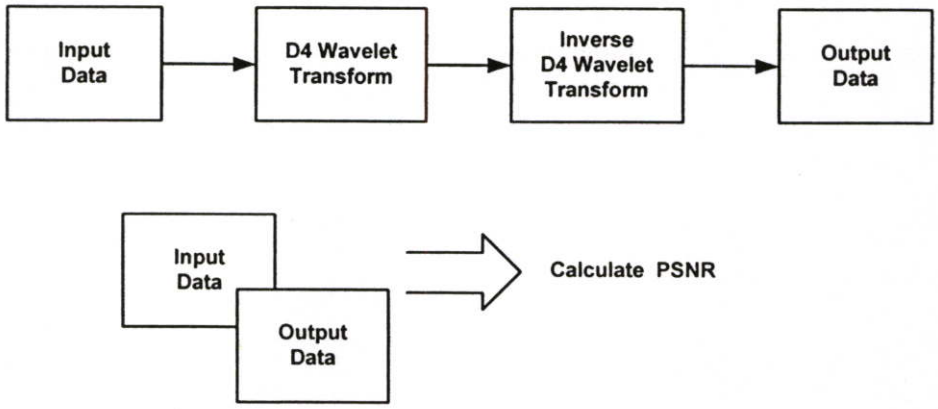
รูปที่ 5.4 แสดงผลลัพธ์ที่ได้จากการจำลองการทำงาน8 * -8

5.2 เปรียบเทียบผลการทดลองวัดประสิทธิภาพการแปลงเวฟเล็ตแต่ละอัลกอริธึม

ในหัวข้อนี้จะทำการทดลองวัดประสิทธิภาพของการแปลงเวฟเล็ตแต่ละอัลกอริธึม ในการตรวจสอบประสิทธิภาพจะวัดจากค่า PSNR และ Zero Moment ดังต่อไปนี้

5.2.1 ผลการทดลองวัดค่า PSNR ของการแปลงเวฟเล็ต

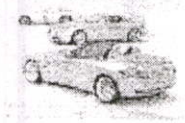
การทดลองวัดค่า PSNR เป็นการวัดประสิทธิภาพของอัลกอริธึมการแปลงเวฟเล็ตแต่ละแบบ ลักษณะการทดสอบค่า PSNR ดังแสดงในรูปที่ 5.5



รูปที่ 5.5 แสดงการตรวจสอบค่า PSNR

จากรูปที่ 5.5 ในการคำนวณค่า PSNR จะใช้ข้อมูลอินพุตมาเปรียบเทียบกับข้อมูลผลลัพธ์ที่ได้จากการแปลงกลับเวฟเล็ต โดยที่ค่า PSNR มีค่าสูงคุณภาพของข้อมูลผลลัพธ์ที่ได้ก็จะสูงตามไปด้วย ตัวอย่างค่า PSNR ที่ได้จากการทดสอบจากการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting และDaubechies4 (D4) ด้วยวิธี Scaling Function และ Wavelet Function ทั้งแบบเลขจำนวนจริงและเลขจำนวนเต็ม ดังแสดงในตารางที่ 5.1

ตารางที่ 5.1 แสดงค่า PSNR ที่วัดได้จากการแปลงเวฟเล็ตแบบต่างๆ

Images	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 1	142.514310	41.127935	140.369886	38.336610
 2	143.003939	41.216575	140.958432	38.816038
 3	148.130505	41.434482	146.218010	43.408084
 4	142.020652	41.220115	139.897484	37.603081

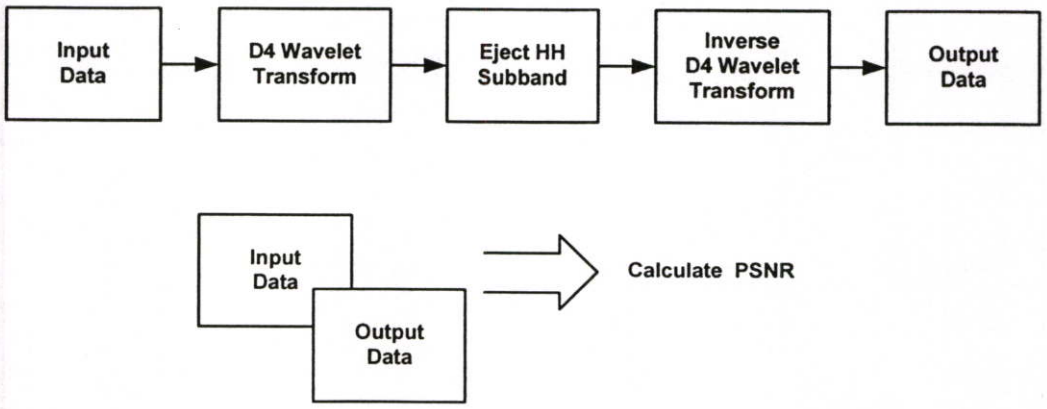
ตารางที่ 5.1 (ต่อ)

Images	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 5	143.196612	41.269923	141.795781	39.766048
 6	141.765803	41.179348	140.124935	37.897556
เฉลี่ย	**143.438636**	**41.241396**	**141.560754**	**39.304569**

จากตัวอย่างผลการทดลองที่แสดงในตารางที่ 5.3 แสดงบางส่วนของผลการทดลอง และสามารถดูผลการทดลองทั้ง 41 ภาพได้ในภาคผนวก ข จากผลการทดลองจะสังเกตเห็นว่าการแปลงเวฟเล็ตด้วยวิธี Lifting จะมีค่า PSNR สูงกว่าการใช้ Matrix (Scaling Function และ Wavelet Function) ไม่ว่าจะเป็นการแปลงเวฟเล็ตแบบเลขจำนวนจริงหรือเลขจำนวนเต็ม ดังนั้นการแปลงเวฟเล็ตแบบ Lifting จะมีคุณภาพสูงกว่า

5.2.2 ผลการทดลองวัดค่า PSNR ของการแปลงเวฟเล็ตเมื่อตัดส่วนที่เป็น HH ทิ้ง

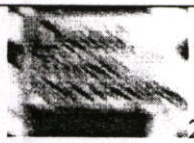
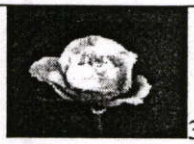
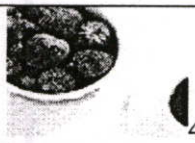
การทดลองนี้จะทำการวัดค่า PSNR เมื่อทำการตัดส่วนที่เป็น HH ทิ้งเพราะว่าข้อมูลในส่วนนี้มีความสำคัญน้อยที่สุด เมื่อตัดส่วนนี้ไปจะทำให้สามารถทำการบีบอัดข้อมูลได้มากขึ้น ในการทดลองจะดูประสิทธิภาพของการแปลงเวฟเล็ตแต่ละแบบว่ามีผลกระทบมากน้อยเพียงไรเมื่อไม่มีส่วน HH ลักษณะการทำงานดังแสดงในรูปที่ 5.6



รูปที่ 5.6 แสดงการตรวจสอบค่า PSNR เมื่อทำการตัดส่วน HH ทิ้ง

ลักษณะการทำงานของรูปที่ 5.6 เมื่อนำข้อมูลอินพุตมาผ่านการแปลงเวฟเล็ตจะทำการตัดค่าสัมประสิทธิ์การแปลงเวฟเล็ตในส่วนที่เป็น HH ทิ้งโดยการแทนข้อมูลในส่วนนี้ด้วยค่าศูนย์ หลังจากนั้นนำข้อมูลที่ทำการตัดส่วน HH มาแปลงกลับเพื่อใช้ในการคำนวณหาค่า PSNR ตัวอย่างค่า PSNR ที่ได้จากการทดลองดังแสดงในตารางที่ 5.2 (สามารถดูผลการทดลองของภาพทั้ง 41 ภาพได้ที่ภาคผนวก ข)

ตารางที่ 5.2 แสดงค่า PSNR ที่วัดได้จากการแปลงเวฟเล็ต เมื่อทำการตัดส่วนที่เป็น HH ทิ้ง

Images	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 1	48.832876	40.969776	48.832875	38.029352
 2	50.856877	41.526907	53.257624	38.706551
 3	44.409980	40.390641	44.409980	40.968991
 4	40.890989	38.330726	40.890989	35.974357
 5	29.126542	28.863702	29.126542	28.831425
 6	45.118549	40.164972	45.118549	37.177885
เฉลี่ย	**43.205968**	**38.374454**	**43.606093**	**36.614760**

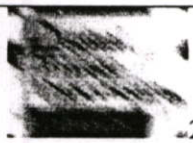
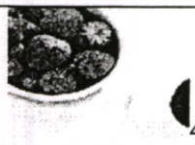
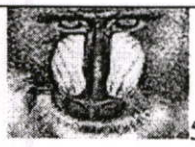
จากผลการทดลองเมื่อทำการตัดส่วน HH ทิ้งจะสังเกตเห็นว่าค่า PSNR ของการแปลงเวฟเล็ตแบบเลขจำนวนจริงทั้งสองแบบลดลงอย่างมาก เพราะว่าค่าสัมประสิทธิ์ของการแปลงเวฟเล็ตสูญหาย จนทำให้การแปลงเวฟเล็ตแบบเลขจำนวนจริงสูญเสียความเป็น PR (Prefect Reconstruction) ไป ส่วนการแปลงเวฟเล็ตแบบเลขจำนวนเต็มค่า PSNR จะลดลงมาไม่มากนักทั้งนี้

ก็ขึ้นอยู่กับรายละเอียดของภาพด้วยว่ามีข้อมูลในแนวทะแยงมุมมากน้อยเพียงไร ถ้ามีรายละเอียดในแนวทแยงมุมมากก็จะทำให้ค่า PSNR ต่ำลงมาก

5.2.3 ผลการทดลองวัดค่า PSNR เมื่อใช้ Zero Moment

การวัดค่า Zero Moment เป็นการนับจำนวนข้อมูลที่มีค่าเป็นศูนย์หรือ น้อยกว่าระดับที่ตั้งเอาไว้ (Threshold) ค่า Zero Moment เป็นค่าที่ใช้บอกประสิทธิภาพในการบีบอัดข้อมูลของการแปลงเวฟเล็ต โดยค่าที่วัดเป็นเปอร์เซ็นต์ของจำนวนข้อมูลทั้งหมดในภาพ ตัวอย่างผลการทดลองวัดค่า Zero Moment ดังแสดงในตารางที่ 5.3 และสามารถดูผลการทดลองของภาพทั้ง 41 ภาพได้ที่ภาคผนวก ค

ตารางที่ 5.3 แสดงค่า Zero Moment ในการแปลงเวฟเล็ต

Images	Zero Moment All (Threshold = 1.5)			
	D4 Lifting (%)	Integer D4 Lifting (%)	D4 Matrix (%)	Integer D4 Matrix (%)
 1	40.048177	18.268229	49.127604	23.290364
 2	38.626098	25.669860	56.159973	28.956604
 3	31.445598	36.089611	37.497615	32.655620
 4	33.878698	19.484574	42.644220	20.433947
 5	11.116027	7.0487976	8.3244323	6.8012237
 6	34.707222	18.201111	41.885833	19.643888
เฉลี่ย	**31.636970**	**20.793697**	**39.273279**	**21.963607**

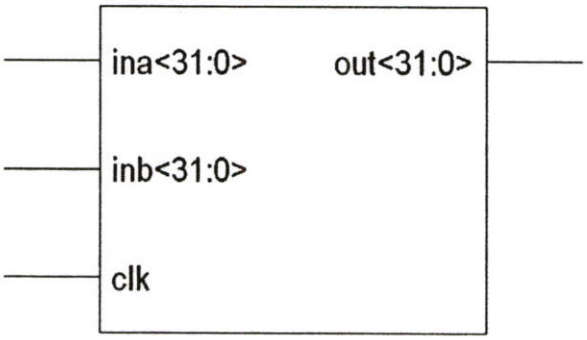
จากตารางที่ 5.3 จะสังเกตได้ว่าค่า Zero Moment จะขึ้นอยู่กับรายละเอียดข้องภาพที่ใช้ในการทดสอบถ้าภาพมีรายละเอียดมากจะทำให้ค่า Zero Moment ต่ำ แต่ว่าการแปลงเวฟเล็ตแบบเลขจำนวนจริงจะให้ค่า Zero Moment สูงกว่าการแปลงเวฟเล็ตแบบเลขจำนวนเต็ม

5.3 แสดงขนาดทรัพยากรของ FPGA ที่ใช้สำหรับโมดูลคำนวณเลขจำนวณจริง

ในหัวข้อนี้จะนำเสนอการตรวจสอบทรัพยากรณ์ที่ใช้การออกแบบโมดูลที่ใช้คำนวณเลขจำนวณจริงซึ่งจะประกอบด้วยหัวข้อดังต่อไปนี้

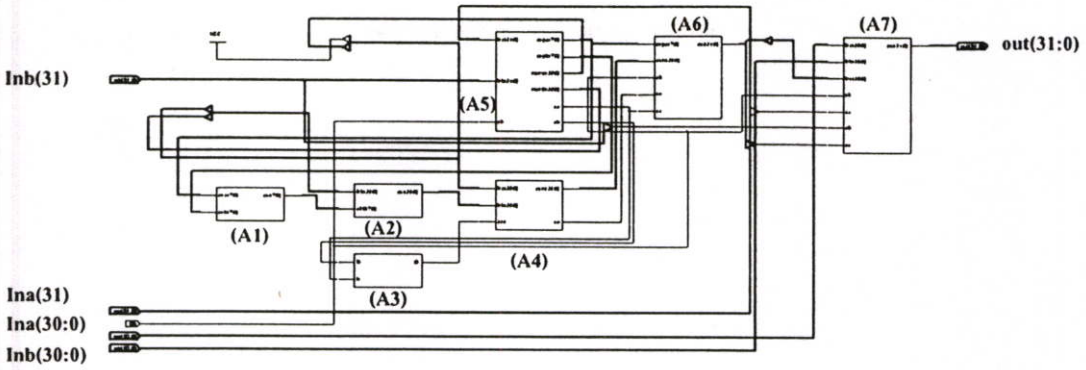
5.3.1 โมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง

โมดูลนี้ใช้สำหรับการบวกและลบเลขจำนวนจริง โดยอ้างอิงตามมาตรฐาน IEEE 754 ลักษณะของโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริงดังแสดงในรูปที่ 5.7



รูปที่ 5.7 ลักษณะของโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง

รูปที่ 5.7 จะประกอบไปด้วย ina และ inb ขนาด 32 บิต เป็นอินพุตของเลขจำนวนจริงที่ต้องการจะทำการบวกหรือลบกัน clk เป็นอินพุตของสัญญาณนาฬิกา และ out มีขนาด 32 บิตเป็นเอาท์พุตของผลลัพธ์ที่ได้จากการคำนวณ ลักษณะโครงสร้างของวงจรที่ใช้ในการบวกและลบเลขจำนวนจริงดังแสดงในรูปที่ 5.8



รูปที่ 5.8 แสดงโครงสร้างของวงจรที่ใช้ในการบวกและลบเลขจำนวนจริง

จากรูปที่ 5.8 จะประกอบไปด้วยโมดูลย่อยต่างๆ ที่ทำหน้าที่ดังต่อไปนี้

A1: เป็นโมดูลที่ทำหน้าที่ลบส่วนยกกำลัง

A2: เป็นโมดูลที่ทำหน้าที่เลื่อนบิตข้อมูล

A3: เป็นโมดูลที่ทำหน้าที่ตรวจสอบบิตเครื่องหมาย

A4: เป็นโมดูลที่ทำหน้าที่บวกหรือลบส่วนทศนิยม

A5: เป็นโมดูลที่ทำหน้าที่ตรวจสอบขนาดของอินพุตทั้งสอง

A6: เป็นโมดูลที่ทำหน้าที่ปรับค่าส่วนยกกำลัง

A7: เป็นโมดูลที่ทำหน้าที่ตรวจสอบผลลัพธ์เมื่อผลลัพธ์ที่ได้มีค่าเป็นศูนย์

โปรแกรมที่ใช้ในการบวกหรือลบเลขจำนวนจริงที่ถูกพัฒนาด้วยภาษา Verilog ดังแสดงในภาคผนวก ง.1

ในการทดสอบจะใช้ชิพ FPGA Spantan3 Xc3s200tq144-4 ของบริษัท Xilinx ผลการ Synthesis ดังแสดงในตารางที่ 5.4

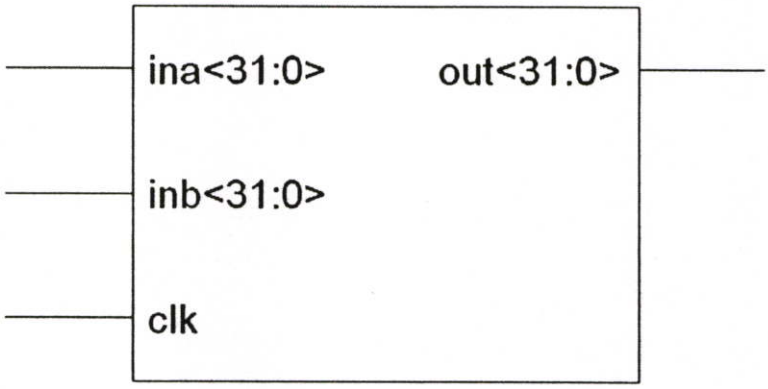
ตารางที่ 5.4 แสดงทรัพยากรที่ใช้ในการออกแบบโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง

	จำนวนทั้งหมด	จำนวนที่ใช้ไป	เปอร์เซ็นต์
Slices	1920	353	18.385%
Flip Flops	3840	132	3.437%
4 input LUT	3840	657	17.109%
GCLK	8	1	12.500%

ความถี่สูงสุดที่วงจรสามารถทำงานได้คือ 56.218MHz (Maximum Frequency: 56.218MHz)

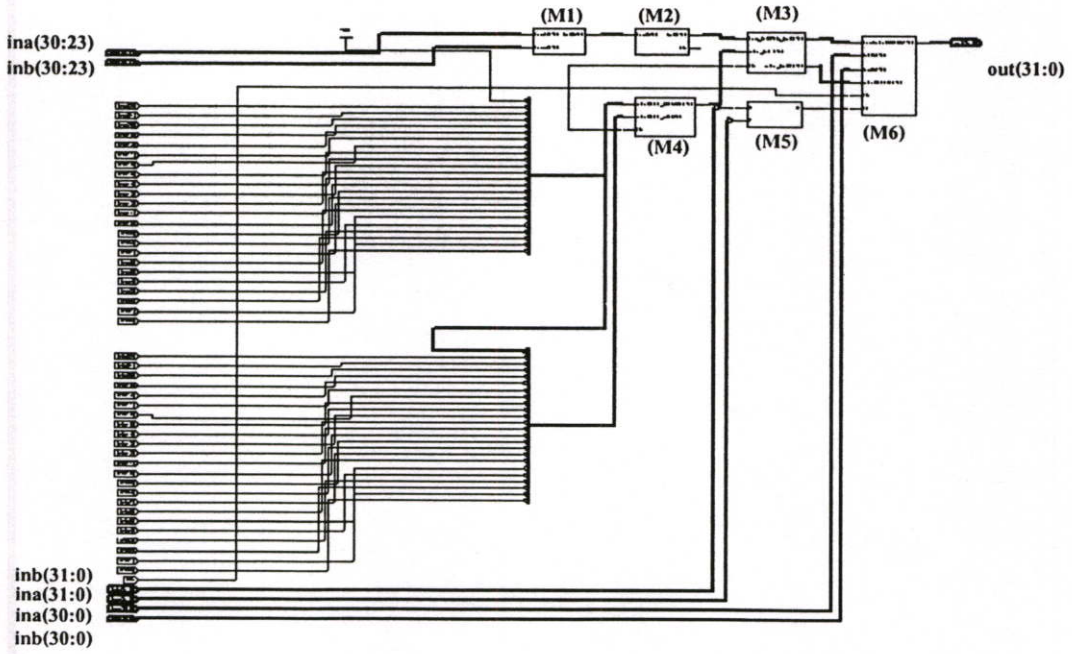
5.3.2 การตรวจสอบโมดูลที่ใช้ในการคูณเลขจำนวนจริง

โมดูลนี้ใช้สำหรับการคูณเลขจำนวนจริง โดยอ้างอิงตามมาตรฐาน IEEE 754 ลักษณะของโมดูลที่ใช้ในการคูณเลขจำนวนจริงดังแสดงในรูปที่ 5.8



รูปที่ 5.9 แสดงลักษณะของโมดูลที่ใช้ในการคูณเลขจำนวนจริง

รูปที่ 5.9 จะประกอบไปด้วย ina และ inb ขนาด 32 บิต เป็นอินพุตของเลขจำนวนจริงที่ต้องการจะคูณกัน clk เป็นอินพุตของสัญญาณนาฬิกา และ out มีขนาด 32 บิตเป็นเอาท์พุตของผลลัพธ์ที่ได้จากการคำนวณ ลักษณะโครงสร้างของวงจรที่ใช้ในการคูณเลขจำนวนจริง ดังแสดงในรูปที่ 5.10



รูปที่ 5.10 แสดงโครงสร้างของวงจรที่ใช้ในการคูณเลขจำนวนจริง

จากรูปที่ 5.10 จะประกอบไปด้วยโมดูลย่อยต่างๆ ที่ทำหน้าที่ดังต่อไปนี้

M1: เป็นโมดูลที่ทำหน้าที่บวกส่วนยกกำลัง

M2: เป็นโมดูลที่ทำหน้าที่ลบส่วนยกกำลังด้วย 127

M3: เป็นโมดูลที่ทำหน้าปรับค่าส่วนยกกำลัง

M4: เป็นโมดูลที่ทำหน้าที่คูณส่วนทศนิยม

M5: เป็นโมดูลที่ทำหน้าที่ตรวจสอบบิตเครื่องหมาย

M6: เป็นโมดูลที่ทำหน้าที่ตรวจสอบผลลัพธ์เมื่อผลลัพธ์ที่ได้มีค่าเป็นศูนย์

โปรแกรมที่ใช้ในการคูณจำนวนจริงที่ถูกพัฒนาด้วยภาษา Verilog ดังแสดงในภาคผนวก ง.2

ในการทดสอบจะใช้ชิพ FPGA Spantan3 Xc3s200tq144-4 ของบริษัท Xilinx ผลการ Synthesis ดังแสดงในตารางที่ ตารางที่ 5.5

ตารางที่ 5.5 แสดงการใช้ทรัพยากรทั้งหมดที่ใช้ในการออกแบบโมดูลคูณเลขจำนวนจริง

	จำนวนทั้งหมด	จำนวนที่ใช้ไป	เปอร์เซ็นต์
Slices	1920	357	18.593%
Flip Flops	3840	182	4.739%
4 input LUT	3840	671	16.692%
MULT18X18	12	4	33.333%
GCLK	8	1	12.500%

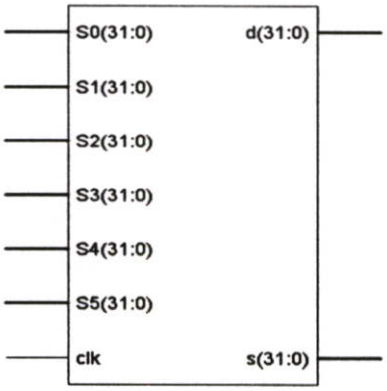
ความถี่สูงสุดที่วงจรสามารถทำงานได้คือ 68.055MHz (Maximum Frequency: 68.055MHz)

5.4 การเปรียบเทียบขนาดทรัพยากรของ FPGA ที่ใช้สำหรับโมดูลแปลงเวฟเล็ตแต่ละอัลกอริธึม

หัวข้อนี้จะเปรียบเทียบทรัพยากรของ FPGA ที่ใช้ในการออกแบบการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting และการแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function ที่มีรูปแบบการคำนวณเป็นเลขจำนวณจริงและเลขจำนวนเต็ม

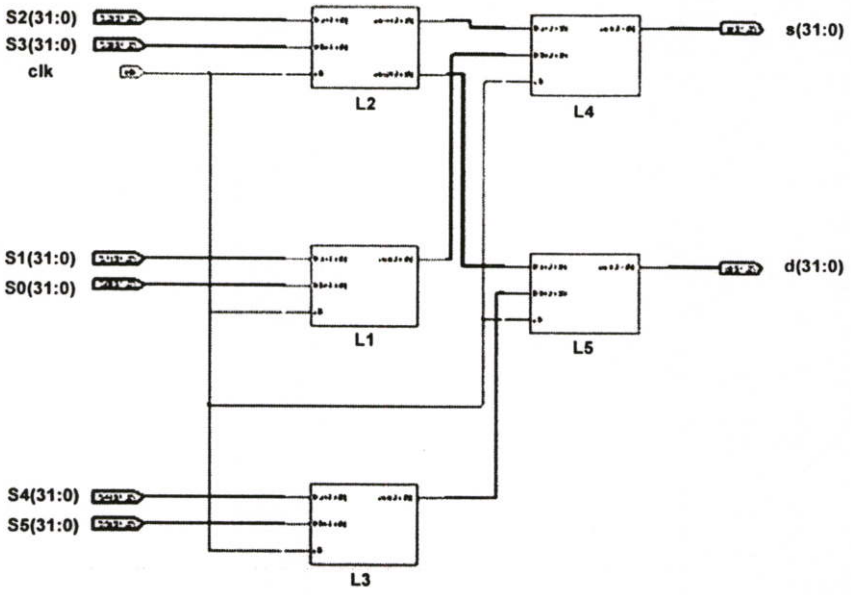
5.4.1 โมดูลที่ใช้ในการแปลงเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนจริง

เมื่อนำโมดูลที่ใช้ในการบวกลบ และคูณเลขจำนวนจริงมาจัดเรียงสมการ Daubechues4 (D4) ด้วยวิธีการของ Lifting โครงสร้างภายนอกของโมดูลดังแสดงอยู่ในรูปที่ 5.11



รูปที่ 5.11 แสดงโครงสร้างภายนอกของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนจริง

จากรูปที่ 5.11 อินพุตจะประกอบไปด้วย S0 ถึง S5 แต่ละตัวมีขนาด 32 บิตเป็นส่วนรับข้อมูลที่ต้องการจะทำการแปลงเวฟเล็ต และ clk เป็นอินพุตของสัญญาณนาฬิกา เอาท์พุตจะประกอบด้วย s มีขนาด 32 บิตใช้แสดงผลลัพธ์ที่ได้จากตัวกรองความถี่ต่ำ (Approximation) และ d มีขนาด 32 บิตใช้แสดงผลลัพธ์ที่ได้จากตัวกรองความถี่สูง (Detail) ลักษณะโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนจริง ดังแสดงในรูปที่ 5.12



รูปที่ 5.12 แสดงโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนจริง

จากรูปที่ 5.12 จะประกอบไปด้วยโมดูลย่อยต่างๆ ที่ทำหน้าที่ดังต่อไปนี้

L1: $out = \frac{\sqrt{3}-2}{4}\left(inb - \sqrt{3}ina\right)$

L2: $out1 = inb - \sqrt{3}\left(ina + \sqrt{3}inb\right)$

$out2 = ina + \sqrt{3}inb$

L3: $out = \left(inb - \frac{\sqrt{3}}{4}inb\right) + \sqrt{3}inb$

L4: $out = \frac{\sqrt{3+1}}{\sqrt{2}}\left(inb - ina\right)$

L5: $out = \frac{\sqrt{3}-1}{\sqrt{2}}\left[\left(ina - inb\right) - \frac{\sqrt{3}-2}{4}ina\right]$

โปรแกรมที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนจริงที่ถูกพัฒนาด้วยภาษา Verilog ดังแสดงในภาคผนวก ง.3

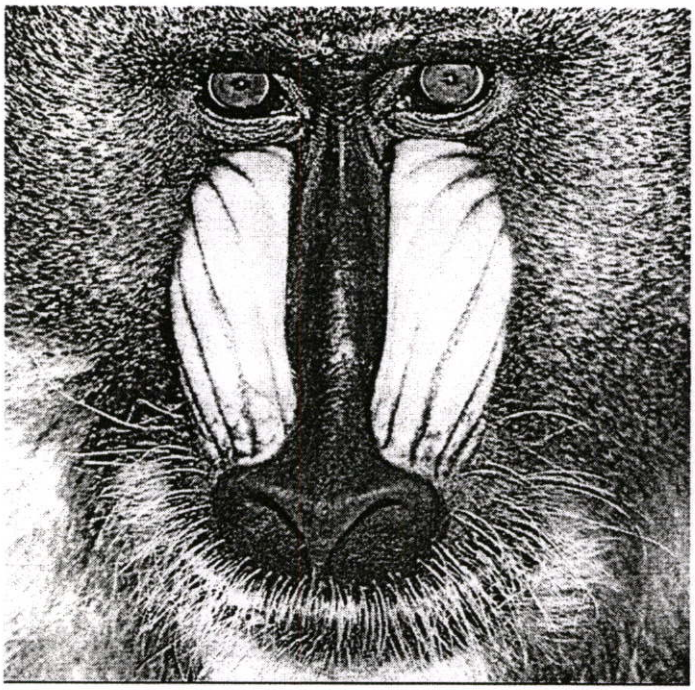
ในการทดสอบจะใช้ชิพ FPGA Spantan3 Xc3s1500tq456-4 ของบริษัท Xilinx ผลการ Synthesis ดังแสดงในตารางที่ ตารางที่ 5.6

ตารางที่ 5.6 แสดงการใช้ทรัพยากรทั้งหมดที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนจริง

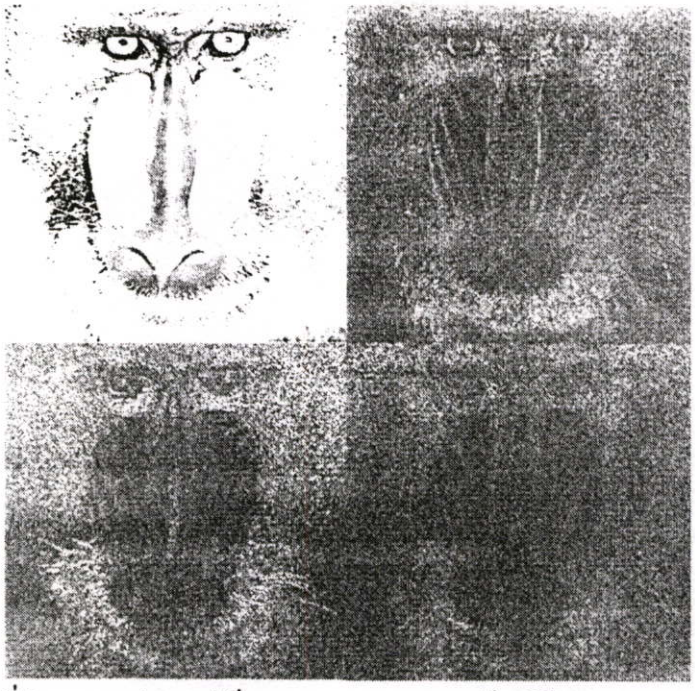
	จำนวนทั้งหมด	จำนวนที่ใช้ไป	เปอร์เซ็นต์
Slices	13312	5659	42.510%
Flip Flops	26624	2493	9.363%
4 input LUT	26624	10553	39.637%
MULT18X18	32	36	112.500%*
GCLK	8	1	12.500%

ความถี่สูงสุดที่วงจรสามารถทำงานได้คือ 41.771MHz (Maximum Frequency: 41.771MHz)

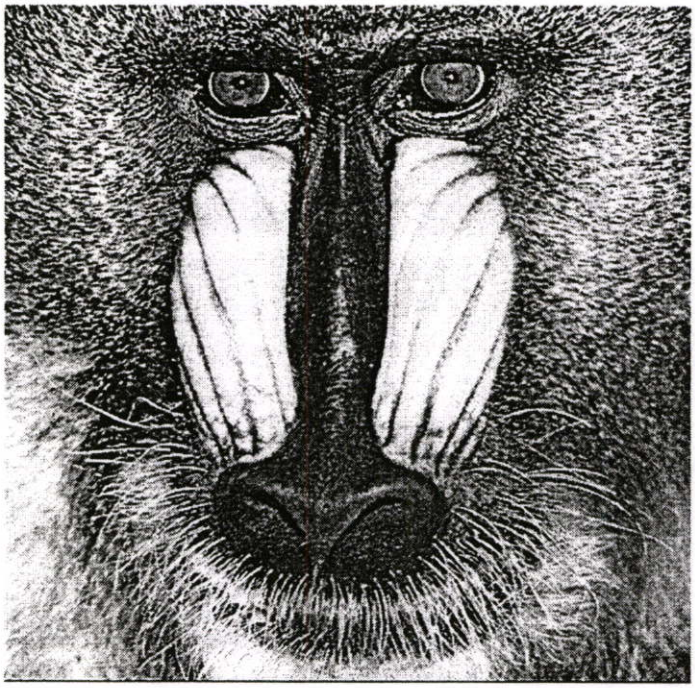
ตัวอย่างภาพผลการทดลองที่ได้การแปลงเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting ดังแสดงในรูปที่ 5.13



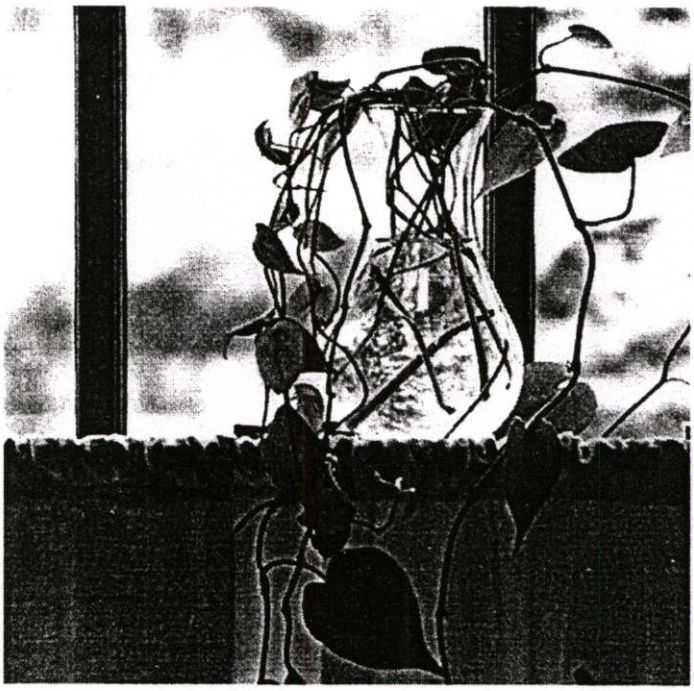
(a) รูป Baboon ต้นฉบับ



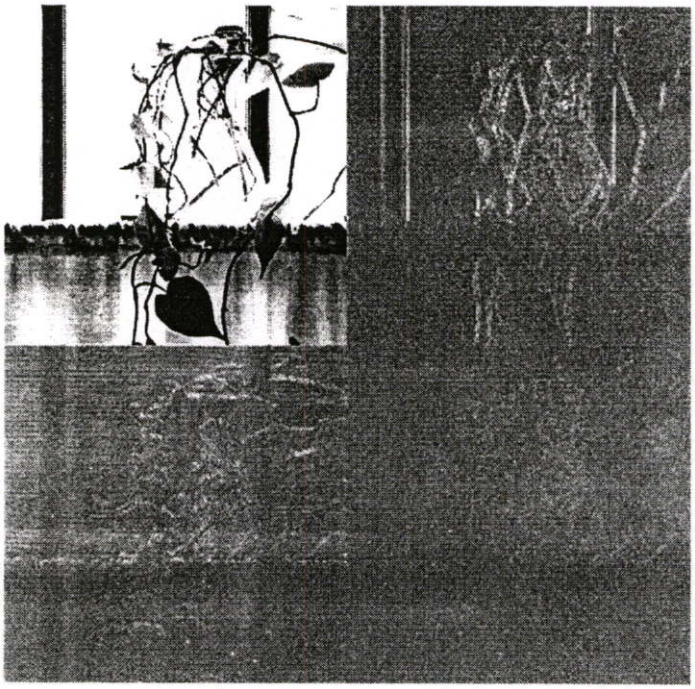
(b) รูป Baboon ที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนจริง



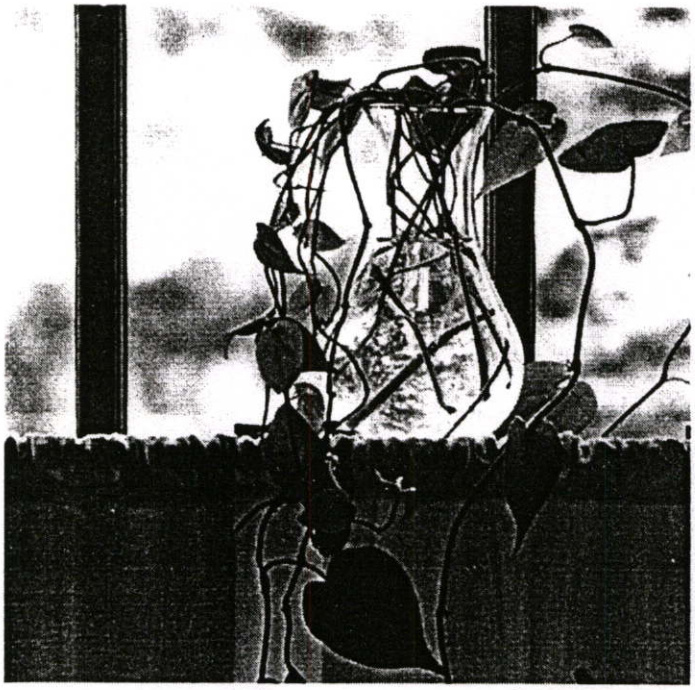
(c) รูป Baboon ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนจริง



(d) รูป Vase ต้นฉบับ



(e) รูป Vaseที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนจริง

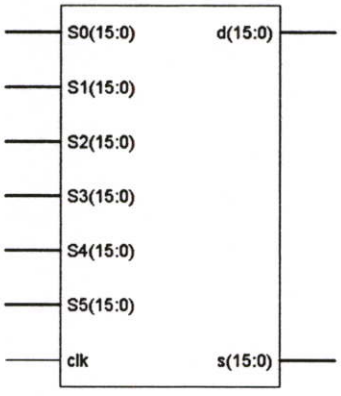


(c) รูป Vase ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนจริง

รูปที่ 5.13 แสดงภาพตัวอย่างผลการทดลองที่ได้จากโมดูล Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนจริง

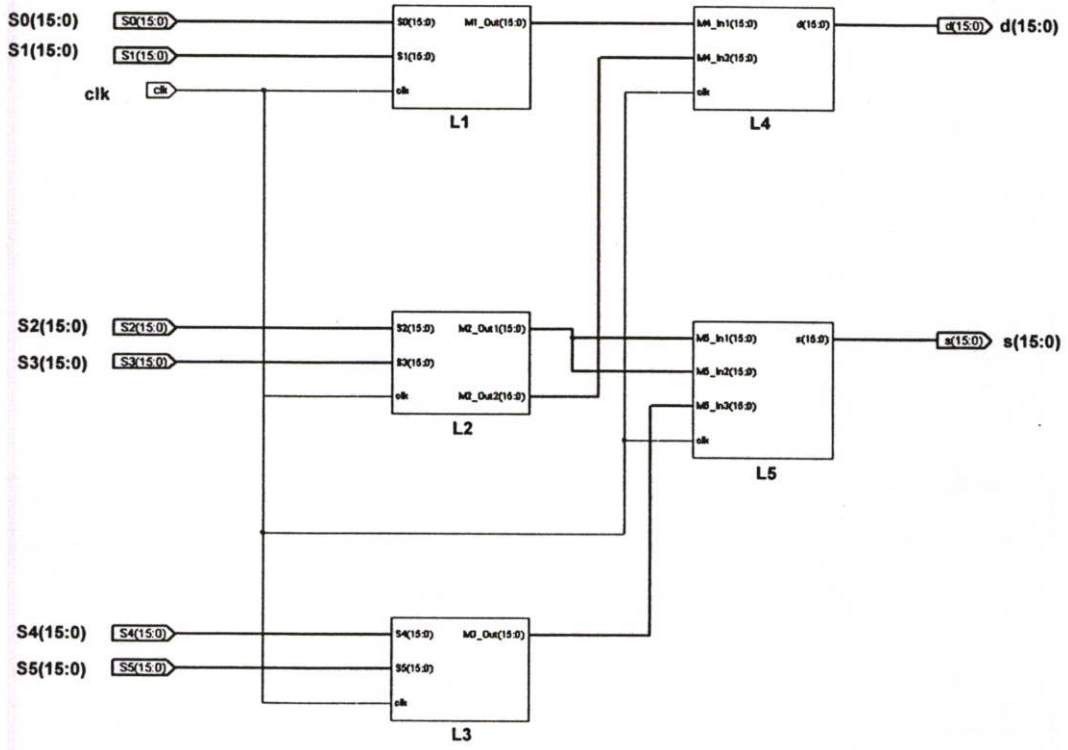
5.4.2 โมดูลที่ใช้ในการแปลงเวฟเล็ต Daubechies4 (D4) วิธี Lifting แบบเลขจำนวนเต็ม

การปรับปรุงสมการที่ใช้ในการแปลงเวฟเล็ต Daubechies4 วิธี Lifting ให้อยู่ในรูปการคำนวณแบบเลขจำนวนเต็มจะมีโครงสร้างภายนอกของโมดูลดังแสดงอยู่ในรูปที่ 5.14



รูปที่ 5.14 แสดงโครงสร้างภายนอกของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนเต็ม

จากรูปที่ 5.14 อินพุตจะประกอบไปด้วย S0 ถึง S5 แต่ละตัวมีขนาด 16 บิตเป็นส่วนรับข้อมูลที่ต้องการจะทำการแปลงเวฟเล็ต และ clk เป็นอินพุตของสัญญาณนาฬิกา เอาท์พุตจะประกอบด้วย s มีขนาด 16 บิตใช้แสดงผลลัพธ์ที่ได้จากตัวกรองความถี่ต่ำ (Approximation) และ d มีขนาด 16 บิตใช้แสดงผลลัพธ์ที่ได้จากตัวกรองความถี่สูง (Detail) ลักษณะโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนเต็ม ดังแสดงในรูปที่ 5.15



รูปที่ 5.15 แสดงโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนเต็ม

จากรูปที่ 5.15 จะประกอบไปด้วยโมดูลย่อยต่างๆ ที่ทำหน้าที่ดังต่อไปนี้

L1: $out = \left[\left(inb - \left(ina \times 55\right) >> 5\right) \times -2\right] >> 5$

L2: $out1 = inb - \left[\left(ina + \left(inb \times 55\right) >> 5\right) \times 55\right] >> 5$

$out2 = ina + \left(inb \times 55\right) >> 5$

L3: $out = \left(inb - \left(inb \times 14\right) >> 5\right) + \left(inb \times 55\right) >> 5$

L4: $out = \left(\left(inb - ina\right) \times 62\right) >> 5$

L5: $out = \left[\left(\left(ina - inb\right) - \left(ina \times -2\right) >> 5\right) \times 17\right] >> 5$

โปรแกรมที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนเต็มที่ถูกพัฒนาด้วยภาษา Verilog ดังแสดงในภาคผนวก ง.4

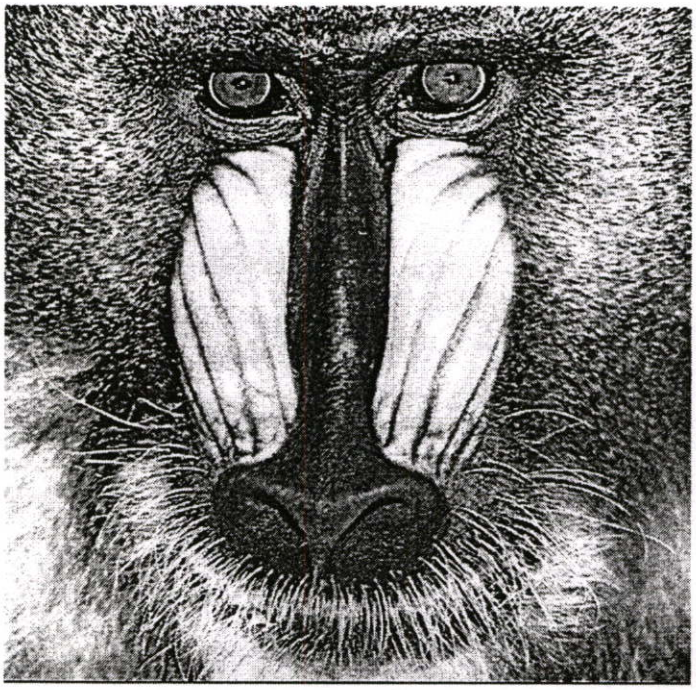
ในการทดสอบจะใช้ชิพ FPGA Spantan3 Xc3s1500tq456-4 ของบริษัท Xilinx ผลการ Synthesis ดังแสดงในตารางที่ ตารางที่ 5.7

ตารางที่ 5.7 แสดงการใช้ทรัพยากรทั้งหมดที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วยวิธีการของ Lifting แบบเลขจำนวนเต็ม

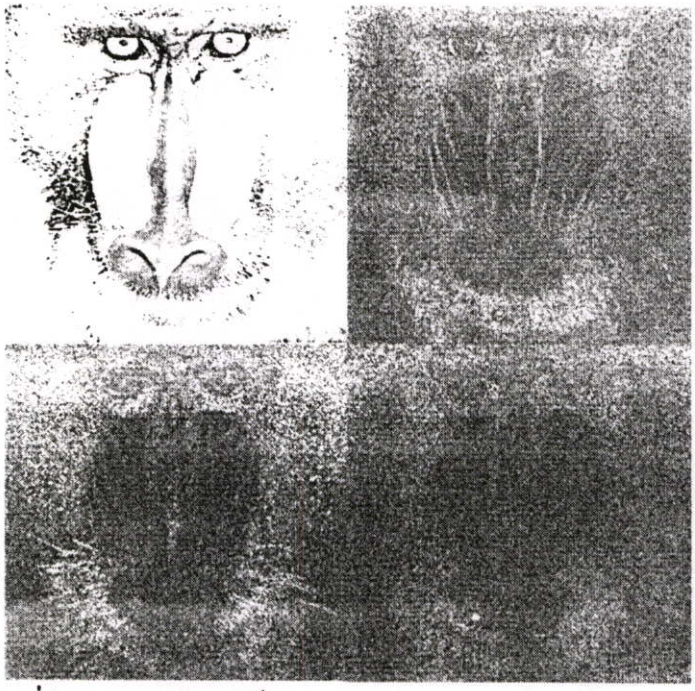
	จำนวนทั้งหมด	จำนวนที่ใช้ไป	เปอร์เซ็นต์
Slices	13312	438	3.290%
Flip Flops	26624	227	0.852%
4 input LUT	26624	623	2.339%
MULT18X18	32	9	28.125%
GCLK	8	1	12.500%

ความถี่สูงสุดที่วงจรสามารถทำงานได้คือ 139.082MHz (Maximum frequency: 139.082MHz)

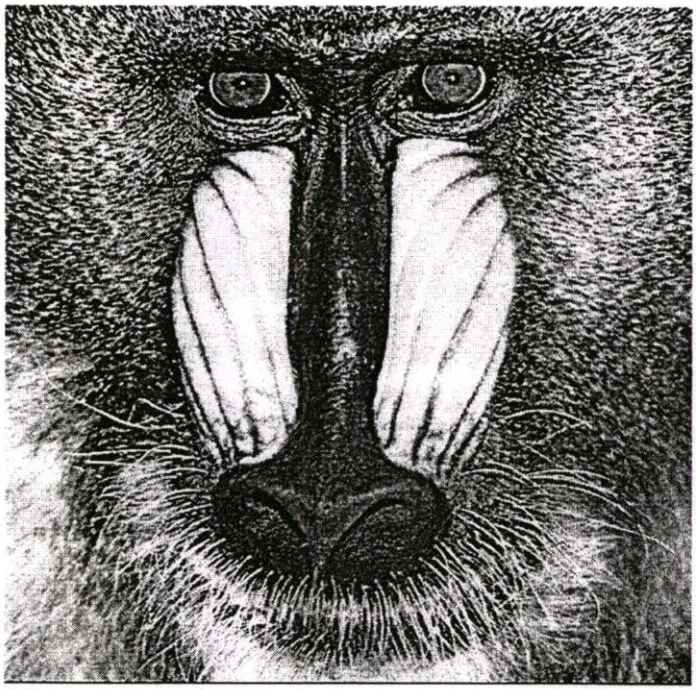
ตัวอย่างผลการทดลองที่ได้ Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนเต็มดังแสดงในรูปที่ 5.16



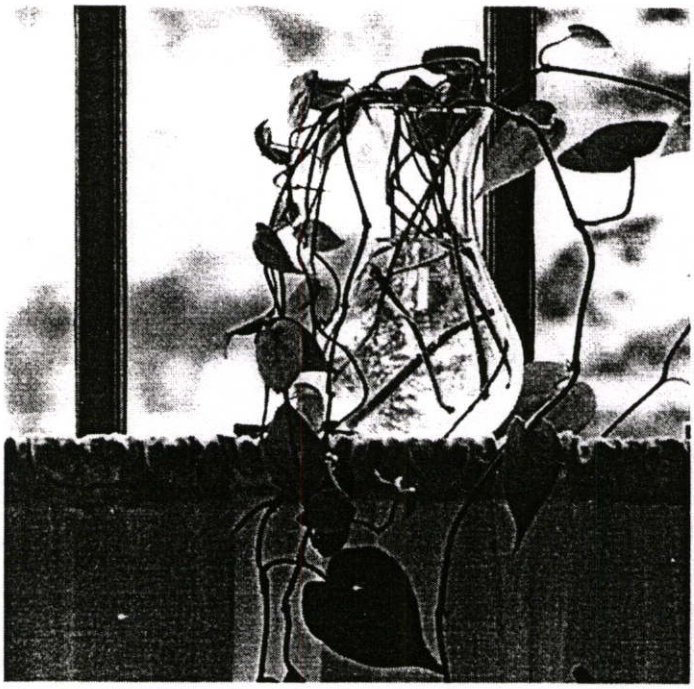
(a) รูป Baboon ต้นฉบับ



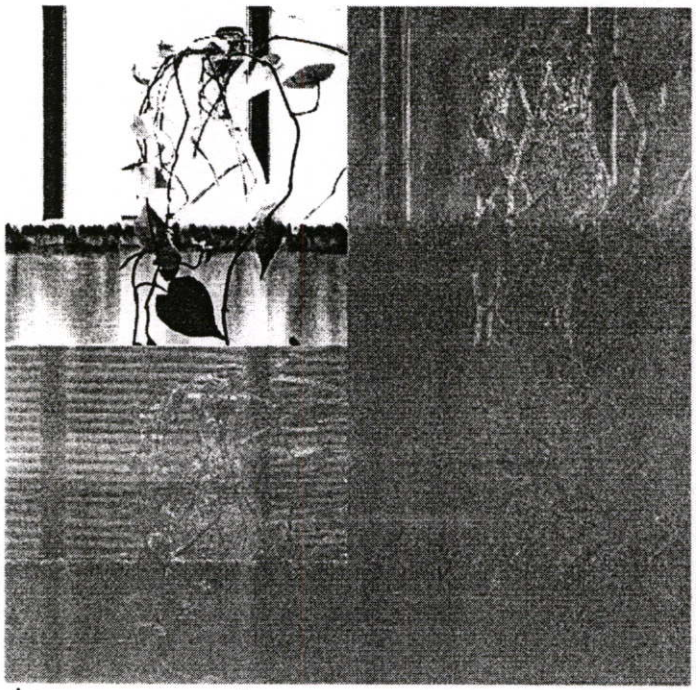
(b) รูป Baboon ที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) วิธี Lifting แบบเลขจำนวนเต็ม



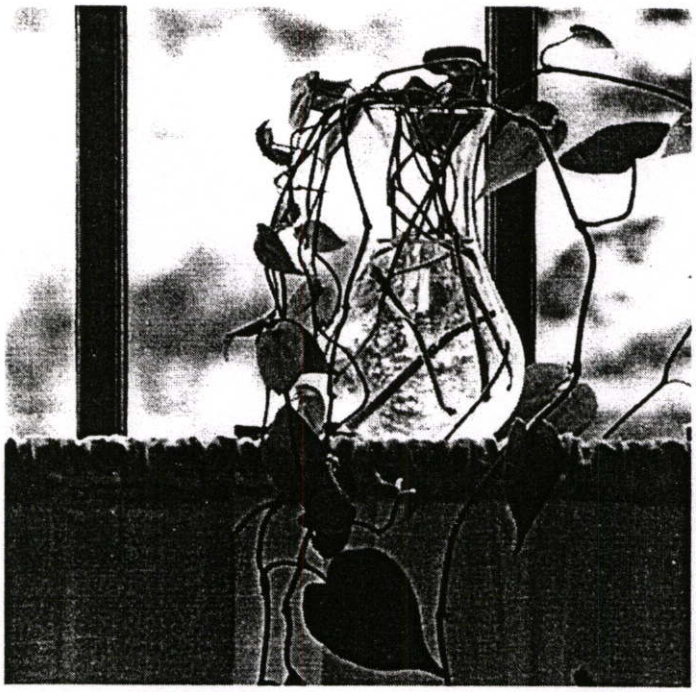
(c) รูป Baboon ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) วิธี Lifting แบบเลขจำนวนเต็ม



(d) รูป Vase ต้นฉบับ



(e) รูป Vaseที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) วิธี Lifting แบบเลขจำนวนเต็ม

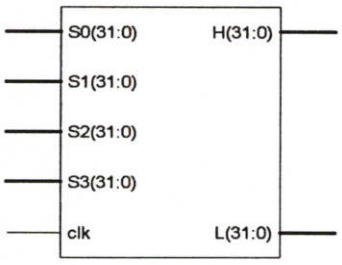


(c) รูป Vase ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) วิธี Lifting แบบเลขจำนวนเต็ม

รูปที่ 5.16 แสดงตัวอย่างภาพผลการทดลองที่ได้จากโมดูล Daubechies4 (D4) ด้วยวิธี Lifting แบบเลขจำนวนเต็ม

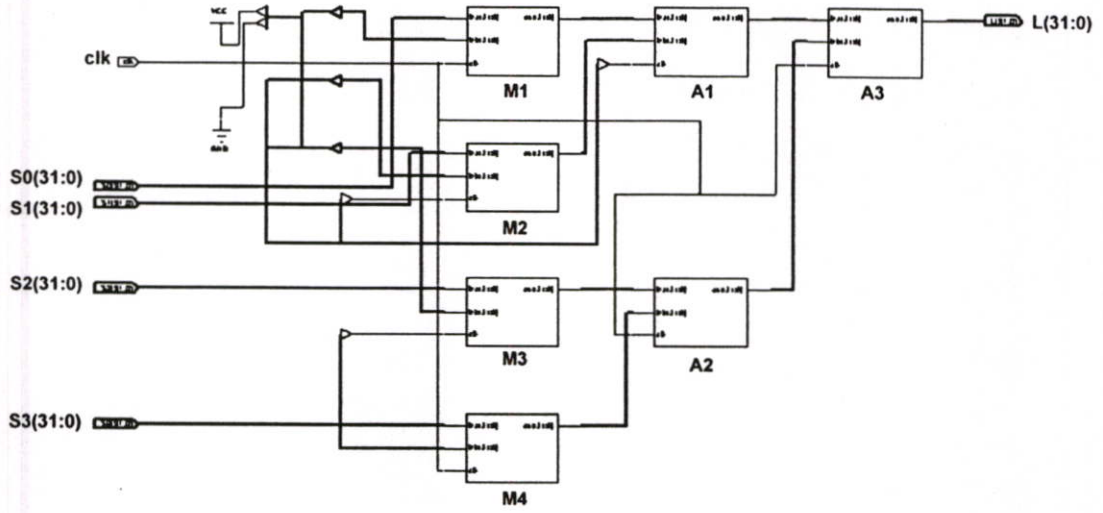
5.4.3 โมดูลที่ใช้ในการแปลงเวฟเล็ต Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

เมื่อนำโมดูลที่ใช้ในการบวกลบ และคูณเลขจำนวนจริงมาจัดเรียงสมการ Daubechues4 (D4) โดยใช้ Matrix (Scaling Function และWavelet Function) โครงสร้างภายนอกของโมดูลดังแสดงอยู่ในรูปที่ 5.17 ลักษณะของโมดูลจะประกอบไปด้วยอินพุตของข้อมูลที่ต้องการจะแปลงเวฟเล็ตขนาด 32 บิตจำนวน 4 ชุด ซึ่งประกอบไปด้วย S0, S1, S2, S3 และ clk คืออินพุตของสัญญาณนาฬิกา เอาท์พุตจะประกอบไปด้วย L และ H แต่ละตัวมีขนาด 32 บิต โดยที่ L คือผลลัพธ์ที่ได้จากตัวกรองความถี่ต่ำ (Approximation) และ H คือผลลัพธ์ที่ได้จากตัวกรองความถี่สูง (Detail)

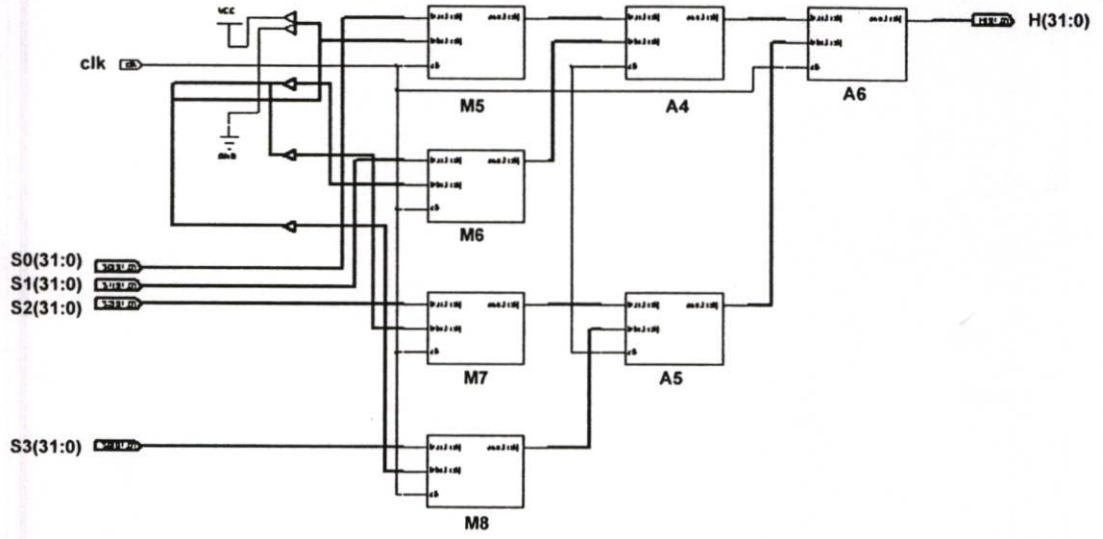


รูปที่ 5.17 แสดงโครงสร้างภายนอกของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

ลักษณะโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง ดังแสดงในรูปที่ 5.18



(a) ส่วนกรองความถี่ต่ำ (Scaling Function)



(b) ส่วนกรองความถี่สูง (Wavelet Function)

รูปที่ 5.18 แสดงโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

จากรูปที่ 5.18 จะประกอบไปด้วยโมดูลที่ใช้ในการคูณเลขจำนวนจริง (M1 ถึง M8) และโมดูลที่ใช้ในการบวกหรือลบเลขจำนวนจริง (A1 ถึง A5) จัดเรียงเพื่อใช้ในการคำนวณหาค่าสัมประสิทธิ์ของการแปลงเวฟเล็ต

โปรแกรมที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริงที่ถูกพัฒนาด้วยภาษา Verilog ดังแสดงในภาคผนวก ง.5

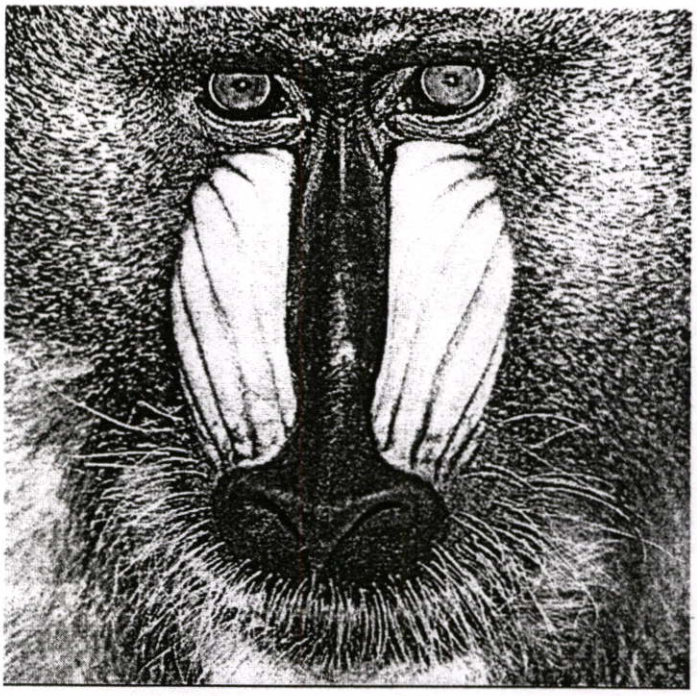
ในการทดสอบทรัพยากรที่ใช้ในการออกแบบจะใช้ชิพ FPGA Spantan3 Xc3s1500tq456-4 ของบริษัท Xilinx ผลการ Synthesis ดังแสดงในตารางที่ ตารางที่ 5.8

ตารางที่ 5.8 แสดงการใช้ทรัพยากรทั้งหมดที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตแบบ Daubechues4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

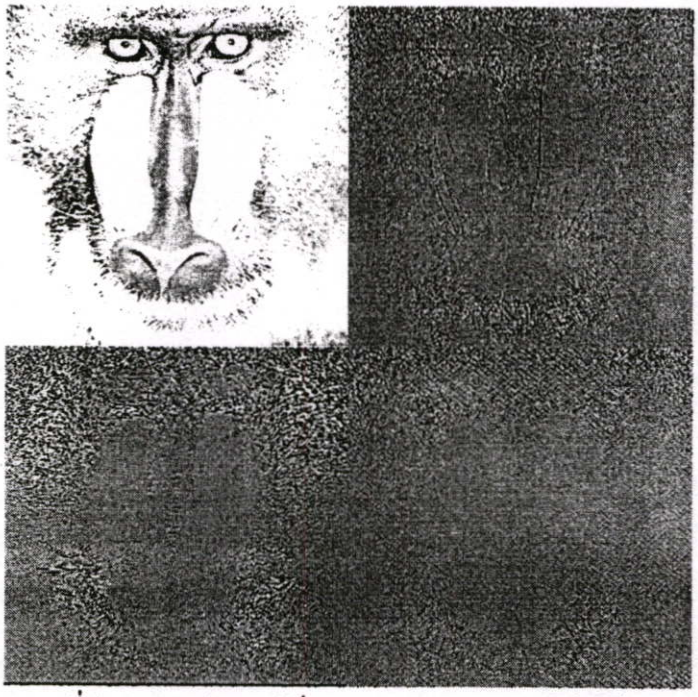
	จำนวนทั้งหมด	จำนวนที่ใช้ไป	เปอร์เซ็นต์
Slices	13312	2810	21.108%
Flip Flops	26624	1261	4.736%
4 input LUT	26624	5174	19.433%
MULT18X18	32	32	100.000%
GCLK	8	1	12.500%

ความถี่สูงสุดที่วงจรสามารถทำงานได้คือ 42.645MHz (Maximum Frequency: 42.645MHz)

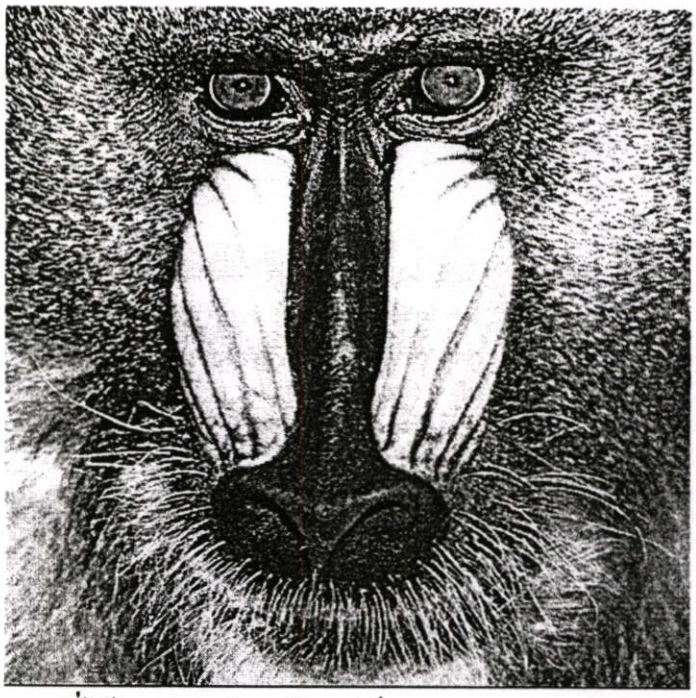
ตัวอย่างผลการทดลองที่ได้ Daubechies4 (D4) ด้วยโดยใช้ Scaling Function และ Wavelet Function แบบเลขจำนวนจริงดังแสดงในรูปที่ 5.19



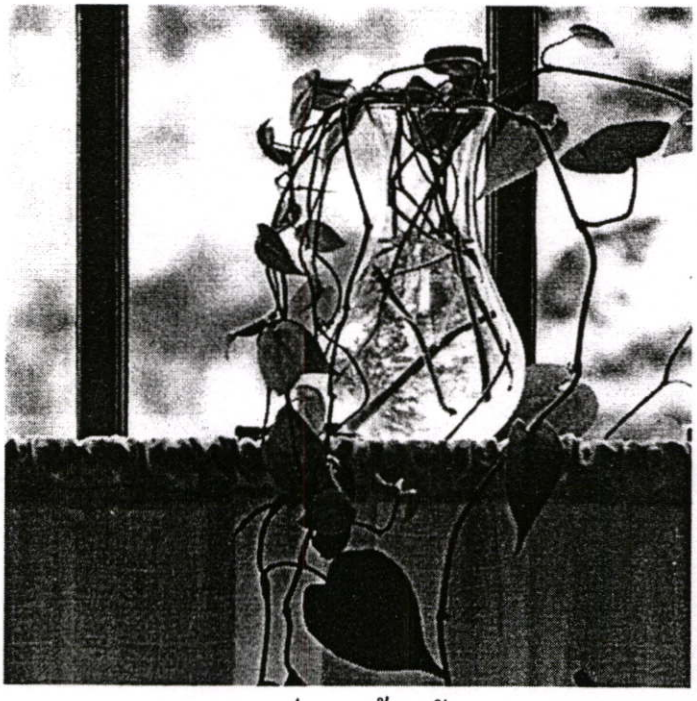
(a) รูป Baboon ต้นฉบับ



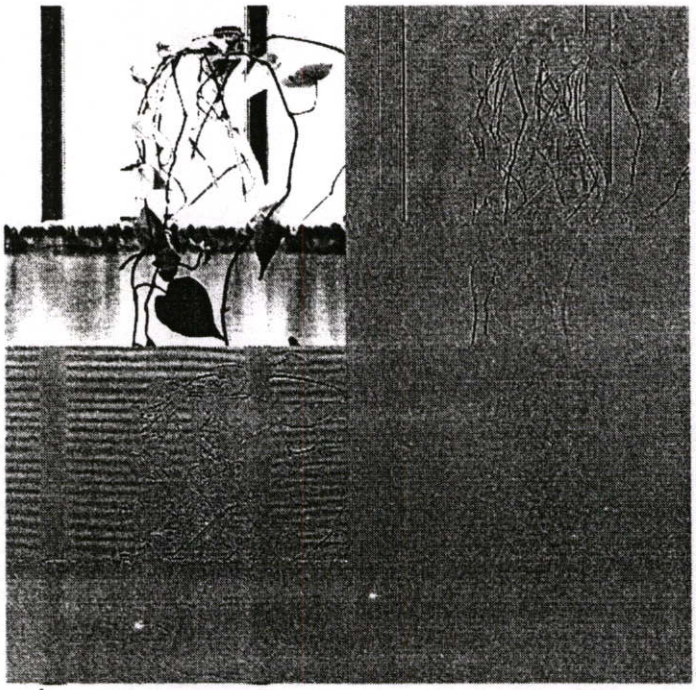
(b) รูป Baboon ที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง



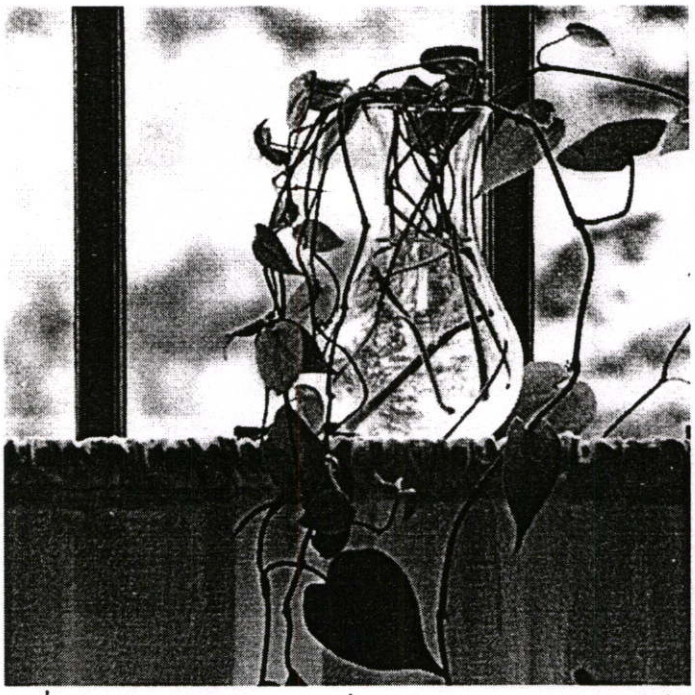
(c) รูป Baboon ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง



(d) รูป Vase ต้นฉบับ



(e) รูป Vaseที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) โดยใช้Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

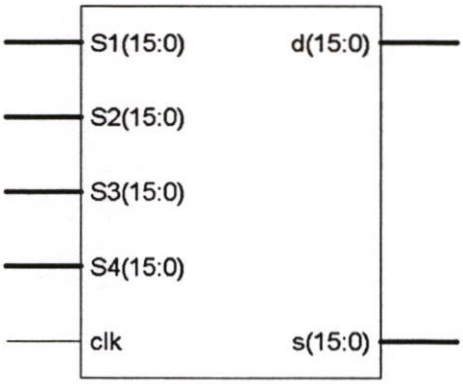


(c) รูป Vase ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

รูปที่ 5.19 แสดงภาพผลการทดลองที่ได้จากโมดูล Daubechies4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

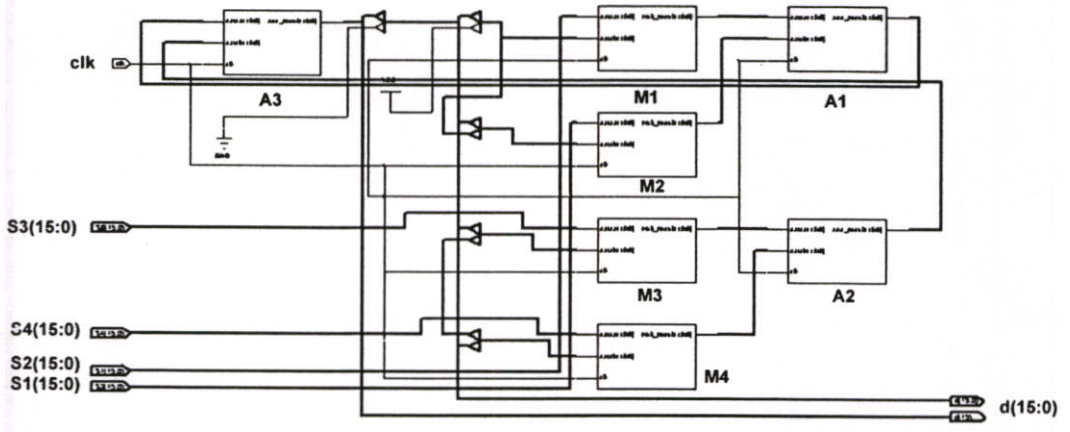
5.4.4 โมดูลที่ใช้ในการแปลงเวฟเล็ต Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนเต็ม

เมื่อทำการปรับปรุงค่าสัมประสิทธิ์ Scaling Function และWavelet Function ให้อยู่ในรูปของเลขจำนวนเต็ม เมื่อฟังก์ชั่นเหล่านี้มาทำการออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ต จะมีโครงสร้างแสดงอยู่ในรูปที่ 5.20 ลักษณะของโมดูลจะประกอบไปด้วยอินพุตของข้อมูลที่ต้องการจะแปลงเวฟเล็ตขนาด 16 บิตจำนวน 4 ชุด ซึ่งประกอบไปด้วย S0, S1, S2, S3 และ clk คืออินพุตของสัญญาณนาฬิกา เอ้าพุตจะประกอบไปด้วย s และ d แต่ละตัวมีขนาด 16 บิตโดยที่ s คือผลลัพธ์ที่ได้จากตัวกรองความถี่ต่ำ (Approximation) และ d คือผลลัพธ์ที่ได้จากตัวกรองความถี่สูง (Detail)

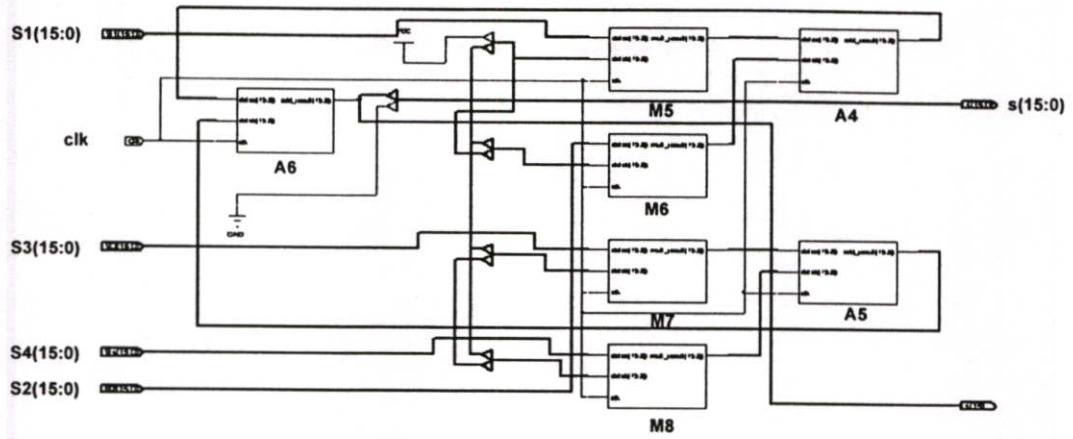


รูปที่ 5.20 แสดงโครงสร้างภายนอกของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และWavelet Function) แบบเลขจำนวนเต็ม

ลักษณะโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนเต็ม ดังแสดงในรูปที่ 5.21



(a) ส่วนกรองความถี่ต่ำ (Scaling Function)



(b) ส่วนกรองความถี่สูง (Wavelet Function)

รูปที่ 5.21 แสดงโครงสร้างของวงจรที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนเต็ม

จากรูปที่ 5.21 จะประกอบไปด้วยโมดูลที่ใช้ในการคูณเลขจำนวนเต็ม (M1 ถึง M8) และโมดูลที่ใช้ในการบวกหรือลบเลขจำนวนเต็ม (A1 ถึง A5) จัดเรียงเพื่อใช้ในการคำนวณหาค่าสัมประสิทธิ์ของการแปลงเวฟเล็ต

โปรแกรมที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechues4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริงที่ถูกพัฒนาด้วยภาษา Verilog ดังแสดงในภาคผนวก ง.4

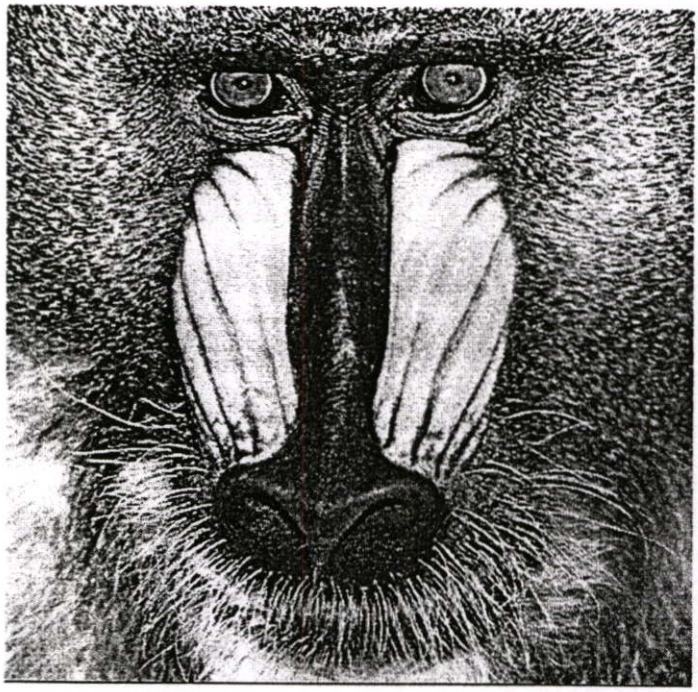
ในการทดสอบทรัพยากรที่ใช้ในการออกแบบจะใช้ชิพ FPGA Spantan3 Xc3s1500tq456-4 ของบริษัท Xilinx ผลการ Synthesis ดังแสดงในตารางที่ ตารางที่ 5.9

ตารางที่ 5.9 แสดงการใช้ทรัพยากรทั้งหมดที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตแบบ Daubechues4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนเต็ม

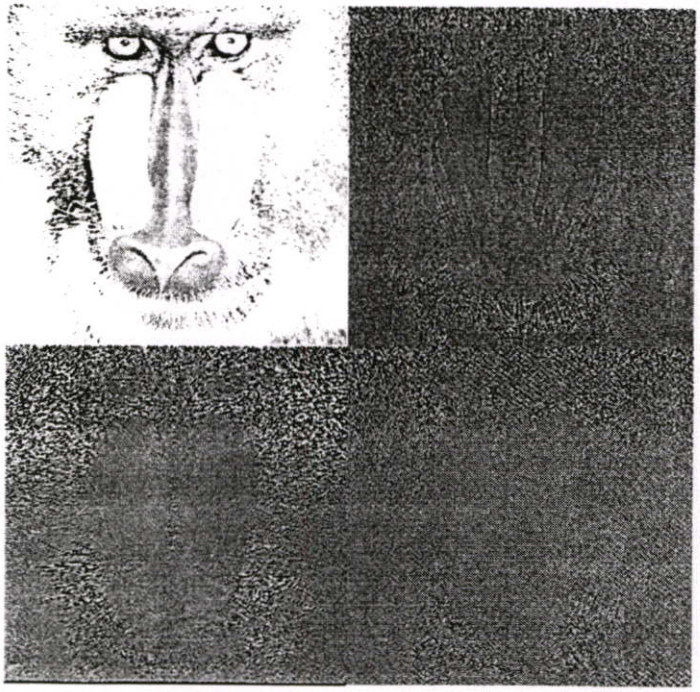
	จำนวนทั้งหมด	จำนวนที่ใช้ไป	เปอร์เซ็นต์
Slices	13312	344	2.584%
Flip Flops	26624	210	0.788%
4 input LUT	26624	522	1.960%
MULT18X18	32	8	25.000%
GCLK	8	1	12.500%

ความถี่สูงสุดที่วงจรสามารถทำงานได้คือ 139.082MHz (Maximum Frequency: 139.082MHz)

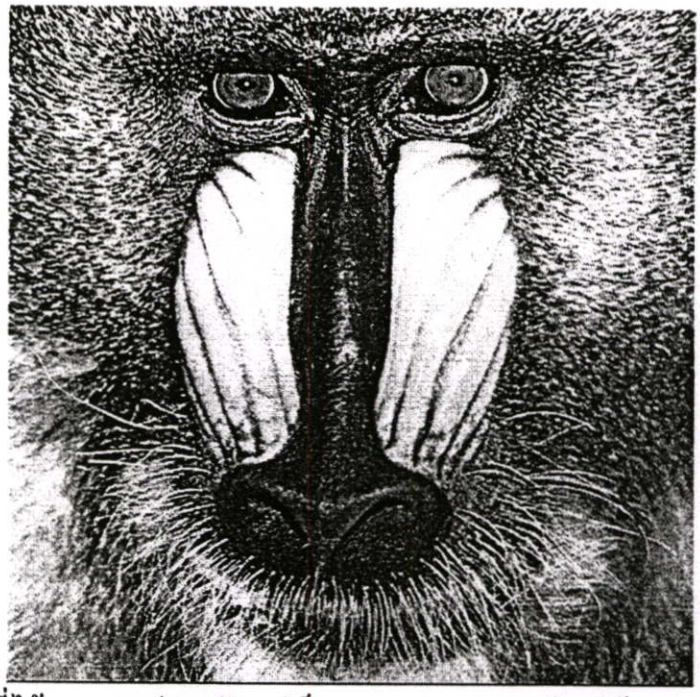
ตัวอย่างผลการทดลองที่ได้ Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนเต็มดังแสดงในรูปที่ 5.22



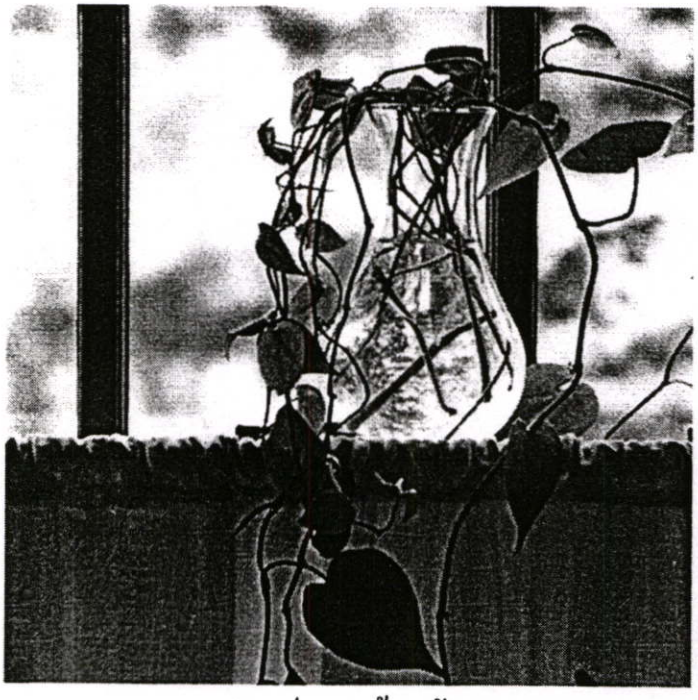
(a) รูป Baboon ต้นฉบับ



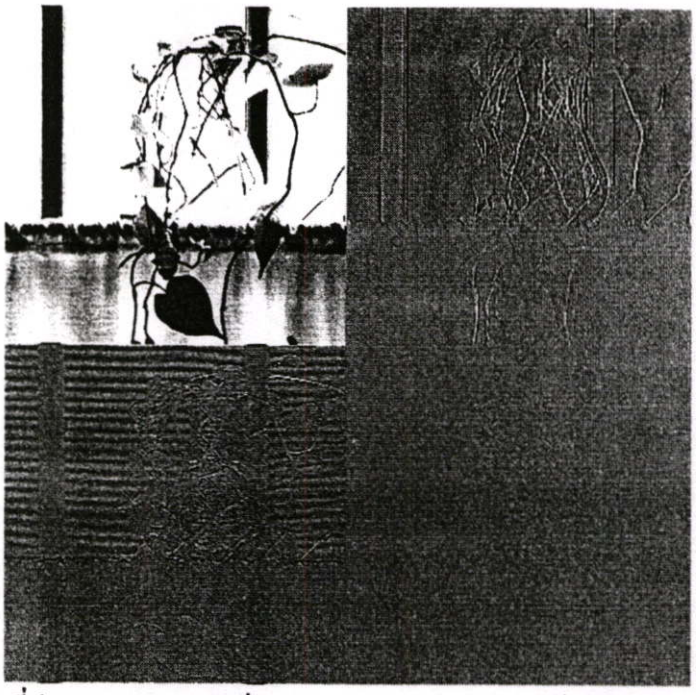
(b) รูป Baboon ที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนเต็ม



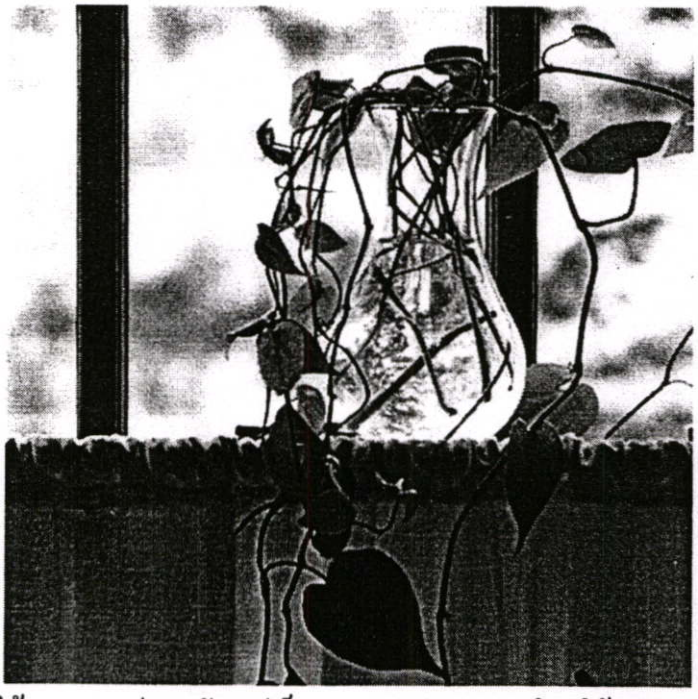
(c) รูป Baboon ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function แบบเลขจำนวนเต็ม



(d) รูป Vase ต้นฉบับ



(e) รูป Vaseที่ผ่านการแปลงเวฟเล็ต Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function แบบเลขจำนวนเต็ม



(c) รูป Vase ที่ได้จากการแปลงกลับเวฟเล็ต Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function แบบเลขจำนวนเต็ม

รูปที่ 5.22 แสดงภาพผลการทดลองที่ได้จากโมดูล Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function แบบเลขจำนวนเต็ม

5.5 สรุปผลการทดลอง

ในหัวข้อนี้จะกล่าวถึงการเปรียบเทียบประสิทธิภาพของโครงสร้างการแปลงเวฟเล็ตที่ได้ทำการออกแบบ โดยจะทำการพิจารณาจากความเร็วในการประมวลผล ทรัพยากรที่ใช้ในการออกแบบ ตลอดจนการนำโมดูลที่ใช้ในการแปลงเวฟเล็ตไปประยุกต์ใช้งานใช้งาน

5.5.1 สรุปผลการทดลองเปรียบเทียบเวลาในการประมวลผล

การเปรียบเทียบความเร็วในการประมวลผล จะใช้เวลาที่ได้จากการประมวลผลของเครื่องคอมพิวเตอร์เปรียบเทียบกับเวลาที่ได้จากการจำลองการทำงานของ FPGA โดยที่ข้อมูลที่นำมาทำการทดสอบคือภาพขนาด 256 x 256 ผลลัพธ์ที่ได้ดังแสดงในตารางที่ 5.10

ตารางที่ 5.10 แสดงค่าเวลาที่ใช้ในการประมวลผลของเรื่องคอมพิวเตอร์เปรียบเทียบกับ FPGA

		PC (mSec.)		FPGA (mSec.)	
		1 ภาพ	30 ภาพ	1 ภาพ	30 ภาพ
Lifting	จำนวนจริง	16.000	480.000	12.500	375.000
	จำนวนเต็ม	11.000	330.000	3.769	113.070
Matrix	จำนวนจริง	15.000	450.000	12.294	368.820
	จำนวนเต็ม	10.000	300.000	3.769	113.070

จากตารางที่ 5.10 จะเห็นได้ว่าค่าเวลาที่ได้จากการจำลองการทำงานของ FPGA ทุกโมดูลสามารถทำการประมวลได้เร็วกว่าเครื่องคอมพิวเตอร์ ทั้งนี้เพราะในการคำนวณของ FPGA จะใช้การประมวลผลแบบขนานทำให้ลดเวลาในการประมวลผล

5.5.2 สรุปผลการทดลองเปรียบเทียบทรัพยากรที่ใช้ในการออกแบบ

ในการเปรียบเทียบทรัพยากรที่ใช้ในการออกแบบจะทำการเปรียบเทียบในส่วนของ Slices, Flip Flops, 4 input LUT, MULT18X18 และ GCLK ดังแสดงในตารางที่ 5.11

ตารางที่ 5.11 แสดงทรัพยากรที่ใช้ในการออกแบบโมดูลแปลงเวฟเล็ตแบบต่างๆ

	Lifting		Matrix	
	จำนวนจริง	จำนวนเต็ม	จำนวนจริง	จำนวนเต็ม
Slices	3.290%	42.510%	21.108%	2.584%
Flip Flops	0.852%	9.363%	4.736%	0.788%
4 input LUT	2.339%	39.637%	19.433%	1.960%
MULT18X18	28.125%	112.500%	100.000%	25.000%
GCLK	12.500%	12.500%	12.500%	12.500%
Clock Speed	41.771 MHz	139.082 MHz	42.645 MHz	139.082 MHz

จากตารางที่ 5.11 การแปลงเวฟเล็ตด้วยวิธี Lifting จะใช้ทรัพยากรในการออกแบบมากกว่าการแปลงเวฟเล็ตโดยใช้ Scaling Function และ Wavelet Function ไม่ว่าจะเป็นแบบเลขจำนวนจริงหรือเลขจำนวนเต็ม เพราะว่าทรัพยากรส่วนใหญ่จะสูญเสียไปกับโมดูลที่ใช้ในการคำนวณ นอกจากนี้โมดูลที่ใช้การแปลงเวฟเล็ตแบบเลขจำนวนจริงจะแตกต่างจากการแปลงเวฟเล็ตแบบเลขจำนวนเต็มตรงที่โมดูลสำหรับใช้ในการคำนวณหาค่าสัมประสิทธิ์การแปลงเวฟเล็ตจะถูกเปลี่ยนจากโมดูลที่ใช้ในการคำนวณแบบเลขจำนวนจริงไปเป็นโมดูลที่ใช้ในการคำนวณแบบเลขจำนวนเต็ม

5.5.3 สรุปผลการทดลองเปรียบเทียบใช้ในเชิงการนำไปใช้งาน

ผลที่ได้จากทดสอบการวัดค่า PSNR ของโมดูลที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting เปรียบเทียบกับโมดูลแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้ Matrix (Scaling Function และ Wavelet Function) เมื่อนำโมดูลที่มีการคำนวณเป็นเลขจำนวนจริงมาเปรียบเทียบกัน โดยพิจารณาจากภาพที่นำมาทดสอบจะเห็นได้ว่า PSNR เฉลี่ยของการแปลงเวฟเล็ตแบบ Lifting (144.598615dB.) จะมีค่าสูงกว่าการแปลงเวฟเล็ตแบบใช้ Matrix (Scaling Function และ Wavelet Function) (139.146234dB.) และเมื่อทำการแปลงโมดูลที่ใช้ในการแปลงเวฟเล็ตทั้งสองให้อยู่ในรูปการคำนวณแบบเลขจำนวนเต็มแล้วทำการวัดค่า PSNR ผลลัพธ์ที่ได้คือ การแปลงเวฟเล็ตโดยใช้วิธี Lifting จะมีค่า PSNR เฉลี่ย (41.579799dB.) สูงกว่าการแปลงเวฟเล็ตแบบใช้ Matrix (Scaling Function และ Wavelet Function) (40.424140dB.) อีกเช่นเดียวกัน

เมื่อทำการตัด HH Subband ทิ้งเพื่อทดสอบผลกระทบจากการควอนไตซ์ (Quantize) ข้อมูลที่ได้จากการแปลงเวฟเล็ต ซึ่งข้อมูลในส่วนนี้มีความสำคัญน้อยที่สุดสามารถทำการตัดทิ้งได้ ผลปรากฏว่าค่า PSNR ของการแปลงเวฟเล็ตแบบเลขจำนวนจริงทั้งสองแบบจะถูกลดทอนลงมามาก เนื่องจากข้อมูลบางส่วนสูญหายไปทำให้สูญเสียความเป็น PR (Perfect Reconstruct) ไป ในการ

แปลงเวฟเล็ตวิธี Lifting แบบเลขจำนวนจริงจะมีค่า PSNR เฉลี่ยลดลงจาก 144.598615dB. เป็น 39.237836dB. และการแปลงเวฟเล็ตโดยใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริงจะมีค่า PSNR เฉลี่ยลดลงจาก 139.146234dB. เป็น 39.296391dB. แต่การแปลงเวฟเล็ตแบบเลขจำนวนเต็มทั้งสองแบบค่า PSNR จะลดลงไม่มากนัก คือ Lifting จะลดลงจาก 41.579799dB.เป็น 35.437061dB. และแบบใช้ Matrix (Scaling Function และ Wavelet Function) ลดลงจาก 40.424140dB เป็น 34.646204dB ดังนั้นแสดงว่าการแปลงเวฟเล็ตแบบ Lifting จะมีข้อมูลอยู่ในส่วน HH Subband มากกว่าการแปลงเวฟเล็ตแบบ Matrix (Scaling Function และ Wavelet Function) ดังนั้นจึงเหมาะที่จะนำไปใช้ในการวิเคราะห์ภาพเช่นการวิเคราะห์ลายนิ้วมือ

การวัดค่า Zero Moment เป็นการตรวจประสิทธิภาพของการแปลงเวฟเล็ตที่ได้นำเสนอว่ามีประสิทธิภาพในการบีบอัดข้อมูลมากน้อยเพียงไร จากผลการทดลองปรากฏว่าการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting แบบการคำนวณแบบเลขจำนวนจริงจะมีค่าเฉลี่ยของ Zero Moment (30.079546) ต่ำกว่าแบบที่ใช้ Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง (30.449212%) และเมื่อทำการแปลงเวฟเล็ตทั้งสองให้อยู่ในรูปการคำนวณแบบเลขจำนวนเต็ม แล้วทำการทดสอบค่า Zero Moment ปรากฏว่าการแปลงเวฟเล็ตแบบ Lifting มีค่า Zero Moment อยู่ที่ 20.714051% ซึ่งมีค่าสูงกว่าการใช้ Matrix (Scaling Function และ Wavelet Function) ซึ่งมีค่า Zero Moment เท่ากับ 20.689938% อยู่เล็กน้อยดังนั้นการแปลงเวฟเล็ตแบบ Matrix (Scaling Function และ Wavelet Function) จึงเหมาะที่จะนำไปประยุกต์ใช้ในการบีบอัดข้อมูล เนื่องจากผลกระทบจากการ Quantize ข้อมูลโดยการตัดส่วนที่ไม่จำเป็นของการแปลงเวฟเล็ตแบบ Matrix (Scaling Function และ Wavelet Function) น้อยกว่าการแปลงเวฟเล็ตแบบ Lifting

บทที่ 6
สรุปผลการทดลอง

วิทยานิพนธ์นี้ ได้นำเสนอการออกแบบการแปลงเวฟเล็ตโดยใช้ FPGA ในการออกแบบจะเลือกใช้อัลกอลิธึมการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting และทำการเปรียบเทียบกับการแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function ในการออกแบบโมดูลที่ใช้ในการแปลงเวฟเล็ตจะกระทำทั้งแบบเลขจำนวนจริงและเลขจำนวนเต็ม เพื่อใช้เปรียบเทียบประสิทธิภาพของการแปลงเวฟเล็ตแต่ละแบบ การวัดประสิทธิภาพของการแปลงเวฟเล็ตแต่ละแบบจะพิจารณาจากการวัดค่า PSNR และ Zero Moment รวมถึงจำนวนของทรัพยากรที่ใช้ในการออกแบบบน FPGA และความเร็วที่ได้ นอกจากนี้จะพิจารณาความแตกต่างของผลลัพธ์ที่ได้จากการประมวลผล ว่ามีความแตกต่างกันอย่างไร

6.1 สรุปและวิเคราะห์ผลการดำเนินงานวิจัย

จากผลการทดลองของงานวิจัยนี้แสดงถึงประสิทธิภาพการแปลงเวฟเล็ตแต่ละโมดูลที่ได้ทำการออกแบบ การวัดประสิทธิภาพของโมดูลแปลงเวฟเล็ตจะพิจารณาจากค่า PSNR และค่า Zero Moment ข้อมูลที่ใช้ในการทดสอบคือ ภาพ Gray Scale 256 ระดับ

จากการทดลองวัดค่าเวลาที่ใช้ในการประมวลผลของ FPGA เทียบกับเครื่องคอมพิวเตอร์ โดยจำลองการทำงานของ FPGA เพื่อวัดเวลาที่ใช้ในการประมวลผล ข้อมูลที่ใช้ในการทดสอบคือ ภาพขนาด 256 X 256 Pixel จากการทดลองจะเห็นได้ว่า FPGA สามารถทำงานได้เร็วกว่าเครื่องคอมพิวเตอร์ที่ใช้ในการทดลองทุกกรณี

จากนั้นเมื่อทำการทดสอบโมดูลแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting จะมีค่าเฉลี่ย PSNR สูง แต่เมื่อทำการตัดส่วนที่เป็น HH ทิ้งเพื่อทดสอบการควอนไตซ์ (Quantize) ข้อมูล จะสังเกตได้ว่าการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธี Lifting จะมีอัตราการลดทอนของค่าเฉลี่ยของ PSNR มากกว่าการแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้ Scaling Function และ Wavelet Function เพราะข้อมูลที่ได้จากการแปลงเวฟเล็ตแบบ Lifting จะมีรายละเอียดของข้อมูลในแต่ละ Subband มากกว่า จึงเหมาะที่จะนำไปใช้ในการวิเคราะห์ภาพ เช่นการวิเคราะห์ลายนิ้วมือ แต่การทำงานได้ช้ากว่า และใช้ทรัพยากรของ FPGA ในการออกแบบมาก

หากพิจารณาโมดูลแปลงเวฟเล็ตแบบ Daubechies4 โดยใช้ Scaling Function และ Wavelet Function จะมีข้อดีคือสามารถทำงานได้เร็วกว่าการแปลงเวฟเล็ตแบบ Lifting และใช้ทรัพยากรของ FPGA ในการออกแบบน้อยกว่า นอกจากนี้จะมีค่าเฉลี่ยของ Zero Moment สูงกว่าการแปลงเวฟเล็ตแบบ Lifting จึงเหมาะที่จะนำไปใช้ในการบีบอัดข้อมูล

ดังนั้นการเลือกโมดูลที่ใช้ในการแปลงเวฟเล็ตจะต้องพิจารณาถึงงานที่จะนำไปใช้เพื่อให้เกิดความเหมาะสม เนื่องจากประสิทธิภาพของการแปลงเวฟเล็ตที่วัดได้จาก PSNR และ Zero Moment รวมถึงจำนวนทรัพยากรที่ใช้ในการออกแบบ และความเร็วที่ใช้ในการประมวลผล ตัวอย่างเช่น การนำการแปลงเวฟเล็ตไปใช้ในการบีบอัดข้อมูลควรนำการแปลงเวฟเล็ตแบบ Scaling Function และ Wavelet Function เพราะให้ค่า Zero Moment ที่ดีกว่าและทำงานได้เร็วกว่า หรือถ้าต้องการที่จะนำไปใช้ในการวิเคราะห์ภาพควรใช้การแปลงเวฟเล็ตแบบ Lifting เพราะว่าให้รายละเอียดของข้อมูลมากกว่า

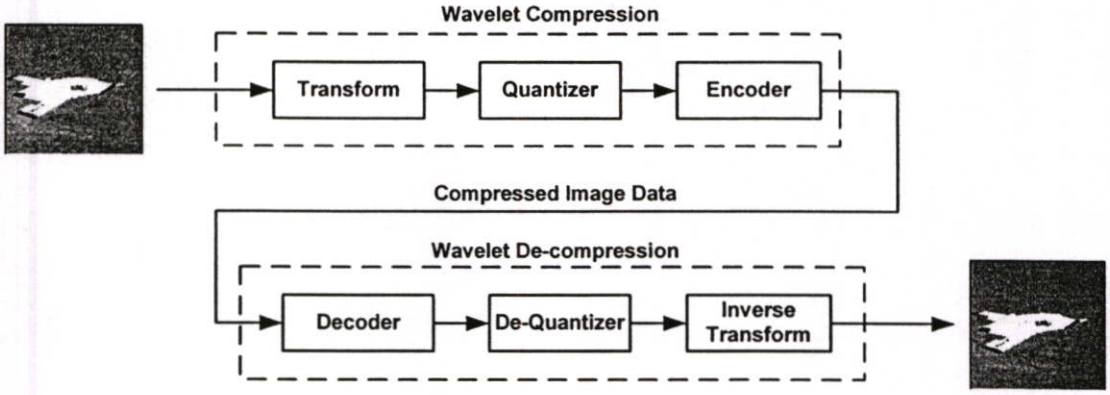
6.2 ปัญหา

ปัญหาแรกที่พบในงานวิจัยนี้คือเรื่องทรัพยากรของ FPGA ซึ่งเป็นอุปกรณ์หลักที่ใช้ในการทดลอง และการทดลองในช่วงแรกจะใช้ FPGA xc3s200 ในการพัฒนาโมดูลที่ใช้ในการคำนวณเลขจำนวนจริงซึ่งทรัพยากรของ xc3s200 มีเพียงพอสำหรับใช้ในการออกแบบเลขจำนวนจริง แต่เมื่อนำค่าทรัพยากรที่ใช้ออกแบบโมดูลคำนวณเลขจำนวนจริงมาใช้ในการคำนวณเพื่อให้ทราบถึงทรัพยากรที่ใช้ในโมดูลแปลงเวฟเล็ต ทำให้ทรัพยากรของ xc3s200 ไม่เพียงพอจึงเปลี่ยนไปใช้ FPGA xc3s1500 ในการออกแบบแทน ซึ่ง FPGA xc3s1500 มีทรัพยากรเพียงพอสำหรับใช้ในการออกแบบโมดูลแปลงเวฟเล็ตโดยใช้ Scaling Function และ Wavelet Function ที่มีการคำนวณแบบเลขจำนวนจริงได้ แต่ใช้ในการออกแบบโมดูลแบบ Lifting ไม่ได้เพราะขาด Mul 18x18 ไป 2 โมดูล

ปัญหาที่พบอีกประการหนึ่งก็คือ การรับส่งข้อมูลเพื่อใช้ในการประมวลผลคือ ในการทดสอบช่วงแรกจะใช้การรับส่งข้อมูลแบบอนุกรม (Serial Port) ปรากฏว่าเกิดความล่าช้าในการรับส่งข้อมูลอย่างมาก หลังจากนั้นจึงได้เปลี่ยนการรับส่งข้อมูลจากแบบอนุกรมเป็นการรับส่งข้อมูลแบบ PCI บัสซึ่งจะได้อัตราการรับส่งข้อมูลที่ดีขึ้นแต่ไม่สามารถใช้ความเร็วในการรับส่งของ PCI บัสได้อย่างเต็มประสิทธิภาพเนื่องจากขาด Windows Device Driver สำหรับควบคุมการรับส่งข้อมูลแบบ DMA ด้วยปัญหาด้านฮาร์ดแวร์ดังที่กล่าวมา งานวิจัยนี้จึงนำเสนอการออกแบบโครงสร้างเวฟเล็ตแต่ละอัลกอริธึมโดยอาศัยการจำลองการทำงานเท่านั้น

6.3 แนวทางในการพัฒนาต่อ

จากการทดลองที่ได้จากงานวิจัยนี้ ผู้สนใจสามารถที่จะนำไปพัฒนาต่อเพื่อใช้ในการบีบอัดข้อมูลภาพดังแสดงในรูปที่ 6.1 โดยอาจทำการพัฒนาในส่วนของ Encoder และ Decoder แล้วนำมาใช้งานร่วมกับโมดูลที่ใช้แปลงเวฟเล็ตที่งานวิจัยนี้ได้นำเสนอ ซึ่งจะได้โมดูลที่ใช้ในการบีบอัดข้อมูลภาพต่อไป



รูปที่ 6.1 แสดงระบบการบีบอัดข้อมูลภาพ

นอกจากนี้ยังสามารถที่จะทำการพัฒนาการรับส่งข้อมูลให้มีความเร็วสูง โดยการพัฒนา Windows Device Driver ให้สามารถควบคุมการติดต่อกับ FPGA บอร์ดได้ดีขึ้น

เอกสารอ้างอิง

[1] S. J. Huang, J.T. Huang and T.M. Yang, "FPGA Realization of Wavelet Transform for Detection of Electric Power System Disturbances", IEEE Transaction on Power Delivery, Volume 17, Issue 2, April 2002. pp.388 – 394.

[2] W. Zhilu and R. Guanghui, "A Study on Implementing Wavelet Transform and FFT with FPGA", Proceeding of the 4th ASIC, October. 2001. pp.486 – 489.

[3] C. Gonzalez and E. Wood, "Wavelet and Multiresolution Processing", Digital Image Processing, Prentice Hall, 2001.

[4] World Wide Web Consortium. "The FBI Fingerprint Image Compression Standard", [Online]. Available: http://www.c3.lanl.gov/~brislawn/FBI/. 2002.

[5] J. Stollnitz, D. DeRose and H. Salesin, "Wavelets for Computer Graphics", IEEE Computer Graphics and Applications, Volume 15, Issue 3, May 1995.pp.76 – 84.

[6] A. Jense and A. la Cour-Harbo, "The Discrete Wavelet Transform", Ripples in Mathematics, Springer, 2001.

[7] W. Sweldent, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions", Proceeding of SPIE, Volume 2569, 1995. pp.68-79.

[8] A. R. Calderbank, "Wavelet Transform That Map Integer To Integer", Mathematics Subject Classification, August 1996.

[9] J. A. Shafer, "Embedded Vector Processor Architecture for Real-Time Wavelet Video Compression", Master Thesis, University of Dayton, May 2004. pp.12 – 4.

[10] Z. Xiong, X. Wu, S. Cheng, and J. Hua, "Lossy-to-Lossless Compression of Medical Volumetric Data Using Three Dimensional Integer Wavelet Transforms", IEEE Transaction on Medical Imaging, Volume 22, Issue 3, March 2003. pp. 459 – 470.

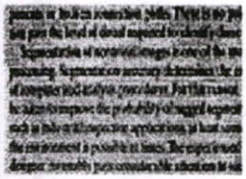
[11] W. Stallings, "Computer Organization and Architecture", Computer Arithmetic, May 2000. pp. 273-317.

[12] IEEE Task P754, "A Proposed Standard for Binary Floating-Point Arithmetic", IEEE Computer, Volume 14, Issue 2, March 1981. pp. 51 – 62.

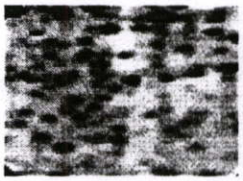
ภาคผนวก ก

ภาพที่ใช้ในการทดสอบโมดูลแปลงเวฟเล็ต

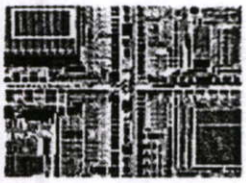
รูปที่ ก.1 ภาพที่ใช้ในการทดลองการแปลงเวฟเล็ต

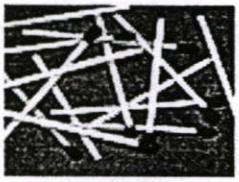

2018 X 918 พิกเซล

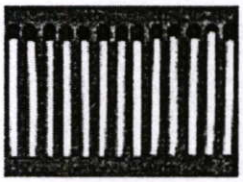

482 X 412 พิกเซล


256 X 256 พิกเซล

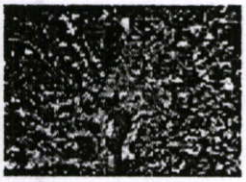

788 X 1242 พิกเซล

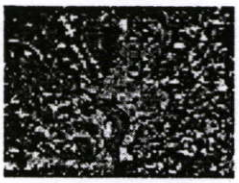

832 X 1314 พิกเซล


558 X 558 พิกเซล

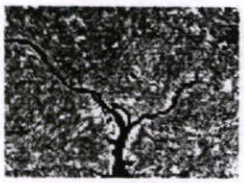

558 X 558 พิกเซล

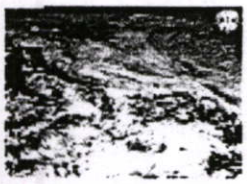

512 X 512 พิกเซล

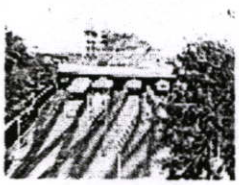

512 X 512 พิกเซล


512 X 512 พิกเซล


512 X 512 พิกเซล


512 X 512 พิกเซล


640 X 480 พิกเซล


490 X 300 พิกเซล


512 X 512 พิกเซล


512X 512 พิกเซล

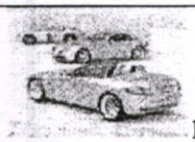
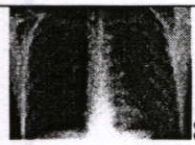
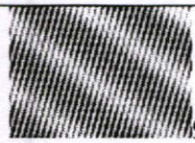

512 X 512 พิกเซล

รูปที่ ก.2 ภาพที่ใช้ในการทดลองการแปลงเวฟเล็ต

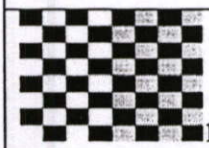
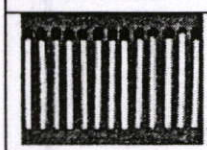
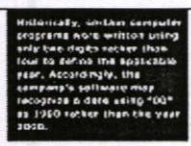
ภาคผนวก ข

แสดงค่า PSNR ที่วัดได้จากการแปลงเวฟเล็ต

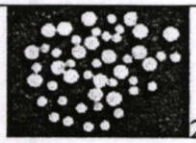
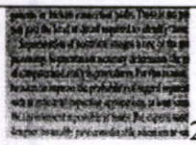
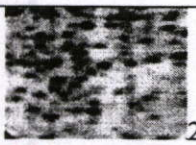
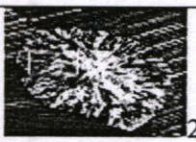
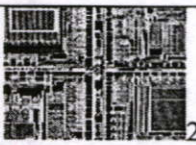
ตารางที่ ข.1 แสดงค่า PSNR ที่วัดได้จากภาพที่นำมาทำการทดลอง

Image	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 1	142.514310	41.127935	140.369886	38.336610
 2	143.003939	41.216575	140.958432	38.816038
 3	148.130505	41.434482	146.218010	43.408084
 4	148.731689	40.557827	146.861387	44.270774
 5	142.112125	41.383441	140.313668	37.875381
 6	141.765803	41.179348	140.124935	37.897556
 7	144.942290	41.915020	135.618886	35.263924
 8	148.222256	41.337985	145.997484	43.758056
 9	143.522353	41.210103	141.457057	39.394943

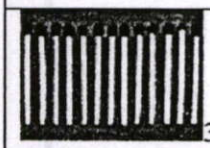
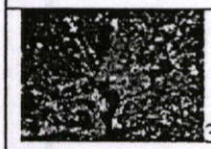
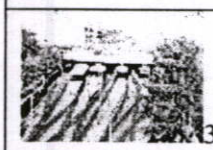
ตารางที่ ข.1 (ต่อ)

Image	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 10	142.135182	41.295508	140.264647	37.990472
 11	142.015193	41.430655	140.302777	37.834311
 12	141.568300	45.837667	139.371948	38.583173
 13	143.681103	41.253090	141.812345	39.624743
 14	142.020652	41.220115	139.897484	37.603081
 15	148.497671	41.244433	146.651058	44.278200
 16	144.400840	41.251353	142.505584	40.454621
 17	144.259608	41.235890	142.413813	40.292865
 18	147.777735	41.216789	145.814721	43.403168
 19	150.080626	50.282224	147.314481	45.282129

ตารางที่ ข.1 (ต่อ)

Image	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 20	145.746271	41.256224	143.924847	42.105752
 21	145.960830	41.233971	144.231160	42.145435
 22	144.793609	41.132580	143.054533	41.140033
 23	147.719289	41.218018	145.864459	43.884566
 24	141.772967	41.175313	140.130290	37.904120
 25	143.594779	41.159766	141.796981	39.839462
 26	141.276885	41.237302	139.396883	37.224413
 27	144.091744	41.163662	142.070402	40.052300
 28	145.498256	41.258697	143.564958	41.414633
 29	143.999553	41.225156	142.122572	39.964164

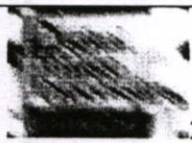
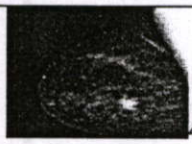
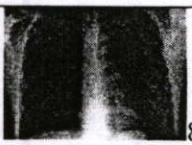
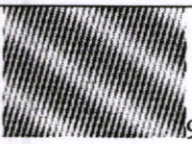
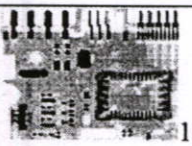
ตารางที่ ข.1 (ต่อ)

Image	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 30	144.400840	41.251353	142.505584	40.454621
 31	144.259608	41.235890	142.413813	40.292865
 32	148.294445	41.248937	146.758726	44.247687
 33	147.936450	41.177394	146.337231	43.958539
 34	148.362256	41.305746	146.729640	44.209241
 35	142.750236	41.240186	140.889844	38.800115
 36	143.536994	41.246178	141.673294	39.636002
 37	143.228941	41.321607	141.167715	38.916283
 38	142.045213	41.343723	140.131709	37.908215
 39	143.196612	41.269923	141.795781	39.766048

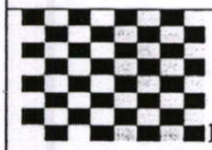
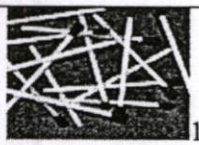
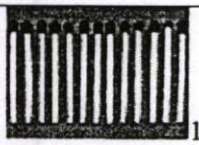
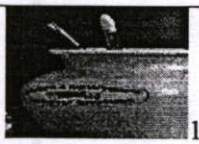
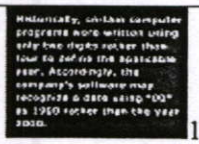
ตารางที่ ข.1 (ต่อ)

Image	PSNR			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 40	145.747350	41.011463	144.166599	42.257701
 41	140.947938	41.428270	139.166531	36.899454
เฉลี่ย	**144.598615**	**41.579799**	**139.146234**	**40.424140**

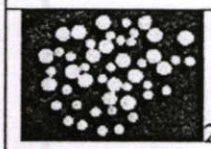
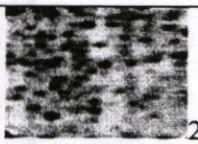
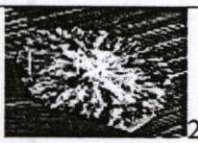
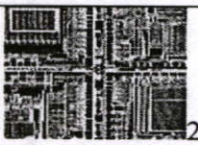
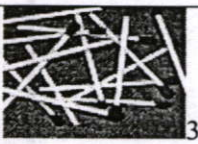
ตารางที่ ข.2 แสดงค่า PSNR เมื่อทำการตัดส่วนที่เป็น HH ทิ้ง

Image	PSNR ตัด HH ทิ้ง			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 1	48.832876	40.969776	48.832875	38.029352
 2	50.856877	41.526907	53.257624	38.706551
 3	44.409980	40.390641	44.409980	40.968991
 4	48.963173	40.854658	48.963172	43.058788
 5	18.046814	18.025301	18.046814	18.040048
 6	45.118549	40.164972	45.118549	37.177885
 7	25.341668	25.245346	25.341668	24.904456
 8	43.834320	39.815644	43.834320	40.910148
 9	64.802482	41.963087	64.802484	39.394943
 10	28.587918	28.394046	28.587918	28.113066

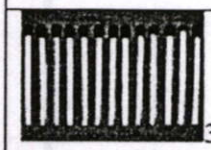
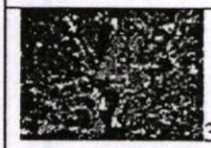
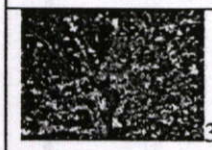
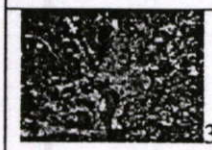
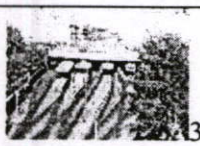
ตารางที่ ข.2 (ต่อ)

Image	PSNR ตัด HH ทิ้ง			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 11	17.151644	17.133357	17.151644	17.124074
 12	49.467658	44.342200	49.467656	38.248696
 13	45.510790	40.298055	45.510789	38.661348
 14	40.890989	38.330726	40.890989	35.974357
 15	49.143163	41.323293	49.143163	43.179433
 16	39.746516	37.705026	39.746516	37.087831
 17	41.648960	38.823891	41.648960	37.960152
 18	48.764128	41.264334	48.764128	42.396962
 19	23.516691	23.510760	23.516691	23.606095
 20	41.337986	38.493761	41.337985	38.600625

ตารางที่ ข.2 (ต่อ)

Image	PSNR ตัด HH ทิ้ง			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 21	35.343891	34.464596	35.343890	34.529970
 22	38.256787	36.680962	38.256787	36.448867
 23	48.248809	40.824433	48.248808	42.666898
 24	45.090867	40.151379	45.090866	37.178106
 25	54.331680	41.453029	54.331679	39.736962
 26	34.593478	33.922176	34.593478	32.727005
 27	49.712084	41.064951	49.712084	39.645613
 28	44.169745	39.757657	44.169744	39.564492
 29	41.816759	38.610370	41.816758	37.751278
 30	39.746516	37.705026	39.746516	37.087831

ตารางที่ ข.2 (ต่อ)

Image	PSNR ตัด HH ทิ้ง			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 31	41.648960	38.823891	41.648960	37.960152
 32	27.081590	26.912448	27.081590	27.079961
 33	27.167021	26.990452	27.167021	27.177183
 34	27.156069	26.983331	27.156069	27.168979
 35	27.969069	27.774619	27.969069	27.639384
 36	29.587675	29.322638	29.587675	29.167808
 37	33.674815	33.103516	33.674816	29.617281
 38	34.147756	33.431825	34.147756	32.596146
 39	29.126542	28.863702	29.126542	28.831425
 40	39.224110	37.318921	39.224110	37.511612

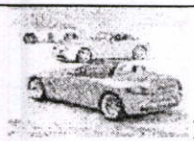
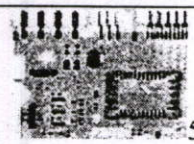
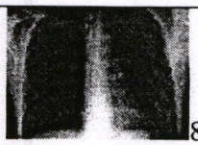
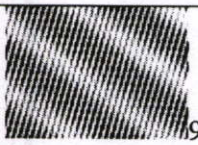
ตารางที่ ข.2 (ต่อ)

Image	PSNR ตัด HH ทิ้ง			
	D4 Lifting (dB.)	Integer D4 Lifting (dB.)	D4 Matrix (dB.)	Integer D4 Matrix (dB.)
 41	44.683893	40.183822	44.683893	36.263639
เฉลี่ย	**39.237836**	**35.437061**	**39.296391**	**34.646204**

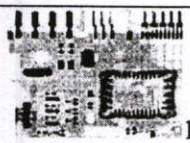
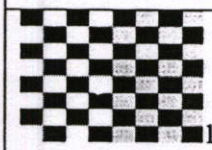
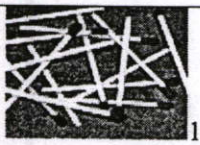
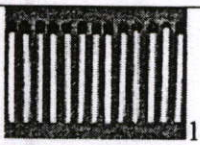
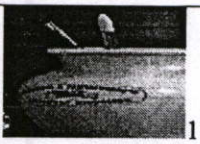
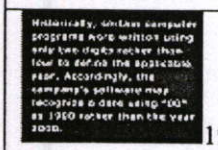
ภาคผนวก ค

แสดงค่า Zero Moment ในการแปลงเวฟเล็ต

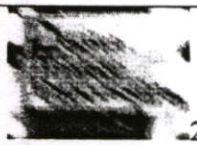
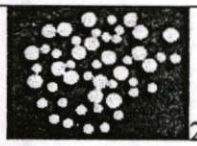
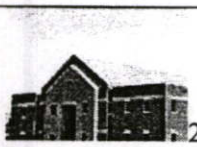
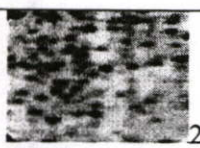
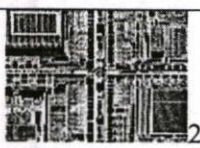
ตารางที่ ค.1 แสดงค่า Zero Moment

Image	Zero Moment All (Threshold = 1.5)			
	D4 Lifting (%)	Integer D4 Lifting (%)	D4 Matrix (%)	Integer D4 Matrix (%)
 1	40.048177	18.268229	49.127604	23.290364
 2	38.626098	25.669860	56.159973	28.956604
 3	31.445598	36.089611	37.497615	32.655620
 4	45.898667	40.385455	53.234330	36.993885
 5	18.022148	3.2948160	3.8682458	3.4121959
 6	34.707222	18.201111	41.885833	19.643888
 7	73.308000	26.334000	66.808800	26.643200
 8	36.598499	28.369947	42.181166	37.314215
 9	23.736572	26.251220	53.372192	32.202148

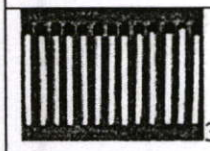
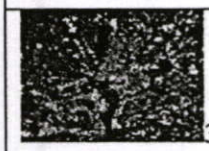
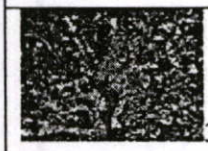
ตารางที่ ค.1 (ต่อ)

Image	Zero Moment All (Threshold = 1.5)			
	D4 Lifting (%)	Integer D4 Lifting (%)	D4 Matrix (%)	Integer D4 Matrix (%)
 10	4.4051146	5.0314616	4.7452278	4.5715632
 11	15.513392	2.4115802	2.5842826	2.3668411
 12	87.500000	59.790039	85.144042	60.357666
 13	22.930145	26.597976	32.297897	26.067352
 14	33.878698	19.484574	42.644220	20.433947
 15	30.492401	38.121795	40.925216	32.210540
 16	10.446936	13.620714	15.650813	12.683226
 17	11.849154	14.924011	17.306753	14.073560
 18	35.011428	31.840408	43.071428	30.793061
 19	90.923423	78.080727	79.659070	78.556792

ตารางที่ ค.1 (ต่อ)

Image	Zero Moment All (Threshold = 1.5)			
	D4 Lifting (%)	Integer D4 Lifting (%)	D4 Matrix (%)	Integer D4 Matrix (%)
 20	14.286892	20.025173	23.370659	19.108940
 21	8.6145235	12.658438	11.708668	11.103903
 22	13.925170	14.704895	16.694641	13.462829
 23	36.487835	38.344939	46.100671	32.871407
 24	34.996944	18.101111	41.929166	19.694444
 25	49.428131	21.193355	55.664541	25.610302
 26	10.742053	8.7851992	11.369999	9.4801192
 27	35.066223	24.249267	44.337463	23.735046
 28	56.370517	25.685606	28.105969	22.661071
 29	56.758759	21.964641	28.199365	22.211520

ตารางที่ ค.1 (ต่อ)

Image	Zero Moment All (Threshold = 1.5)			
	D4 Lifting (%)	Integer D4 Lifting (%)	D4 Matrix (%)	Integer D4 Matrix (%)
 30	10.446936	13.620714	15.650813	12.683226
 31	11.849154	14.924011	17.306753	14.073560
 32	4.6768188	6.5967559	5.7750701	5.8013916
 33	6.6711425	6.2744140	5.4431915	5.4153442
 34	5.4885864	6.9820404	6.1698913	6.1027526
 35	5.3436279	6.1786651	5.7289123	5.6301116
 36	5.8582305	7.3738098	6.6829681	6.6932678
 37	13.769856	15.199869	19.134765	14.708658
 38	16.642176	10.086394	18.061904	10.853061
 39	11.116027	7.0487976	8.3244323	6.8012237

ตารางที่ ค.1 (ต่อ)

Image	Zero Moment All (Threshold = 1.5)			
	D4 Lifting (%)	Integer D4 Lifting (%)	D4 Matrix (%)	Integer D4 Matrix (%)
 40	25.067520	18.763732	20.128631	16.374206
 41	49.544143	17.746734	44.364547	19.984436
เฉลี่ย	**30.079546**	**20.714051**	**30.449212**	**20.689938**

ภาคผนวก ง

ง.1 โปรแกรมภาษา Verilog ของโมดูลที่ใช้ในการบวกและลบเลขจำนวนจริง

```
module floating_adder(ina,inb,out,clk);
input [31:0] ina,inb;
input clk;
output [31:0] out;
wire c;
wire [7:0] expa,expb,s_left;
wire [22:0] mantissab,mantissaa;
wire [23:0] shift_result;
wire sa,sb,add;
wire [23:0] mantissa;
wire [30:0] t_ina,t_inb;
wire t_sa,tsb;
wire  t_sc;
wire [30:0] t_inc;
// Swap input a anb b if b > a
swap_input swap1        (
                        .ina(ina),
                        .inb(inb),
                        .sa(sa),
                        .sb(sb),
                        .expa(expa),
                        .expb(expb),
                        .manta(mantissaa),
                        .mantb(mantissab),
                        .clk(clk)
                        );

sub_8bit sub1    (
                .exa(expa),
                .exb(expb),
```

```
                .out(s_left)
                );

shift_left shift    (
                .inb({1'b1,mantissab}),
                .shift(s_left),
                .out(shift_result)
                );

assign add = sa ^ sb;
add_sub add_sub1        (
                        .add(add),
                        .ina({1'b1,mantissaa}),
                        .inb(shift_result),
                        .sum(mantissa),
                        .co(c)
                        );

add_sub_normalize nor1 (
                        .sa(sa),
                        .expa(expa),
                        .sum(mantissa),
                        .co(c),
                        .out({t_sc,t_inc}),
                        .clk(clk)
                        );
assign {t_sa,t_ina} = ina;
assign {t_sb,t_inb} = inb;

add_sub_zero zero1      (
                        .ina(t_ina),
                        .inb(t_inb),
```

```
                .inc(t_inc),
                .sa(t_sa),
                .sb(t_sb),
                .sc(t_sc),
                .out(out),
                .clk(clk)
                );

endmodule

module add_sub(add,ina,inb,sum,co);
input    add;
input    [23 : 0] ina,inb;
output   [23 : 0] sum;
output   co;
assign {co,sum} = add ? (ina - inb) : (ina + inb) ;
endmodule

module add_sub_normalize(sa,expa,sum,co,out,clk);
input sa,co,clk;
input [7:0] expa;
input [23:0] sum;
output [31:0] out;
reg [4:0] Adject;
reg [7:0] temp_exp;
reg [23:0] temp_mant;
initial Adject = 0;
always @(posedge clk)
begin
        casex(sum)

        24'b1??????????????????????? : Adject = 0;
```

```
        24'b01??????????????????????? : Adject = 1;
        24'b001??????????????????????? : Adject = 2;
        24'b0001??????????????????????? : Adject = 3;
        24'b00001??????????????????????? : Adject = 4;
        24'b000001??????????????????????? : Adject = 5;
        24'b0000001??????????????????? : Adject = 6;
        24'b00000001??????????????????? : Adject = 7;
        24'b000000001??????????????????? : Adject = 8;
        24'b0000000001??????????????? : Adject = 9;
        24'b00000000001??????????????? : Adject = 10;
        24'b000000000001??????????? : Adject = 11;
        24'b0000000000001??????????? : Adject = 12;
        24'b00000000000001????????? : Adject = 13;
        24'b000000000000001????????? : Adject = 14;
        24'b0000000000000001??????? : Adject = 15;
        24'b00000000000000001??????? : Adject = 16;
        24'b000000000000000001????? : Adject = 17;
        24'b0000000000000000001????? : Adject = 18;
        24'b00000000000000000001???? : Adject = 19;
        24'b000000000000000000001??? : Adject = 20;
        24'b0000000000000000000001?? : Adject = 21;
        24'b00000000000000000000001? : Adject = 22;
        24'b000000000000000000000001 : Adject = 23;

endcase

end

always @(posedge clk)
begin

        if (co)
```

```
begin
        temp_exp = expa + 1;
        temp_mant = sum >> 1;
 end
else
begin
        temp_exp = expa - Adject;
        temp_mant = sum << Adject;
end
end

assign  out [31] = sa;
assign  out [30:23] = temp_exp;
assign  out [22] =  temp_mant[22];
assign  out [21] =  temp_mant[21];
assign  out [20] =  temp_mant[20];
assign  out [19] =  temp_mant[19];
assign  out [18] =  temp_mant[18];
assign  out [17] =  temp_mant[17];
assign  out [16] =  temp_mant[16];
assign  out [15] =  temp_mant[15];
assign  out [14] =  temp_mant[14];
assign  out [13] =  temp_mant[13];
assign  out [12] =  temp_mant[12];
assign  out [11] =  temp_mant[11];
assign  out [10] =  temp_mant[10];
assign  out [9] =  temp_mant[9];
assign  out [8] =  temp_mant[8];
assign  out [7] =  temp_mant[7];
assign  out [6] =  temp_mant[6];
assign  out [5] =  temp_mant[5];
assign  out [4] =  temp_mant[4];
```

```
        assign   out [3] =  temp_mant[3];
        assign   out [2] =  temp_mant[2];
        assign   out [1] =  temp_mant[1];
        assign   out [0] =  temp_mant[0];

endmodule

module add_sub_zero(ina,inb,inc,sa,sb,sc,out,clk);
input [30:0] ina,inb,inc;
input sa,sb,sc,clk;
output [31:0] out;
reg [31:0] out;
wire a_zero,b_zero;
wire sign;
wire equ_inp;
wire equ;
assign sign = sa ^ sb;
assign equ_inp = !(&(ina ^ inb));
assign equ = sign & equ_inp;
assign a_zero = |ina;
assign b_zero = |inb;
always @(posedge clk)
begin
        if(equ)
        case({a_zero,b_zero})
                2'b00   :       out = 0;
                2'b01   :       out = {sb,inb};
                2'b10   :       out = {sa,ina};
                2'b11   :       out = {sc,inc};
        endcase

        else
```

```
            case({a_zero,b_zero})
                    2'b00   :       out = 0;
                    2'b01   :       out = {sb,inb};
                    2'b10   :       out = {sa,ina};
                    2'b11   :       out = {sc,inc};
            endcase
    end

endmodule
module shift_left(inb,shift,out);
input [23:0] inb;
input [7:0] shift;
output [23:0] out;
assign out = inb >> shift;
endmodule

module sub_8bit(exa,exb,out);
input [7:0] exa,exb;
output [7:0] out;
assign out = exa - exb;
endmodule

module swap_input(ina,inb,sa,sb,expa,expb,manta,mantb,clk);
input [31:0] ina,inb;
input clk;
output sa,sb;
output [7:0] expa,expb;
output [22:0] manta,mantb;
reg sa,sb;
reg [7:0] expa,expb;
reg [22:0] manta,mantb;
always @(posedge clk)
```

```
begin
        if( ina [30:0] < inb [30:0] )
        begin
                sa <= inb [31];
                expa <= inb [30:23];
                manta <= inb [22:0];
                sb <= ina [31];
                expb <= ina [30:23];
                mantb <= ina [22:0];
        end

        else
        begin
                sa <= ina [31];
                expa <= ina [30:23];
                manta <= ina [22:0];
                sb <= inb [31];
                expb <= inb [30:23];
                mantb <= inb [22:0];
         end
        end

endmodule
```

ง.2 โปรแกรมภาษา Verilog ของโมดูลที่ใช้ในการคูณเลขจำนวนจริง

```
module floating_mul(ina,inb,out,clk);
input [31:0] ina,inb;
input clk;
output [31:0] out;
wire sa,sb;
wire [7:0] expa,expb;
```

```
wire [22:0] man_a,man_b;
wire [8:0] exponent_add;
wire [7:0] exp_out1,exp_out;
wire [22:0] man_out;
wire [47:0] mul_ab;
wire sign;
assign {sa,expa,man_a} = ina;
assign {sb,expb,man_b} = inb;
assign sign = sa ^ sb;

mul_add_exp exp_add1  (
                      .expa(expa),              // input 8 bit
                      .expb(expb),              // input 8 bit
                      .out(exponent_add)        // output 9 bit
                      );
subtract_bias sub_bias1  (
                      .exp(exponent_add),       // input 9 bit
                      .out(exp_out1),           // output 8 bit
                      .co()                     // output carry 1 bit use to overflow
                      );

multiplier mul_man1     (
                      .mantissa_a({1'b1,man_a}),      // input 24 bit
                      .mantissa_b({1'b1,man_b}),      // input 24 bit
                      .result(mul_ab),                // output 48 bit
                      .clk(clk)                       // input 1 bit
                      );

mul_normalize mul_nor1        (
                              .exp_in(exp_out1),      // input 8 bit
                              .man_in(mul_ab),        // mantisa 48 bit
                              .exp_out(exp_out),      // output 8 bit
```

```
                .man_out(man_out),      // output 23 bit
                .clk(clk)               // input 1 bit
                );

mul_zero mul_zero1    (
                .ina({expa,man_a}),     // input 31 bit
                .inb({expb,man_b}),     // input 31 bit
                .sc(sign),              // input 1 bit
                .exponent(exp_out),     // input 8 bit
                .mantissa(man_out),     // input 23 bit
                .clk(clk),              // input 1 bit
                .out(out)               // output 32 bit
                );
endmodule

module mul_add_exp(expa,expb,out);
input [7:0] expa,expb;
output [8:0] out;
assign out = expa + expb;
endmodule

module mul_normalize(exp_in,man_in,exp_out,man_out,clk);
input clk;
input [47:0] man_in;
input [7:0] exp_in;
output [22:0] man_out;
output [7:0] exp_out;
reg [22:0] man_out;
reg [7:0] exp_out;
reg [47:0] reg_temp;
reg [5:0] adject;
```

```
always @(posedge clk)
begin
        casex(man_in)
        48'b1???????????????????????????????????????????? : adject = 1;
        48'b01??????????????????????????????????????????? : adject = 2;
        48'b001?????????????????????????????????????????? : adject = 3;
        48'b0001????????????????????????????????????????? : adject = 4;
        48'b00001???????????????????????????????????????? : adject = 5;
        48'b000001??????????????????????????????????????? : adject = 6;
        48'b0000001?????????????????????????????????????? : adject = 7;
        48'b00000001????????????????????????????????????? : adject = 8;
        48'b000000001???????????????????????????????????? : adject = 9;
        48'b0000000001??????????????????????????????????? : adject = 10;
        48'b00000000001?????????????????????????????????? : adject = 11;
        48'b000000000001????????????????????????????????? : adject = 12;
        48'b0000000000001???????????????????????????????? : adject = 13;
        48'b00000000000001??????????????????????????????? : adject = 14;
        48'b000000000000001???????????????????????????????? : adject = 15;
        48'b0000000000000001??????????????????????????????? : adject = 16;
        48'b00000000000000001???????????????????????????? : adject = 17;
        48'b000000000000000001??????????????????????????? : adject = 18;
        48'b0000000000000000001???????????????????????????? : adject = 19;
        48'b00000000000000000001??????????????????????????? : adject = 20;
        48'b000000000000000000001???????????????????????? : adject = 21;
        48'b0000000000000000000001??????????????????????? : adject = 22;
        48'b00000000000000000000001???????????????????????? : adject = 23;
        48'b000000000000000000000001??????????????????????? : adject = 24;
        48'b0000000000000000000000001???????????????????? : adject = 25;
        48'b00000000000000000000000001??????????????????? : adject = 26;
        48'b000000000000000000000000001???????????????????? : adject = 27;
        48'b0000000000000000000000000001??????????????????? : adject = 28;
        48'b00000000000000000000000000001??????????????????? : adject = 29;
```

```
        48'b000000000000000000000000000001??????????????????? :adject = 30;
        48'b000000000000000000000000000001???????????????????? :adject = 31;
        48'b000000000000000000000000000001????????????????????? :adject = 32;
        48'b000000000000000000000000000001?????????????????? :adject = 33;
        48'b000000000000000000000000000001??????????????????? :adject = 34;
        48'b000000000000000000000000000001???????????????? :adject = 35;
        48'b000000000000000000000000000001???????????????? :adject = 36;
        48'b000000000000000000000000000001???????????? :adject = 37;
        48'b000000000000000000000000000001????????????? :adject = 38;
        48'b000000000000000000000000000001????????? :adject = 39;
        48'b000000000000000000000000000001???????? :adject = 40;
        48'b000000000000000000000000000001??????? :adject = 41;
        48'b000000000000000000000000000001?????? :adject = 42;
        48'b000000000000000000000000000001????? :adject = 43;
        48'b000000000000000000000000000001???? :adject = 44;
        48'b000000000000000000000000000001??? :adject = 45;
        48'b000000000000000000000000000001?? :adject = 46;
        48'b000000000000000000000000000001? :adject = 47;
        48'b000000000000000000000000000001 :adject = 48;
endcase
if(man_in[47])
begin
        exp_out = exp_in + 1;
        reg_temp <= man_in >> 1;
end
else
begin
        exp_out = exp_in;
        reg_temp <= man_in << adject;
end

man_out[22] = reg_temp[47];
```

```
man_out[21] = reg_temp[46];
man_out[20] = reg_temp[45];
man_out[19] = reg_temp[44];
man_out[18] = reg_temp[43];
man_out[17] = reg_temp[42];
man_out[16] = reg_temp[41];
man_out[15] = reg_temp[40];
man_out[14] = reg_temp[39];
man_out[13] = reg_temp[38];
man_out[12] = reg_temp[37];
man_out[11] = reg_temp[36];
man_out[10] = reg_temp[35];
man_out[9] = reg_temp[34];
man_out[8] = reg_temp[33];
man_out[7] = reg_temp[32];
man_out[6] = reg_temp[31];
man_out[5] = reg_temp[30];
man_out[4] = reg_temp[29];
man_out[3] = reg_temp[28];
man_out[2] = reg_temp[27];
man_out[1] = reg_temp[26];
man_out[0] = reg_temp[25];

 end

endmodule

module mul_zero(ina,inb,sc,exponent,mantissa,clk,out);
input [30:0] ina;
input [30:0] inb;
input sc,clk;
input [7:0] exponent;
```

```
input [22:0] mantissa;
output [31:0] out;
wire temp;
wire zero_a,zero_b;
assign zero_a = |ina;
assign zero_b = |inb;
assign temp =     zero_a & zero_b;
assign out = temp ? {sc,exponent,mantissa} : 32'b00000000_00000000_00000000_00000000;
endmodule

module multiplier(mantissa_a,mantissa_b,result,clk);
input clk;
input [23:0] mantissa_a,mantissa_b;
output [47:0] result;
reg [47:0] prod,result;
always @(posedge clk)
        prod <= #1 mantissa_a * mantissa_b;

always @(posedge clk)
        result <= #1 prod;
endmodule

module subtract_bias(exp,out,co);
input [8:0] exp;
output [7:0] out;
output co;
assign {co,out} = exp - 127;
endmodule
```

ง.3 โปรแกรมภาษา Verilog ที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วย Lifting แบบเลขจำนวนจริง

```
// Define Data Input
// S0 = S [ 2n - 2 ]
// S1 = S [ 2n - 1 ]
// S2 = S [ 2n ]
// S3 = S [ 2n + 1 ]
// S4 = S [ 2n + 2 ]
// S5 = S [ 2n + 5 ]

//Define Constant Value
// (Sqr(3) - 2 ) / 4  = BD 89 30 A3 = 32'b1011_1101_1000_1001_0011_0000_1010_0011
// Sqr(3) / 4         = 3E DD B3 D7 = 32'b0011_1110_1101_1101_1011_0011_1101_0111
// Sqr(3)             = 3F DD B3 D7 = 2'b0011_1111_1101_1101_1011_0011_1101_0111
// (Sqr(3) - 1) / 2   = 3E BB 67 AF = 32'b0011_1110_1011_1011_0110_0111_1010_1111
// (Sqr(3) + 1) / 2   = 3F AE D9 EC = 32'b0011_1111_1010_1110_1101_1001_1110_1100

module Daubechies4(S0,S1,S2,S3,S4,S5,s,d,clk);

input [31:0] S0,S1,S2,S3,S4,S5;
input clk;
output [31:0] s,d;
wire [31:0]temp1,temp2,temp3,temp4;

M1 module1     (
               .ina(S1),
               .inb(S2),
               .clk(clk),
               .out(temp1)
               );
```

```
M2 module2     (
               .ina(S2),
               .inb(S3),
               .clk(clk),
               .out1(temp2),
               .out2(temp3)
               );

M3 module3     (
               .ina(S3),
               .inb(S5),
               .clk(clk),
               .out(temp4)
               );

M4 modual4     (
               .ina(temp2),
               .inb(temp1),
               .clk(clk),
               .out(s)
               );

M5 modual5     (
               .ina(temp3),
               .inb(temp4),
               .clk(clk),
               .out(d)
               );
Endmodule

module M1(ina,inb,clk,out);
input [31:0] ina,inb;
```

```
input clk;
output [31:0] out;
wire [31:0] temp1,temp2;

floating_mul m1mul1    (
                       .ina(ina),
                       .inb(32'b0011_1111_1101_1101_1011_0011_1101_0111),
                .out(temp1),
                       .clk(clk)
                       );

floating_adder m1add1  (
                       .ina(inb),
                       .inb({~temp1[31],temp1[30:0]}),
                       .out(temp2),
                       .clk(clk)
                       );

floating_adder m1add2  (
                       .ina(temp2),
                       .inb(32'b1011_1101_1000_1001_0011_0000_1010_0011),
                       .out(out),
                       .clk(clk)
                       );

endmodule

module M2(ina,inb,clk,out1,out2);
input [31:0] ina,inb;
input clk;
output [31:0] out1,out2;
wire [31:0] temp1,temp2,temp3;
```

```
floating_mul m2mul1    (
                       .ina(inb),
                       .inb(32'b0011_1111_1101_1101_1011_0011_1101_0111),
                       .out(temp1),
                       .clk(clk)
                       );

floating_adder m2add1  (
                       .ina(ina),
                       .inb(temp1),
                       .out(out2),
                       .clk(clk)
                       );

floating_mul m2mul2    (
                       .ina(out2),
                       .inb(32'b0011_1110_1101_1101_1011_0011_1101_0111),
                       .out(temp3),
                       .clk(clk)
                       );

floating_adder m2add2  (
                       .ina(inb),
                       .inb({temp3[31],temp3[30:0]}),
                       .out(out1),
                       .clk(clk)
                       );
endmodule

module M3(ina,inb,clk,out);
input [31:0] ina,inb;
input clk;
```

```
output [31:0] out;
wire [31:0] temp1,temp2,temp3;

floating_mul m3mul1    (
                       .ina(ina),
                       .inb(32'b0011_1110_1101_1101_1011_0011_1101_0111),
                       .out(temp1),
                       .clk(clk)
                       );

floating_adder m3add1  (
                       .ina(inb),
                       .inb({~temp1[31], temp1[30:0]}),
                       .out(temp2),
                       .clk(clk)
                       );

floating_mul m3mul2    (
                       .ina(inb),
                       .inb(32'b0011_1111_1101_1101_1011_0011_1101_0111),
                       .out(temp3),
                       .clk(clk)
                       );

floating_adder m3add2  (
                       .ina(temp2),
                       .inb(temp3),
                       .out(out),
                       .clk(clk)
                       );

endmodule
```

```
module M4(ina,inb,clk,out);
input [31:0] ina,inb;
input clk;
output [31:0] out;
wire [31:0] temp1;

floating_adder m4add1  (
                       .ina(inb),
                       .inb({ina[31],ina[30:0]}),
                       .out(temp1),
                       .clk(clk)
                       );

floating_mul m4mul1    (
                       .ina(temp1),
                       .inb(32'b0011_1111_0010_1110_1101_1001_1110_1100),
                       .out(out),
                       .clk(clk)
                       );

Endmodule

module M5(ina,inb,clk,out);
input [31:0] ina,inb;
input clk;
output [31:0] out;
wire [31:0] temp1,temp2,temp3;
floating_adder m5add1  (
                       .ina(ina),
                       .inb({~inb[31],inb[30:0]}),
                       .out(temp1),
                       .clk(clk)
```

```
                    );

floating_mul m5mul1    (
                    .ina(ina),
                    .inb(32'b1011_1101_1000_1001_0011_0000_1010_0011),
                    .out(temp2),
                    .clk(clk)
                    );

floating_adder m5add2  (
                    .ina(temp1),
                    .inb({~temp2[31],temp2[30:0]}),
                    .out(temp3),
                    .clk(clk)
                    );

floating_mul m5mul2    (
                    .ina(temp3),
                    .inb(32'b0011_1110_0011_1011_0110_0111_1010_1110),
                    .out(out),
                    .clk(clk)
                    );

endmodule
```

ง.4 โปรแกรมภาษา Verilog ที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วย Lifting แบบเลขจำนวนเต็ม

```
// S0 = S[2n-2]
// S1 = S[2n-1]
// S2 = S[2n]
// S3 = S[2n+1]
```

```
// S4 = S[2n+2]
// S5 = S[2n+3]

module int_d4_lifting(S0,S1,S2,S3,S4,S5,s,d,clk);
input [15:0] S0,S1,S2,S3,S4,S5;
input clk;
output [15:0] s,d;
wire [15:0] temp1,temp2,temp3,temp4,temp5,temp6,temp7;

M1 M1          (
               .S0(S0),
               .S1(S1),
               .M1_Out(temp1),
               .clk(clk)
               );

M2 M2          (
               .S2(S2),
               .S3(S3),
               .M2_Out1(temp2),
               .M2_Out2(temp3),
               .clk(clk)
               );

M3 M3          (
               .S4(S4),
               .S5(S5),
               .M3_Out(temp4),
               .clk(clk)
               );

M4 M4          (
```

```
                .M4_In1(temp1),
                .M4_In2(temp3),
                .d(d),
                .clk(clk)
                );

M5 M5           (
                .M5_In1(temp2),
                .M5_In2(temp2),
                .M5_In3(temp4),
                .s(s),
                .clk(clk)
                );

endmodule

module M1(S0,S1,M1_Out,clk);
input [15:0] S0,S1;
input clk;
output [15:0] M1_Out;
wire [15:0] temp1,temp2,temp3,temp4;

signed_mult mul1        (
                        .dataa(S1),
                        .datab(16'b0000_0000_0011_0111),
                        .mult_result(temp1),
                        .clk(clk)
                        );

assign temp2[15] = ~temp1[15];
assign temp2[14:0] = temp1[14:0] >> 5;
```

```
sign_adder appro_a3     (
                        .dataa(S0),
                        .datab(temp2),
                        .add_result(temp3),
                        .clk(clk)
                        );

signed_mult mul2        (
                        .dataa(temp3),
                        .datab(16'b1000_0000_0000_0010),
                        .mult_result(temp4),
                        .clk(clk)
                        );

assign M1_Out[15] = temp4[15];
assign M1_Out[14:0] = temp4[14:0] >> 5;
endmodule

module M2(S2,S3,M2_Out1,M2_Out2,clk);
input [15:0] S2,S3;
input clk;
output [15:0] M2_Out1,M2_Out2;
wire [15:0] temp1,temp2,temp3,temp4,temp5;

signed_mult mul1        (
                        .dataa(S3),
                        .datab(16'b0000_0000_0011_0111),
                        .mult_result(temp1),
                        .clk(clk)
                        );

assign temp2[15] = temp1[15];
```

```
assign temp2[14:0] = temp1[14:0] >> 5;

sign_adder appro_a1     (
                        .dataa(S2),
                        .datab(temp2),
                        .add_result(temp3),
                        .clk(clk)
                        );

assign M2_Out1 = temp3;

signed_mult mul2        (
                        .dataa(temp3),
                        .datab(16'b0000_0000_0000_1110),
                        .mult_result(temp4),
                        .clk(clk)
                        );

assign temp5[15] = ~temp4[15];
assign temp5[14:0] = temp4[14:0] >> 5;

sign_adder appro_a2     (
                        .dataa(S3),
                        .datab(temp5),
                        .add_result(M2_Out2),
                        .clk(clk)
                        );

endmodule

module M3(S4,S5,M3_Out,clk);
input [15:0] S4,S5;
```

```
input clk;
output [15:0] M3_Out;
wire  [15:0] temp1,temp2,temp3,temp4,temp5;

signed_mult mul1        (
                        .dataa(S4),
                        .datab(16'b0000_0000_0000_1110),
                        .mult_result(temp1),
                        .clk(clk)
                        );

assign temp2[15] = temp1[15];
assign temp2[14:0] = temp1[14:0] >> 5;

sign_adder appro_a1     (
                        .dataa(temp2),
                        .datab({~S5[15],S5[14:0]}),
                        .add_result(temp3),
                        .clk(clk)
                        );

signed_mult mul2        (
                        .dataa(S5),
                        .datab(16'b0000_0000_0000_1110),
                        .mult_result(temp4),
                        .clk(clk)
                        );

assign temp5[15] = temp4[15];
assign temp5[14:0] = temp4[14:0] >> 5;

sign_adder appro_a2     (
```

```
                    .dataa(temp3),
                    .datab(temp5),
                    .add_result(M3_Out),
                    .clk(clk)
                    );

Endmodule

module M4(M4_In1,M4_In2,d,clk);
input [15:0] M4_In1,M4_In2;
input clk;
output [15:0] d;
wire [15:0] temp1,temp2;

sign_adder appro_a1     (
                    .dataa(M4_In2),
                    .datab({~M4_In1[15],M4_In1[14:0]}),
                    .add_result(temp1),
                    .clk(clk)
                    );

signed_mult mul1        (
                    .dataa(temp1),
                    .datab(16'b0000_0000_0100_0010),
                    .mult_result(temp2),
                    .clk(clk)
                    );

assign d[15] = temp2[15];
assign d[14:0] = temp2[14:0] >> 5;

endmodule
```

```
module M5(M5_In1,M5_In2,M5_In3,s,clk);
input [15:0] M5_In1,M5_In2,M5_In3;
input clk;
output [15:0] s;
wire [15:0] temp1,temp2,temp3,temp4,temp5;

sign_adder appro_a1     (
                        .dataa(M5_In2),
                        .datab({~M5_In3[15],M5_In3[14:0]}),
                        .add_result(temp1),
                        .clk(clk)
                        );

signed_mult mul1        (
                        .dataa(M5_In1),
                        .datab(16'b1000_0000_0000_0010),
                        .mult_result(temp2),
                        .clk(clk)
                        );

assign temp3[15] = temp2[15];
assign temp3[14:0] = temp2[14:0] >> 5;

sign_adder appro_a2     (
                        .dataa(temp3),
                        .datab({~temp1[15],temp1[14:0]}),
                        .add_result(temp4),
                        .clk(clk)
                        );

signed_mult mul2        (
                        .dataa(temp4),
```

```
                .datab(16'b1000_0000_0001_0001),
                .mult_result(temp5),
                .clk(clk)
                );

assign s[15] = temp5[15];
assign s[14:0] = temp5[14:0] >> 5;

endmodule
```

ง.5 โปรแกรมภาษา Verilog ที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วย Matrix (Scaling Function และ Wavelet Function) แบบเลขจำนวนจริง

```
// h0 = 0.4829629  = 0011_1110_1111_0111_0100_0110_1110_1010
// h1 = 0.8365163  = 0011_1111_0101_0110_0010_0101_1110-1111
// h2 = 0.2241439  = 0011_1110_0110_0101_1000_0101_1111_1011
// h3 = -0.1294095 = 1011_1110_0000_0100_1000_0011_1110_1101
// g0 = -0.1294095 = 1011_1110_0000_0100_1000_0011_1110_1101
// g1 = -0.2241439 = 1011_1110_0110_0101_1000_0101_1111_1011
// g2 =  0.8365163 = 0011_1111_0101_0110_0010_0101_1110_1111
// g3 = -0.4829629 = 1011 1110 1111 0111 0100 0110 1110 1010

module d4_matrix(S0,S1,S2,S3,L,H,clk);
input [31:0] S0;
input [31:0] S1;
input [31:0] S2;
input [31:0] S3;
input clk;
output [31:0] L;
output [31:0] H;
wire [31:0] T_L1,T_L2,T_L3,T_L4,T_L5,T_L6;
wire [31:0] T_H1,T_H2,T_H3,T_H4,T_H5,T_H6;
```

```
////////////////////////// LOW PASS FILTER //////////////////////////////

floating_mul M1        (
                       .ina(S0),
                       .inb(32'b0011_1110_1111_0111_0100_0110_1110_1010),
                       .out(T_L1),
                       .clk(clk)
                       );

floating_mul M2        (
                       .ina(S1),
                       .inb(32'b0011_1111_0101_0110_0010_0101_1110_1111),
                       .out(T_L2)
                       );

floating_adder A1      (
                       .ina(T_L1),
                       .inb(T_L2),
                       .out(T_L3)
                       );
floating_mul M3        (
                       .ina(S2),
                       .inb(32'b0011_1110_0110_0101_1000_0101_1111_1011),
                       .out(T_L4)
                       );

 floating_mul M4       (
                       .ina(S3),
                       .inb(32'b1011_1110_0000_0100_1000_0011_1110_1101),
                       .out(T_L5),
                       .clk(clk)
                       );
```

```
floating_adder A2       (
                        .ina(T_L4),
                        .inb(T_L5),
                        .out(T_L6),
                        .clk(clk)
                        );


floating_adder A3       (
                        .ina(T_L3),
                        .inb(T_L6),
                        .out(L),
                        .clk(clk)
                        );

////////////////////////////// HIGH PASS FILTER //////////////////////////////

floating_mul M5(
                        .ina(S0),
                        .inb(32'b1011_1110_ 0000_0100_1000_0011_1110_1101),
                        .out(T_H1),
                        .clk(clk)
                        );

floating_mul M6         (
                        .ina(S1),
                        .inb(32'b1011_1110_0110_0101_1000_0101_1111_1011),
                        .out(T_H2),
                        .clk(clk)
                        );
```

```
floating_adder A4       (
                                .ina(T_H1),
                                .inb(T_H2),
                                .out(T_H3),
                                .clk(clk)
                        );

floating_mul M7         (
                        .ina(S2),
                        .inb(32'b0011_1111_0101_0110_0010_0101_1110_1111),
                        .out(T_H4),
                        .clk(clk)
                        );

floating_mul M8         (
                        .ina(S3),
                        .inb(32'b1011_1110_1111_0111_0100_0110_1110_1010),
                        .out(T_H5),
                        .clk(clk)
                        );

floating_adder A5       (
                        .ina(T_H4),
                        .inb(T_H5),
                        .out(T_H6),
                        .clk(clk)
                        );

floating_adder A6       (
```

```
                .ina(T_H3),
                .inb(T_H6),
                .out(H),
                .clk(clk)
                );

endmodule
```

ง.6 โปรแกรมภาษา Verilog ที่ใช้ในการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วย Scaling Function และ Wavelet Function แบบเลขจำนวนเต็ม

```
module int_d4_matrix(S1,S2,S3,S4,d,clk);
input [15:0] S1;
input [15:0] S2;
input [15:0] S3;
input [15:0] S4;
input clk;
output [15:0] d;
output [15:0] s;
wire [15:0]temp_a1,temp_a2,temp_a3,temp_a4,temp_a5,temp_a6,temp_a7;
wire [15:0]temp_d1,temp_d2,temp_d3,temp_d4,temp_d5,temp_d6,temp_d7;

//############## Approximation Path ##################
signed_mult appro_m1    (
                .dataa(S1),
                .datab(16'b1000_0000_0000_1000),
                .mult_result(temp_a1),
                .clk(clk)
                );

signed_mult appro_m2    (
                .dataa(S2),
```

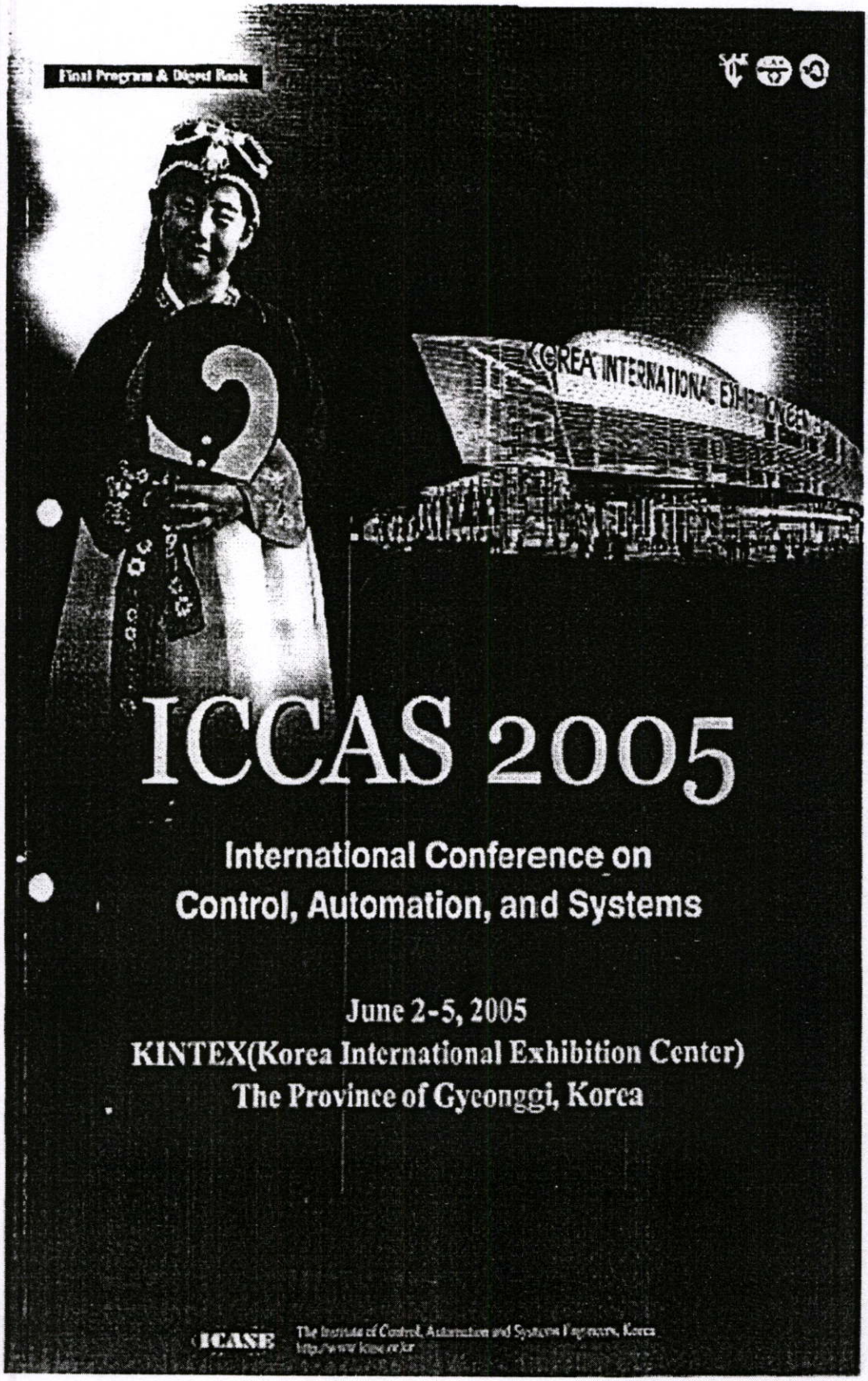
```
                .datab(16'b0000_0000_0000_1110),
                .mult_result(temp_a2),
                .clk(clk)
                );

signed_mult appro_m3    (
                .dataa(S3),
                .datab(16'b0000_0000_0011_0110),
                .mult_result(temp_a3),
                .clk(clk)
                );

signed_mult appro_m4    (
                .dataa(S4),
                .datab(16'b0000_0000_0001_1111),        //31
                .mult_result(temp_a4),
                .clk(clk)
                );

sign_adder appro_a1     (
                .dataa(temp_a1),
                .datab(temp_a2),
                .add_result(temp_a5),
                .clk(clk)
                );

sign_adder appro_a2     (
                .dataa(temp_a3),
                .datab(temp_a4),
                .add_result(temp_a6),
                .clk(clk)
                );
```

```
sign_adder appro_a3     (
                        .dataa(temp_a5),
                        .datab(temp_a6),
                        .add_result(temp_a7),
                        .clk(clk)
                        );

assign s[15] = temp_a7[15];
assign s[14:0] = temp_a7[14:0] >> 6;

//########## Detail Path #########

 signed_mult detail_m1  (
                        .dataa(S1),
                        .datab(16'b0000_0000_0001_1111),
                        .mult_result(temp_d1),
                        .clk(clk)
                        );
signed_mult detail_m2   (
                        .dataa(S2),
                        .datab(16'b0000_0000_0011_0110),
                        .mult_result(temp_d2),
                        .clk(clk)
                        );

signed_mult detail_m3   (
                        .dataa(S3),
                        .datab(16'b0000_0000_0000_1110),
                        .mult_result(temp_d3),
                        .clk(clk)
                        );
```

```
signed_mult detail_m4   (
                        .dataa(S4),
                        .datab(16'b1000_0000_0000_1000),
                        .mult_result(temp_d4),
                        .clk(clk)
                        );

sign_adder detail_a1    (
                        .dataa(temp_d1),
                        .datab(temp_d2),
                        .add_result(temp_d5),
                        .clk(clk)
                        );

sign_adder detail_a2    (
                        .dataa(temp_d3),
                        .datab(temp_d4),
                        .add_result(temp_d6),
                        .clk(clk)
                        );

sign_adder detail_a3    (
                        .dataa(temp_d5),
                        .datab(temp_d6),
                        .add_result(temp_d7),
                        .clk(clk)
                        );

assign d[15] = temp_d7[15];
assign d[14:0] = temp_d7[14:0] >> 6;

endmodule
```

ภาคผนวก จ

ผลงานวิจัยที่ได้รับการตีพิมพ์



A Design of Parallel Processing for Wavelet Transformation on FPGA (ICCAS 2005)

Kraiuek Ngowsuwan*, Orachat Chisobhuk**, and Charoen Vongchumyen***

Department of Computer Engineering Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang
Ladkrabang, Bangkok, Thailand, 10520

Abstract: In this paper we introduce a design of parallel architecture for wavelet transformation on FPGA. We implement wavelet transforms though lifting scheme and apply Daubechies4 transform equations. This technique has an advantage that we can obtain perfect reconstruction of the data. We divide our process to high pass filter and low pass filter. With this division, we can find coefficients from low and high pass filters simultaneously using parallel processing properties of FPGA to reduce processing time. From the equations, we have to design real number computation module, referred to IEEE754 standard. We choose 32 bit computation that is fine enough to reconstruct data. After that we arrange the real number module according to Daubechies4 transform though lifting scheme.

1. INTRODUCTION

As we known the digital image processing system requires high speed and efficient processing unit to run in real-time systems. Traditional approach can be processed using PC with the high speed and performance such as mainframe. Therefore, the regular personal computer is not suitable if we want to work in real time system. At present, people prefer a small system which can process correctly and fast. Therefore, our research present a designing of wavelet transform through lifting scheme implementing on small FPGA system. The wavelet transform is widely used in signal processing. Moreover, wavelet transform can be applied in many applications [1, 2]. The accuracy of wavelet transform depends on the design of low and high pass filter, which represent the scaling function of the Mother wavelet. The result from scaling function is down-sampled [3]. The advantage of the proposed method is that it can process wavelet transformation of an image fast and efficient.

Typically, the image and digital signal processing applications require high calculation throughput [4, 5]. The arithmetic operators presented here are implemented for real time signal processing. We design floating-point arithmetic based on IEEE754 standard [6]. We designed 32-bit floating-point computation which is fine enough for image and digital signal processing application. After that, we arrange our floating-point arithmetic unit according to Daubechies4 transform though lifting scheme equation [7, 8]. All constants of equations are transformed to the format according to IEEE754 before processed to reduce complexity of the computation. The results of Daubechies4 equations consist of approximation and detail. In this module, we can use parallel processing properties of FPGA to calculate approximation and detail simultaneously.

2. OVERVIEW OF THE PROPOSED DWT ARCHITECTURE ON FPGA

The overview structure of DWT is shown in Fig.1. First, this module receives 32-bit input which is already in IEEE754 standard format. Next, the input is processed with Daubechies4 equations within module. Internal structure of the module composes of floating-point units which can compute addition, subtraction and multiplication according to the equations. The results are the coefficients of both low and high filters. We call the results as "approximation" and "detail" respectively. The designing and internal structure will be described in next section.

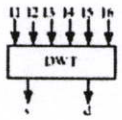


Fig. 1 Internal structure of DWT

2.1 Floating Point Format Representation

The designing of real number computation module is referred to IEEE754 standard. We use 32 bit format as shown in Fig.2. "s" is sign bit, "e" is exponent field and "f" is mantissa field to collect floating-point. The value of each part can be calculated as shown in Eq. (1)

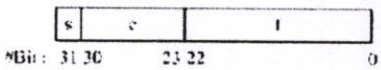


Fig. 2 32 Bit Floating Point Format.

$$V = -1^{s} 2^{(e-127)} (1.f) \quad (1)$$

2.2 Floating Addition and Subtraction

Most of addition and subtraction process of real number follows the same processing step to get the result. However, there are some different parts needed to be changed from minus to addition as shown in Fig. 4 in stage 2 above 24-bit register (R24). Fig.3 shows all fields of input data. V1 is the first input data, which can be an addends or a subtrahend. V2 is the second input data, which can be an addends or a subtrahend also. Addition and Subtraction process can be separated into 3 stages as followed:

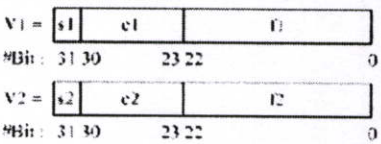


Fig. 3 Show all field of input data

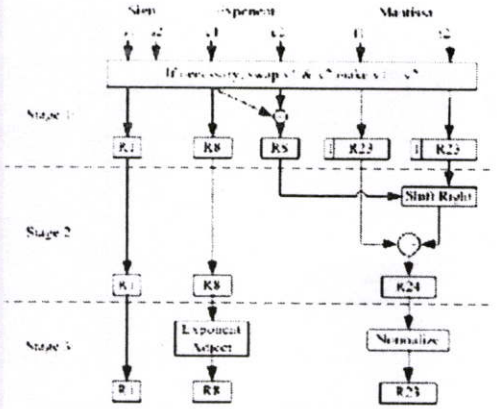


Fig. 4 Three stage 32 bit floating-point adder/subtractor.

Stage: 1

- Consider the both of input data if absolute value of V1 less than absolute value of V2, then alternate V1 and V2. The absolute value can be obtained by edit the part of exponent and mantissa term.
- Minus the exponent term of both input data for specified shifting position of f2.

Stage: 2

- Shift bit of 1.f2 to the right.
- If the value of s1 equal to s2 then combine the value of 1.f1 and 1.f2 together.
- If s1 and s2 are not equal then minus the value of 1.f1 and 1.f2.
- Keep sign bit (s1) from result of input data (V1).

Stage: 3

- Normalize value in 24-bit register(R24) by shifting bit to left side until the left most bit of this register has the value of '1'.
- Adjust exponent value by subtract exponent term with the time of shifting bit in R24.

2.3 Floating-Point Multiplication

Real number multiplication is similar to integer multiplication because real number data is collected sign and value (Sign-Magnitude). Real number multiplication is done without sign and adapted some values to get accurate result as shown in Fig. 4. We separate real number multiplication process to 3 states.

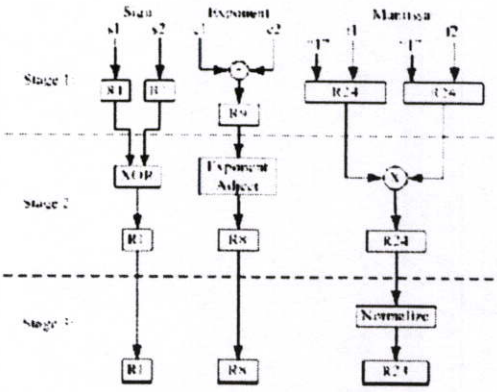


Fig. 5 Three stage 32 bit floating-point multiplier.

Stage: 1

- Add c1 and c2 then keep the result in 9-bit register (R9).
- Add left most bit of f1 and f2 then keep the result in 24-bit register (R24).
- Keep value of s1 and s2 in 1-bit register (R1).

Stage: 2

- Apply integer multiplication with input 1.f1 and 1.f2 then keep the result in 24-bit registers (R24).
- Adjust value of exponent term by subtracting with bias.
- Compare sign bit s1 and s2 if sign bits are the same so the result is positive, otherwise is negative.

Stage: 3

- Normalize value for 23-bit register (R23).
- The result consists of three parts, which are sign bit, exponent and mantissa term.

2.4 Implementing the DWT through Lifting

The calculation applied to all wavelet transforms through lifting can be implemented in two different basic methods. First method is initiated by determining the value of $s^{(1)}[n]$, then substituting this value on $d^{(1)}[n]$. The value from the equation (2) is substituted to equation (3) and so on until equation (6). The advantage of this method is that it can be programmed easily but requires more memory space to store result. As a result, this method may not be suitable for designing FPGA.

$$s^{(1)}[n] = S[2n] - \sqrt{3}S[2n+1] \quad (2)$$

$$d^{(1)}[n] = S[2n-1] - \frac{1}{4}\sqrt{3}s^{(1)}[n] - \frac{\sqrt{3}-2}{4}s^{(1)}[n-1] \quad (3)$$

$$s^{(2)} = s^{(1)}[n] - d^{(1)}[n+1] \quad (4)$$

$$s[n] = \frac{\sqrt{3}-1}{\sqrt{2}} s^{(2)}[n] \quad (5)$$

$$d[n] = \frac{\sqrt{3}-1}{\sqrt{2}} d^{(1)}[n] \quad (6)$$

In the second approach, we will deduce the Eq. (2) to Eq. (6). it becomes two equations for high pass and low pass filters in order to determine approximation and detail coefficients. The advantage of this method is that it can be adapted to fit with data flow of FPGA.

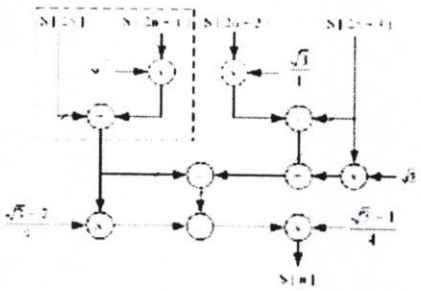
2.5 Design and Architecture

To design data flow of DWT though lifting scheme using Daubechies4 equations, we need to arrange the equations. With appropriate arrangement, the proposed system becomes less complex. Consider equation (7) and (8), which is obtained from solving Daubechies4 equations. There are only 2 equations left to find approximation and detail respectively.

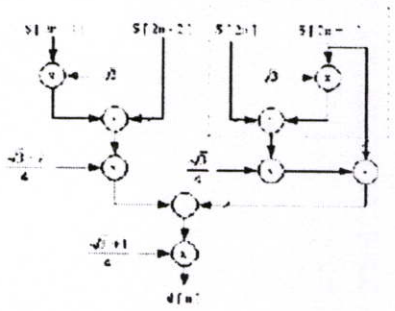
$$s[n] = \frac{\sqrt{3}-1}{\sqrt{2}}\{(S[2n] + \sqrt{3}S[2n+1]) - (S[2n-3] - \frac{1}{4}\sqrt{3}(S[2n+2] + \sqrt{3}S[2n+3]) - \frac{\sqrt{3}-2}{4}(S[2n] + \sqrt{3}S[2n-1]))\} \quad (7)$$

$$d[n] = \frac{\sqrt{3}+1}{\sqrt{2}}\{S[2n-1] - \frac{1}{4}\sqrt{3}(S[2n] - \sqrt{3}S[2n+1]) - \frac{\sqrt{3}-2}{4}(S[2n-2] + \sqrt{3}S[2n-1])\} \quad (8)$$

After that, these equations are used to make data flow as shown in Fig. 5(a) and (b). From both figures, we can notice that there are some overlapped parts. Therefore, we can combine these parts together to save environment resource as shown in Fig. 7.



(a)



(b)

Fig. 6 Data flow of low pass filter and high pass filter used to find approximation (a) and detail (b).

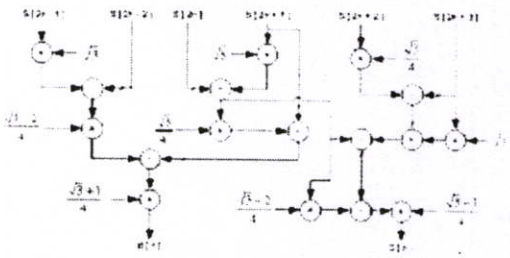


Fig. 6 Complete data flow.

3. EXPERIMENT

In our experiment, we selected images form the set of standard ISO test image with resolution of 256 x 256 pixels. Example image used to test DWT transform is show in Fig.8. We begin by loading an image to the memory. Next, image in memory is read by FPGA to process as shown in Fig. 8 (b). Then, we increase image brightness by 128 to the parts of LH, HL, and HH for clarity of details. After that we use result data to be an input and reconstruct image to check the error. The process is tested on real time system. The proposed architecture is implemented as explained in previous sections. From the experiment, we found that our system can process 30 images within an average of 1 second as shown in table 1.

Our experiment system use devices and equipments as followed:

- FPGA Board: FPGA Xilinx Virtex-II 1000 with SDRAM 64MByte.
- PC: CPU Intel Pentium III 1.13 GHz (Tulatin), SDRAM 768MByte

(a) (b) (c)

Table 1 Processing time for our system.

	1 Image	Error	30 Images	Error
PC.	0.089843 Sec.	0	2.644531 Sec.	0
FPGA	0.015236 Sec.	0	0.5470958 Sec.	0

4. CONCLUSION

In this paper, a design of parallel architecture for wavelet transformation on FPGA presented. We begin the design with floating-point computational unit, this unit is essential because it is arranged according to Daubechies4 though lifting. The objective of this paper is to apply FPGA to DWT process to reduce processing time. From the experiment, we found that our system can work on real-time system. System can process 30 images in less than 1 second. However, processing time depends on many factors such as memory speed, CPU speed on the board, because data will be sent and received from CPU all the time. The proposed architecture can be applied on the variety systems such as real time image compression, image quantization, and JPEG 2000. Furthermore, the proposed DWT can perfectly reconstruct data and can be applied in lossless applications.

REFERENCES

[1] CK Chui, "An Introduction to Wavelet," *1 ed. Wavelet analysis and its application*, Vol. 1, 1992.

[2] S Mallat, "A wavelet tour of signal processing," 2 ed. 1998.

[3] N Fliege, Multiraten-Signalverarbeitung, Informationstechnik, ed. N. Fliege, 1993.

[4] J.A. Elden and C. Robertson, "A Floating Point Format for Signal Processing," *Proceedings IEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 717-720, 1982.

[5] N. Shirazi, *Implementation of a 2-D Fast Fourier Transform on an FPGA Base Computing Platform*, VPI&SU Masters Thesis in progress, VPI&SU Masters Thesis in progress.

[6] IEEE Task P754, "A Proposed for Binary Floating-Point Arithmetic," *IEEE Computer*, Vol. 14, No. 12, pp.51-62, March 1981.

[7] I Daubechies and W. Sweldens, *Factoring wavelet into lifting step*, J. Fourier Anal. Appl., pp. 27, 1998.

[8] W Sweldens, "The lifting scheme: Anew philosophy in biorthogonal wavelet constructions," *In Wavelet Application in Signal and Image Processing III*, 1995.

[9] J.M Aronold, D.A Buell, and E.G. Davis, "Splash 2," Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 316-322, June 1992.

[10] Xilinx, Inc., *Programmable Logic Design Quick Start Handbook*, Karen Parnell and Nick Mehta, 2003.

ประวัติผู้เขียน

ชื่อ นาย ไกรฤกษ์ โง้วสุวรรณ

เกิดวันที่ 6 ตุลาคม 2521 ที่จังหวัด ขอนแก่น

ประวัติ จบการศึกษาระดับ ป.ว.ช. สาขาอิเล็กทรอนิกส์ สถาบันเทคโนโลยีราชมงคลวิทยาเขตขอนแก่น ปี พ.ศ. 2541 เกรดเฉลี่ย 3.15

จบการศึกษาระดับ ป.ว.ส. สาขาอิเล็กทรอนิกส์ สถาบันเทคโนโลยีราชมงคลวิทยาเขตขอนแก่น ปี พ.ศ. 2542 เกรดเฉลี่ย 3.43

จบการศึกษาระดับปริญญาตรี คณะวิศวกรรมศาสตร์ ภาควิชาวิศวกรรมโทรคมนาคม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปี พ.ศ. 2544 เกรดเฉลี่ย 2.79