การนำภาพ SAR มาบีบอัดข้อมูลด้วยวิธี SPIHT บนอุปกรณ์ FPGA

Implementation of SPIHT SAR Image Compression on FPGA

โดย

นางสาวธัชภร สงวนพวก

นายภาคภูมิ ศิริอภัยพันธ์

นายวุฒิพงศ์ คำภาบุตร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ผ่านการตรวจชิ้นงานแล้ว
(ลงชื่อ).....ผู้ตรวจ

ภาควิชา
วิศวกรรมโทรคมนาคม

การนำภาพ SAR มาบีบอัดข้อมูลด้วยวิธี SPIHT บนอุปกรณ์ FPGA

Implementation of SPIHT SAR Image Compression on FPGA

โดย

นางสาวธัชภร สงวนพวก 46010272

นายภาคภูมิ ศิริอภัยพันธ์ 46010568

นายวุฒิพงศ์ คำภาบุตร 46010755

อาจารย์ที่ปรึกษา

รศ.ดร.กอบชัย เดชหาญ

อาจารย์ ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การนำภาพ SAR มาบีบอัดข้อมูลด้วยวิธี SPIHT บนอุปกรณ์ FPGA**

Implementation of SPIHT SAR Image Compression on FPGA

ผู้จัดทำ

1. นางสาวธัชภร สงวนพวก	**46010272**
2. นายภาคภูมิ ศิริอภัยพันธ์	**46010568**
3. นายวุฒิพงศ์ คำภานุตร	**46010755**

..... อาจารย์ที่ปรึกษา

(รศ.ดร. กอบชัย เดชหาญ)

..... อาจารย์ที่ปรึกษา

(อาจารย์ ศรวัฒน์ ชิวปรีชา)

การนำภาพ SAR มาบีบอัดข้อมูลด้วยวิธี SPIHT บนอุปกรณ์ FPGA

Implementation of SPIHT SAR Image Compression On FPGA

โดย นางสาวธัชภร สงวนพวก 46010272

นายภาคภูมิ ศิริอภัยพันธ์ 46010568

นายวุฒิพงศ์ คำภาบุตร 46010755

อาจารย์ที่ปรึกษา รศ.ดร.กอบชัย เดชหาญ

อ.ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

โครงงานนี้นำเสนอ การจำลองภาพถ่ายที่ได้นำมาบีบอัดข้อมูล ซึ่งในปัจจุบันการบีบอัดข้อมูลภาพ SAR มีความสำคัญมาก เพื่อลดค่าใช้จ่ายในการ เก็บและส่งข้อมูล โดยโครงงานนี้เสนอการออกแบบ และการสร้างตัวประมวลผลเพื่อใช้บีบอัดข้อมูลภาพด้วยวิธีการ Set Partitioning in Hierarchical Trees (SPIHT) ซึ่งเป็นวิธีที่ได้รับความสนใจเป็นอันมาก ในฐานะเทคนิคสำหรับการเข้ารหัสภาพที่มีประสิทธิภาพ

ABSTRACT

This project proposes the duplicated SAR image compression. At present SAR image compression is very important in reducing the costs of data storage and transmission. We present an FPGA implementation of image compression use set partitioning in hierarchical trees (SPIHT) algorithm that attracted great attention as a technique for efficient image coding.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงไปได้ด้วยดี ด้วยความกรุณาอย่างดียิ่ง จาก รศ.ดร.กอบชัย เดชหาญ และ อาจารย์ ศรวัฒน์ ชิวปรีชา ที่กรุณาสนับสนุนช่วยเหลือในทุกๆด้าน ทั้งด้านความรู้ ให้คำปรึกษา แนะนำ คอยดูแลเอาใจใส่และเมตตาแก่ศิษย์ด้วยดีเสมอมา ขอขอบคุณสำหรับกำลังใจและให้คำแนะนำเกี่ยวกับปัญหาต่างๆ ระหว่างการทำงานนี้ ขอขอบคุณ ดร.มนตรี คำเงิน และ อาจารย์สิรภพ ตู้ประกาย อาจารย์ทุกท่านที่ประสิทธิ์ประสาทวิชาความรู้ต่างๆ ให้แก่ศิษย์ทั้งโดยทางตรงและทางอ้อม

สุดท้ายนี้ กราบขอบพระคุณ บิดา มารดา ผู้เป็นที่เคารพยิ่ง ที่ให้ความรักความเข้าใจ ให้กำลังใจ คอยช่วยเหลือ จนการจัดทำปริญญานิพนธ์ ครั้งนี้สำเร็จลงได้ด้วยดี

คณะผู้จัดทำ

สารบัญ

สารบัญ (ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

สา รบัญรูปภาพ (ต่อ)

หน้า

สารบัญรูปภาพ (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาของหัวข้อปริญญานิพนธ์

ในหลายๆสาขา ภาพดิจิตอลมักถูกใช้แทนที่ภาพอนาลอก เช่น ภาพถ่ายหรือ ภาพ X-rays ปริมาณข้อมูลต้องมีจำนวนมากเพื่อบรรยายลักษณะรูปภาพจึงทำให้การส่งข้อมูลช้า ดังนั้นข้อมูลจะต้องถูกบีบอัดอย่างมีประสิทธิภาพ จุดประสงค์หลักของการบีบอัดข้อมูลคือเพื่อที่จะลดบิตเรท (Bit Rate) โดยยังคงคุณภาพของข้อมูลภาพไว้ การบีบอัดข้อมูลภาพสามารถ Transform ข้อมูลและโปรเจคชัน ในรูปของ Basis Function แล้วจึง นำมาเข้ารหัส (Encode) ต่อไป เพราะธรรมชาติของสัญญาณภาพ และการมองเห็นของมนุษย์นั้น การ transform จะต้องถูกวางในตำแหน่ง ทั้ง Space และ Frequency Domain โดยในปริญญานิพนธ์ฉบับนี้ใช้เทคนิค SPIHT (Set Partitioning in Hierarchical Trees) โดยการใช้ Set Partitioning และการทดสอบบิตที่สำคัญ (Significance) บนโครงสร้าง Hierarchical โดย Said and Pearlman เป็นผู้คิดค้น ซึ่งปรับปรุงมาจาก Shapiro's EZW (Embedded Zerotree Wavelet) Algorithm

ในปริญญานิพนธ์ฉบับนี้ได้นำภาพSAR (Synthetic Aperture Radar) มาทำการบีบอัด ซึ่งภาพ SARข้อมูลภาพของ SAR คือ ข้อมูลภาพ 2 มิติ ที่บันทึกโดยจานเรดาร์ยาว 10 เมตรบันทึกข้อมูลเป็นแนวกว้าง (Swath width) 100 กิโลเมตรที่มุมตกกระทบ (Incident angle) 23 องศา มีความละเอียดเชิงพื้นที่เท่ากับ 23 x 30 เมตร และเป็นสัญญาณสะท้อนกลับจากวัตถุ ซึ่งก็คือค่าการกระจัดกระจายกลับ (Backscatter) ของวัตถุบนพื้นโลกนั้นเองโดยจะขึ้นอยู่กับคุณสมบัติความเรียบและความขรุขระของผิวหน้าวัตถุเป็นสำคัญ ยิ่งเรียบจะให้ค่าการกระจัดกระจายกลับต่ำ ขณะที่ความขรุขระให้การกระจัดกระจายกลับที่สูงขึ้น แต่ทั้งนี้ต้องพิจารณามุมตกกระทบ (Incident Angle) ขณะที่บันทึกข้อมูลด้วยและข้อมูลหลักนี้จะนำไปใช้ประโยชน์ในการสำรวจทรัพยากรและสิ่งแวดล้อมทั่วๆไปขนาดของภาพมาตรฐานคือ 100x100 ตารางกิโลเมตรข้อมูลชนิดนี้ไม่มีการบันทึกทางดาวเทียม เนื่องจากปริมาณข้อมูลสูงมาก โดยจะส่งสัญญาณกลับมาบันทึกข้อมูลที่สถานีรับสัญญาณภาคพื้นดินเท่านั้น

1.2 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์นี้เป็นการนำ เทคนิค SPIHT มาบีบอัดข้อมูลภาพ ทั้งแบบ มี list และ without list และได้ทำการจำลองการทำงานผ่านโปรแกรม Xilinx-Project Navigator ซึ่งกระบวนการทำงานทั้งหมดจะถูกบรรยายพฤติกรรมการทำงานโดยใช้การเขียนด้วยภาษา VHDL ในการออกแบบและทำการจำลองผลการทดลองออกมา และจะนำไปเปรียบเทียบกับผลที่ได้จากโปรแกรมMATLAB

1.3 เนื้อหาของปริญญานิพนธ์

ในบทที่ 2 เป็นส่วนของทฤษฎีพื้นฐานการแปลงเวฟเลต, ทฤษฎีพื้นฐานของวงจรกรองความถี่และความสัมพันธ์ของทฤษฎีทั้งสอง, ทฤษฎีของการบีบอัดข้อมูลภาพ SPIHT ,ทฤษฎีของภาษา VHDL

ในบทที่ 3 เป็นส่วนของการคำนวณและการสร้าง โดยจะแบ่งเป็นส่วนของการ Simulate โดยใช้โปรแกรมแมทแล็ปและ โปรแกรม Xilinx-Project Navigator โดยใช้ภาษา VHDL ในการออกแบบ และส่วนของการ Implement ลงบนอุปกรณ์ FPGA

ในบทที่ 4 เป็นส่วนของผลการทดลอง ซึ่งประกอบด้วยส่วนของการ Simulate โดยใช้โปรแกรม MATLAB, ส่วนของการ Simulate ด้วยโปรแกรม Xilinx-Project Navigator โดยใช้ภาษา VHDL, ส่วนของการ Implement ลงบนอุปกรณ์ FPGA และส่วนของการแสดงผลผ่านจอคอมพิวเตอร์

ในบทที่ 5 เป็นของ บทวิจารณ์และบทสรุป

หนังสืออ้างอิง

บทที่ 2
ทฤษฎีและหลักการ

2.1 ความรู้เบื้องต้นของเวฟเลต

เมื่อกล่าวถึงคลื่น (wave) จะมีลักษณะเป็นสัญญาณที่มีการเปลี่ยนแปลงอยู่ตลอดเวลาหรือที่เรียกว่าเกิดการออสซิเลต (oscillate) เช่น สัญญาณรูปไซน์ (sinusoid) ซึ่งจะใช้การวิเคราะห์โดยการแปลงฟูเรียร์(Fourier Transform) คลื่นโดยทั่วไปจะมีการกระจายพลังงานอย่างไม่มีที่สิ้นสุดเนื่องจากสัญญาณเปลี่ยนแปลงอยู่ตลอดเวลา ถ้าเทียบกับเวฟเลต (wavelet) เปรียบเสมือนคลื่นเล็กๆ ที่เกิดขึ้นชั่วขณะหนึ่ง มีการเปลี่ยนแปลงแบบไม่คงที่ และมีการแปรผันตามเวลา และมีค่าพลังงานรวมกันอยู่ในช่วงใดช่วงหนึ่ง และสามารถวิเคราะห์ได้ทั้งในเทอมของเวลาและความถี่ในเวลาเดียวกัน โดยที่การกระจายเวฟเลต (wavelet expansion) สามารถเขียนให้อยู่ในรูปสมการที่ (2.1) ดังนี้

$$f(t)=\sum_{k}\sum_{j}a_{j,k}\psi_{j,k}(t) \tag{2.1}$$

โดยที่ $a_{j,k}$ จะแทนสัมประสิทธิ์การกระจาย และ $\psi_{j,k}(t)$ จะแทนฟังก์ชันการกระจายของเวฟเลต โดยที่เราจะเรียกเซตของสัมประสิทธิ์การกระจายนี้ว่า การแปลงเวฟเลตแบบไม่ต่อเนื่อง (Discrete Wavelet Transform) และสามารถแปลงกลับได้ตามสมการข้างต้น

2.1.1 คุณสมบัติพิเศษของเวฟเลต (Specific Characteristics of Wavelet Systems)

1. เวฟเลตทุกประเภทสามารถสร้างได้จากสเกลลิ่งฟังก์ชันหรือเวฟเลตเพียงตัวเดียว โดยการแบ่งระดับ (scaling) และการเลื่อนตำแหน่ง (translation) โดยที่เราจะเรียกเวฟเลตต้นกำเนิดนี้ว่า "เวฟเลตแม่" (Mother wavelet) หรือ $\psi(t)$ โดยที่ j จะแทนระดับต่างของสัญญาณ และ k จะแทนการเลื่อนตำแหน่ง ดังสมการที่ (2.2)

$$\psi_{j,k}(t)=2^{j/2}\psi(2^{j}t-k) \tag{2.2}$$

2. เวฟเลตเกือบทุกแบบสามารถใช้เงื่อนไขมัลติรีโซลูชัน (multiresolution) หมายความว่าถ้าสัญญาณใดสัญญาณหนึ่งสามารถแสดงให้อยู่ในเทอมของฟังก์ชันสเกลลิ่ง (scaling function) $\varphi(t-k)$ ได้ สัญญาณนี้ก็สามารถแสดงให้อยู่ในรูป $\varphi(2t-k)$ ได้เช่นกัน หรืออีกนัยหนึ่งคือการกระจายของเวฟเลตจะแบ่งสัญญาณเดิมออกเป็นส่วนย่อยๆได้
3. สัมประสิทธิ์การกระจายในส่วนย่อยๆสามารถคำนวณได้จากสัมประสิทธิ์ของส่วนที่ใหญ่กว่าตามวิธีการแตกกิ่งก้านสาขา หรือ ฟิลเตอร์แบงค์ (filter bank) ทำให้การคำนวณสัมประสิทธิ์การกระจายมีประสิทธิภาพมากยิ่งขึ้น

การวิเคราะห์เวฟเลตจะเหมาะสมกับสัญญาณที่เกิดขึ้นชั่วขณะหนึ่ง ซึ่งจากลักษณะจำเพาะอยู่เฉพาะส่วนของเวฟเลต ทำให้สัมประสิทธิ์การกระจายมีค่าน้อย ซึ่งจะเป็นประโยชน์กับการนำไปประยุกต์ใช้งาน

2.2 การแปลงเวฟเลตแบบไม่ต่อเนื่อง (The Discrete Wavelet Transform)

จุดประสงค์ของการแปลงเวฟเลต คือ การกระจายสัญญาณออกเป็นเซตของฟังก์ชันๆหนึ่ง โดยที่สามารถกระจายให้อยู่ในเทอมของเวลาและความถี่ ดังสมการที่ (2.3)

$$f(t)=\sum_{j,k} a_{j,k}\, 2^{j/2}\psi\left(2^{j}t-k\right) \tag{2.3}$$

โดยที่ $a_{j,k}$ เป็นเซตของสัมประสิทธิ์การกระจายที่มีสองมิติ เรียกว่า การแปลงเวฟเลตแบบไม่ต่อเนื่อง หรือ Discrete wavelet transforms (DWT) จะเห็นได้ว่าการกระจายแบบนี้มีตัวระบุตำแหน่งอยู่สองตัวคือ j และ k โดยที่ตัวที่ใช้ในการเลื่อนตำแหน่งหรือ translation คือ k และตัวที่ใช้ในการแบ่งระดับหรือ scaling คือ j

การพิสูจน์ Mother Wavelet ดังสมการที่ (2.2) ได้จากทฤษฎีการวิเคราะห์ Multiresolution เมื่อเราพูดถึง wavelet จะหมายถึงของ scaling function, φ ซึ่งสามารถเขียนให้อยู่ในรูปสมการที่ (2.4)

$$\varphi(t)=\sqrt{2}\sum_{k\in z} h_k\varphi(2t-k) \qquad t, h_k \epsilon R \tag{2.4}$$

สมการที่ (2.4) นี้จะแสดงให้เห็นดังรูปที่ 2.1 อาจกล่าวได้ว่า φ คือ eigenfunction โดยมีค่า eigenvalue เป็น 1 โดยค่า eigenfunction นี้จะมีค่าเดียวเมื่อมี amplitude ดังกล่าว ดังนั้น scaling function จะถูก normalized ด้วยค่า $\sqrt{2}$ เสมอ ดังสมการที่ (2.5)

$$\sum_{k\in z}\varphi(k)=\sqrt{2} \tag{2.5}$$

สำหรับ Wavelet function จะถูกสร้างจาก scaling function ในรูปของ ฟิลเตอร์ g ดังรูปที่ 2.1

$$\psi(t)=\sqrt{2}\sum_{k\in z} g_k\varphi(2t-k) \tag{2.6}$$

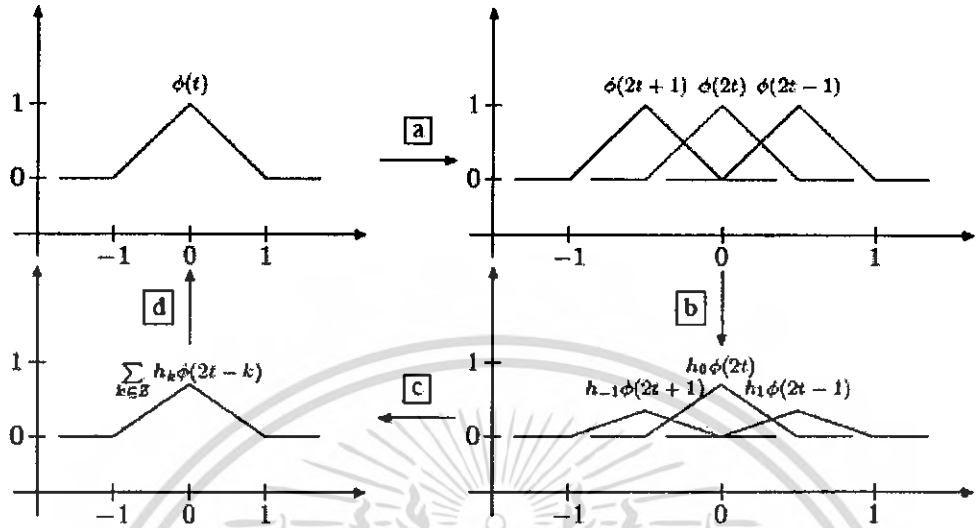
จากสมการที่ (2.4) และ (2.6) อาจแปลงมาเป็น สมการที่ (2.7) และ (2.8) โดยถูกขยายด้วย $\sqrt{2}$

$$\psi_{j,l}(t)=2^{j/2}\psi(2^{j}t-l) \tag{2.7}$$

$$\varphi_{j,l}(t)=2^{j/2}\varphi\left(2^{j}t-l\right) \tag{2.8}$$

j แทน การสเกล โดยยิ่ง j มีค่ามาก ความถี่ยิ่งสูง ยอดของ Wavelet ยิ่งบางลง

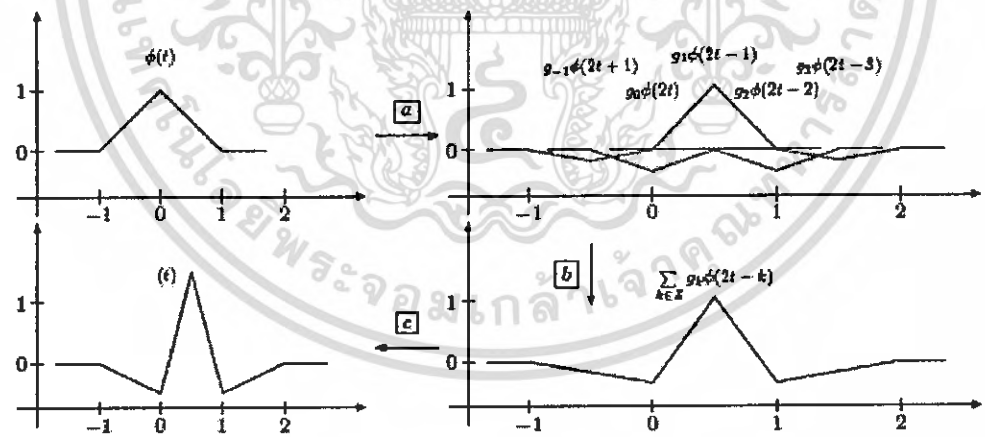
l แทนการเลื่อนตำแหน่ง โดยยิ่ง l มีค่ามากขึ้น จะยิ่งถูกเลื่อนไปทางขวามือ



รูปที่ 2.1 แสดง scaling function ดังใน step a ถึง d โดยในที่นี้ใช้ ค่าสัมประสิทธิ์

$$h_{-1} = \frac{1}{\sqrt{2}} \times \frac{1}{2}, h_0 = \frac{1}{\sqrt{2}} \times 1$$

$h_1 = \frac{1}{\sqrt{2}} \times \frac{1}{2}$ อาจเรียกว่า CDF 5-3 สำหรับการแปลง lossless compression



รูปที่ 2.2 การสร้าง wavelet function จาก scaling functions ดังใน step a ถึง c โดย ฟิลเตอร์ g นำมาจาก CDF 5-3 โดย

$$g_{-1} = \sqrt{2} \times \frac{1}{8}, g_0 = \sqrt{2} \times \left(\frac{-1}{4}\right), g_1 = \sqrt{2} \times \frac{3}{4}, g_2 = \sqrt{2} \times \left(\frac{-1}{4}\right), g_3 = \sqrt{2} \times \left(\frac{-1}{8}\right)$$

2.2.1 สเกลลิ่งฟังก์ชัน (The Scaling Function)

เป็นวิธีการหนึ่งในวิธีมัลติรีโซลูชัน โดยการแบ่งสัญญาณออกเป็นส่วนย่อย หรือเป็นระดับแล้วจึงอธิบายเวฟเลตในเทอมของสเกลลิ่งฟังก์ชัน ดังสมการที่ (2.9)

$$f(t)=\sum_{k} a_k \varphi_k(t) \qquad \text{โดยที่} \quad f(t)\in V_0 \tag{2.9}$$

เราสามารถเพิ่มขนาดของส่วนย่อยนี้โดยการเปลี่ยนค่าระดับในสเกลลิ่งฟังก์ชัน โดยสามารถแสดงให้อยู่ในสองมิติ คือ การสเกลลิ่งและการทรานสเลชัน ได้ดังสมการที่ (2.10)

$$\varphi_{j,k}(t)=2^{j/2}\varphi\left(2^{j}t-k\right) \tag{2.10}$$

โดยที่ ถ้า $j\rangle 0$ จะทำให้มีการแบ่งมากขึ้น ทำให้ $\varphi_{j,k}(t)$ มีขนาดแคบหรือเล็กลง และมีการเลื่อนตำแหน่งที่น้อยลง ดังนั้นจะได้รายละเอียดดีขึ้น ในทางตรงกันข้ามถ้า $j\langle 0$ ทำให้ $\varphi_{j,k}(t)$ จะขนาดกว้างขึ้นและเลื่อนตำแหน่งมากขึ้น สรุปได้ว่าในส่วนสเกลลิ่งฟังก์ชันจะทำให้ได้ข้อมูลแบบหยาบๆ

2.2.2 เวฟเลตฟังก์ชัน (The Wavelet Function)

เป็นอีกขั้นตอนหนึ่งในวิธีมัลติรีโซลูชัน โดยการกำหนดค่าความแตกต่างระหว่างสเกลลิ่งฟังก์ชันในแต่ละระดับ ดังนั้นจึงจะอธิบายให้อยู่ในเทอมของสเกลลิ่งฟังก์ชัน ข้อได้เปรียบของการทำสเกลลิ่งฟังก์ชันและเวฟเลตคือ การเป็นฟังก์ชันเชิงตั้งฉากซึ่งกันและกัน ทำให้ง่ายต่อการหาสัมประสิทธิ์ของแต่ละชุด โดยจะแทนเวฟเลตนี้ด้วยสัญญลักษณ์ W_j สามารถอธิบายได้ดังนี้

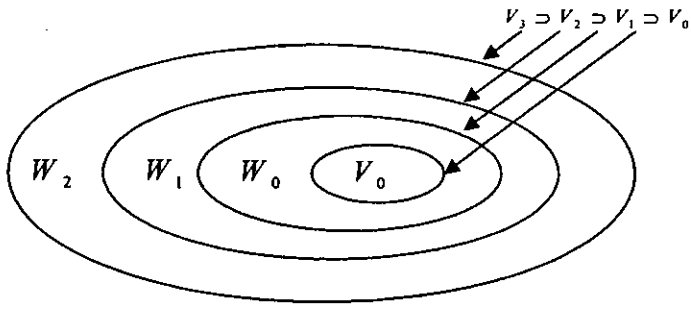
$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset L^2 \tag{2.11}$$

ดังนั้น
$$V_1 = V_0 \oplus W_0 \tag{2.12}$$

ส่วนต่อไปคือ
$$V_2 = V_0 \oplus W_0 \oplus W_1 \tag{2.13}$$

สามารถอธิบายให้อยู่ในรูปทั่วไปได้ดังนี้

$$L^2 = V_0 \oplus W_0 \oplus W_1 \oplus W_2 \oplus \cdots \tag{2.14}$$



รูปที่ 2.3 สเกลลิ่งฟังก์ชันและเวฟเลต

จากที่อธิบายมาข้างต้น สามารถหาค่า เวฟเลตฟังก์ชันได้ดังนี้

$$\psi(t) = \sum_n h_1(n)\sqrt{2}\varphi(2t-n) \tag{2.15}$$

โดยที่ $h_1(n)$ จะหมายถึงสัมประสิทธิ์ของฟังก์ชันเวฟเลต

2.3 หลักการวิเคราะห์ (Analysis-From Fine Scale to Coarse Scale)

ในการที่หาสัมประสิทธิ์ของการแปลงเวฟเลต จะหาได้จากความสัมพันธ์ของสัมประสิทธิ์การกระจายระหว่างระดับต่ำและระดับสูง หรืออาจกล่าวได้ว่าจากระดับที่มีความละเอียดไปสู่ระดับหยาบ โดยสามารถคำนวณได้จากสมการดังต่อไปนี้

$$\varphi(t) = \sum_n h(n)\sqrt{2}\varphi(2t-n) \tag{2.16}$$

ทำการเปลี่ยนระดับและการเลื่อนตำแหน่ง โดยการแทนค่า $t = 2^j t - k$

$$\varphi(2^j t-k) = \sum_n h(n)\sqrt{2}\varphi(2(2^j t-k)-n) = \sum_n h(n)\sqrt{2}\varphi(2^{j+1}t-2k-n) \tag{2.17}$$

แทนค่า $m = 2k+n$

$$\varphi(2^j t-k) = \sum_m h(m-2k)\sqrt{2}\varphi(2^{j+1}t-m) \tag{2.18}$$

ถ้าเราแทนพจน์ที่เกิดจากการกระจายด้วย V_j

$$V_j = Span\left\{2^{j/2}\varphi(2^j t-k)\right\} \tag{2.19}$$

ดังนั้น จะได้ว่า

$$f(t) \in V_{j+1} \Rightarrow f(t) = \sum_k c_{j+1}(k) 2^{(j+1)/2} \varphi(2^{j+1}t - k) \tag{2.20}$$

จาก $V_{j+1} = V_j \oplus W_j$

$$f(t) = \sum_k c_j(k) 2^{j/2} \varphi(2^j t - k) + \sum_k d_j(k) 2^{j/2} \psi(2^j t - k) \tag{2.21}$$

สามารถหาค่าสัมประสิทธิ์ของสเกลลิ่งฟังก์ชันได้ดังนี้

$$c_j(k) = \sum_m h(m - 2k) c_{j+1}(m) \tag{2.22}$$

และจากความสัมพันธ์สามารถหาค่าสัมประสิทธิ์ของเวฟเลตได้ดังนี้

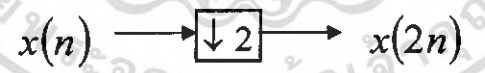
$$d_j(k) = \sum_m h_1(m - 2k) c_{j+1}(m) \tag{2.23}$$

2.4 การกรองสัญญาณและการสุ่มค่าตัวอย่าง (Filtering and Down-Sampling or Decimating)

จากหลักการของการประมวลผลสัญญาณดิจิตอล การกรองสัญญาณสามารถทำได้โดยการทำคอนโวลูชั่นระหว่างสัญญาณอินพุตกับสัมประสิทธิ์ของวงจรกรองความถี่ (Filter coefficients or impulse response) โดยถ้าสมมติให้อินพุต คือ $x(n)$ และสัมประสิทธิ์ของวงจรกรองความถี่คือ $h(n)$ จะได้เอาต์พุตคือ $y(n)$ ดังสมการ

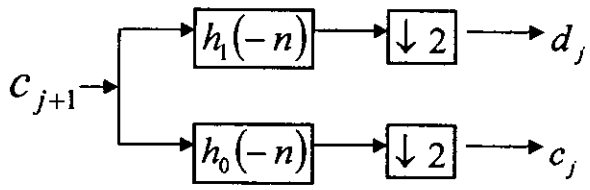
$$y(n) = \sum_{k=0}^{N-1} h(k) x(n - k) \tag{2.24}$$

การสุ่มค่าตัวอย่างจะเป็นการนำอินพุต $x(n)$ สร้างออกมาเป็นเอาต์พุต $y(n) = x(2n)$ ดังรูป



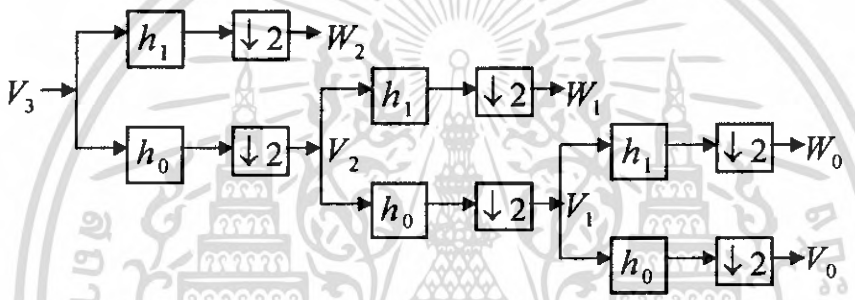
รูปที่ 2.4 ตัวสุ่มค่าตัวอย่าง (Down Sampler or Decimator)

จะเห็นได้ว่าสัมประสิทธิ์ของสเกลลิ่งและเวฟเลตในระดับที่ต่างกันเกิดจากการทำคอนโวลูชั่นระหว่างสัมประสิทธิ์การกระจายในระดับ j กับสัมประสิทธิ์ของวงจรกรองความถี่ $h_0(-n)$ และ $h_1(-n)$ จากนั้นจึงนำมาสุ่มค่าตัวอย่าง จึงจะได้สัมประสิทธิ์การกระจายในลำดับถัดไป คือ $j-1$ หรืออาจกล่าวได้ว่า สัมประสิทธิ์ในระดับ j ถูกกรองโดยวงจรกรองความถี่สองตัวซึ่งมีสัมประสิทธิ์คือ $h_0(-n)$ และ $h_1(-n)$ และหลังจากผ่านการสุ่มค่าตัวอย่างก็จะได้สัมประสิทธิ์สเกลลิ่งและเวฟเลตในระดับที่ละเอียดน้อยกว่า ดังรูป



รูปที่ 2.5 แสดงการแตกกิ่งก้านสาขาแบบสองแถบของภาคส่ง

การแตกออก การกรองสัญญาณและการสุ่มค่าตัวอย่างจะทำซ้ำในส่วนของสัมประสิทธิ์สเกลลิ่ง เพื่อให้ได้โครงสร้างตามรูปที่ 2.6 ซึ่งเราเรียกกว่า iterating the filter bank ในขั้นแรกจะแบ่งส่วนของสเกลลิ่งฟังก์ชันออกเป็นส่วนของวงจรกรองความถี่ต่ำ (Lowpass Band) และส่วนของวงจรกรองความถี่สูง (Highpass Band) ซึ่งจะทำให้ได้สัมประสิทธิ์ของสเกลลิ่งและสัมประสิทธิ์ของเวฟเลตในระดับต่ำกว่า



รูปที่ 2.6 แสดงการแตกกิ่งก้านสาขาแบบสองแถบสามขั้นของภาคส่ง

2.4.1 หลักการสังเคราะห์ (Synthesis-From Coarse Scale to Fine Scale)

ในการที่จะนำสัญญาณเดิมกลับคืนมาสามารถทำได้โดยการรวมกันของสัมประสิทธิ์สเกลลิ่งฟังก์ชันและเวฟเลตในระดับที่มีความละเอียดน้อยกว่า โดยสามารถเขียนเป็นสมการที่อยู่ในรูปของสเกลลิ่งฟังก์ชันในระดับ $j+1$ ได้ดังนี้

$$f(t)=\sum_k c_{j+1}(k)2^{(j+1)/2}\varphi\left(2^{j+1}t-k\right) \tag{2.25}$$

หรืออาจจะเขียนในรูปของระดับถัดไป ซึ่งจะมีส่วนของเวฟเลตด้วยดังนี้

$$f(t)=\sum_k c_j(k)2^{j/2}\varphi\left(2^j t-k\right)+\sum_k d_j(k)2^{j/2}\psi\left(2^j t-k\right) \tag{2.26}$$

เมื่อแทนค่าสมการที่ (2.16) และ (2.17) ลงในสมการที่ (2.26) จะได้

$$f(t)=\sum_k c_j(k)\sum_n h(n)2^{(j+1)/2}\varphi\left(2^{j+1}t-2k-n\right)+\sum_k d_j(k)\sum_n h_1(n)2^{(j+1)/2}\varphi\left(2^{j+1}t-2k-n\right) \tag{2.27}$$

เนื่องจากฟังก์ชันทั้งหมดนี้เป็นฟังก์ชันเชิงตั้งฉาก จึงทำการคูณสมการที่ (2.18) และ (2.19) ด้วย $\varphi\left(2^{j+1}t-k\right)$ และทำการอินทิเกรตหาค่าสัมประสิทธิ์ได้ดังนี้

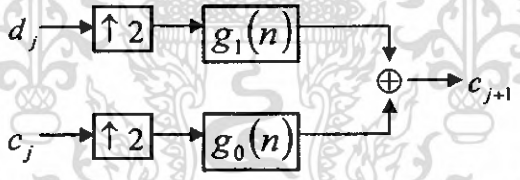
$$c_{j+1}(k)=\sum_m c_j(m)h(k-2m)+\sum_m d_j(m)h_1(k-2m) \tag{2.28}$$

2.4.2.1การกรองสัญญาณและการเพิ่มค่าตัวอย่าง (Filtering and Up-Sampling or Stretching)

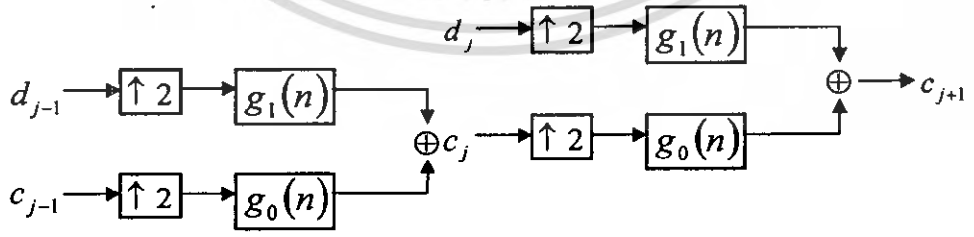
ในส่วนของฟิลเตอร์แบงค์ด้านสังเคราะห์จะประกอบด้วยการเพิ่มค่าตัวอย่างและการกรองสัญญาณ ซึ่งหมายความว่าอินพุตของฟิลเตอร์จะมีศูนย์แทรกอยู่ระหว่างสัญญาณเดิม ดังสมการ

$$y(2n)=x(n) \qquad y(2n+1)=0 \tag{2.29}$$

ทำให้สัญญาณอินพุตถูกขยายออกเป็นสองเท่าของสัญญาณเดิม และค่าศูนย์ก็ถูกแทรกเข้ามา และจากสมการที่ (2.28) ซึ่งเกิดจากการเพิ่มค่าตัวอย่างในระดับ j จากนั้นจึงทำการคอนโวลูชั่นกับสัมประสิทธิ์ของวงจรกรองความถี่ $g(n)$ ทั้งในส่วนของสเกลลิ่งฟังก์ชันและเวฟเลต แล้วนำมาบวกกันได้เป็นสัมประสิทธิ์ในระดับ $j+1$ ดังรูป



รูปที่ 2.7 แสดงการแตกกิ่งก้านสาขาแบบสองแถบของภาครับ



รูปที่ 2.8 แสดงการแตกกิ่งก้านสาขาแบบสองแถบสองขั้นของภาครับ

2.5 การแปลง wavelet transform ประยุกต์โดยใช้กับภาพ (2D-DWT)

พิจารณาขนาด $N \times N$ โดยหลังจากที่แปลง ระดับ 1 จะได้จำนวนสัมประสิทธิ์ $\frac{N}{2}$ โดย $0 \le l,k \le \frac{N}{2}$

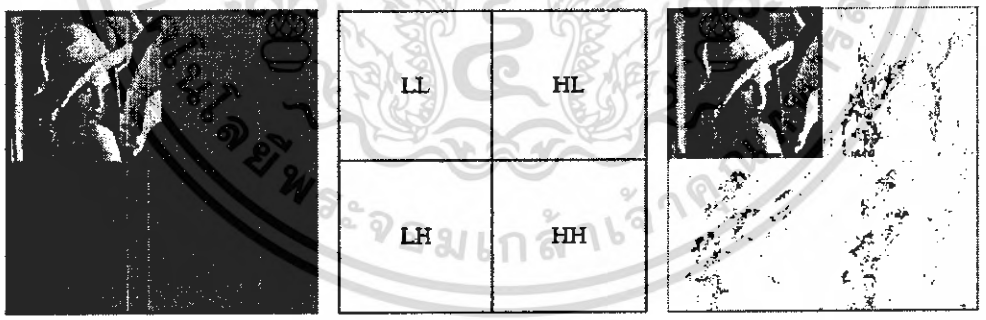
$$(c_{0,0}, d_{0,0}, c_{0,1}, d_{0,1}, ..., c_{0,N-1}, d_{0,N-1}) \tag{2.30}$$

$$r^{(0)} = (c_{0,0}, c_{0,1}, ..., c_{0,N-1}, d_{0,0}, d_{0,1} ....., d_{0,N-1}) \tag{2.31}$$

ดังนั้นจะได้ Array ขนาดใหม่โดยมีโครงสร้างดังรูปที่ 2.9

(a) Lena 512, depth 8 (b) CDF 5-3 apply กับ row a (c) low (L,$c_{a,k}$)และ high(H,$d_{a,k}$)

รูปที่ 2.9 การแปลง 1 มิติ (one dimensional) CDF (5-3) ของ wavelet transform ถูกนำประยุกต์ตามแถวและหลัก ของรูป Lena



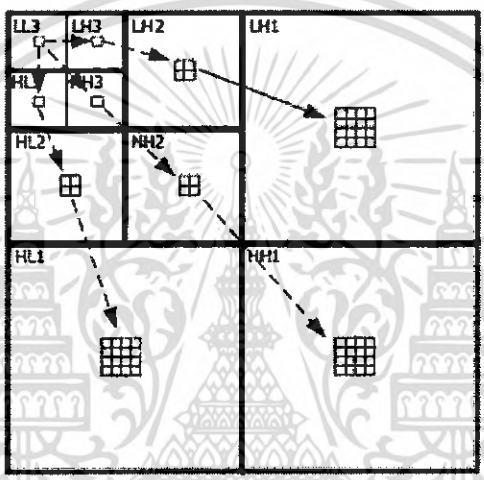
(a) CDF5-3 ทั้งแถวและหลัก (b) Block ความถี่ที่ต่างกัน (c) set coefficient υ ในLH,HL,HH โดย $-20 \le \upsilon \le 20$ เพื่อเป็นสีขาว

รูปที่ 2.10 การแปลง 1 มิติ (one dimensional) CDF (5-3) wavelet transform ถูกนำประยุกต์ตามแถวของรูป Lena และส่วนสะท้อนตามบริเวณใกล้เคียง

2.6 การบีบอัดภาพด้วยขั้นตอนวิธีSPIHT (Set Partitioning In Hierarchical Tree)

นิยามคำจำกัดความที่สำคัญๆของเทคนิคการบีบอัดภาพวิธีSPIHT มีดังต่อไปนี้

1.พ่อแม่ (Parents) จะเป็นสัมประสิทธิ์จุดภาพที่อยู่ในแถบภาพที่อยู่ในแถบความถี่หยาบ (Coarse Scale Subband) หรือแถบความถี่ใดๆยกเว้นแถบต่ำสุด ยกตัวอย่างเช่น จุดภาพที่อยู่แถบความถี่LL3เป็นพ่อแม่ของจุดภาพที่อยู่ปลายหัวลูกศรในแถบความถี่LH3, HH3, HL3 ดังแสดงในรูปที่1 และจุดภาพที่อยู่ใน LH3 เป็นพ่อแม่ของ 4 จุดภาพใน LH2



รูปที่2.11 โครงสร้างต้นไม้ที่ใช้ในขั้นตอนวิธีSPIHT ซึ่งออกแบบมาจากความสัมพันธ์ระหว่างจุดภาพพ่อแม่ไปยังจุดภาพลูกหลาน

2.ลูก (Children) จะเป็นสัมประสิทธิ์จุดภาพที่อยู่ในแถบความถี่ที่ละเอียดถัดมา1ระดับขั้นจากแถบความถี่พ่อแม่อยู่ โดยที่พ่อแม่1จุดภาพที่ตำแหน่ง (x, y) จะมีลูกได้4จุดภาพที่ตำแหน่ง $(2x, 2y)$, $(2x+1, 2y)$, $(2x, 2y+1)$ และ $(2x+1, 2y+1)$

3.หลาน (Descendent) จะหมายถึงจุดภาพทั้งหมดที่อยู่ภายใต้พ่อแม่เดียวกัน นั่นคือลูกจะเป็นพ่อแม่ของหลานและต่อกันไปลักษณะความสัมพันธ์ของโครงสร้างต้นไม้ศูนย์จากพ่อแม่ถึงลูกหลาน (Parent-Children-Descendents) ซึ่งได้แสดงให้เห็นในรูปที่ 2.11 กลุ่มที่ลูกศรชี้ไปทั้งหมดจะเป็นลูกหลานของพ่อแม่ที่ต้นลูกศร

4.จุดภาพสำคัญ (Significant Pixel) จุดภาพที่จะเรียกว่าเป็นจุดภาพที่มีความสำคัญได้ก็ต่อเมื่อค่าสัมประสิทธิ์ของจุดภาพนั้นๆที่สูงกว่าระดับอ้างอิง $|C| \geq T_i$ โดยค่าระดับอ้างอิงสามารถคำนวณได้จากสมการที่ (2.32)

$$T_{i+1} = T_i / 2 \ , \ i = 0,1,2,... \tag{2.32}$$

และค่าระดับอ้างอิงระดับแรกสามารถคำนวณได้จากสมการที่(2.33)

$$T_0 = 2^{[\log_2(MaxCoeff)]} \tag{2.33}$$

เมื่อ $MaxCoef$ คือค่าสัมประสิทธิ์ที่มีค่าสูงที่สุดของทั้งภาพ

5.จุดภาพที่ไม่สำคัญ (Insignificant Pixel) หมายถึงจุดภาพที่มีค่าสัมประสิทธิ์ไม่ถึงค่าระดับอ้างอิง $|C| \geq T_i$

สำหรับตัวอย่างของการเข้ารหัสSPIHTนี้มีเพื่อแสดงการเข้ารหัส zerotree โดยจะเริ่มจากwavelet 3 ระดับ โดยเป็นภาพ8×8 ดังรูปที่2.12 สัมประสิทธิ์ที่มีค่ามากที่สุดคือ 63 เราจึงเลือกค่าระดับอ้างอิงในการผ่านครั้งแรกให้อยู่ระหว่าง 31.5-63 โดยให้ $T_1 = 32$ ตารางที่2.1 แสดงการผ่านครั้งแรกของกระบวนการเข้ารหัสแบบSPIHT โดยมีคำอธิบายดังนี้

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

รูปที่2.12 ตัวอย่างของการแสดง wavelet 3ระดับ ของภาพ8×8

(1) ค่าสัมประสิทธิ์ 63 มีค่ามากกว่าระดับอ้างอิง 32 และมีค่าบวก ดังนั้นจะทำการใส่บิทสำคัญเป็น 1 จากนั้นจะตามด้วยบิทเครื่องหมายเป็น 0 ภายหลังจากการถอดรหัสสัญลักษณ์เหล่านี้ ผู้ที่ทำการถอดรหัสจะทราบว่าสัมประสิทธิ์จะมีค่าระหว่าง 32 และ 64 และใช้ค่ากลาง 48 ในการประมาณค่า

(2) กลุ่มหลานทางด้าน -34 นั้นมีความสำคัญ จึงทำการใส่บิทสำคัญเป็น 1 และตามด้วยการทดสอบความสำคัญของลูกๆทั้ง4 ได้แก่ {49, 10, 14, -13}

(3) กลุ่มหลานทางด้าน -31 นั้นมีความสำคัญ จึงทำการใส่บิทสำคัญเป็น 1 และตามด้วยการทดสอบความสำคัญของลูกๆทั้ง 4 ได้แก่ {15, 14, -9, -7}

(4) กลุ่มหลานทางด้าน 23 นั้นไม่มีความสำคัญ จึงทำการใส่บิทสำคัญเป็น 0

(5) กลุ่มหลานปู่ทางด้าน -34 นั้นไม่มีความสำคัญ จึงทำการใส่บิทสำคัญเป็น 0

(6) กลุ่มหลานปู่ทางด้าน -31 นั้นมีความสำคัญ จึงทำการใส่บิทสำคัญเป็น 0

(7) กลุ่มหลานของสัมประสิทธิ์ 15 นั้นไม่มีความสำคัญ จึงทำการใส่บิทสำคัญเป็น 0 โดยจะเป็นการใส่บิท 0 แค่ตัวเดียวสำหรับการผ่านหลานของสัมประสิทธิ์15

(8) กลุ่มหลานของสัมประสิทธิ์ 14 นั้นมีความสำคัญ จึงทำการใส่บิทสำคัญเป็น 0 แล้วตามด้วยการทดสอบลูกๆของสัมประสิทธิ์ 14 ได้แก่ {-1, 47, -3, 2}

(9) สัมประสิทธิ์ -31 มีลูกอยู่ 4 ได้แก่ {15, 14 , -9, 7} กลุ่มหลานของ 15 และ 14 ได้ถูกทดสอบแล้ว ต่อไปก็จะเป็นการทดสอบกลุ่มหลานที่ยังเหลืออยู่ คือ -9 และ -7

ตารางที่ 2.1 กระบวนการผ่านครั้งแรกของการเข้ารหัส SPIHT ที่มีระดับอ้างอิง $T_1 = 32$

คู่อันดับ	ค่าสัมประสิทธิ์	Binary Bit	ค่าการแปลงกลับ	คำอธิบาย
(0,0)	63	1		(1)
		0	48	
(1,0)	-34	1		
		1	-48	
(0,1)	-31	0	0	
(1,1)	23	0	0	
(1,0)	-34	1		(2)
(2,0)	49	1		
		0	48	
(3,0)	10	0	0	
(2,1)	14	0	0	
(3,1)	-13	0	0	
(0,1)	-31	1		(3)
(0,2)	15	0	0	
(1,2)	14	0	0	
(0,3)	-9	0	0	
(1,3)	-7	0	0	
(1,1)	23	0		(4)
(1,0)	-34	0		(5)
(0,1)	-31	1		(6)
(0,2)	15	0		(7)
(1,2)	14	1		(8)
(2,4)	-1	0	0	
(3,4)	47	1		
		0	48	
(2,5)	-3	0	0	
(3,5)	2	0	0	
(0,3)	-9	0		(9)
(1,3)	-7	0		

จากตัวอย่างดังกล่าว ตัวเข้ารหัสได้ทำการสร้าง 29 บิท ในการผ่านครั้งแรก ตลอดกระบวนการนี้มีสัมประสิทธิที่สำคัญได้แก่ {63, -34, 49, 47} ตัวถอดรหัสจะทำการแปลงค่าสัมประสิทธิ์กลับโดยอาศัยบิทเหล่านี้ โดยตัวถอดรหัสรู้ว่าสัมประสิทธิ์แต่ละค่าจะอยู่ในช่วง -32 ถึง 32 ซึ่งจะใช้ค่ากลางคือ 0 ในการประมาณ ผลจากการแปลงกลับจะเป็นดังรูปที่ 2.13(a) และ 2.13(b)

48	-48	48	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	48	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(a)

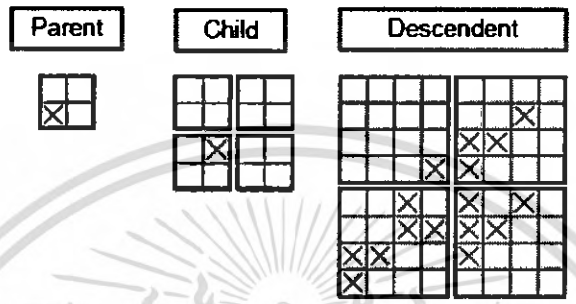
56	-40	56	0	0	0	0	0
-24	24	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	40	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b)

รูปที่2.13 การแปลงกลับภายหลังจาก (a) การผ่านครั้งแรก (b) การผ่านครั้งที่2

2.7 การปรับปรุงการบีบอัดวิธีSPIHT ด้วย Block Coding

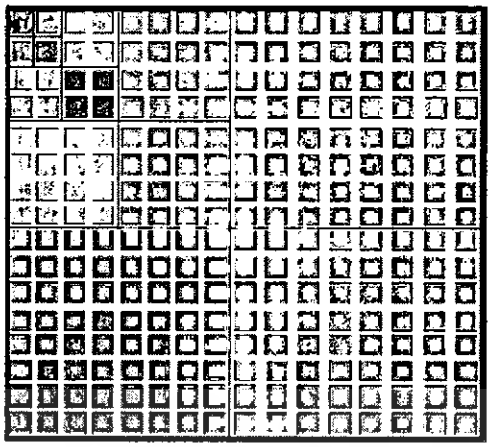
เนื่องจากโครงสร้างที่ถูกบังคับให้เป็นลักษณะจัตุรัสจากพ่อแม่ไปยังลูก และจากลูกไปยังหลาน ดังแสดงในรูปที่ 2.14 ข้อกำหนดนี้ทำให้เกิดลักษณะของจุดภาพประเภท Isolate Zero (IZ) ขึ้นในภาพที่มีรายละเอียดเส้นหรือขอบที่มีความคมชัดมากๆ และถ้ามีเป็นจำนวนมากจะทำให้การเข้ารหัสแบบ EZW หรือSPIHT ไม่มีประสิทธิภาพเท่าที่ควร ภาพลักษณะดังกล่าวเมื่อผ่านการแปลงแบบเวฟเล็ทแล้วพิจารณาขยายเฉพาะจุดภาพที่อยู่บริเวณขอบคม จะมีลักษณะดังรูปที่2.14



รูปที่2.14 ลักษณะการเกิดจุดภาพสำคัญบริเวณที่มีขอบคม

การเกิดจุดภาพลักษณะเช่นนี้ จะทำให้การเข้ารหัสภาพต้องส่งข้อมูลประเภทIsolate Zero (IZ) เป็นจำนวนมาก ซึ่งจะส่งผลให้ประสิทธิภาพการบีบอัดต่ำลง ดังนั้นวิธีหนึ่งที่สามารถช่วยเพิ่มประสิทธิภาพการบีบอัดวิธีSPIHT คือการลดจำนวนข้อมูลประเภทนี้ ซึ่งเป็นสมมติฐานหลักในการปรับปรุงโครงสร้างการเข้ารหัสภาพของบทความนี้ ซึ่งมีรายละเอียดดังต่อไปนี้

เริ่มต้นจากการแบ่งภาพที่ผ่านการแปลงเวฟเล็ทที่มีขนาด $(M \times N)$ ออกเป็นบล็อกย่อยๆที่มีขนาด32×32จุดภาพเท่าๆกันดังรูปที่2.15 (ขนาดบล็อกสามารถเปลี่ยนแปลงได้ แต่32×32เป็นขนาดที่พอเหมาะ) การแบ่งภาพเป็นส่วนย่อยๆจะทำได้ง่ายต่อการวิเคราะห์เพื่อแยกแยะหาบริเวณที่ไม่มีรายละเอียดซึ่งไม่จำเป็นที่จะต้องเข้ารหัสส่งข้อมูลและสามารถช่วยลดเหตุการณ์ที่มีการเกิดจุดภาพประเภท Isolate Zero (IZ) ลงได้ และขั้นตอนวิธีใหม่นี้ได้ปรับเปลี่ยนโครงสร้างโดยยึดสมมติฐานที่ว่า "ลักษณะการเกิดจุดภาพที่สำคัญมิใช่มีเพียงความสัมพันธ์ระหว่างจุดภาพพ่อแม่ไปยังจุดภาพลูกหลานแต่เพียงอย่างเดียว แต่ยังมีความสัมพันธ์ระหว่างจุดภาพในบริเวณข้างเคียงอีกด้วย"



รูปที่2.15 การแบ่งภาพภายหลังจากผ่ายการแปลงเวฟเล็ท

ขั้นตอนวิธีที่ได้ปรับปรุงนี้ ได้เปลี่ยนโครงสร้างการเข้ารหัสตำแหน่งจุดภาพสำคัญขึ้นใหม่ โดยแบ่งขั้นตอนการเข้ารหัสออกเป็น2ส่วน คือ

2.7.1.ขั้นตอนค่าสูงสุดในบล็อก (Max-of-Block Process)

ขั้นตอนนี้เป็นการค้นหาและจำแนกความสำคัญของแต่ละบล็อกโดยการส่งข้อมูลไปยังตัวถอดรหัสเพื่อบ่งบอกว่าบล็อกใดมีความสำคัญ ซึ่งการจำแนกนี้สามารถทำได้โดยการเข้ารหัสจุดภาพในภาพ Max-of-Block (MOB) ที่มีขนาด $(M/32 \times N/32)$ โดยที่ค่าสัมประสิทธิ์จุดภาพ (i,j) ของภาพ MOB จะเก็บค่าสัมประสิทธิ์ที่สูงสุดของบล็อก (i,j) ในภาพที่ผ่านการแปลงเวฟเล็ทแล้ว ดังสมการที่ (2.34)

$$MOB(i,j) = \max\left\{Block_{i,j}(x,y)\right\} \tag{2.34}$$

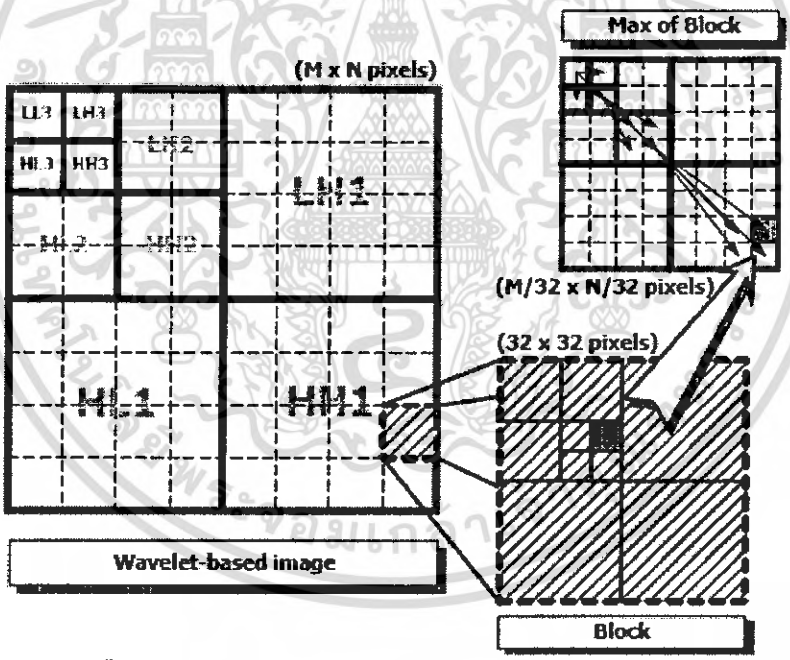
โดยที่ $MOB(j,j)$ คือค่าสัมประสิทธิ์จุดภาพ (i,j) ของภาพMOBและ $Block_{i,j}(x,y)$ คือบล็อกแถวที่ i และหลักที่ j ในภาพที่ผ่านการแปลงเวฟเล็ทแล้ว ส่วน (x,y)เป็นตำแหน่งในการหาค่าสูงสุดในบล็อกโดย x,y จะมีค่าอยู่ในช่วง 0-31

เนื่องจากจุดภาพใน MOB เก็บค่าสัมประสิทธิ์สูงสุดในแต่ละบล็อกไว้นั้น จึงเปรียบเสมือนจุดภาพนั้นเป็นตัวแทนความสำคัญของบล็อกนั้น ซึ่งหมายความว่าเมื่อจุดภาพใดใน MOBได้ถูกพิจารณาว่าเป็นจุดภาพสำคัญ จะแสดงว่าในบล็อกนั้นจะต้องมีจุดภาพที่สำคัญอย่างน้อยหนึ่งจุดเกิดขึ้น การเข้ารหัสจุดภาพ MOB เพื่อส่งให้ตัวถอดรหัสนี้ ได้เลือกใช้ขั้นตอนวิธีเช่นเดียวกับ SPIHT ดังรูปที่ 2.16 ด้านขวาบน เนื่องจากลักษณะภาพใน MOBจะมีลักษณะที่คล้ายหรือใกล้เคียงกับภาพต้นแบบที่ผ่านการแปลงเวฟเล็ท แต่ถูกย่อขนาดเล็กลง ในกรณีภาพที่มีลายเส้นหรือขอบที่เด่นชัด ภาพ MOB เหมาะที่จะใช้การเข้ารหัสวิธี SPIHT เนื่องจากการทำภาพ MOB จะสร้างความสัมพันธ์ระหว่างพ่อแม่และลูกหลานที่สำคัญของบล็อกเข้าด้วยกัน ซึ่งจะลดจำนวนข้อมูลประเภท Isolate Zero (IZ) ลงได้

2.7.2 ขั้นตอนบล็อก (Block Process)

ภายหลังจากที่ผ่านการพิจารณาในส่วน MOB แล้ว ในส่วนถัดไปจะเป็นการเข้ารหัสเพื่อบอกตำแหน่งของจุดภาพที่สำคัญภายในบล็อกโดยในบทความนี้ได้เลือกใช้โครงสร้างต้นไม้แบ่งสี่(Quad Tree) ซึ่งมีลักษณะดังรูปที่ 2.16 ด้านขวาล่างและตามด้วยการเข้ารหัสแบบArithmetic Codingเนื่องจากตามปกติทั่วไปจุดภาพสำคัญมักจะเกิดในบริเวณใกล้เคียงหรือที่ติดกัน ซึ่งเหมาะกับลักษณะการเข้ารหัสโครงสร้างต้นไม้แบ่งสี่ โดยขั้นตอนวิธีในส่วนนี้จะทำการเรียงลำดับการสำรวจหาจุดภาพสำคัญเพื่อเข้ารหัส โดยจะทำการสำรวจในบริเวณใกล้เคียงกับจุดภาพที่เคยที่มีการเกิดจุดภาพสำคัญมาแล้วก่อนหน้า เนื่องจากรายละเอียดหรือลายเส้นของภาพส่วนใหญ่มักจะเกิดในบริเวณที่ติดกันหรือใกล้เคียงกัน ดังนั้นจึงมีความน่าจะเป็นที่จะเกิดจุดภาพสำคัญมากกว่าบริเวณอื่น และภาพหลังจากการสำรวจบริเวณใกล้เคียงจุดภาพสำคัญเดิมเสร็จสิ้นแล้ว ก็จะทำการสำรวจบริเวณอื่นๆในบล็อกจนหมด

ตัวอย่างในรูปที่ 2.16 แสดงลักษณะความสัมพันธ์ระหว่างภาพต้นแบบที่ผ่านการแปลงเวฟเล็ทขนาด $M \times N$ (โดย M และ N ควรจะหารด้วย32ลงตัว เพื่อไม่ให้เหลือเศษของบล็อก) กับภาพMOBที่มีขนาด $(M/32 \times N/32)$ และบล็อกที่มีขนาด 32×32 โดยในรูปด้านขวาล่างจะแสดงตัวอย่างบล็อกหนึ่งที่ผ่านการแบ่งในบริเวณย่านความถี่ HH1ซึ่งค่าสัมประสิทธิ์สูงสุดภายในบล็อกนั้นจะถูกเก็บไว้ในภาพ MOB ซึ่งแสดงไว้ในรูปขวาบน



รูปที่ 2.16 ขั้นตอนการเข้ารหัสภาพขนาด $M \times N$ จุดภาพโดยแบ่งออกเป็น2ส่วน คือ Block และ Max-of-Block

2.8 ประวัติความเป็นมาของภาษาวีเอชดีแอล

วีเอชดีแอล ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และสามารถเขียนได้หลายรูปแบบด้วยเหตุผลนี้จึงทำให้ภาษาวีเอชดีแอล เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกรได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชั่นการทำงานของวงจรอย่างสังเขป โดยไม่คำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนี้วีเอชดีแอลยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มีความซับซ้อนได้ทั้งหมด ดังนั้น วีเอชดีแอลจึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งานเป็นอย่างยิ่ง สำหรับมาตรฐานของภาษาที่ใช้บรรยายพฤติกรรมวงจร หรือฮาร์ดแวร์ สามารถสรุปได้ดังนี้

-ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอลและมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่องคอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก

- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้

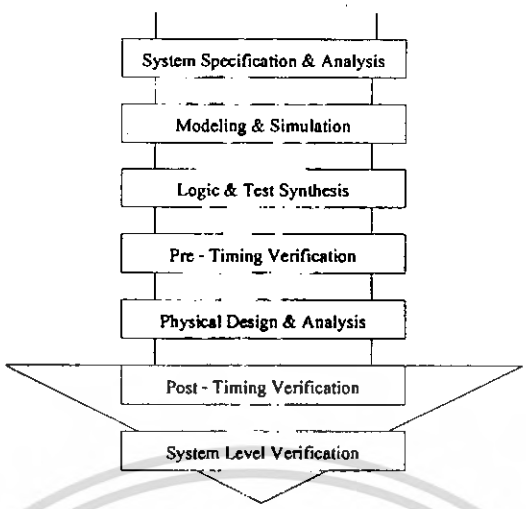
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่น เดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรมภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า ภาษาโปรแกรมระดับสูง

2.8.1 การออกแบบจากบนลงล่าง (Top-Down Design)

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของบล็อกไดอะแกรมก่อนที่ทำวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษาวีเอชดีแอลนั้นอนุญาตให้อธิบายและวิเคราะห์การทำงานของแต่ละบล็อก รวมถึงการปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามต้องการนอกจากนี้ยัง สามารถเพิ่มเติมในรายละเอียดในแต่ละขั้นตอนได้ ซึ่งหลักการนี้สอดคล้องกับหลักการออกแบบจากบนลงล่างนั่นเอง ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom-Up Design) จะเห็นได้ว่า การออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เนื่องจากเป็นการวางวงจรด้วยอุปกรณ์ต่างๆ (Schematic Capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบก่อน แล้วจึงทำการจำลองการทำงาน และตรวจสอบความถูกต้อง

วีเอชดีแอลกับหลักการออกแบบจากบนลงล่างจึงเป็นทางออกให้กับวิศวกรให้สามารถ ออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 2.17 แสดงขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.17 แสดงถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อแตกต่างไปจากนี้บ้าง เล็กน้อยเนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอน การออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

1) ความต้องการของระบบและการวิเคราะห์ คือ การสร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

2) รูปแบบและการจำลองการทำงาน คือ การเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา วีเอชดีแอล หรือ ภาษา เอชดีแอล อื่นๆ สำหรับใช้ในการบรรยายพฤติกรรมการทำงาน พร้อมทั้งทำการจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3) ลอจิกและการทดสอบการสังเคราะห์ คือ หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้วหลักการนี้จะถูกเพิ่มเติมรายละเอียดลงมาเป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจรจริง หรือทำการสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่จะเขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของโครงข่ายการเชื่อมต่อ ที่สามารถนำไปผลิตอุปกรณ์อื่นได้

4) การตรวจสอบเวลาก่อนการออกแบบ คือ หลังจากได้ทำการสังเคราะห์วงจรให้อยู่ในระดับเกท หรือ โครงข่ายการเชื่อมต่อแล้ว ข้อมูลนี้จะถูกนำไปใช้สำหรับการจำลองการทำงานในเรื่องความถูกต้องของฟังก์ชันพร้อมกับนำข้อมูลที่เกี่ยวข้องกับเวลาเข้ามาใช้ในการประกอบในการพิจารณาด้วย ซึ่งตามปกติแล้วอุปกรณ์ ทางอิเล็กทรอนิกส์ทุกชิ้นจะต้องมีเวลาหน่วงของการแพร่กระจาย (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็น เวลาที่น้อยมากในระดับนาโนวินาทีก็ตาม แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันต่างๆ จำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจ

ทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไป หรือไม่สามารถที่จะทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

5) การออกแบบทางกายภาพและการวิเคราะห์ คือ ขั้นตอนในการผลิตเป็นวงจรจริง (Technology and device mapping) โดยจะนำข้อมูลที่ได้จากการสังเคราะห์ มาใช้ในการผลิตเป็นวงจรรวม ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวมเอซิก (ASIC)

6) การตรวจสอบเวลาหลังการออกแบบ คือ การทำการตรวจสอบการทำงานด้วยตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้ายก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้ วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุทและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

7) การตรวจสอบระบบ คือ การนำวงจรที่ออกแบบไว้ประกอบเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบที่สมบูรณ์ แล้วทำการทดสอบการทำงานทั้งระบบรวมกับอุปกรณ์อื่นๆ อีกครั้งเพื่อควบคุมคุณภาพของผลิตภัณฑ์

2.9 หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจาย

โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) หรือเรียกอย่างย่อๆว่า "DA" เป็นการจัดรูปแบบทางคณิตศาสตร์ของสมการที่อยู่ในรูปของผลบวกของผลคูณ (Sum of Product) ของเลขฐานสิบให้กระจายออกเป็นบิทหรือในรูปแบบของเลขฐานสอง เพื่อให้การคำนวณทางคณิตศาสตร์สามารถแปลงเป็นวงจรดิจิตอลอิเล็กทรอนิกส์ได้ โดยจะปรากฏอยู่ในงานทางด้านการประมวลผลสัญญาณเชิงเลข หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจายที่นำมาใช้งานด้านการประมวลผลสัญญาณเชิงเลข คือการแปลงฟังก์ชันถ่ายโอน (Transfer Function) ซึ่งเป็นสมการที่อยู่ในรูปผลบวกของผลคูณระหว่างสัญญาณอินพุทกับค่าสัมประสิทธิ์ของฟังก์ชั่นถ่ายโอนของระบบ โดยในการคูณกันระหว่างค่าสัมประสิทธิ์ของฟังก์ชั่นถ่ายโอนกับสัญญาณอินพุท จะใช้การคูณเลขฐานสองแบบส่วนเติมเต็มสอง (2's complement) และการคูณจะใช้แบบเปิดตาราง (Look-up table) โดยค่าผลบวกของผลคูณระหว่างสัมประสิทธิ์และสัญญาณอินพุทจะถูกเก็บไว้ในหน่วยความจำ EPROM และจะใช้สัญญาณอินพุทเป็นแอดเดรสของ EPROM โดยตรง ทั้งค่าสัมประสิทธิ์ของวงจรกรองและสัญญาณอินพุทจะถูกแทนด้วยเลขส่วนเติมเต็มสอง ดังนั้นโครงสร้างเลขคณิตกระจายจึงมีทฤษฎีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง (2's complement Multiplication)

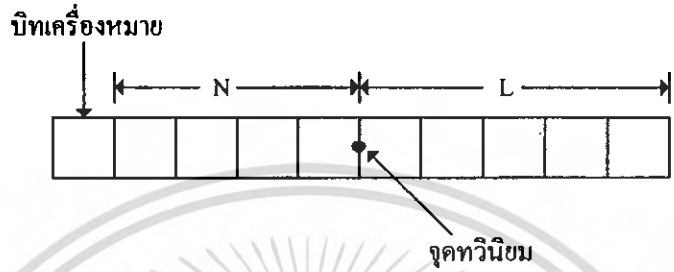
2.9.1 ระบบตัวเลข

สำหรับระบบเชิงเลข ตัวเลขต่างๆจะถูกแทนด้วยเลขฐานสอง ซึ่งโดยทั่วไปมีรูปแบบที่นิยมใช้กันอยู่ 2 รูปแบบ คือ รูปแบบจำนวนโดยตรง (Fixed point format) และ รูปแบบจำนวนอิงครรชนี (Floating point format) ซึ่งรูปแบบจำนวนโดยตรงจะมีวงจรฮาร์ดแวร์ที่ใช้ในการคำนวณที่ง่ายกว่า แต่ให้ค่าจากการคูณค่อนข้างจำกัด ส่วนรูปแบบจำนวนอิงครรชนีจะสามารถแทนค่าของสัญญาณ คือให้ย่านพลวัตร

(Dynamic range) ได้มากกว่า แต่ต้องใช้วงจรฮาร์ดแวร์ที่สลับซับซ้อน แพงกว่า และให้ความเร็วในการประมวลผลที่ลดลง

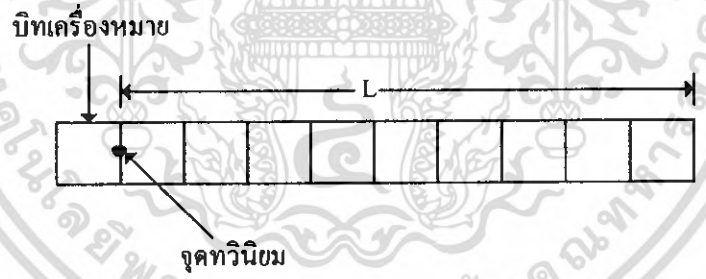
2.9.1.1 รูปแบบจำนวนโดยตรง

รูปแบบจำนวนโดยตรงปกติจะประกอบไปด้วย 3 ส่วน คือ บิทเครื่องหมาย (Sign bit) 1 บิท, บิทจำนวนเต็ม (Integer bit) N บิท และบิทเศษส่วน (Fractional bit) L บิท โดยจะมีจุดทวินิยม (Binary point) อยู่ระหว่างบิทจำนวนเต็มและบิทเศษส่วนดังแสดงในรูปที่ 2.18



รูปที่ 2.18 แสดงการจัดรูปแบบจำนวนโดยตรงที่ประกอบด้วยบิทจำนวนเต็มและบิทเศษส่วน

จำนวนบิท N เป็นตัวกำหนดย่านพลวัตรที่ต้องการ โดยถ้าเลือกให้มีจำนวนน้อยอาจทำให้เกิดการล้น (Overflow) จากการคำนวณได้ แต่ถ้าเลือกให้มีจำนวนมากความเที่ยงตรงก็จะน้อยลง ซึ่งในการสร้างวงจรกรองสัญญาณเชิงเลขโดยการแทนด้วยรูปแบบจำนวนโดยตรงนั้น นิยมที่จะทำมาตราส่วน (Scaling) เพื่อให้ขนาดของสัญญาณมีค่าอยู่ระหว่าง $-1 \le x < 1$ คือมีบิทเครื่องหมาย 1 บิท และบิทเศษส่วน L บิท ดังแสดงในรูปที่ 2.19



รูปที่ 2.19 แสดงการจัดรูปแบบจำนวนโดยตรงที่มีแต่บิทเศษส่วน

โดยทั่วไปเลขฐานสองแบบจำนวนโดยตรงแบ่งออกได้เป็น 3 รูปแบบด้วยกัน คือ (1) แบบขนาดและเครื่องหมาย (Sign magnitude), (2) แบบส่วนเติมเต็มหนึ่ง (1's complement) และ (3) แบบส่วนเติมเต็มสอง (2's complement) โดยคุณลักษณะที่สำคัญบางประการของการแทนตัวเลขด้วยเลขฐานสองแบบจำนวนโดยตรงทั้ง 3 รูปแบบสามารถสรุปได้ดังตารางที่ 2.2

ตารางที่ 2.2 แสดงคุณสมบัติที่สำคัญของรูปแบบจำนวนโดยตรง

Features	Sign and magnitude	2' complement	1'complement
Range	$-(1-2^{-L}) \le x \le (1-2^{-L})$	$-1 \le x \le (1-2^{-L})$	$-(1-2^{-L}) \le x \le (1-2^{-L})$
Representation of zero	0.000 and 1.000	0.000	0.000 and 1.111
Arithmetic rules	Simple must be kept track of, separately	Simple; negative numbers elegantly handled	Simple, but "end around carry" should be carefully handled
Suitability for serial arithmetic	Not so good	Excellent	Good

ใน 3 รูปแบบนี้ตัวเลขแบบส่วนเติมเต็มสองเป็นที่นิยมใช้กันมากในระบบการประมวลผลสัญญาณเชิงเลข ทั้งนี้เนื่องมาจาก

1. มีการแทนค่าเลขศูนย์ได้เพียงค่าเดียว

2. การสร้างวงจรฮาร์ดแวร์สำหรับการบวก ลบ และคูณ ของเลขส่วนเติมเต็มสองทำได้ง่ายโดยในการคูณสามารถใช้หลักการเลื่อนและบวก (Shift and add)

3. ในระหว่างผลการบวกย่อย (Partial sum) ของการบวกเลขส่วนเติมเต็มสอง สามหรือสี่จำนวนถึงแม้อาจจะเกิดการล้น (ตัวทดจากผลการบวกล้นข้ามไปทับบิทเครื่องหมาย) แต่ผลลัพธ์สุดท้ายมักให้ค่าถูกต้องเสมอ ถ้าผลบวกอยู่ในช่วง -1 ถึง $1-2^{-L}$ ดังตัวอย่าง

7/8	0.111	
+4/8	0.100	
11/8	1.011	ผลบวกย่อยที่ผิดเนื่องจากเกิดการล้น
6/8	1.010	
5/8	0.101	ผลบวกที่ถูกต้อง

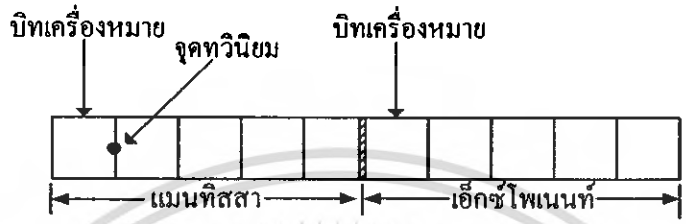
2.9.1.2 รูปแบบจำนวนอิงดรรชนี

รูปแบบจำนวนโดยตรงมีข้อเสียที่สำคัญ 2 ประการ คือ (1) ย่านพลวัตรของตัวเลขมีค่าน้อย เช่น การแทนด้วยเลขส่วนเติมเต็มสอง ค่าที่น้อยที่สุดคือ -1 และค่าที่มากที่สุด คือ $1-2^{-L}$ เปอร์เซ็นต์ความผิดพลาดที่เกิดจากการตัด (Truncation) หรือการปัด (Rounding) จะเพิ่มมากขึ้นเมื่อขนาดของตัวเลขมีค่าลดลง ตัวอย่างเช่น ถ้าจำนวน 0.11011010 และ 0.000110101 ถูกตัดให้จำนวนบิทเศษส่วนเหลือเพียง 4 บิท เปอร์เซ็นต์ความผิดพลาดจะเป็น 4.59 % และ 39.6 % ตามลำดับ โดยข้อเสียนี้สามารถแก้ไขได้โดยการใช้รูปแบบจำนวนอิงดรรชนี ซึ่งตัวเลข X แสดงได้โดย

$$X = M \times 2^{e} \qquad (2.35)$$

โดย e เป็นจำนวนเต็ม และ $\frac{1}{2} \le |M| < 1$

M และ e เรียกว่า แมนทิสสา (Mantissa) และ เอ็กซ์โพเนนท์ (Exponent) ตามลำดับ ตัวอย่างเช่น จำนวน 0.00110101 และ 01001.11 สามารถแทนได้โดย 0.110101×2^{-2} และ 0.100111×2^{4} ตามลำดับ ส่วนจำนวนที่มีค่าเป็นลบก็ทำในลักษณะเดียวกัน รูปแบบจำนวนอิงครรชนีสามารถแสดงได้ดังรูปที่ 2.20 โดยแบ่งเป็น 2 ส่วน คือส่วนหนึ่งสำหรับแมนทิสสา และอีกส่วนสำหรับเอ็กซ์โพเนนท์



รูปที่ 2.20 แสดงการจัดรูปแบบจำนวนอิงครรชนี

ข้อดีของการใช้จำนวนอิงครรชนี คือแทนค่าของสัญญาณได้ละเอียดกว่า และแม่นยำกว่าแบบจำนวนโดยตรง แต่การบวก ลบ หรือคูณจะยุ่งยากกว่ามาก วงจรจึงซับซ้อนและแพงกว่าแบบจำนวนโดยตรงมาก นอกจากนี้ความเร็วในการประมวลผลยังช้ากว่าด้วย ดังนั้นสำหรับการประมวลผลแบบเวลาจริง (Real time) จึงนิยมใช้ระบบตัวเลขแบบจำนวนโดยตรง

2.9.2 ทฤษฎีเลขคณิตกระจาย

จากที่ได้กล่าวมาแล้วว่า โครงสร้างเลขคณิตกระจายมีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง ดังนั้นในหัวข้อนี้จะได้อธิบายถึงหลักการของการคูณเลขส่วนเติมเต็มสอง

ให้เลขส่วนเติมเต็มสองของ X ซึ่งแทนด้วย $\overline{X}$ และนิยามโดย

$$\bar{x} = \begin{cases} x & \text{ถ้า } X \ge 0 \\ 2 - |x| & \text{ถ้า } X < 0 \end{cases} \qquad (2.36)$$

โดย X เป็นเลขที่เป็นเศษส่วน (Fractional number)

ในระบบเลขส่วนเติมเต็มสองจะใช้บิทที่มีนัยสำคัญสูงสุด (MSB) เป็นบิทแสดงเครื่องหมาย โดยถ้าเป็นบวกแทนด้วย "0" และถ้าเป็นลบแทนด้วย "1" ถ้าให้ X แทนด้วยเลขฐานสองขนาด L+1 บิท ดังนั้นรูปแบบของเลขส่วนเติมเต็มสองจะเขียนได้ดังนี้

$$\overline{X} = X_0.X_1.X_2...X_L \qquad (2.37)$$

ค่าของ $\overline{X}$ ในรูปของเลขฐานสิบสามารถหาได้ดังนี้

$$X = -X_0 + \sum_{i=1}^{L} X_i 2^{-i} \tag{2.38}$$

จากนั้นพิจารณาผลคูณต่อไปนี้

$$Y = Xm \tag{2.39}$$

ให้ $\overline{Y}$, $\overline{X}$ และ $\overline{m}$ เป็นเลขส่วนเติมเต็มสองของ Y, X และ m ตามลำดับ จากนั้นพิจารณาจากสมการที่ (2.38) และ สมการที่ (2.39) จะได้

$$\begin{aligned} Y &= -Y_0 + \sum_{i=1}^{L} Y_i 2^{-i} \\ &= -X_0 m + \sum_{i=1}^{L} X_i m 2^{-i} \end{aligned} \tag{2.40}$$

ดังนั้น

$$\begin{aligned} \overline{Y} &= \text{ส่วนเติมเต็มสองของ} \left(-X_0 m + 2^{-1} X_1 m + 2^{-2} X_2 m + 2^{-3} X_3 m + \ldots + 2^{-L} X_L m\right) \\ &= \text{ส่วนเติมเต็มสองของ} \left(-X_0 m + 2^{-1}(X_1 m + \ldots + 2^{-1}(X_{L-1} m + 2^{-1}(X_L m)))\right) \end{aligned} \tag{2.41}$$

ต่อไปพิจารณาส่วนเติมเต็มสองของ $2^{-1}U$ โดย

$\overline{U} = U_0.U_1U_2\ldots U_M$; สำหรับ $U \geq 0$ (หรือ $U_0 = 0$)

ส่วนเติมเต็มสองของ $(2^{-1}U) = 2^{-1}\overline{U}$; และสำหรับ $U < 0$ (หรือ $U_0 = 1$)

ส่วนเติมเต็มสองของ $(2^{-1}U) = 2 - \left|2^{-1}U\right| = 1 + 2^{-1}\left(2 - |U|\right) = 1 + 2^{-1}\overline{U}$

ดังนั้นสรุปได้ว่า

$$\text{ส่วนเติมเต็มสองของ} (2^{-1}U) = \begin{cases} 2^{-1}\overline{U} & ; U_0 = 0 \\ 1 + 2^{-1}\overline{U} & ; U_0 = 1 \end{cases} \tag{2.42}$$

สมการที่ (2.42) นี้แสดงให้เห็นได้ว่า ส่วนเติมเต็มสองของ $(2^{-1}U)$ เป็นการเลื่อนข้อมูลของ $\overline{U}$ ไปทางขวา 1 บิท

$$\therefore \text{ส่วนเติมเต็มสองของ} (2^{-1}U) = 2_2^{-1}\overline{U} \tag{2.43}$$

โดย $2_2^{-1}\overline{U}$ แสดงถึงการเลื่อนข้อมูลของ $\overline{U}$ ไปทางขวา 1 บิท แบบเลขส่วนเติมเต็มสอง ซึ่งสัญลักษณ์ 2_2^{-1} (ซึ่งโดยทั่วไปนิยมเขียนเป็น 2^{-1}) เป็นการแสดงว่าในกรณีที่ $\overline{U}$ เป็นเลขบวก ซึ่งบิท

เครื่องหมายจะเป็นเลขศูนย์ โดยหลังจากเลื่อนข้อมูลไปทางขวา 1 บิทแล้ว บิทที่มาแทนบิทเครื่องหมายก็ยังคงเป็นเลขศูนย์ ส่วนในกรณีที่ $\overline{U}$ เป็นเลขลบ หลังจากเลื่อนข้อมูลไปทางขวา 1 บิทแล้ว บิทที่มาแทนบิทเครื่องหมายจะเป็นเลขหนึ่ง (จาก $1+2^{-1}\overline{U}$) ซึ่งในการสร้างเพื่อใช้งานจริงจำเป็นจะต้องมีวงจรที่ใช้ในการตรวจสอบบิทเครื่องหมาย (Sign digit) ทุกครั้งที่มีการเลื่อนข้อมูล

จากนั้นพิจารณาสมการที่ (2.42) และสมการที่ (2.43) จะได้ว่า

$$\begin{aligned}\overline{Y} &= -X_0\overline{m} + 2^{-1}X_1\overline{m} + 2^{-2}X_2\overline{m} + 2^{-3}X_3\overline{m} + \ldots + 2^{-L}X_L\overline{m} \\ &= -X_0\overline{m} + 2^{-1}(X_1\overline{m} + \ldots + 2^{-1}(X_{L-1}\overline{m} + 2^{-1}(X_L\overline{m})))\end{aligned} \qquad (2.44)$$

ซึ่งจากสมการที่ (2.44) จะเห็นได้ว่าผลคูณจากสมการที่ (2.19) สามารถหาได้โดยการใช้หลักการเลื่อนและบวก (Shift and add) โดยผลลัพธ์ที่ได้จากการคูณแบบเลขส่วนเติมเต็มสอง สามารถหาได้ตามขั้นตอนดังนี้

1. เคลียร์ค่าข้อมูลในแอคคิวมูเลเตอร์รีจิสเตอร์
2. บวก $X_L\overline{m}$ กับค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์
3. เลื่อนค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ไปทางขวา 1 บิท
4. ทำซ้ำข้อ 2 และ 3 สำหรับค่า $X_{L-1},\ldots,X_1$
5. ลบค่า $X_0\overline{m}$ ออกจากค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ (ลบแบบเลขส่วนเติมเต็มสอง)

ตัวอย่างการทำงานตามอัลกอริธึมนี้

$Y = Xm = 0.8125(-0.390625)$ โดยสมมุติให้ใช้แอคคิวมูเลเตอร์รีจิสเตอร์ขนาด 12 บิท

$m = -0.390625$

$\overline{m} = 2 - |m|$ $\because$ m เป็นเลขลบ

$= 2 - 0.390625$

$= 1.609375$

$\therefore\ \overline{m} = 1.100111$

$X = 0.8125 = \overline{X}$ $\because$ X เป็นเลขบวก

$\therefore \overline{X} = 0.1101 = X_0.X_1X_2X_3X_4$

โดยมีขั้นตอนการทำงาน ดังตารางต่อไปนี้

ตารางที่ 2.3 แสดงขั้นตอนการคูณเลขส่วนเติมเต็มสอง

การดำเนินการ	ข้อมูลในแอคคิวมูเลเตอร์รีจิสเตอร์
เคลียร์ ACC	0.000 0000 0000
ACC+ $X_4\overline{m}$	1.100 1110 0000
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.110 0111 0000
ACC+ $X_3\overline{m}$	1.110 0111 0000
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.111 0011 1000
ACC+ $X_2\overline{m}$	1.100 0001 1000
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.110 0000 1100
ACC+ $X_1\overline{m}$	1.010 1110 1100
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.101 0111 0110
ACC- $X_0\overline{m}$	1.101 0111 0110

$$\therefore \overline{Y} = 1.101\ \ 0111\ \ 0110\ \ = Y_0.Y_1Y_2\ldots Y_{11}$$

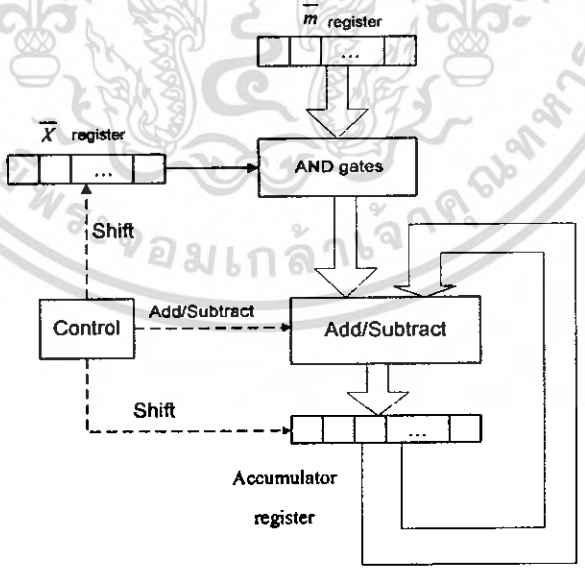
จะได้

$$Y = -Y_0 + \sum_{i=1}^{11} Y_i\, 2^{-i}$$

$$= -1 + \left(2^{-1} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-9} + 2^{-10}\right)$$

$$= -0.3173828125$$

จากอัลกอริธึมดังกล่าวสามารถออกแบบการทำงานและสร้างวงจรแสดงได้ดังรูปที่ 2.21



รูปที่ 2.21 แสดงการคูณแบบเลขส่วนเติมเต็มสองโดยใช้เลขคณิตกระจาย

2.10 การอินเตอร์เฟสตามมาตรฐาน RS-232

มาตรฐาน RS-232 เป็นมาตรฐานที่ได้รับการพัฒนามานานและถูกใช้งานอย่างแพร่หลาย เราใช้ RS-232 เชื่อมต่อ DTE (Data Terminal Equipment) เช่น คอมพิวเตอร์หรือเทอร์มินัล (Terminal) เป็นต้น เข้ากับ DCE (Data Communication Equipment) เช่น โมเด็ม (Modem) ทีเอ อะเด็บเตอร์ (TA adapter) พล็อตเตอร์ (Plotter) เป็นต้น ตัวอย่างการเชื่อมต่อเช่น การต่อเทอร์มินัลเข้ากับโมเด็ม

มาตรฐาน RS-232 จะใช้สัญญาณเส้นเดียวในการส่งสัญญาณโดยจะสัญญาณจะส่งไปในทิศทางเดียวกัน สำหรับการแทนค่าแรงดันในการส่งสัญญาณเป็นดังนี้

- สัญญาณของลอจิก "1" แทนด้วยระดับแรงดันไฟฟ้าระหว่าง -3 ถึง -25 โวลต์
- สัญญาณของลอจิก "0" แทนด้วยระดับแรงดันไฟฟ้า ระหว่าง 3 ถึง 25 โวลต์
- ส่วนแรงดันไฟฟ้าระหว่าง 3 ถึง -3 โวลต์ ไม่มีการนิยาม

การเชื่อมต่อกับพอร์ตสื่อสารของคอมพิวเตอร์ส่วนบุคคลจะเลือกใช้พอร์ตสื่อสารแบบอนุกรม 9 ขา (DB-9) ซึ่งสามารถทำการส่งสัญญาณข้อมูลได้ตามมาตรฐาน RS-232 โดยลักษณะของคอนเน็คเตอร์แบบ DB-9 สามารถแสดงได้ดังรูปที่ 2.22 และการเชื่อมต่อของพอร์ตสื่อสารสำหรับคอนเน็คเตอร์แบบ DB-9 สามารถแสดงได้ดังตารางที่ 2.4

รูปที่ 2.22 แสดงลักษณะของคอนเน็กเตอร์แบบ DB-9

ตารางที่ 2.4 การเชื่อมต่อของพอร์ตสื่อสารสำหรับคอนเน็คเตอร์แบบ DB-9

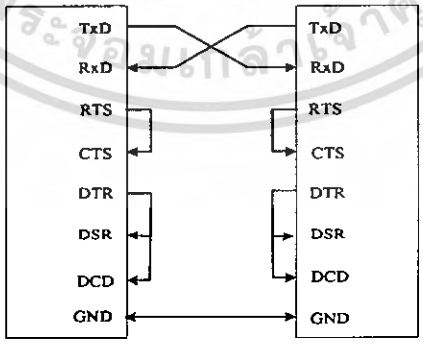
ตำแหน่งขาของ DB-9	สัญญาณ
1	Data Carrier Detect : DCD
2	Received Data : RxD
3	Transmitted Data : TxD
4	Data Terminal Ready : DTR
5	Signal Ground : GND
6	Data Set Ready : DSR
7	Request To Send : RST
8	Clear To Send : CTS
9	Ring Indicator : RI

เปรียบเทียบข้อดีข้อเสียของการสื่อสารข้อมูลแบบอนุกรมและขนาน

- การสื่อสารข้อมูลแบบอนุกรมสามารถสื่อสารได้ระยะทางที่ไกลกว่า
- การสื่อสารข้อมูลแบบอนุกรมใช้สายสัญญาณที่ประหยัดกว่า
- การสื่อสารข้อมูลแบบขนานสามารถสื่อสารข้อมูลได้ทีละมากๆ และรวดเร็วกว่า

2.10.1 นันโมเด็ม (Null Modems)

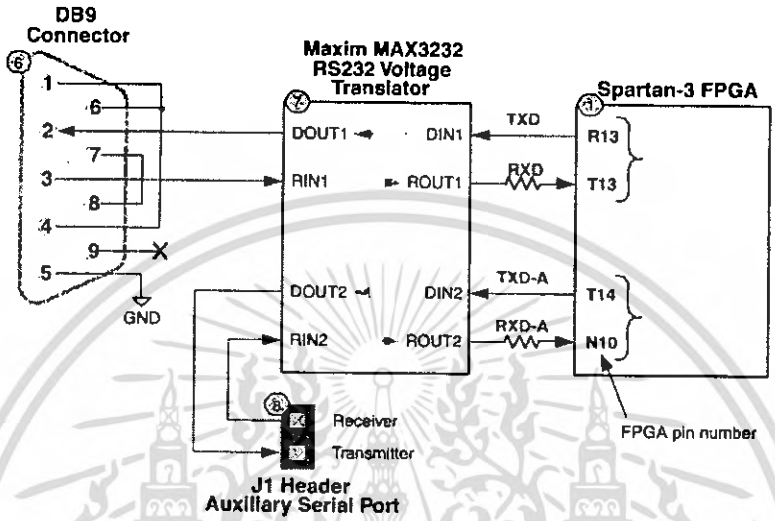
นันโมเด็ม (Null Modems) เป็นการเชื่อมต่อระหว่าง DTE (Data Terminal Equipment) เข้าด้วยกัน ซึ่งสามารถแสดงการเชื่อมต่อแบบนันโมเด็มได้ดังรูปที่ 2.23



รูปที่ 2.23 แสดงบล็อกไดอะแกรมการเชื่อมต่อแบบนันโมเด็ม

2.10.2 การเชื่อมต่อระหว่าง DB-9 กับ FPGA

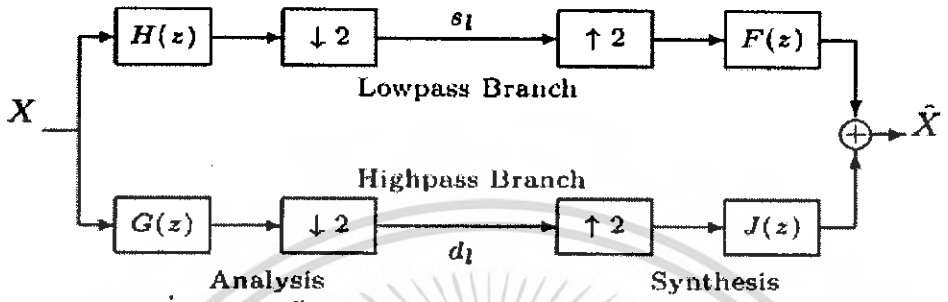
โครงงานนี้ได้ใช้บอร์ด FPGA สปาร์ตัน 3 สตาร์เตอร์คิท (Spartan-3 Starter Kit Board) การเชื่อมต่อระหว่างคอนเน็คเตอร์แบบ DB-9 และ บอร์ด FPGA จะต้องผ่าน MAX3232 ก่อนเพื่อปรับแรงดันที่รับมาได้จากพอร์ต RS232 ให้เหมาะสมกับระดับแรงดันภายในบอร์ด ซึ่งสามารถแสดงการเชื่อมต่อระหว่างคอนเน็คเตอร์แบบ DB-9 และ บอร์ด FPGA โดยผ่าน MAX3232 ได้ดังรูปที่ 2.24



รูปที่ 2.24 แสดงการเชื่อมต่อระหว่างคอนเน็คเตอร์แบบ DB-9 และ บอร์ด FPGA โดยผ่าน MAX3232

บทที่ 3
การคำนวณและการสร้าง

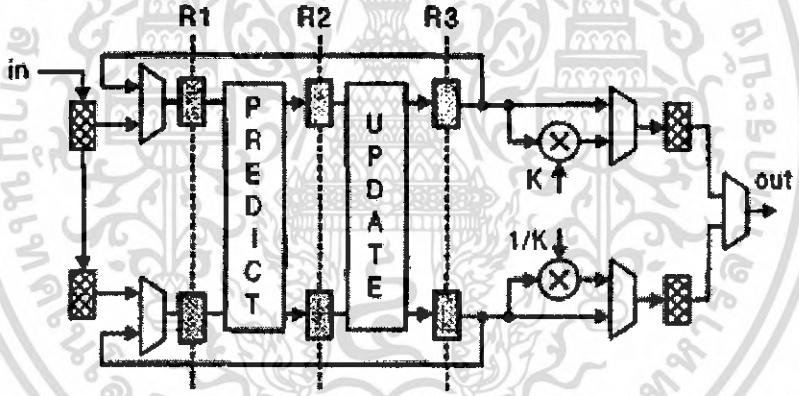
3.1 โครงสร้างและการคำนวณในส่วนของการหาค่าสัมประสิทธิ์ Discrete Wavelet Transform (DWT) โดยใช้โครงสร้างแบบ Lifting Scheme



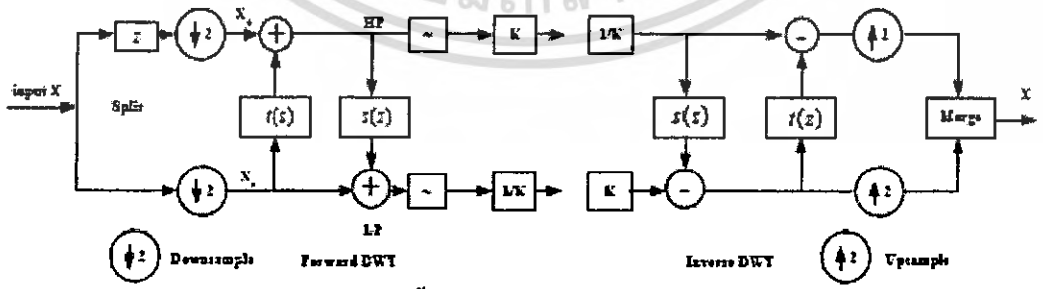
รูปที่ 3.1 โครงสร้างคอนโวลูชัน(Convolution) ของ Biorthogonal DWT

Perfect Reconstruction filter bank ทั้งในส่วนการวิเคราะห์ (Analysis) และการสังเคราะห์ (Synthesis)

การแปลงเวฟเลตโดยใช้โครงสร้างแบบ Lifting มีลักษณะดังนี้



รูปที่ 3.2 โครงสร้าง Lifting ทางด้านวิเคราะห์ (Analysis)



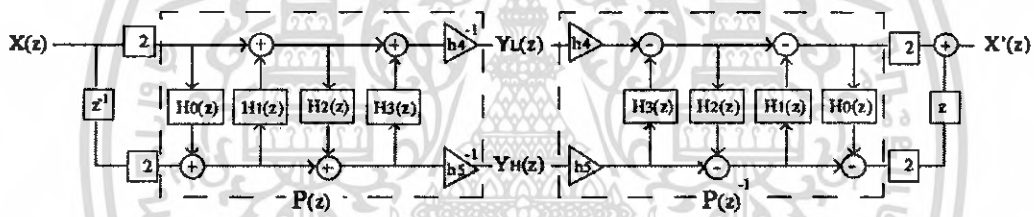
รูปที่ 3.3 โครงสร้างโดยละเอียด Lifting ทั้งทางด้านวิเคราะห์ (Analysis) และด้านสังเคราะห์ (Synthesis)

โดยจากรูปที่ 3.3 สามารถอธิบายได้ดังนี้

1) Split step ซึ่งสัญญาณถูกแบ่งเป็น จำนวนคู่และจำนวนคี่
2) Predict step ซึ่งจำนวนคู่ถูกคูณด้วย t(z) ใน time domain และนำไปรวมกับจำนวนคี่
3) Update step ซึ่งจำนวนคี่ถูกคูณด้วย s(z) ใน time domain และนำไปรวมกับจำนวนคู่
4) Scaling step ซึ่งจำนวนคู่ถูกคูณด้วยค่าคงที่ 1/K และจำนวนคี่ถูกคูณด้วย K

จากรูปที่ 3.3 สามารถพิจารณาได้ว่า DWT และ IDWT สมมาตรกันโดย IDWT สามารถทำได้โดย transverse ในทิศทางตรงกันข้าม เปลี่ยนค่า K เป็น 1/K และ 1/K เป็น K และ เปลี่ยนค่าเครื่องหมายของ ค่าสัมประสิทธิ์ใน t(z) และ s(z) สำหรับ Biorthogonal wavelet ของ Daubichies 9/7 wavelet ฟิลเตอร์ ในรูปของ lifting scheme

$$\tilde{P}(z)=\begin{bmatrix}1 & \alpha\left(1+z^{-1}\right)\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \beta(1+Z) & 1\end{bmatrix}\begin{bmatrix}1 & \gamma\left(1+z^{-1}\right)\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & \gamma\left(1+z^{-1}\right)\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \delta(1+z) & 1\end{bmatrix}\begin{bmatrix}K & 0\\ 0 & 1/K\end{bmatrix} \tag{3.1}$$



รูปที่ 3.4 Two channel Lifting Scheme

จากรูปที่ 3.4 จะเห็นว่าประกอบด้วย 2 tap FIR ฟิลเตอร์ แบบ linear phase

$$\begin{aligned}H_0(z)&=h_0(z+1)\\ H_1(z)&=h_1\left(1+z^{-1}\right)\\ H_2(z)&=h_2(z+1)\\ H_3(z)&=h_3\left(1+z^{-1}\right)\end{aligned} \tag{3.2}$$

สัมประสิทธิ์ของ 9/7 ฟิลเตอร์ที่นำมาคูณคือ

$$\begin{aligned}h_0&=-1.5863\ 43420\ 59924\\ h_1&=-0.05298\ 01185\ 72961\\ h_2&=0.88291\ 10755\ 30934\\ h_3&=0.44350\ 68520\ 43971\\ h_4&=1.23017\ 41049\ 14001\\ h_5&=1.62578\ 86132\ 23192\end{aligned} \tag{3.3}$$

ในส่วนการวิเคราะห์ ของ การเข้ารหัส (Encode) โดยเมื่อ ป้อนสัญญาณ $X(z)$ เป็น 2 แบนด์ $Y_L(z), Y_H(z)$

$$\begin{bmatrix} Y_L(z) \\ Y_H(z) \end{bmatrix} = P(z) \begin{bmatrix} \downarrow 2 & 0 \\ 0 & \downarrow 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} x(z) \tag{3.4}$$

ในส่วนการสังเคราะห์ ของ การถอดรหัส (Decode)

$$x'(z) = \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix}^T \begin{bmatrix} \uparrow 2 & 0 \\ 0 & \uparrow 2 \end{bmatrix} P^{-1}(z) \begin{bmatrix} Y_L(z) \\ Y_H(z) \end{bmatrix} \tag{3.5}$$

$$P(z) = K \times H_3(z) H_2(z) H_1(z) H_0(z) \tag{3.6}$$

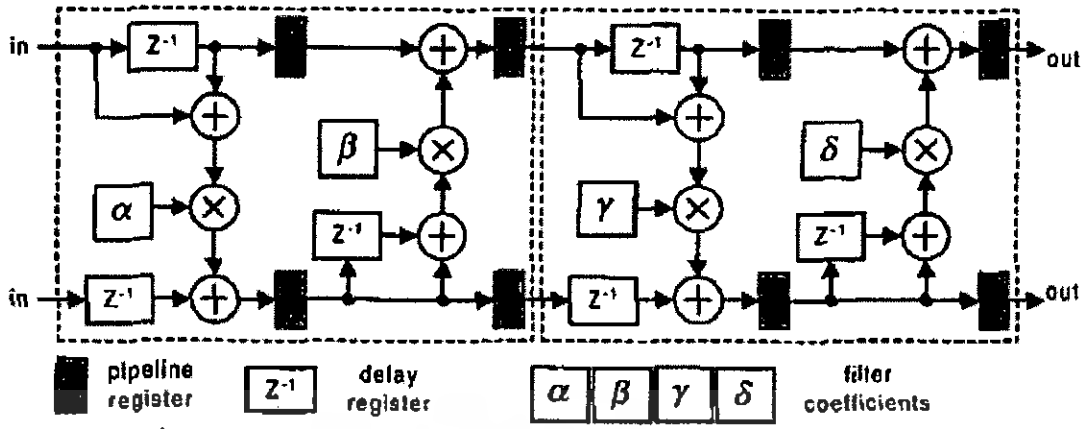
$$H_i(z) = \begin{bmatrix} 1 & H_i(z) \\ 0 & 1 \end{bmatrix}, i \in odd \tag{3.7}$$

$$H_i(z) = \begin{bmatrix} 1 & 0 \\ H_i(z) & 1 \end{bmatrix}, i \in even \tag{3.8}$$

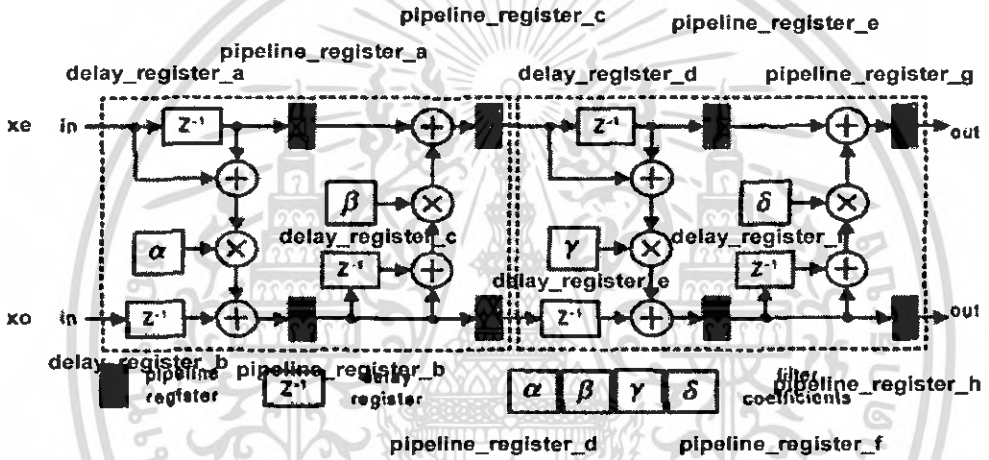
$$K = \begin{bmatrix} h_4^{-1} & 0 \\ 0 & h_5^{-1} \end{bmatrix} \tag{3.9}$$

ตารางที่ 3.1 ค่าสัมประสิทธิ์ 9-7 filter ที่ถูก quantize แล้ว

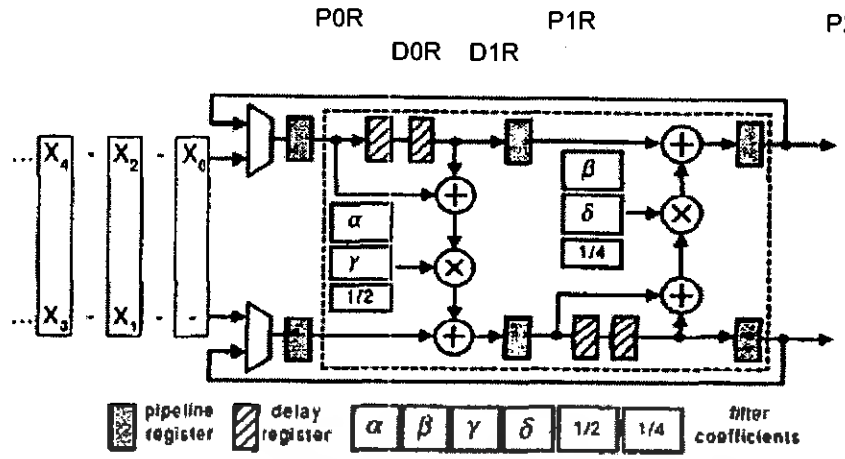
(a)		(b)	
α	-1.5	α	-1.5
β	-0.0625	β	-0.0625
γ	0.7998046875	$1/\gamma$	1.25
δ	0.46875	$\gamma\delta$	0.375
ζ	0.7998046875	ζ	0.80078125
$-1/\zeta$	-1.25030517578125	$-\gamma/\zeta$	-0.9990234375
LS	$\sqrt{2}$	LS	$\sqrt{2}$



รูปที่ 3.5 โครงสร้างการออกแบบ Lifting Discrete Wavelet Transform (DWT)



รูปที่ 3.6 การสังเคราะห์โครงสร้าง Lifting บนฮาร์ดแวร์ (Hardware)



รูปที่ 3.7 การสังเคราะห์โครงสร้าง Lifting บนฮาร์ดแวร์ (Hardware) ออกแบบให้จำนวนเกตลดลง

ตารางที่ 3.2 แสดงค่าสัมประสิทธิ์ของค่า แอลฟา (α) และ แกมมา (γ)

Alfa	Gamma
-1.5861343420693648	0.8829110755411875
- 2^(0)	+ 2^(0)
- 2^(-1)	
- 2^(-3)	- 2^(-3)
+ 2^(-5)	
+ 2^(-7)	+ 2^(-7)

ตารางที่ 3.3 แสดงค่าสัมประสิทธิ์ของค่า เบตา (β) และ เดลตา (δ)

Beta	Delta
-0.0529801185718856	0.4435068520511142
	+ 2^(-1)
- 2^(-4)	- 2^(-4)
+ 2^(-7)	+ 2^(-7)
+ 2^(-9)	- 2^(-9)
- 2^(-12)	+ 2^(-12)

ตารางที่ 3.4 แสดงค่าสัมประสิทธิ์ของค่า 1/K และ K

1/k	K
0.86986445162478	1.1496043988602418
+ 2^(0)	+ 2^(0)
- 2^(-3)	+ 2^(-3)
- 2^(-7)	+ 2^(-6)
+ 2^(-9)	+ 2^(-7)

ตารางที่ 3.5 แสดงการทำงานของ Discrete Wavelet Transform (DWT) 1 เสตท

1-D DWT with RAM		
Write Image data		
initialize 1	0	Set state_RX = 6 Set state_TX = 1 Set state_write = 11 Set address_temp=0 Set state_temp=0 Set state_read = 22
RX	1 5	state_RX
Write	6 10	state_write
Read	11 17	state_read
update address_temp	22	address_temp + 1 if address_temp =1023 then goto state 18
TX	18 21	state_TX
1-DWT		
initialize 2	23	Set state_read = 24 Set address_temp=0 Set address_read=0 Set address_write_a=2^15 Set address_write_b=2^16 Set state_write = 26
Read	11 17	address_temp state_read
9/7 Lifting	24	
Write ye if sel_dwt = 0	25	address_temp<=address_write_a data_temp<=ye Set state_write = 26
Write	6 10	state_write
Write yo if sel_dwt = 0	26	address_temp<=address_write_b data_temp<=yo Set state_write = 27

ตารางที่ 3.5 (ต่อ)

		data_temp<=yo Set state_write = 27
Write	6 10	state_write
update address_####	27	address_write_a<=address_write_a+1 address_write_b<=address_write_b+1 address_read<=address_read+1
Set address_temp for read	28	address_tempa<=read if address_read = end image goto state 29 else goto 11
output data to MatLab		
LED ON	29	Set state_RX = 30 Set state_TX = 1 Set state_read =31 Set address_write_a=2^15 Set address_write_b=2^16
RX	1 5	state_RX
Set read ye	30	address_temp<=address_write_a Set state_read = 31
Read	11 17	state_write
Set read yo	31	address_temp<=address_write_b Set state_read = 32 ye<=data_temp
Read	11 17	state_write
update address_####	32	address_write_a<=address_write_a+1 address_write_b<=address_write_b+1 yo<=data_temp
TX 4 time	18 21	state_TX

ตารางที่ 3.6 แสดงการทำงานของตัวรับ ตัวส่ง (Tx-Rx) ของ Discrete Wavelet Transform (DWT) เมื่อสัญญาณขาเข้าเป็นภาพ (สัญญาณ 2 มิติ)

1-D DWT		
initialize 1	0	Set stete_RX = 6 Set stete_TX = 1
RX 1 time	1	
	5	stete_RX
Split input	6	
9/7 Lifting	7	
TX 4 time	8	
	12	stete_TX

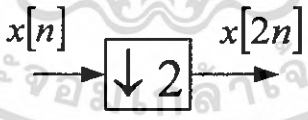
3.2 ส่วนประกอบภายในของฟิลเตอร์

3.2.1 วงจรหน่วงเวลา (Delay)

รูปที่ 3.8 บล็อกไดอะแกรมของวงจรหน่วงเวลา

วงจรหน่วงเวลามีหลักการทำงาน คือ จะทำการเลื่อนเวลาของอินพุตที่เข้ามาให้มีเวลาที่ช้ากว่าเดิมอยู่ 1 หน่วย ดังรูปที่ 3.8

3.2.2 วงจรสุ่มค่าตัวอย่าง (Down-Sampling)



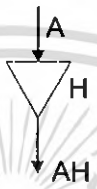
รูปที่ 3.9 บล็อกไดอะแกรมของวงจรสุ่มค่าตัวอย่าง

จากรูปที่ 3.9 จะเห็นได้ว่าวงจรนี้จะทำการลดขนาดของอินพุตลงเหลือครึ่งหนึ่งเท่านั้น

3.2.3 วงจรบวก (Adder)

รูปที่ 3.10 บล็อกไดอะแกรมของวงจรบวก

3.2.4 วงจรคูณ (Multiplier)

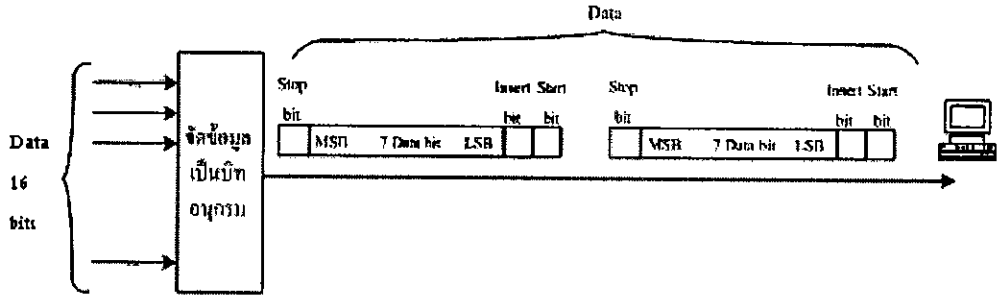


รูปที่ 3.11 บล็อกไดอะแกรมของวงจรคูณ

ในส่วนของวงจรคูณนั้น อินพุตที่เข้ามาจะทำการคูณกับค่าคงที่ที่มีอยู่แล้ว ซึ่งจะได้ผลลัพธ์เป็นผลคูณระหว่างอินพุตกับค่าคงที่

3.3 ส่วนของการแปลงข้อมูลส่งออกพอร์ตอนุกรมจากบอร์ด FPGAไปยังคอมพิวเตอร์

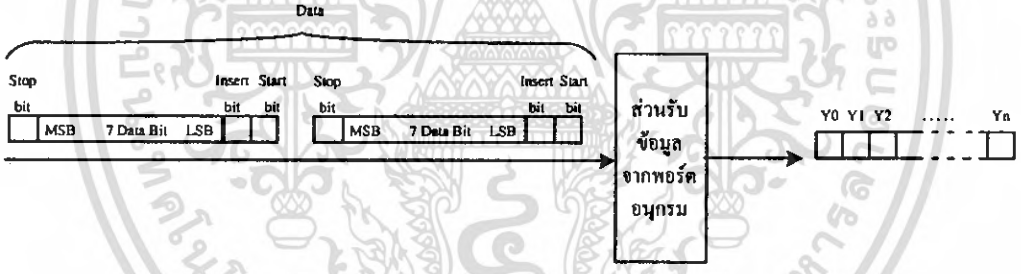
ส่วนของการแปลงข้อมูลส่งออกพอร์ตอนุกรมจากบอร์ด FPGA ไปยังคอมพิวเตอร์ ทำหน้าที่แปลงบิทข้อมูลแบบขนานขนาด 16 บิท ที่ได้จากแปลงเวฟเลตทรานสฟอร์ม ให้เป็นบิทข้อมูลแบบอนุกรม โดยแบ่งเป็น 2 เฟรม โดยแต่ละเฟรมข้อมูลประกอบด้วย สตาร์ทบิท(Start bit) 1 บิท , สตอบบิท(Stop bit) 2 บิท , บิทข้อมูล(Data bit) 7 บิท และแทรกบิท(Insert bit) 1 บิท เพื่อแก้ไขกรณีการส่งบิทบิทข้อมูลที่เป็น 0 ทั้งหมด หลังจากนั้นส่งเฟรมข้อมูลดังกล่าวออกพอร์ตอนุกรม(serial port) ไปยังคอมพิวเตอร์ตามความถี่บอดเรต(Baud Rate) สามารถแสดงบล็อกไดอะแกรมการแปลงข้อมูลส่งออกพอร์ตอนุกรมจากบอร์ด FPGAไปยังคอมพิวเตอร์ดังรูปที่ 3.12



รูปที่ 3.12 แสดงบล็อกไดอะแกรมการแปลงข้อมูลส่งออกพอร์ตอนุกรม
จากบอร์ด FPGA ไปยังคอมพิวเตอร์

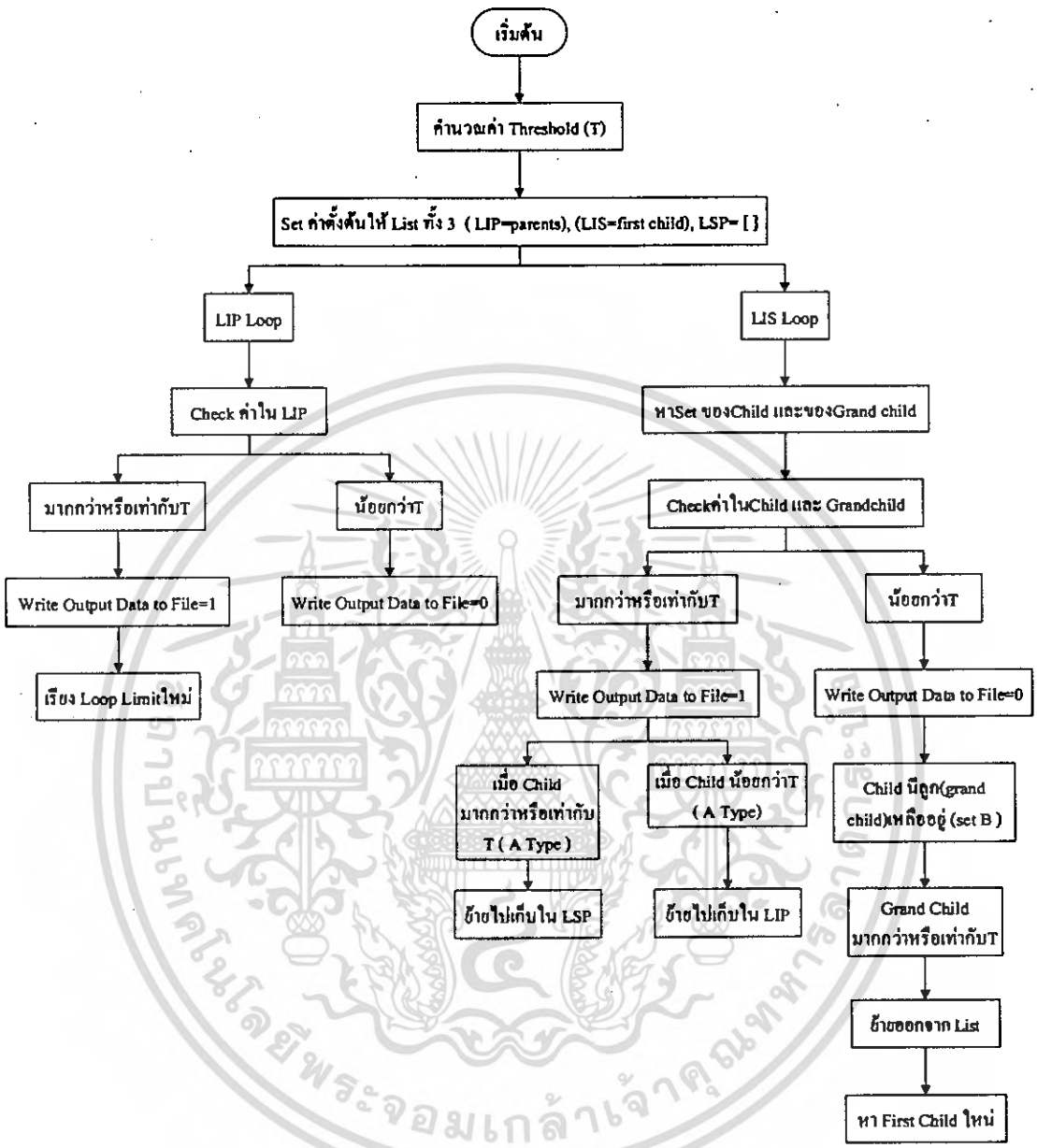
3.3.1 ส่วนของการรับบิทข้อมูลจากบอร์ด FPGA ด้วยโปรแกรม MATLAB ผ่านทางพอร์ตอนุกรม

ส่วนของการรับข้อมูลแบบอนุกรมจากบอร์ต FPGA ด้วยโปรแกรม MATLAB ทำหน้าที่รับเฟรมบิตของข้อมูลจากพอร์ตอนุกรมตามความถี่บอดเรต (Baud Rate) มาทำการตัดสตาร์ทบิท (Start bit) และสตอบบิท (Stop bit) และบิทแทรก (Insert bit) ทิ้ง พร้อมทั้งรวมเฟรมบิทข้อมูลที่แยกกันอยู่เพื่อให้ได้ค่าของข้อมูล 1 ค่า แล้วนำค่าที่ได้เก็บให้ครบทุกค่า สามารถแสดงบล็อกไดอะแกรมการรับบิทข้อมูลผ่านทางพอร์ตอนุกรมได้ดังรูปที่ 3.13

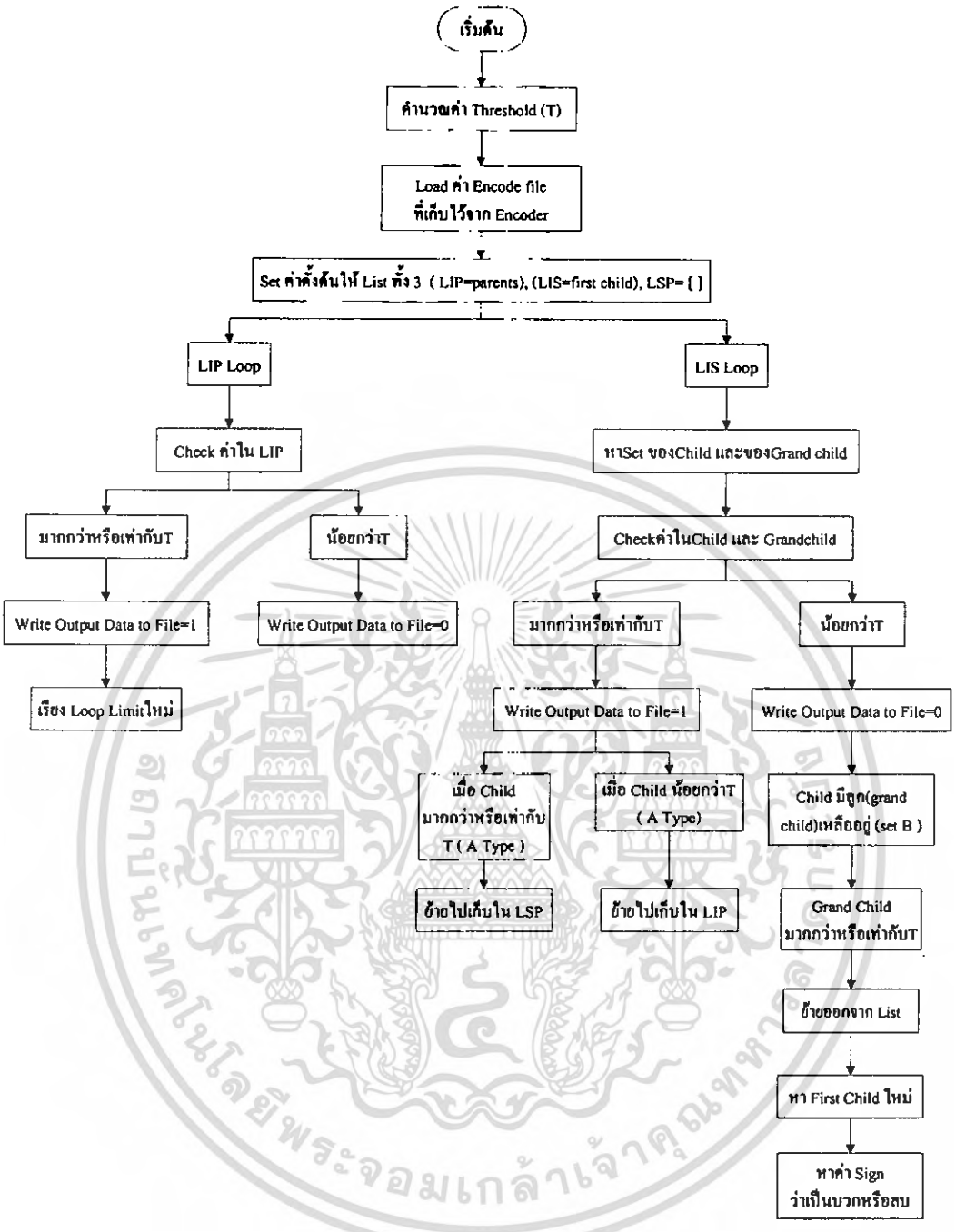


รูปที่ 3.13 แสดงบล็อกไดอะแกรมการรับบิทข้อมูลจากบอร์ด FPGA ผ่านพอร์ตอนุกรม

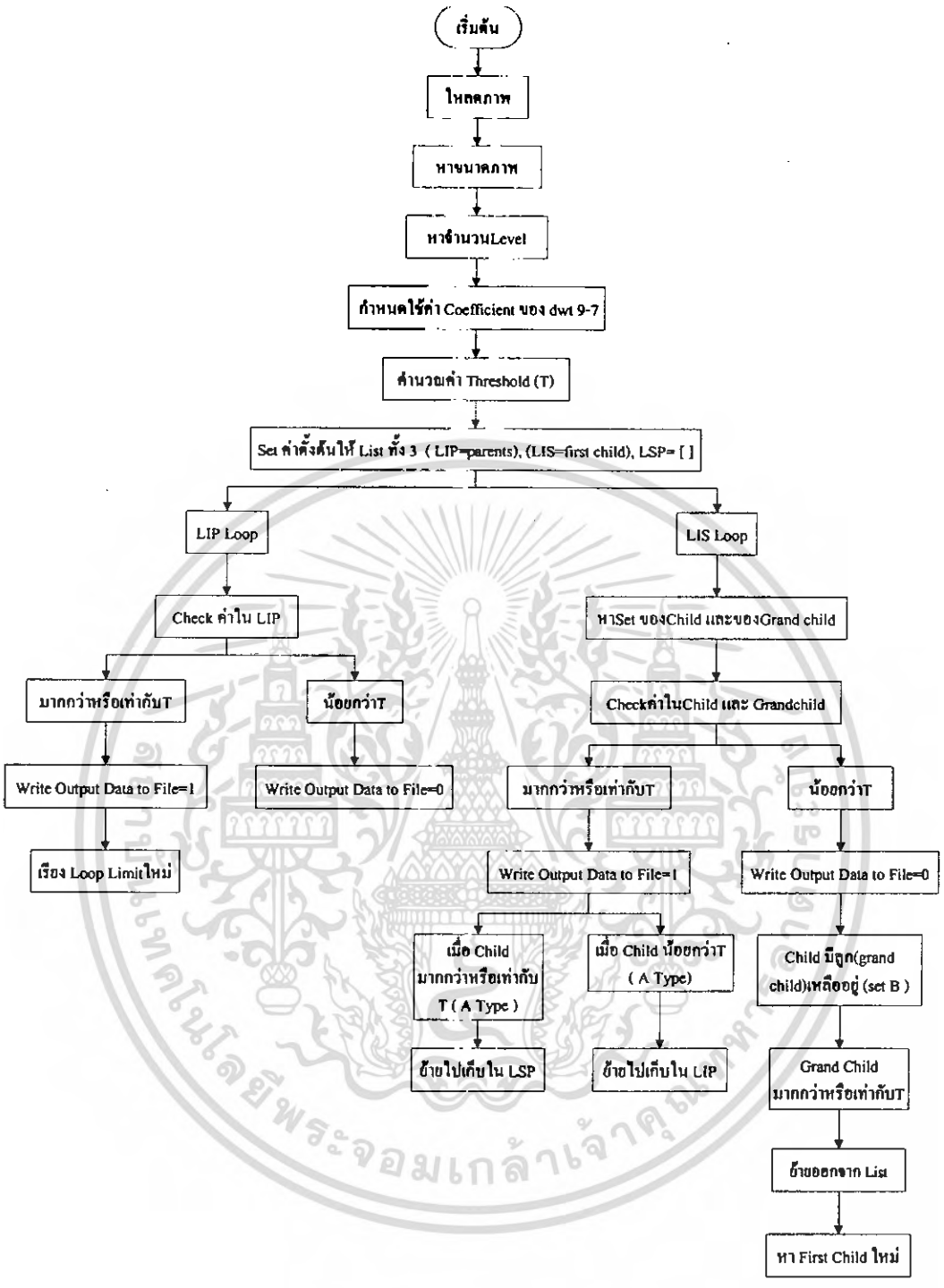
3.4 ส่วนของ บล็อกไดอะแกรม (Block Diagram) ของการสร้าง อัลกอริทึม SPIHT



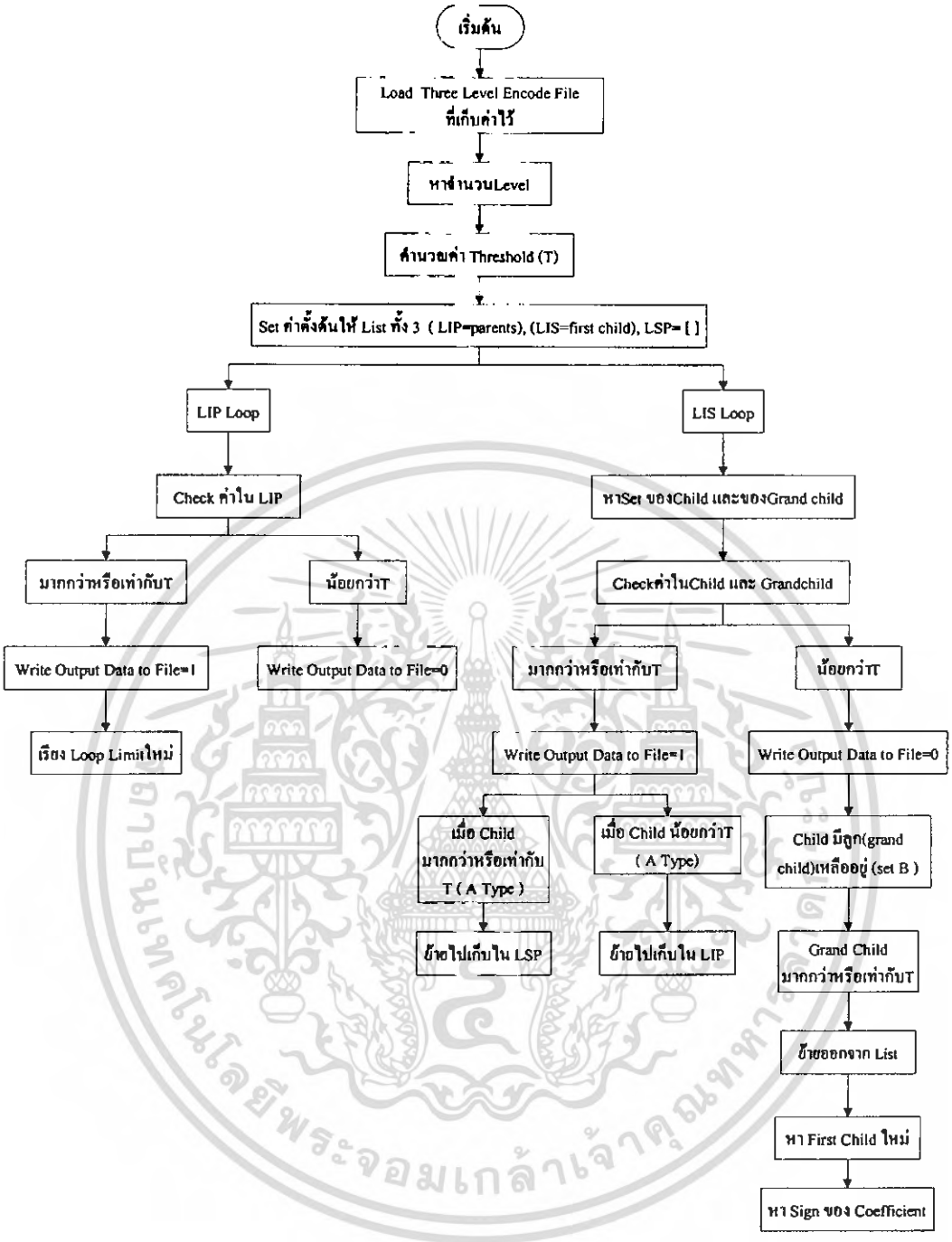
รูปที่ 3.14 บล็อกไดอะแกรม แสดงการทำ SPIHT encode 2 level



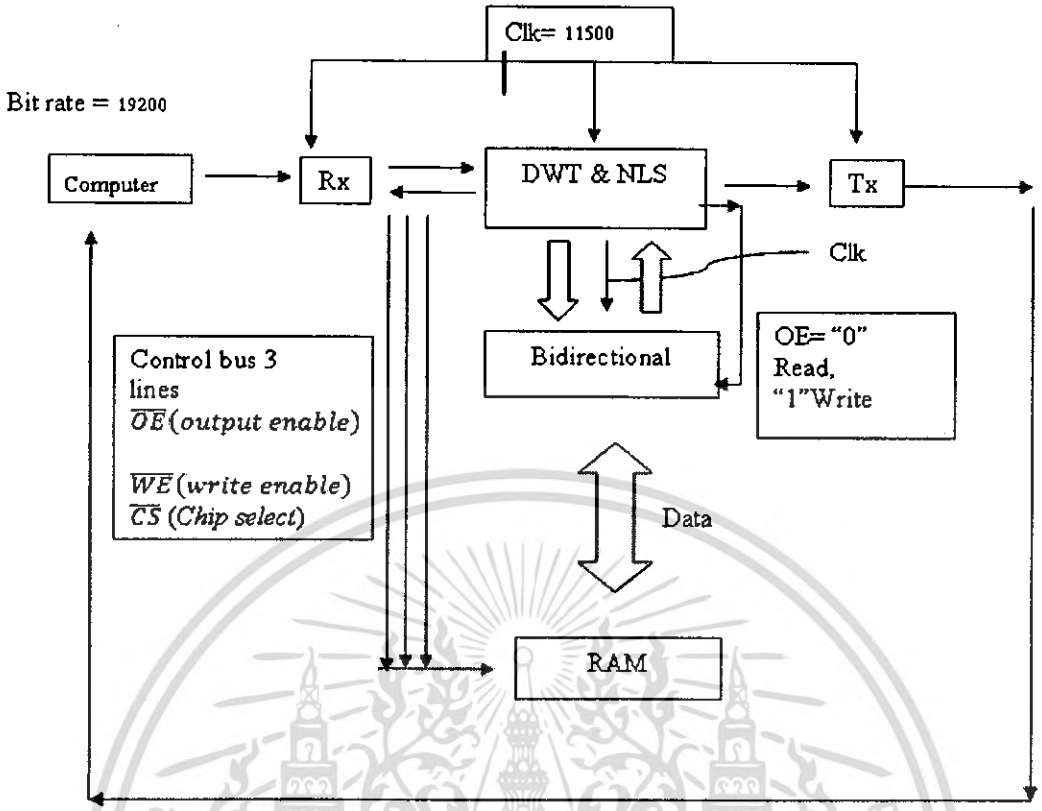
รูปที่ 3.15 บล็อกไดอะแกรม แสดงทำ SPIHT decode 2 level



รูปที่ 3.16 บล็อกไดอะแกรมแสดงการทำ SPIHT encode 3 level เมื่อป้อนสัญญาณ 2 มิติ (ภาพ)



รูปที่ 3.17 บล็อกไดอะแกรม แสดงการทำ SPIHT decode 3 level เมื่อป้อนสัญญาณ 2 มิติ (ภาพ)



รูปที่ 3.18 บล็อกไดอะแกรม ในการสร้างฮาร์แวร์ (Hardware)

บทที่ 4
ผลการทดลอง

4.1 ส่วนของการจำลองโดยใช้โปรแกรม MATLAB

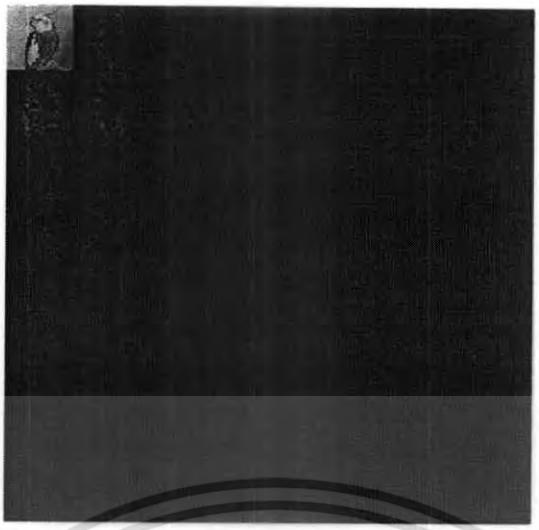
4.1.1 เมื่อสัญญาณภาพ ผ่านการเข้ารหัส (Encode) และ ถอดรหัส (Decode) โดยใช้ อัลกอริทึม (Algorithm) SPIHT 3 ระดับโดยใช้ภาพตัวอย่างต่างๆ ขนาด 256× 256 พิกเซล (pixel)

รูปที่ 4.1 ภาพ "Elaine" ดั้งเดิม (original) ขนาด 256×256 พิกเซล (pixel)

รูปที่ 4.2 ภาพ "Elaine" เมื่อผ่านการเข้ารหัส SPIHT (encode) 3 ระดับ

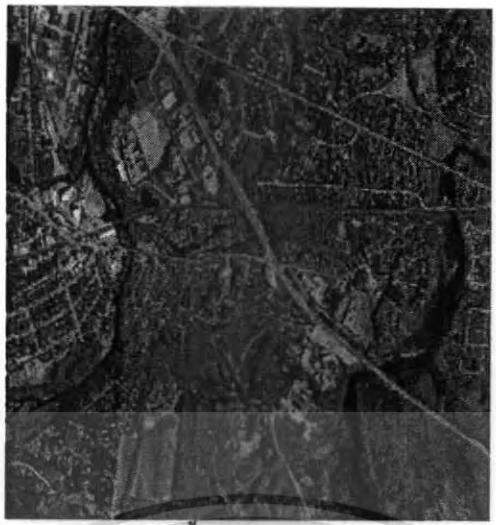
รูปที่ 4.3 ภาพ "Elaine" เมื่อผ่านการถอดรหัส SPIHT (decode) 3 ระดับ

รูปที่ 4.4 ภาพ "Bird" ดั้งเดิม (original) ขนาด 256 × 256 พิกเซล (pixel)



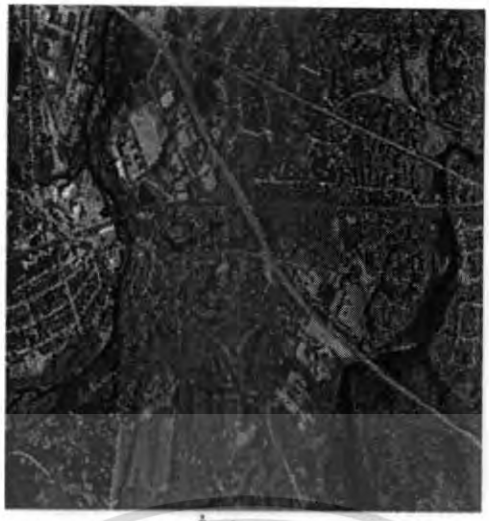
รูปที่ 4.5 ภาพ "Bird" เมื่อผ่านการเข้ารหัส SPIHT (encode) 3 ระดับ

รูปที่ 4.6 ภาพ "Bird" เมื่อผ่านการถอดรหัส SPIHT (decode) 3 ระดับ

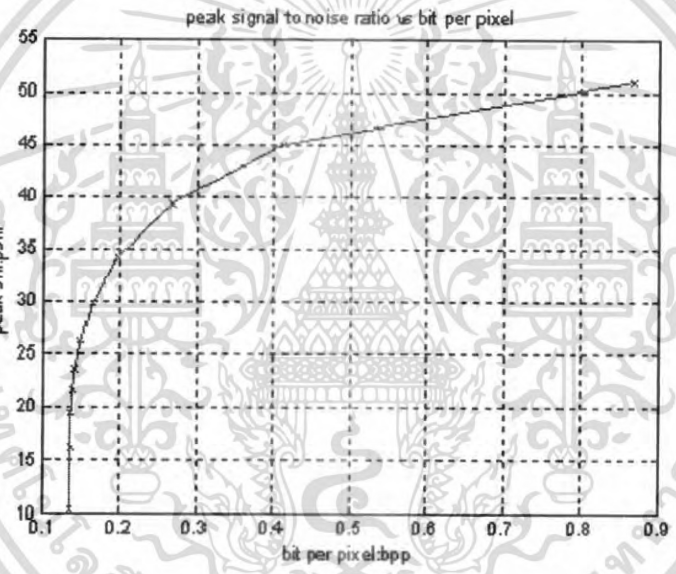


รูปที่ 4.7 ภาพถ่าย SAR "Concord" ดั้งเดิม (original) ขนาด 256×256 พิกเซล (pixel)

รูปที่ 4.8 ภาพถ่าย SAR "Concord" เมื่อผ่านการเข้ารหัส SPIHT (encode) 3 ระดับ

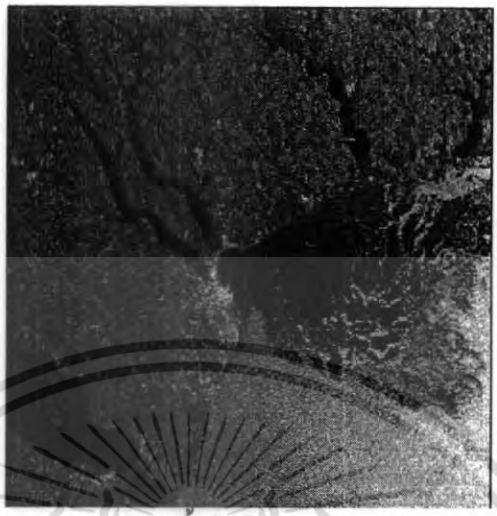


รูปที่ 4.9 ภาพถ่าย SAR "Concord" เมื่อผ่านการถอดรหัส SPIHT (decode) 3 ระดับ



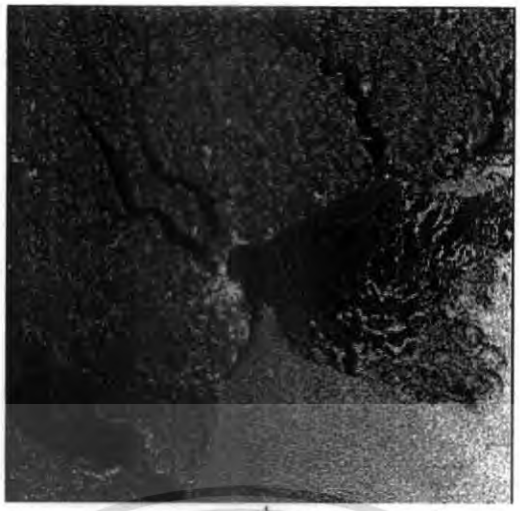
รูปที่ 4.10 กราฟความสัมพันธ์ระหว่าง peak signal to noise ratio และ bit per pixel ของ ภาพถ่าย SAR "Concord"

การเข้ารหัส (Encode) และ การถอดรหัส (Decode) SPIHT 3 ระดับโดยใช้ภาพถ่าย SAR โดยเมื่อพิจารณาผลของ Peak Signal to Noise Ratio (PSNR)

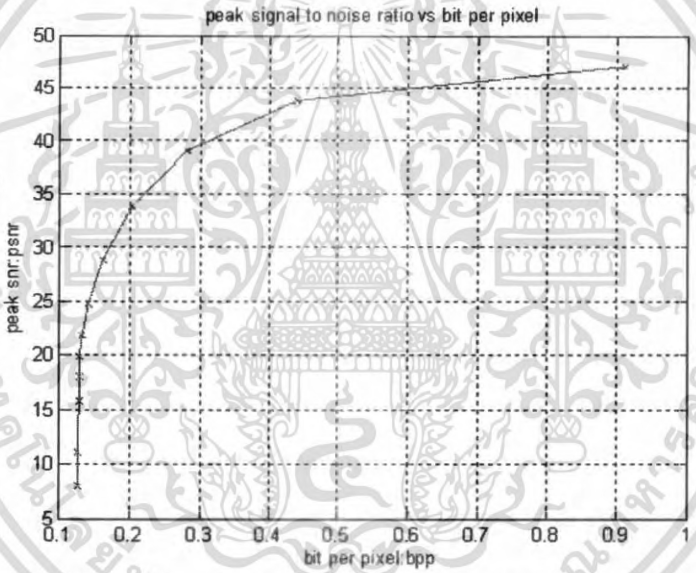


รูปที่ 4.11 ภาพถ่าย SAR "OdessaUkraine" ดั้งเดิม(original) ขนาด 256×256 พิกเซล (pixel)

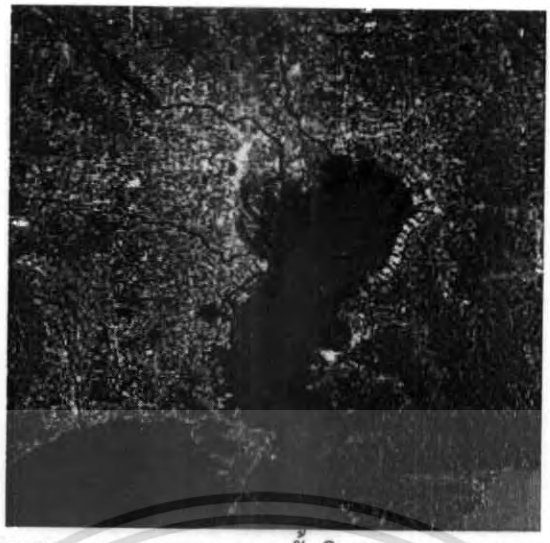
รูปที่ 4.12 ภาพถ่าย SAR "OdessaUkraine" เมื่อผ่านการเข้ารหัส SPIHT encode 3 ระดับ



รูปที่ 4.13 ภาพถ่าย SAR "OdessaUkraine" เมื่อผ่านการถอดรหัส SPIHT decode 3 ระดับ

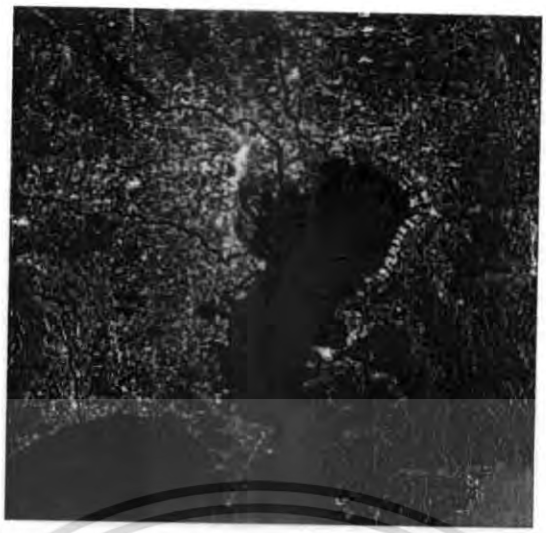


รูปที่ 4.14 กราฟความสัมพันธ์ระหว่าง peak signal to noise ratio และ bit per pixel ของ ภาพถ่าย SAR "OdessaUkraine"

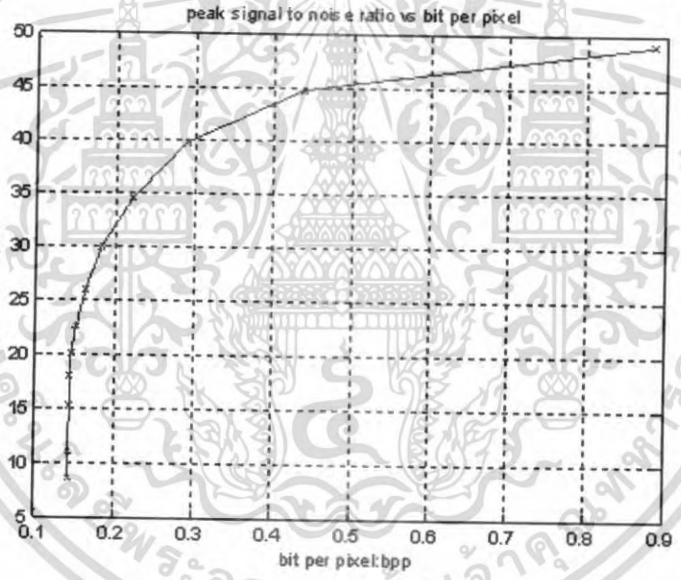


รูปที่ 4.15 ภาพถ่าย SAR "Bay Tokyo Yokohama" ดั้งเดิม (original) ขนาด 256 × 256 พิกเซล (pixel)

รูปที่ 4.16 ภาพถ่าย SAR "Bay Tokyo Yokohama" เมื่อผ่านการเข้ารหัส SPIHT encode 3 ระดับ



รูปที่ 4.17 ภาพถ่าย SAR "Bay Tokyo Yokohama" เมื่อผ่านการถอดรหัส SPIHT decode 3 ระดับ

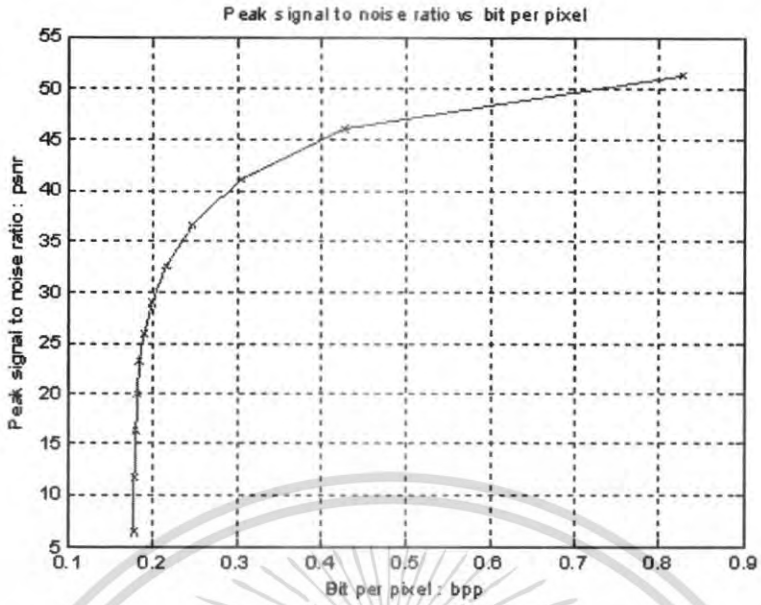


รูปที่ 4.18 กราฟความสัมพันธ์ระหว่าง peak signal to noise ratio และ bit per pixel ของภาพถ่าย SAR "Bay Tokyo Yokohama"

รูปที่ 4.19 ภาพ "lena" ดั้งเดิม (original) ขนาด พิกเซล (pixel)

รูปที่ 4.20 ภาพ "lena"ผ่านการเข้ารหัส SPIHT encode 3 ระดับ

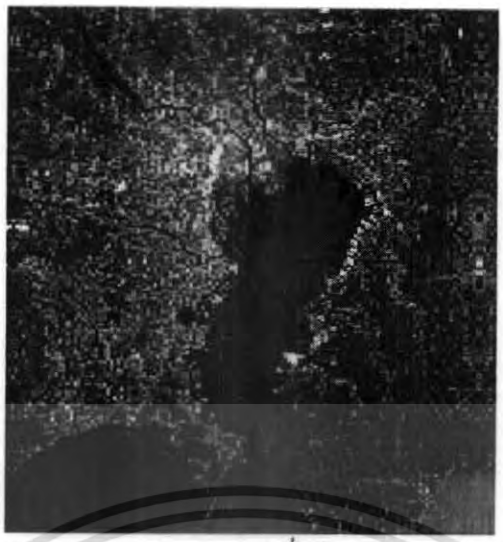
รูปที่ 4.21 ภาพถ่าย "lena" เมื่อผ่านการถอดรหัส SPIHT decode 3 ระดับ



รูปที่ 4.22 กราฟความสัมพันธ์ระหว่าง peak signal to noise ratio และ bit per pixel ของภาพ "lena"

4.1.2 ผลของภาพเมื่อผ่านการแปลง Discrete Wavelet Transform (DWT) โครงสร้าง Lifting ดังรูปที่ 3.7 โดยใช้โปรแกรม MATLAB

รูปที่ 4.23 ภาพถ่าย SAR "Bay Tokyo Yokohama" ดั้งเดิม (original) ขนาด 256×256 พิกเซล (pixel)



รูปที่ 4.24 ภาพถ่าย SAR "Bay Tokyo Yokohama" เมื่อผ่าน Discrete Wavelet Transform (DWT) โครงสร้าง Lifting รูปที่ 3.7 ส่วน Low Pass Filter

รูปที่ 4.25 ภาพถ่าย SAR "Bay Tokyo Yokohama" เมื่อผ่าน Discrete Wavelet Transform (DWT) โครงสร้าง Lifting รูปที่ 3.7 ส่วน High Pass Filter

รูปที่ 4.26 ภาพถ่าย SAR "Westconcord" ดั้งเดิม (original) ขนาด 256×256 พิกเซล (pixel)

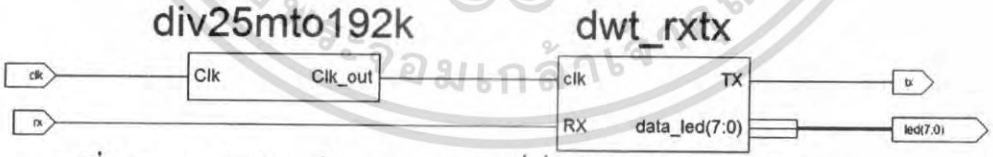
รูปที่ 4.27 ภาพถ่าย SAR "Westconcord" เมื่อผ่าน Discrete Wavelet Transform (DWT) โครงสร้าง Lifting รูปที่ 3.7 ส่วน Low Pass Filter

รูปที่ 4.28 ภาพถ่าย SAR "Westconcord" เมื่อผ่าน Discrete Wavelet Transform (DWT) โครงสร้าง Lifting รูปที่ 3.7 ส่วน High Pass Filter

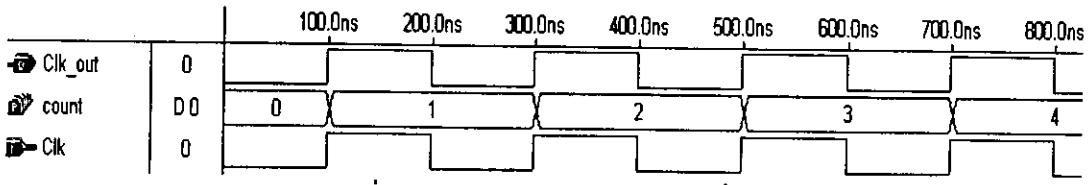
4.2 ส่วนของการจำลองผลโดยใช้ โปรแกรม VHDL

4.2.1 ส่วนของวงจรหารความถี่และ ส่วนรับส่งข้อมูลของ Discrete Wavelet Transform (DWT)

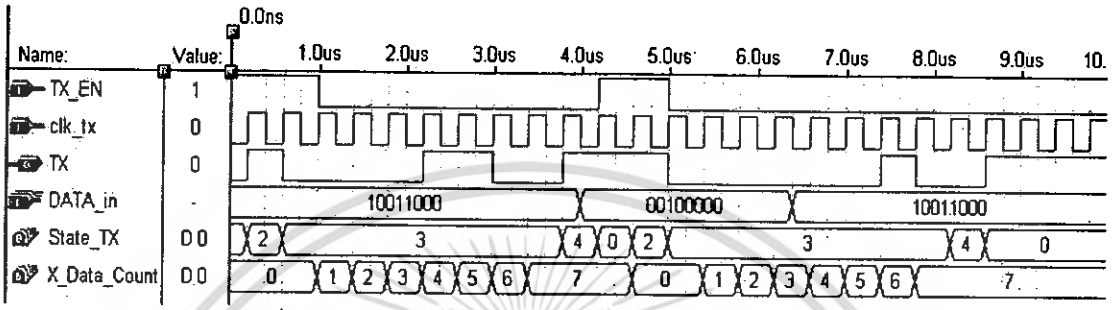
ส่วนของวงจรหารความถี่ ทำหน้าที่หารความถี่ที่ของระบบให้ได้เอาท์พุทเป็นความถี่บอดเรด(Baud Rate) ที่ใช้ในการรับ-ส่งบิทข้อมูลทางพอร์ตอนุกรม โครงงานนี้ใช้แหล่งกำเนิดความถี่ 25 MHz สำหรับใช้เป็นความถี่ของระบบ แต่เราต้องหารความถี่เพื่อให้ได้เท่ากับความถี่บอดเรด 19200 บิทต่อวินาทีสามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรหารความถี่ได้และนำมาต่อ กับ ส่วนรับส่งข้อมูลของ DWT (Discrete Wavelet Transform) โดยใช้ การรับส่งทีละ 16 บิต โดยในส่วนของ DWT ออกแบบตามโครงสร้าง รูปที่ 3.7 ซึ่งเป็น โครงสร้าง Lifting ซึ่งช่วยประหยัดจำนวนเกตลง



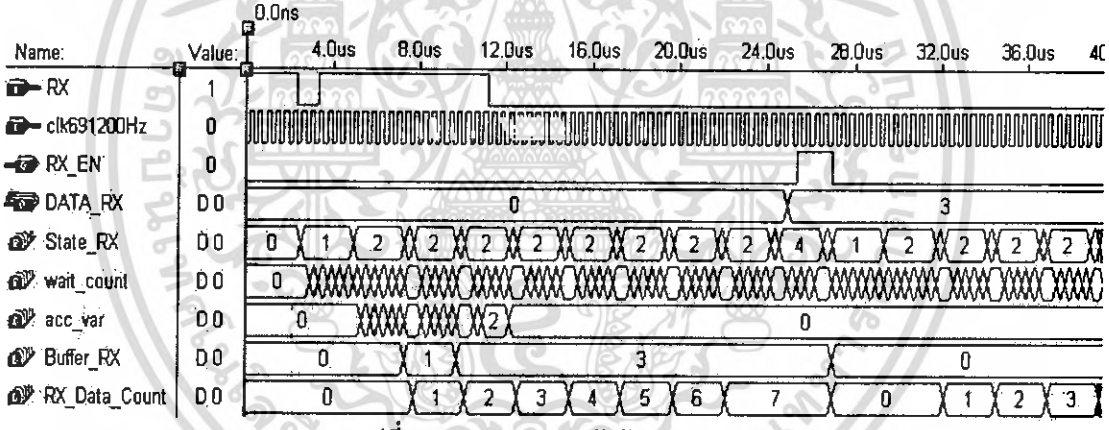
รูปที่ 4.29 แสดงสัญลักษณ์ของวงจรหารความถี่เชื่อมต่อกับส่วนการรับส่งข้อมูลของ dwt



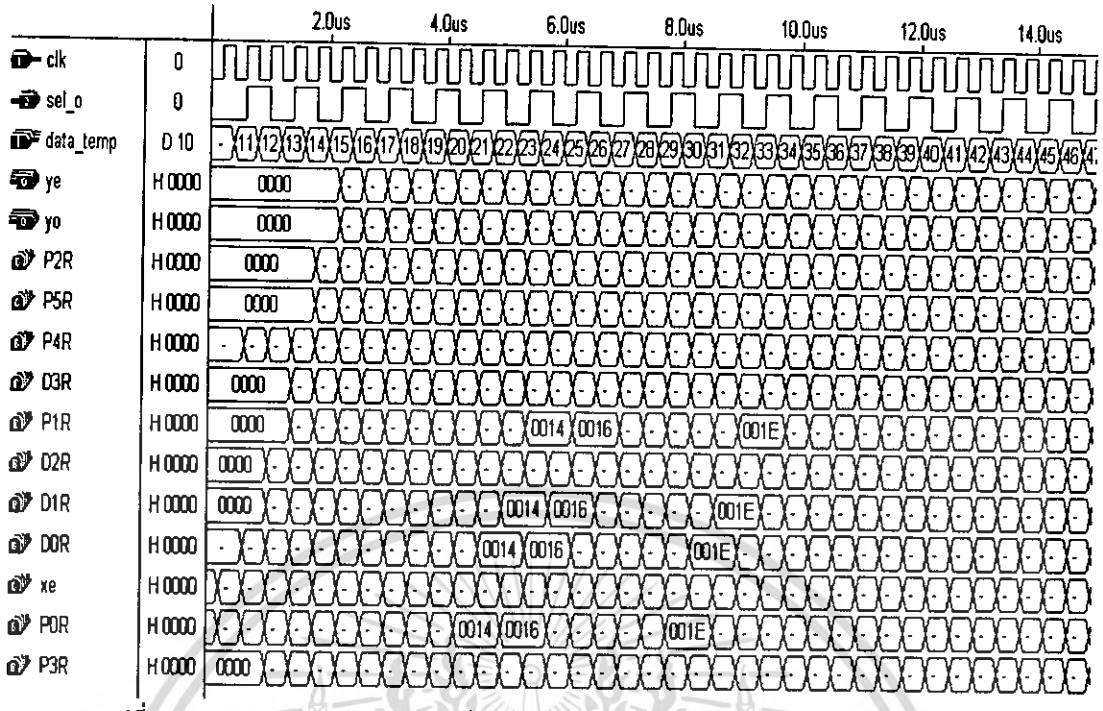
รูปที่ 4.30 แสดงผลของวงจรหารความถี่



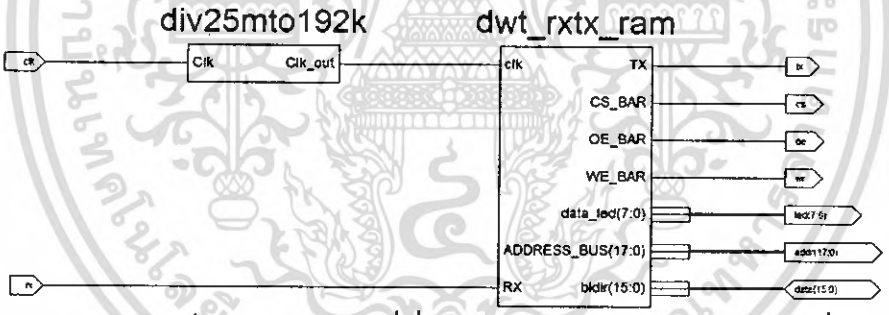
รูปที่ 4.31 แสดงผลของตัวส่ง (Tx) โดยใช้ clock 19200 Hz



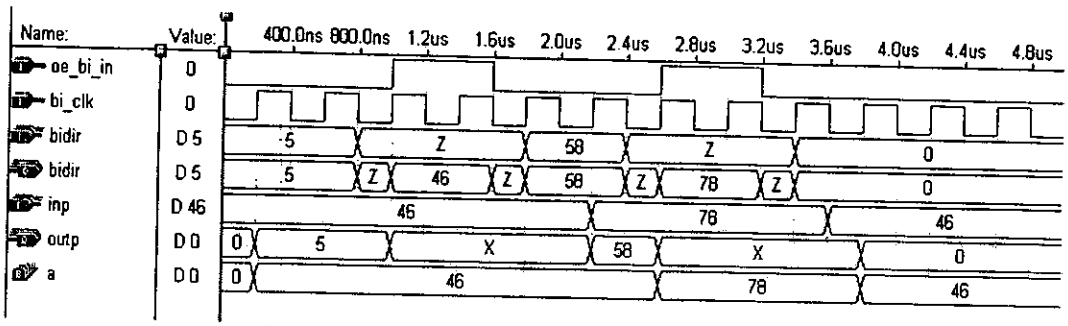
รูปที่ 4.32 แสดงผลของตัวรับ (Rx)



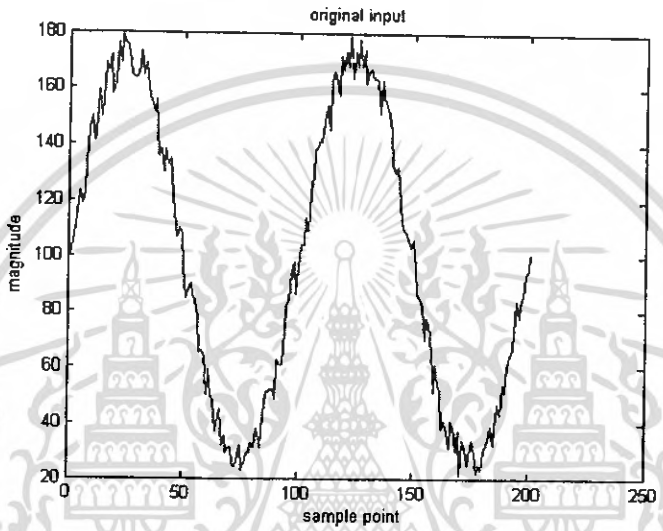
รูปที่ 4.33 แสดงผลของส่วน โครงสร้าง Discrete Wavelet Transform (DWT) 1 ระดับ



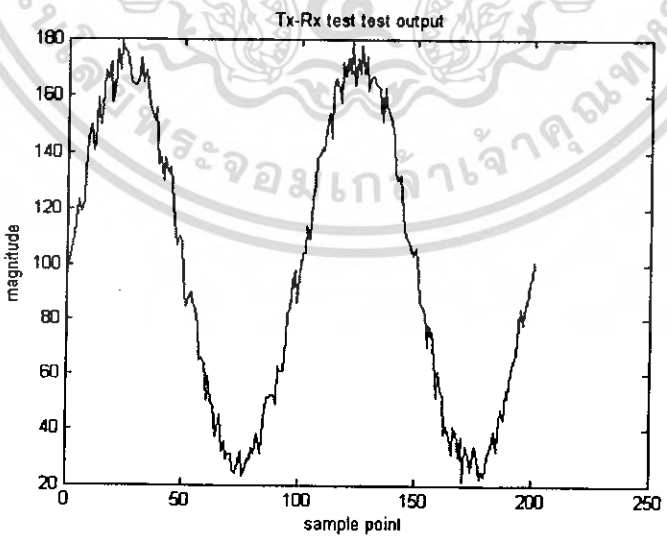
รูปที่ 4.34 แสดงสัญลักษณ์ของวงจรหารความถี่เชื่อมต่อกับส่วนการรับส่งข้อมูลของ dwt ที่ใช้อ่านเขียนแรม (Ram)



รูปที่ 4.35 แสดงผลของ Bidirectional จากโครงสร้างรูปที่ 3.8

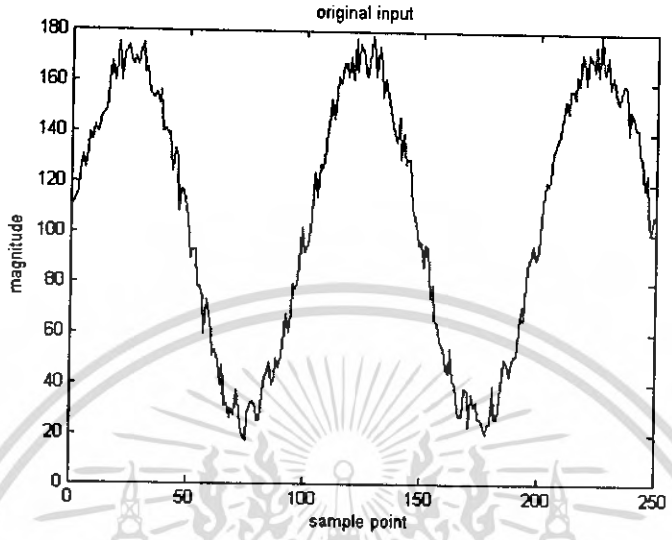


รูปที่ 4.36 แสดงผลของ เมื่อป้อนสัญญาณ sinusoidal ผ่าน Rs-232

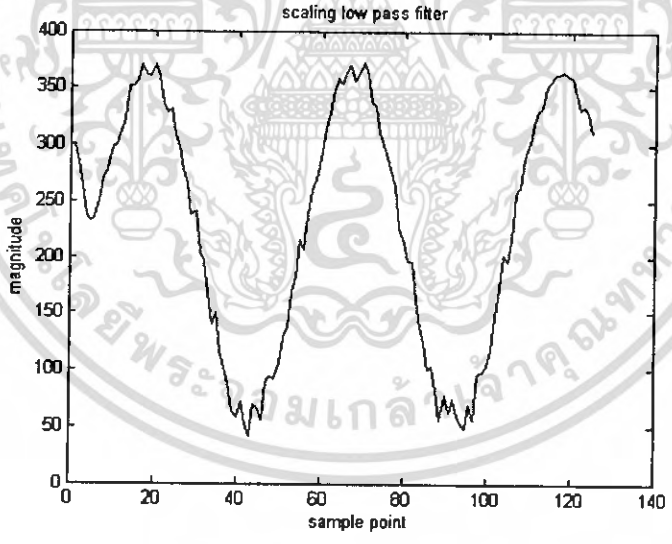


รูปที่ 4.37 แสดงผลสัญญาณ ของตัวรับ ผ่าน Rs-232

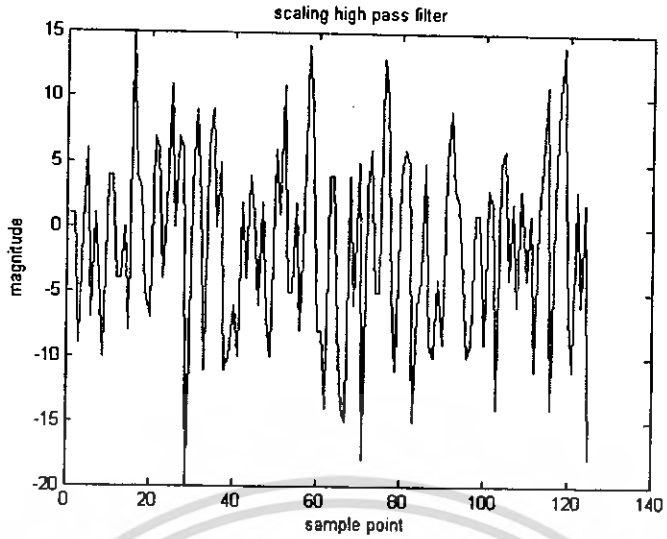
4.2.2 ผลของสัญญาณเมื่อผ่าน Lifting 1 ระดับด้านวิเคราะห์ (Analysis) จากโครงสร้างรูปที่ 3.7 เมื่อ simulate ลงบอร์ด FPGA ผ่าน Rs-232



รูปที่ 4.38 แสดงผลของ แสดงผลของ เมื่อป้อนสัญญาณ sinusoidal ผ่าน Rs-232



รูปที่ 4.39 ผลของสัญญาณเมื่อผ่านโครงสร้าง Lifting ส่วน Low Pass Filter รูปที่ 3.7



รูปที่ 4.40 ผลของสัญญาณเมื่อผ่านโครงสร้าง Lifting ส่วน High Pass Filter รูปที่ 3.7

บทที่ 5
บทวิจารณ์และบทสรุป

ผลการทดลองสำหรับภาพระดับสีเทา (grayscale) ขนาด 256×256 พิกเซล (pixel) ของภาพ SAR และภาพ ต่างๆ เช่น ภาพ "Elaine" ภาพ "Bird" ภาพ "Lena" ดังแสดงในบทที่ 3ใช้ค่า สัมประสิทธิ์ ของ 9/7 ไบออโธกอนอล เวฟเล็ต (Biorthogonal Wavelet) บนมาตรฐานของ JPEG 2000 โดย จากกราฟ แสดงผลระหว่าง Peak Signal to noise Ratio (PSNR) และ Bit per pixel (Bpp) เห็นได้ว่า ค่า Peak Signal to Noise Ratio มีค่าค่อนข้างสูง แสดงว่า error ค่อนข้างน้อย และยังให้ Linear Phase อีกด้วย อัลกอริทึมนี้ สามารถ สร้างบนฮาร์ดแวร์แบบ SPTHT Without List อีกด้วย ซึ่งทำให้ มีการลดจำนวนเกตลงได้อีก ถึง 3 เท่า ดังนั้น SPIHT จึงเป็น อัลกอริทึมที่นิยม ในปัจจุบันในการบีบอัดข้อมูลภาพ ในการพัฒนา อัลกอริทึมนี้ ยังสามารถ นำผลที่ได้มาผ่าน การเข้ารหัสแบบ Arithmatic Code ได้อีกด้วย เพราะจะทำให้ได้ ผล PSNR มีค่าสูงขึ้นอีก ประมาณ 0.2 dB

บรรณานุกรม

[1] Gilbert Strang, Truong Nguyen, *"Wavelets and Filter Banks"*, Wellesley – Cambridge Press, 1996

[2] N.J. Fliege, *"Multirate Digital Signal Processing"*, John Wiley & Sons, 1995

[3] A.Said and William A. Pearlman, *"A New, Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees"* IEEE Trans. Circuit Syst. 243-250, 1996

[4] Frederick W.Wheeler and William A. Pearlman, *"Spiht Image Compression Without Lists"* Acoustic.Speech.and Signal Processing.2000.ICASSP '00.Proceedings.2000 IEEE International Conference on Volume 6, 5-9 June 2000 Page(s):2047-2050

[5] Al Bovik, *"Handbook of Image and Video Processing"*, Academic Press, 2000

[6] Frederick W.Wheeler and William A. Pearlman , *"Low-Memory Packetized Spiht Image Compression"* Signals.Systems and Computers.1999. Conference Record of the Thirty-Third Asilomar Conference on Volume 2, 24-07 Oct.1999 Page(s):1193-1197

[7] Amir Said, *"Example of Application for Image Compression"* , 1999

[8] Thomas W.Fry and Scott A.Hauck, Senior Member, IEEE, *"Spiht Image Compression on FPGAs"* IEEE Transactions on circuits and systems for video technology Volume 15, NO 9, September 2005

[9] Chung- Jr Lian, Ktian-Ftr Chen, Hong- Hui Chen, and Liang- Gee Chen, *"Lifting based discrete Wavelet transform architecture for JPEG2000"* Circuits and Systems.2001.ISCAS 2001 IEEE International Symposium on Volume 2, 6-9 May 2001 Page(s):445-448

[10] Dustin Lennon, *"(Bi)orthogonal Wavelets"* Oct.2005

[11] Satyabrata Rout, *"Orthoganal vs. Biorthogonal Wavelets for Image Compression"* August.2003

[12] Hong LIU, Lin-pei Zhai, Ying Gao, Wen-ming Li, Jiu-fei Zhou, *"Image Compression Based on Biorthogonal Wavelet Transform"*, Proceedinfs of Iscit2005

[13] Sherman D Riemenschneidez, Zuowei Shen, *"Construction of compactly supported"*

[14] Charles D.Crusere, Member, IEEE, and Sanjit K. Mitra, Fellow,IEEE , *"Image Coding Using Wavelets Based on Perfect Reconstruction IIR Filter Bank"* IEEE Transactions on circuits and systems for video technology Vol 6, No 5, October 1996

[15] Wim Sweldens, *"The Lifting Scheme A New Philosophy in Biorthogonal Wavelet Constructions"*

[16] สุขสันต์ จิรเชวง และ วุฒิพงศ์ อารีกุล, *"การปรับปรุงการบีบอัดภาพวิธี SPIHT โดยใช้การเข้ารหัสบล็อกสำหรับภาพลายเส้นคม"*, การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่24, สจล, หน้า1214-1219, November 2001

[17] ชำนาญ ปัญญาใส และ วัชรากร หนูทอง, *"ภาษาVHDL สำหรับการออกแบบวงจรดิจิตอล"*, ซีเอ็ดยูเคชั่น, กรุงเทพฯ, 2004

ภาคผนวก

โปรแกรม SPIHT_encode_3 level with header

```
clear all,clc
format short
tic

x=imread('elaine_64.tif');

x=double(x);
I=x;

im_size=length(x);
mat=zeros(size(x));

dwt_level=3;

lev=log2(im_size)-dwt_level; %frist level

   if lev < 1
      break
   end;

%%%% 2D-DWT %%%%
dwtmode('per','nodisp');

mapping_index=im_size;

 for ii=1:dwt_level
    [a b c d]=dwt2(I,'bior3.7');
    mat(1:2^(log2(mapping_index)-1),1:2^(log2(mapping_index)-1))=a;
    mat(1:2^(log2(mapping_index)-1),2^(log2(mapping_index)-1)+1:2^log2(mapping_index))=b;
    mat(2^(log2(mapping_index)-1)+1:2^log2(mapping_index),1:2^(log2(mapping_index)-1))=c;
    mat(2^(log2(mapping_index)-1)+1:2^log2(mapping_index),2^(log2(mapping_index)-
1)+1:2^log2(mapping_index))=d;
```

```
  I=a;
  mapping_index=mapping_index/2;
end;

clear mapping_index

%figure
%imshow(mat2gray(mat));

clear a;  clear b;   clear c;  clear d;
clear x;  clear I;

T=2^fix(log2(max(abs(mat(:)))));  %Th

LIP=[]; LIS=[]; % 0*0
OP=[]; % Output
stop_run=0;

fid = fopen('three_level_output_encode.txt','a'); % Open outout file for write.

fprintf(fid,'%d\n',dwt_level); % Write DWT-level data to file.
fprintf(fid,'%d\n',im_size); % Write image size to file.
fprintf(fid,'%d\n',T); % Write the Th to file.

% ========== Initialization ========== %

  %LIP(row,colum)
  for m=1:2^lev
     for n=1:2^lev
        LIP=[LIP ;[m,n]];
     end
  end

  %% LIS(row,colum,type{A=0,B=1})
```

```
%% Keep only first child of (i,j) in LIP for addressing mode
  for m=1:2:2^lev
    for n=(2^lev)+1:2:(2^(lev+1))
      LIS=[LIS ;[m,n,0]];
    end
  end

  for m=(2^lev)+1:2:(2^(lev+1))
    for n=1:2:2^lev
      LIS=[LIS ;[m,n,0]];
    end
  end

  for m=(2^lev)+1:2:(2^(lev+1))
    for n=(2^lev)+1:2:(2^(lev+1))
      LIS=[LIS ;[m,n,0]];
    end
  end
      LSP=[];
% ================================================================

 %!!!!! Loop here for Bit Per Pixel

%  while bits_expected >= stop_run ; %stop running when bpp is ok

 while T>=1; % Stop when refinement all
        % This mean that you run losses compression
 T
 %for irk=1:6; % for test stop running

% ================== Sorting Pass ================================ %

% ################## LIP Loop ####################
```

```
temp_size=size(LIP); tem=temp_size(1); clear temp_size;
ii=1;
while ii<=tem

   temp=LIP(ii,:);
   temp_a=temp(1);
   temp_b=temp(2);

    if abs(mat(temp_a,temp_b))>=T % To test significant pixel in LIP
      OP=1;
      fprintf(fid,'%d\n',OP); % Write outout data to file.

      LSP=[LSP ; LIP(ii,:)];   % If 'significant' move (i,j) to LSP

      % Removing 'significant' address from LIP
      LIP = [LIP(1:ii-1,:) ; LIP(ii+1:end,:)];

      %%% Output sign %%%
      if mat(temp_a,temp_b)>=0
       OP=0; %%% positive = 0
       fprintf(fid,'%d\n',OP); % Write output data to file.
      else
        OP=1; %%% negative = 1
        fprintf(fid,'%d\n',OP); % Write output data to file.
      end
      %%%%%%%%%%%%%%%%%%%

    else
      OP=0;
      fprintf(fid,'%d\n',OP); % Write output data to file.
      ii=ii+1;
    end

    temp_size=size(LIP);  % Rearrange loop limit number,
```

```
    tem=temp_size(1);    % because address number has changed
    clear temp_size;

  end

% ################ End of LIP Loop ################

% ################## LIS Loop ####################

 child=[]; child_addr=[]; g_child=[];
 check_sig=[];

 temp_size=size(LIS);  tem=temp_size(1); clear temp_size;

 ii=1;
 while ii<=tem
   temp=LIS(ii,:);
   temp_a=temp(1);
   temp_b=temp(2);
   temp_c=temp(3);

   %^^^^^^^^^^ A-type ^^^^^^^^^^^^%

   if temp_c==0  %if 2

     child=[mat(temp_a,temp_b);...     % Find the set of child
          mat(temp_a,temp_b+1);...
          mat(temp_a+1,temp_b);...
          mat(temp_a+1,temp_b+1)];

     child_addr=[[temp_a,temp_b];...   % Find the set of child address
           [temp_a,temp_b+1];...
           [temp_a+1,temp_b];...
           [temp_a+1,temp_b+1]];
```

```
%  check_sig=(abs(child)>=T) % To test significant pixel

      if  temp_a <(2^(log2(im_size)-1))+1 & temp_b <(2^(log2(im_size)-1))+1 % Checking that (i,j)
in LIP have
                                                % grand child or not
       % Find the set of child
       g_child=[ mat(((temp_a-1)*2)+1,((temp_b-1)*2)+1);...
             mat(((temp_a-1)*2)+1,((temp_b-1)*2)+2);...
             mat(((temp_a-1)*2)+1,((temp_b-1)*2)+3);...
             mat(((temp_a-1)*2)+1,((temp_b-1)*2)+4);...

             mat(((temp_a-1)*2)+2,((temp_b-1)*2)+1);...
             mat(((temp_a-1)*2)+2,((temp_b-1)*2)+2);...
             mat(((temp_a-1)*2)+2,((temp_b-1)*2)+3);...
             mat(((temp_a-1)*2)+2,((temp_b-1)*2)+4);...

             mat(((temp_a-1)*2)+3,((temp_b-1)*2)+1);...
             mat(((temp_a-1)*2)+3,((temp_b-1)*2)+2);...
             mat(((temp_a-1)*2)+3,((temp_b-1)*2)+3);...
             mat(((temp_a-1)*2)+3,((temp_b-1)*2)+4);...

             mat(((temp_a-1)*2)+4,((temp_b-1)*2)+1);...
             mat(((temp_a-1)*2)+4,((temp_b-1)*2)+2);...
             mat(((temp_a-1)*2)+4,((temp_b-1)*2)+3);...
             mat(((temp_a-1)*2)+4,((temp_b-1)*2)+4)];
       else
        g_child=0; % Setup of grand child as zero,
                % because (i,j) in LIP don't have grand child
       end;

     if max([max(abs(g_child)) max(abs(child))])>=T    % To test significant set

       OP=1;
       fprintf(fid,'%d\n',OP); % Write output data to file.
```

```
  check_sig=(abs(child)>=T); % To test significant pixel
for hj=1:length(check_sig)
  if check_sig(hj)==1
    OP=1;
    fprintf(fid,'%d\n',OP); % Write output data to file.

    % Moving the child that is significant pixel to LSP
    LSP=[LSP; child_addr(hj,:)];

    %%% Output sign %%%
     if child(hj) >= 0
       OP=0;
       fprintf(fid,'%d\n',OP); % Write outout data to file.
     else
       OP=1;
       fprintf(fid,'%d\n',OP); % Write outout data to file.
     end;
    %%%%%%%%%%%%%%%%%%%%%%

  else
    OP=0;
    fprintf(fid,'%d\n',OP); % Write output data to file.

    % Moving the child that is significant pixel to LIP
    LIP=[LIP; child_addr(hj,:)];
  end;
end;

 if  temp_a <(2^(log2(im_size)-1))+1 & temp_b <(2^(log2(im_size)-1))+1

    % If (i,j) has child, move (i,j) to end of LIS as B-type
    LIS(ii,3) = 1;
    LIS = [LIS ; LIS(ii,:)];
      % Removing (i,j) as A-type from LIS
```

```
            LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

        else
            % If (i,j) don't has child, remove (i,j) from LIS
            LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

        end;
    else
      OP=0;
      fprintf(fid,'%d\n',OP); % Write outout data to file.
      ii=ii+1;
    end;  % if I

end; % if 2


%^^^^^^^^^^^^^^End of A-type^^^^^^^^^^^^^^

%^0^_^0^_^0^_^0^_^0^_^0^_^ B-type ^0^_^0^_^0^_^0^_^0^_^0^_^

if temp_c==1
  % Find the set of grand child
  % !!! No need Checking, because we have checked at A-type loop already
  g_child=[ mat(((temp_a-1)*2)+1,((temp_b-1)*2)+1);...
          mat(((temp_a-1)*2)+1,((temp_b-1)*2)+2);...
          mat(((temp_a-1)*2)+1,((temp_b-1)*2)+3);...
          mat(((temp_a-1)*2)+1,((temp_b-1)*2)+4);...

          mat(((temp_a-1)*2)+2,((temp_b-1)*2)+1);...
          mat(((temp_a-1)*2)+2,((temp_b-1)*2)+2);...
          mat(((temp_a-1)*2)+2,((temp_b-1)*2)+3);...
          mat(((temp_a-1)*2)+2,((temp_b-1)*2)+4);...

          mat(((temp_a-1)*2)+3,((temp_b-1)*2)+1);...
```

```
mat(((temp_a-1)*2)+3,((temp_b-1)*2)+2);...
 mat(((temp_a-1)*2)+3,((temp_b-1)*2)+3);...
 mat(((temp_a-1)*2)+3,((temp_b-1)*2)+4);...

 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+1);...
 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+2);...
 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+3);...
 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+4)];

if max(abs(g_child))>=T

 OP=1;
 fprintf(fid,'%d\n',OP); % Write outout data to file.

 % Removing (i,j) from LIS
 LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

 % Add new frist child to LIS as A-type
 LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+1,0]];
 LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+3,0]];
 LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+1,0]];
 LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+3,0]];

else

 OP=0;
 fprintf(fid,'%d\n',OP); % Write output data to file.
 ii=ii+1;
end;

end;

temp_size=size(LIS);  % Rearrange loop limit number,
tem=temp_size(1);     % because address number has changed
```

```
    clear temp_size;

  end; % while

  %^0^_^0^_^0^_^0^_^0^_^ End of B-type ^0^_^0^_^0^_^0^_^0^_^0^_^


% ################ End of LIS Loop ################

% ============== End of Sorting Pass ==============================%


% ============== Refinement Pass ============================== %

    temp_size=size(LSP);
    tem=temp_size(1);
    clear temp_size;

      Th_temp=dec2binvec(T);
      length_temp=length(Th_temp);

    for ii=1:tem
        temp=LSP(ii,:);
      temp_a=temp(1);
      temp_b=temp(2);
          mat_temp=dec2binvec(abs(mat(temp_a,temp_b)),2^fix(log2(max(abs(mat(:))))));
      OP=mat_temp(length_temp);
      fprintf(fid,'%d\n',OP); % Write outout data to file.
       end;

% ============== End of Refinement Pass ============================== %
  T=T/2; %  Update T
 %  Go to Sorting Pass
 %  Stop when we have exact 'bit per pixel'
```

```
  end; %!!!!! end Loop here for Bit Per Pixel

  fclose(fid);
 toc
end;
```

โปรแกรม SPIHT_Decode 3 level with header

```
clear all,clc
format short
tic

IP=load('three_level_output_encode.txt');

dwt_level=IP(1);
IP=IP(2:end);

im_size=IP(1);
IP=IP(2:end);

mat=zeros(im_size,im_size);
mat_sign=mat;

lev=log2(im_size)-dwt_level; %frist level

  if lev < 1
     break
  end;

T=IP(1); %Th
IP=IP(2:end);

LIP=[]; LIS=[]; % 0*0

% import encoded data here
```

```
% ========== Initialization ========== %

 %LIP(row,colum)
 for m=1:2^lev
    for n=1:2^lev
       LIP=[LIP ;[m,n]];
    end
 end

 %% LIS(row,colum,type{A=0,B=1})
 %% Keep only frist child of (i,j) in LIP for addressing mode
  for m=1:2:2^lev
    for n=(2^lev)+1:2:(2^(lev+1))
       LIS=[LIS ;[m,n,0]];
    end
  end

  for m=(2^lev)+1:2:(2^(lev+1))
    for n=1:2:2^lev
       LIS=[LIS ;[m,n,0]];
    end
  end

  for m=(2^lev)+1:2:(2^(lev+1))
    for n=(2^lev)+1:2:(2^(lev+1))
       LIS=[LIS ;[m,n,0]];
    end
  end
       LSP=[];
% ================================================

 %!!!!! Loop here for Bit Per Pixel
```

```
%!!!!!  while length(IP)>0 ; stop running when bpp is ok

while T>=1;

%for irk=1:6  ; %for stop running

T
% ================== Sorting Pass ==================================%

% ################### LIP Loop ######################
 temp_size=size(LIP);  tem=temp_size(1);  clear temp_size;
 ii=1;
while ii<=tem

   temp=LIP(ii,:);
   temp_a=temp(1);
   temp_b=temp(2);

     if IP(1)==1;
      IP=IP(2:end);

      LSP=[LSP ; LIP(ii,:)];    % If 'significant' move (i,j) to LSP

     %%% Input sign %%%
      mat_sign(temp_a,temp_b)=IP(1);
      IP=IP(2:end);
      %%%%%%%%%%%%%%%%%%%%

      % Removing 'significant' address from LIP
      LIP = [LIP(1:ii-1,:) ; LIP(ii+1:end,:)];

    else
      IP=IP(2:end);
      ii=ii+1;
```

```
    end

    temp_size=size(LIP);  % Rearrange loop limit number,
    tem=temp_size(1);     % because address number has changed
    clear temp_size;
  end

% ################ End of LIP Loop ################

% ################## LIS Loop ####################

 child=[]; child_addr=[]; g_child=[];
 check_sig=[];

 temp_size=size(LIS);  tem=temp_size(1); clear temp_size;

 ii=1;
 while ii<=tem
   temp=LIS(ii,:);
   temp_a=temp(1);
   temp_b=temp(2);
   temp_c=temp(3);

   %^^^^^^^^^^ A-type ^^^^^^^^^^^^^%

   if temp_c==0   %if 2

     if IP(1)==1    % To test significant set
       IP=IP(2:end);

       child_addr=[[temp_a,temp_b];...   % Find the set of child address
             [temp_a,temp_b+1];...
             [temp_a+1,temp_b];...
             [temp_a+1,temp_b+1]];
```

```
for hj=1:4
  if IP(1)==1
    IP=IP(2:end);
    % Moving the child that is significant pixel to LSP
    LSP=[LSP; child_addr(hj,:)];

    %%% Input sign %%%
    tb=child_addr(hj,:);
    mat_sign(tb(1),tb(2))= IP(1);
    IP=IP(2:end);
    %%%%%%%%%%%%%%%%%%%%

  else
    % Moving the child that is significant pixel to LIP
    LIP=[LIP; child_addr(hj,:)];
    IP=IP(2:end);
  end;
end;

  if   temp_a <(2^(log2(im_size)-1))+1 & temp_b <(2^(log2(im_size)-1))+1

    % If (i,j) has child, move (i,j) to end of LIS as B-type
    LIS(ii,3) = 1;
    LIS = [LIS ; LIS(ii,:)];

    % Removing (i,j) as A-type from LIS
    LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];
  else
    % If (i,j) don't has child, remove (i,j) from LIS
    LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];
  end;

else
```

```
    IP=IP(2:end);
    ii=ii+1;
  end;  % if 1

end; % if 2


%^^^^^^^^^^^^^^^End of A-type^^^^^^^^^^^^^^^

%^0^_^0^_^0^_^0^_^0^_^0^_^ B-type ^0^_^0^_^0^_^0^_^0^_^0^_^

if temp_c==1

      if IP(1)==1
        IP=IP(2:end);
       % Removing (i,j) from LIS
       LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

       % Add new frist child to LIS as A-type
       LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+1,0]];
       LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+3,0]];
       LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+1,0]];
       LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+3,0]];

      else

        IP=IP(2:end);
        ii=ii+1;
      end;

 end;

 temp_size=size(LIS);  % Rearrange loop limit number,
 tem=temp_size(1);     % because address number has changed
```

```
    clear temp_size;

  end; % while

  %^0^_^0^_^0^_^0^_^0^_^ End of B-type ^0^_^0^_^0^_^0^_^0^_^0^_^


% ################ End of LIS Loop ################

% ============== End of Sorting Pass ============================================ %


% ================= Refinement Pass ============================================ %
    temp_size=size(LSP);
    tem=temp_size(1);
    clear temp_size;
%     temp_Refinement=

      Th_temp=dec2binvec(T);
      length_temp=length(Th_temp);

    for ii=1:tem
          temp=LSP(ii,:);
      temp_a=temp(1);
      temp_b=temp(2);
      mat(temp_a,temp_b)=mat(temp_a,temp_b)+ (IP(1)*T);
      IP=IP(2:end);
        end;


% ================ End of Refinement Pass ======================================= %
  T
  T=T/2; %  Update T
```

```
  % Go to Sorting Pass
  % Stop when we have exact 'bit per pixel'

  end; %!!!!! end Loop here for Bit Per Pixel

% delete('output_encode.txt')

mat_sign=mat_sign.*(-2);
mat_sign=mat_sign+1;
mat_a=mat_sign.*mat;
xx=[];

figure
imshow(mat2gray(mat_a));

dwtmode('per','nodisp');

for ii=1:dwt_level

   if ii==1
     xx=idwt2(mat_a(1:2^lev,1:2^lev) ,...
       mat_a(1:2^lev,(2^lev)+1:2^(lev+1)) ,...
       mat_a((2^lev)+1:2^(lev+1),1:(2^lev)) ,...
       mat_a((2^lev)+1:2^(lev+1),(2^lev)+1:2^(lev+1))  ,'bior3.7');
   else
     xx=idwt2(xx ,...
       mat_a(1:2^lev,(2^lev)+1:2^(lev+1)) ,...
       mat_a((2^lev)+1:2^(lev+1),1:(2^lev)) ,...
       mat_a((2^lev)+1:2^(lev+1),(2^lev)+1:2^(lev+1))  ,'bior3.7');
   end;
     lev=lev+1;
end;
figure
```

```
%imshow(mat2gray(xx));
imshow((xx/255));

toc
```

โปรแกรม SPIHT encode 3 level with header PSNR

```
clear all,clc
format short
tic
x=imread('baboon_64.gif');
x=double(x);
I=x;

im_size=length(x);
mat=zeros(size(x));

dwt_level=3;

lev=log2(im_size)-dwt_level; %frist level

  if lev < 1
    break
  end;

%%%% 2D-DWT %%%%
dwtmode('per','nodisp');

mapping_index=im_size;

for ii=1:dwt_level
  [a b c d]=dwt2(I,'bior3.7');
  mat(1:2^(log2(mapping_index)-1),1:2^(log2(mapping_index)-1))=a;
  mat(1:2^(log2(mapping_index)-1),2^(log2(mapping_index)-1)+1:2^log2(mapping_index))=b;
```

```
   mat(2^(log2(mapping_index)-1)+1:2^log2(mapping_index),1:2^(log2(mapping_index)-1))=c;
   mat(2^(log2(mapping_index)-1)+1:2^log2(mapping_index),2^(log2(mapping_index)-
1)+1:2^log2(mapping_index))=d;
   I=a;
   mapping_index=mapping_index/2;
end
clear mapping_index

figure
imshow(mat2gray(mat));

clear a;  clear b;   clear c;  clear d;
clear x;  clear I;

T=2^fix(log2(max(abs(mat(:)))));  %Th

LIP=[]; LIS=[]; % 0*0
OP=[]; % Output
stop_run=0;

fid = fopen('three_level_output_encode.txt','a'); % Open outout file for write.

fprintf(fid,'%d\n',dwt_level); % Write DWT-level data to file.
fprintf(fid,'%d\n',im_size); % Write image size to file.
fprintf(fid,'%d\n',T); % Write the Th to file.

% ========================== Initialization ========================== %

 %LIP(row,colum)
 for m=1:2^lev
    for n=1:2^lev
       LIP=[LIP ;[m,n]];
    end
 end
```

```
%% LIS(row,colum,type{A=0,B=1})
%% Keep only first child of (i,j) in LIP for addressing mode
 for m=1:2:2^lev
   for n=(2^lev)+1:2:(2^(lev+1))
      LIS=[LIS ;[m,n,0]];
   end
 end

 for m=(2^lev)+1:2:(2^(lev+1))
   for n=1:2:2^lev
      LIS=[LIS ;[m,n,0]];
   end
 end

 for m=(2^lev)+1:2:(2^(lev+1))
   for n=(2^lev)+1:2:(2^(lev+1))
      LIS=[LIS ;[m,n,0]];
   end
 end
      LSP=[];
% ================================================================

 %!!!!! Loop here for Bit Per Pixel

% while bits_expected >= stop_run ; %stop runing when bpp is ok

 while T>=1; % Stop when refinement all
        % This mean that you run losses compression
 T
 %for irk=1:5; % for test stop runing

% ========================== Sorting Pass ==========================%
```

```
% ################### LIP Loop ####################

 temp_size=size(LIP);  tem=temp_size(1);  clear temp_size;
 ii=1;
while ii<=tem

    temp=LIP(ii,:);
    temp_a=temp(1);
    temp_b=temp(2);

     if abs(mat(temp_a,temp_b))>=T % To test significant pixel in LIP
        OP=1;
        fprintf(fid,'%d\n',OP); % Write outout data to file.

        LSP=[LSP ; LIP(ii,:)];   % If 'significant' move (i,j) to LSP

        % Removing 'significant' address from LIP
        LIP = [LIP(1:ii-1,:) ; LIP(ii+1:end,:)];

        %%% Output sign %%%
        if mat(temp_a,temp_b)>=0
          OP=0; %%% positive = 0
          fprintf(fid,'%d\n',OP); % Write output data to file.
        else
          OP=1; %%% negative = 1
          fprintf(fid,'%d\n',OP); % Write output data to file.
        end
        %%%%%%%%%%%%%%%%%%%

      else
        OP=0;
        fprintf(fid,'%d\n',OP); % Write output data to file.
        ii=ii+1;
```

```
    end

    temp_size=size(LIP);  % Rearrange loop limit number,
    tem=temp_size(1);     % because address number has changed
    clear temp_size;

  end

% ################ End of LIP Loop ################

% ################### LIS Loop ###################

 child=[]; child_addr=[]; g_child=[];
 check_sig=[];

 temp_size=size(LIS);  tem=temp_size(1); clear temp_size;

 ii=1;
 while ii<=tem
   temp=LIS(ii,:);
   temp_a=temp(1);
   temp_b=temp(2);
   temp_c=temp(3);

   %^^^^^^^^^^ A-type ^^^^^^^^^^^^^%

   if temp_c==0  %if 2

     child=[mat(temp_a,temp_b);...     % Find the set of child
          mat(temp_a,temp_b+1);...
          mat(temp_a+1,temp_b);...
          mat(temp_a+1,temp_b+1)];

     child_addr=[[temp_a,temp_b];...   % Find the set of child address
```

```
        [temp_a,temp_b+1];...
        [temp_a+1,temp_b];...
        [temp_a+1,temp_b+1]];

    %   check_sig=(abs(child)>=T) % To test significant pixel

        if temp_a <(2^(log2(im_size)-1))+1 & temp_b <(2^(log2(im_size)-1))+1 % Checking that (i,j)
in LIP have
                                                    % grand child or not
        % Find the set of child
        g_child=[ mat(((temp_a-1)*2)+1,((temp_b-1)*2)+1);...
            mat(((temp_a-1)*2)+1,((temp_b-1)*2)+2);...
            mat(((temp_a-1)*2)+1,((temp_b-1)*2)+3);...
            mat(((temp_a-1)*2)+1,((temp_b-1)*2)+4);...

            mat(((temp_a-1)*2)+2,((temp_b-1)*2)+1);...
            mat(((temp_a-1)*2)+2,((temp_b-1)*2)+2);...
            mat(((temp_a-1)*2)+2,((temp_b-1)*2)+3);...
            mat(((temp_a-1)*2)+2,((temp_b-1)*2)+4);...

            mat(((temp_a-1)*2)+3,((temp_b-1)*2)+1);...
            mat(((temp_a-1)*2)+3,((temp_b-1)*2)+2);...
            mat(((temp_a-1)*2)+3,((temp_b-1)*2)+3);...
            mat(((temp_a-1)*2)+3,((temp_b-1)*2)+4);...

            mat(((temp_a-1)*2)+4,((temp_b-1)*2)+1);...
            mat(((temp_a-1)*2)+4,((temp_b-1)*2)+2);...
            mat(((temp_a-1)*2)+4,((temp_b-1)*2)+3);...
            mat(((temp_a-1)*2)+4,((temp_b-1)*2)+4)];
        else
         g_child=0; % Setup of grand child as zero,
                % because (i,j) in LIP don't have grand child
        end;
```

```
if max([max(abs(g_child)) max(abs(child))])>=T    % To test significant set

 OP=1;
 fprintf(fid,'%d\n',OP); % Write outout data to file.


     check_sig=(abs(child)>=T); % To test significant pixel
   for hj=1:length(check_sig)
     if check_sig(hj)==1
       OP=1;
       fprintf(fid,'%d\n',OP); % Write outout data to file.

        % Moving the child that is significant pixel to LSP
        LSP=[LSP; child_addr(hj,:)];

       %%% Output sign %%%
        if child(hj) >= 0
         OP=0;
         fprintf(fid,'%d\n',OP); % Write outout data to file.
        else
         OP=1;
         fprintf(fid,'%d\n',OP); % Write outout data to file.
         end;
       %%%%%%%%%%%%%%%%%%%%

    else
       OP=0;
       fprintf(fid,'%d\n',OP); % Write outout data to file.

       % Moving the child that is significant pixel to LIP
       LIP=[LIP; child_addr(hj,:)];
    end;
  end;

 if  temp_a <(2^(log2(im_size)-1))+1 & temp_b <(2^(log2(im_size)-1))+1
```

```
            % If (i,j) has child, move (i,j) to end of LIS as B-type
            LIS(ii,3) = 1;
            LIS = [LIS ; LIS(ii,:)];

            % Removing (i,j) as A-type from LIS
            LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

        else
            % If (i,j) don't has child, remove (i,j) from LIS
            LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

        end;
    else
      OP=0;
      fprintf(fid,'%d\n',OP); % Write outout data to file.
      ii=ii+1;
    end;  % if 1

end; % if 2


%^^^^^^^^^^^^^^^End of A-type^^^^^^^^^^^^^^^

%^0^_^0^_^0^_^0^_^0^_^0^_^ B-type ^0^_^0^_^0^_^0^_^0^_^0^_^

if temp_c==1
  % Find the set of grand child
  % !!! No need Checking, because we have checked at A-type loop already
  g_child=[ mat(((temp_a-1)*2)+1,((temp_b-1)*2)+1);...
          mat(((temp_a-1)*2)+1,((temp_b-1)*2)+2);...
          mat(((temp_a-1)*2)+1,((temp_b-1)*2)+3);...
          mat(((temp_a-1)*2)+1,((temp_b-1)*2)+4);...
```

```
 mat(((temp_a-1)*2)+2,((temp_b-1)*2)+1);...
 mat(((temp_a-1)*2)+2,((temp_b-1)*2)+2);...
 mat(((temp_a-1)*2)+2,((temp_b-1)*2)+3);...
 mat(((temp_a-1)*2)+2,((temp_b-1)*2)+4);...

 mat(((temp_a-1)*2)+3,((temp_b-1)*2)+1);...
 mat(((temp_a-1)*2)+3,((temp_b-1)*2)+2);...
 mat(((temp_a-1)*2)+3,((temp_b-1)*2)+3);...
 mat(((temp_a-1)*2)+3,((temp_b-1)*2)+4);...

 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+1);...
 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+2);...
 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+3);...
 mat(((temp_a-1)*2)+4,((temp_b-1)*2)+4)];

if max(abs(g_child))>=T

 OP=1;
 fprintf(fid,'%d\n',OP); % Write outout data to file.

 % Removing (i,j) from LIS
 LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

 % Add new frist child to LIS as A-type
 LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+1,0]];
 LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+3,0]];
 LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+1,0]];
 LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+3,0]];

else

 OP=0;
 fprintf(fid,'%d\n',OP); % Write outout data to file.
 ii=ii+1;
```

```
        end;

    end;

    temp_size=size(LIS);  % Rearrange loop limit number,
    tem=temp_size(1);     % because address number has changed
    clear temp_size;

end; % while

%^0^_^0^_^0^_^0^_^0^_^ End of B-type ^0^_^0^_^0^_^0^_^0^_^0^_^


% ################ End of LIS Loop ################

% ========================= End of Sorting Pass ==========================%

% ========================= Refinement Pass ========================== %

    temp_size=size(LSP);
    tem=temp_size(1);
    clear temp_size;

        Th_temp=dec2binvec(T);
        length_temp=length(Th_temp);

    for ii=1:tem
            temp=LSP(ii,:);
      temp_a=temp(1);
      temp_b=temp(2);
            mat_temp=dec2binvec(abs(mat(temp_a,temp_b)),2^fix(log2(max(abs(mat(:))))));
      OP=mat_temp(length_temp);
      fprintf(fid,'%d\n',OP); % Write outout data to file.
```

```
        end;

% ========== End of Refinement Pass ==========%
  T=T/2; % Update T
 % Go to Sorting Pass
 % Stop when we have exact 'bit per pixel'

 end; %!!!!! end Loop here for Bit Per Pixel

 fclose(fid);
 toc
end;
```

โปรแกรม SPIHT Decode 3 level with header PSNR

```
clear all,clc
format short
tic

Im_org=imread('baboon_64.gif');
Im_org=double(Im_org);

IP=load('three_level_output_encode.txt');

dwt_level=IP(1);
IP=IP(2:end);

im_size=IP(1);
IP=IP(2:end);

mat=zeros(im_size,im_size);
mat_sign=mat;

lev=log2(im_size)-dwt_level; %frist level
```

```
lev_temp=lev;

  if lev < 1
    break
  end;

T=IP(1); %Th
IP=IP(2:end);

LIP=[]; LIS=[]; % 0*0

% import encoded data here

%================================ Initialization ================================%

 %LIP(row,colum)
 for m=1:2^lev
    for n=1:2^lev
      LIP=[LIP ;[m,n]];
    end
 end

 %% LIS(row,colum,type{A=0,B=1})
 %% Keep only frist child of (i,j) in LIP for addressing mode
  for m=1:2:2^lev
   for n=(2^lev)+1:2:(2^(lev+1))
     LIS=[LIS ;[m,n,0]];
   end
  end

  for m=(2^lev)+1:2:(2^(lev+1))
   for n=1:2:2^lev
     LIS=[LIS ;[m,n,0]];
   end
```

```
    end

    for m=(2^lev)+1:2:(2^(lev+1))
      for n=(2^lev)+1:2:(2^(lev+1))
        LIS=[LIS ;[m,n,0]];
      end
    end
        LSP=[];
%=================================================================

 %!!!!! Loop here for Bit Per Pixel

 %!!!!!  while length(IP)>0 ; stop runing when bpp is ok

   number_IP=[];PSNR_plot=[];
 while T>=1;
   number_IP=[number_IP; length(IP)];

%  for irk=1:5  ; %for stop runing

%================== Sorting Pass ==========================================%

% ################### LIP Loop ######################
 temp_size=size(LIP);  tem=temp_size(1);  clear temp_size;
 ii=1;
while ii<=tem

   temp=LIP(ii,:);
   temp_a=temp(1);
   temp_b=temp(2);

    if IP(1)==1;
      IP=IP(2:end);
```

```
        LSP=[LSP ; LIP(ii,:)];   % If 'significant' move (i,j) to LSP

        %%% Input sign %%%
        mat_sign(temp_a,temp_b)=IP(1);
        IP=IP(2:end);
        %%%%%%%%%%%%%%%%%%%%

        % Removing 'significant' address from LIP
        LIP = [LIP(1:ii-1,:) ; LIP(ii+1:end,:)];

      else
        IP=IP(2:end);
        ii=ii+1;
      end

      temp_size=size(LIP);  % Rearrange loop limit number,
      tem=temp_size(1);     % because address number has changed
      clear temp_size;
    end

% ################ End of LIP Loop ################

% ################## LIS Loop ####################

 child=[]; child_addr=[]; g_child=[];
 check_sig=[];

 temp_size=size(LIS);  tem=temp_size(1); clear temp_size;

 ii=1;
 while ii<=tem
   temp=LIS(ii,:);
   temp_a=temp(1);
   temp_b=temp(2);
```

```
temp_c=temp(3);

%^^^^^^^^^^ A-type ^^^^^^^^^^^^^%

if temp_c==0  %if 2

  if IP(1)==1   % To test significant set
   IP=IP(2:end);

   child_addr=[[temp_a,temp_b];...   % Find the set of child address
         [temp_a,temp_b+1];...
         [temp_a+1,temp_b];...
         [temp_a+1,temp_b+1]];

   for hj=1:4
    if IP(1)==1
     IP=IP(2:end);
     % Moving the child that is significant pixel to LSP
     LSP=[LSP; child_addr(hj,:)];

     %%% Input sign %%%
     tb=child_addr(hj,:);
     mat_sign(tb(1),tb(2))= IP(1);
     IP=IP(2:end);
     %%%%%%%%%%%%%%%%%%%%

    else
     % Moving the child that is significant pixel to LIP
     LIP=[LIP; child_addr(hj,:)];
     IP=IP(2:end);
    end;
   end;

    if   temp_a <(2^(log2(im_size)-1))+1 & temp_b <(2^(log2(im_size)-1))+1
```

```
            % If (i,j) has child, move (i,j) to end of LIS as B-type
            LIS(ii,3) = 1;
            LIS = [LIS ; LIS(ii,:)];

            % Removing (i,j) as A-type from LIS
            LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];
        else
            % If (i,j) don't has child, remove (i,j) from LIS
            LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];
        end;

    else
      IP=IP(2:end);
      ii=ii+1;
    end;  % if 1

end; % if 2

%^^^^^^^^^^^^^^^End of A-type^^^^^^^^^^^^^^^

%^0^_^0^_^0^_^0^_^0^_^0^_^ B-tpye ^0^_^0^_^0^_^0^_^0^_^0^_^

if temp_c==1

    if IP(1)==1
      IP=IP(2:end);
      % Removing (i,j) from LIS
      LIS = [LIS(1:ii-1,:) ; LIS(ii+1:end,:)];

      % Add new frist child to LIS as A-type
      LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+1,0]];
      LIS = [LIS ;[((temp_a-1)*2)+1,((temp_b-1)*2)+3,0]];
```

```
            LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+1,0]];
            LIS = [LIS ;[((temp_a-1)*2)+3,((temp_b-1)*2)+3,0]];

          else

            IP=IP(2:end);
            ii=ii+1;
          end;

        end;

        temp_size=size(LIS);  % Rearrange loop limit number,
        tem=temp_size(1);     % because address number has changed
        clear temp_size;

      end; % while

      %^0^_^0^_^0^_^0^_^ End of B-type ^0^_^0^_^0^_^0^_^0^_^

    % ################ End of LIS Loop ################

    % ================= End of Sorting Pass ================================%

    % ================= Refinement Pass ================================%
        temp_size=size(LSP);
        tem=temp_size(1);
        clear temp_size;
    %     temp_Refinement=

          Th_temp=dec2binvec(T);
          length_temp=length(Th_temp);
```

```
        for ii=1:tem
                temp=LSP(ii,:);
          temp_a=temp(1);
          temp_b=temp(2);
          mat(temp_a,temp_b)=mat(temp_a,temp_b)+ (IP(1)*T);
          IP=IP(2:end);
           end;

% ========== End of Refinement Pass ==========%
   T
   T=T/2; % Update T

 % Go to Sorting Pass
 % Stop when we have exact 'bit per pixel'

mat_sign_temp=mat_sign.*(-2);
mat_sign_temp=mat_sign_temp+1;
mat_a=mat_sign_temp.*mat;
xx=[];
lev_temp=lev;

dwtmode('per','nodisp');

for iii=1:dwt_level

  if iii==1
    xx=idwt2(mat_a(1:2^lev_temp,1:2^lev_temp) ,...
      mat_a(1:2^lev_temp,(2^lev_temp)+1:2^(lev_temp+1)) ,...
      mat_a((2^lev_temp)+1:2^(lev_temp+1),1:(2^lev_temp)) ,...
      mat_a((2^lev_temp)+1:2^(lev_temp+1),(2^lev_temp)+1:2^(lev_temp+1)) ,'bior3.7');
  else
    xx=idwt2(xx ,...
      mat_a(1:2^lev_temp,(2^lev_temp)+1:2^(lev_temp+1)) ,...
      mat_a((2^lev_temp)+1:2^(lev_temp+1),1:(2^lev_temp)) ,...
```

```
        mat_a((2^lev_temp)+1:2^(lev_temp+1),(2^lev_temp)+1:2^(lev_temp+1)) ,'bior3.7');
    end;
      lev_temp=lev_temp+1;
 end;

% figure
% imshow(mat2gray(xx));
% imshow((xx/255));


err=(Im_org(:)-xx(:));
perf=mean(err(:).^2);
PSNR=10*log10((255^2)/perf);
PSNR_plot=[PSNR_plot;PSNR];

 end; %!!!!! end Loop here for Bit Per Pixel

%  delete('output_encode.txt')

BPP=(im_size^2)./number_IP;
%figure
%plot(BPP,PSNR_plot,'m-x'),grid on

toc

fid = fopen('bpp_plot_file.txt','a'); % Open file for write.
fprintf(fid,'%d\n',BPP); % Write data to file.
fclose(fid);

fid = fopen('psnr_plot_file.txt','a'); % Open file for write.
fprintf(fid,'%d\n',PSNR_plot); % Write data to file.
fclose(fid);

psnr_plot_temp=[];
```

```
BPP_plot_temp=[];
psnr_plot_temp=load('psnr_plot_file.txt','a');
BPP_plot_temp=load('bpp_plot_file.txt','a');

figure
plot(BPP_plot_temp,psnr_plot_temp,'m-x'),grid on
xlabel('Bit per pixel : bpp');
ylabel('Peak signal to noise ratio : psnr');
title('Peak signal to noise ratio vs bit per pixel');

end;
```

โปรแกรม Lifting coefficient 9-7 จำลองจาก VHDL

```
clear all
clc;
close all;

format long;

delay_register_a=0;
delay_register_b=0;
delay_register_c=0;
delay_register_d=0;
delay_register_e=0;
delay_register_f=0;

pipeline_register_a=0;
pipeline_register_b=0;
pipeline_register_c=0;
pipeline_register_d=0;
pipeline_register_e=0;
pipeline_register_f=0;
pipeline_register_g=0;
```

```
pipeline_register_h=0;

alpha = -1.5861343420693648;
beta  = -0.0529801185718856;
gamma =  0.8829110755411875;
delta =  0.4435068520511142;
k     =  1.1496043988602418;


xe=[];
xo=[];

ye=[];
yo=[];

n=[1:5000];

x=sin(n*2*pi*(1/550))+(0.05*rand(1,length(n)));

%x=sin(n*2*pi*(1/550));

x=x/max(x);
x=x+1;
x=x*0.5;

x=x*(2^8);
x=round(x);

xx= fliplr(x);

xxe=xx(1:2:end);
xxo=xx(2:2:end);
```

```
%xe=[xxe(end-0:1:end) x(1:2:end)];
%xo=[xxo(end-0:1:end) x(2:2:end)];

xe=[x(1:2:end) xxe(end-4:1:end)];
xo=[x(2:2:end) xxe(end-4:1:end)];

for i=1:length(xe)
yo(i)=  k *pipeline_register_h;
ye(i)=(1/k)*pipeline_register_g;

%%%%% 2'rd state %%%%%%%%%
pipeline_register_g=((pipeline_register_f+delay_register_f)*delta)+pipeline_register_e;
pipeline_register_h=pipeline_register_f;
delay_register_f=pipeline_register_f;

pipeline_register_e=delay_register_d;
pipeline_register_f=((delay_register_d+pipeline_register_c)*gamma)+delay_register_e;

delay_register_d=pipeline_register_c;
delay_register_e=pipeline_register_d;

%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% 1'ft state %%%%%%%%%
pipeline_register_c=((pipeline_register_b+delay_register_c)*beta)+pipeline_register_a;
pipeline_register_d=pipeline_register_b;
delay_register_c=pipeline_register_b;

pipeline_register_a=delay_register_a;
pipeline_register_b=((delay_register_a+xe(i))*alpha)+delay_register_b;

delay_register_a=xe(i);
delay_register_b=xo(i);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

figure
plot(xe(10:end))

figure
plot(yo(10:end))

figure
plot(ye(10:end))


[cA,cD] = dwt(x,'bior4.4');

figure
plot(cA)

figure
plot(cD)
```

โปรแกรม LIFTING coefficient 9-7 โครงสร้างแบบ low gate

```
clear all
clc;
close all;
format long;

D0R=0; D1R=0; D2R=0; D3R=0;
P0R=0; P1R=0; P2R=0; P3R=0; P4R=0; P5R=0;

alpha = -1.5861343420693648;
beta  = -0.0529801185718856;
```

```
gamma =  0.8829110755411875;
delta =  0.4435068520511142;
k     =  1.1496043988602418;

xe=0; xo=0;
ye=0; yo=0;

sel=0;

I=imread('lena128.tif');
im_size=size(I);
I=double(I);
%y=zeros(size(I));

for m=1:im_size(1)
   x=I(m,:);
   xf(m,:)=x(1:2:end);
%  x=[x(9),x(8),x(7),x(6),x(5),x(4),x(3),x(2), x ,x(length(x)-1),x(length(x)-2),x(length(x)-
3),x(length(x)-4),x(length(x)-5),x(length(x)-6),x(length(x)-7),x(length(x)-8),x(length(x)-9),x(length(x)-
10)];
   x=[x(5),x(4),x(3),x(2), x ,x(length(x)-1),x(length(x)-2),x(length(x)-3),x(length(x)-4),x(length(x)-
5),x(length(x)-6),x(length(x)-7),x(length(x)-8),x(length(x)-9),x(length(x)-10)];

   fprintf('%.0f',m);

   D0R=0; D1R=0; D2R=0; D3R=0;
   P0R=0; P1R=0; P2R=0; P3R=0; P4R=0; P5R=0;
   kk=0;
for n=1:1:length(x)

    if sel==0
      kk=kk+1;
      ye(kk)=(1/k)*P2R;
      yo(kk)=(k)*P5R;
```

```
    end

%   if rem(n,2) == 0
      P2R_temp=P2R;
      P5R_temp=P5R;
%   end;

    if sel == 1
      P2R = P1R+((P4R+D3R)*delta);
    else
      P2R = P1R+((P4R+D3R)*beta);
    end;

    P5R=D3R;

    D3R=D2R;
    D2R=P4R;

    if sel == 1
      P4R=P3R+((P0R+D1R)*alpha);
    else
      P4R=P3R+((P0R+D1R)*gamma);
    end;

    P1R=D1R;

    D1R=D0R;
    D0R=P0R;

    if rem(n,2) == 1
      xe=x(n);
    else
      xo=x(n);
    end;
```

```
      if sel==0
        P0R=xe;
        P3R=xo;
        sel=1;
      else
        sel=0;
        P0R=P2R_temp;
        P3R=P5R_temp;
      end;
    end;
   clc
%   y(m,:)=[ye,yo];
   y(m,:)=[ye(8:1:end)];
   xx(m,:)=x(1:2:end);
end;

 figure
 imshow(xf/255)

 figure
 imshow(xx/255)

 figure
 imshow(y/255)
```

โปรแกรม VHDL : Discrete Wavelet Transform (DWT) Rx-Tx

```
library ieee;
use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;

Entity dwt_rxtx is
GENERIC(N : INTEGER :=15);

Port( clk          : in   std_logic; -- fpga clk
          RX           : in   std_logic; -- fpga RX
          TX           : out  std_logic; -- fpga TX
    data_led      : out  std_logic_vector( 7 downto 0));

End;

Architecture rtl of dwt_rxtx is

  SIGNAL state          : integer range 0 to 63;
        SIGNAL state_index_RX   : integer range 0 to 63;
        SIGNAL state_index_TX   : integer range 0 to 63;

 SIGNAL data_temp       : STD_LOGIC_VECTOR(15 downto 0);
 SIGNAL xo, xe, yo, ye  : std_logic_vector( N downto 0);
 Begin

 Process(clk ,RX)
  ----- Rx & Tx variables ------
   variable RX_Data_Count  : integer range 0 to 7; -- RX data count variable
   variable wait_count     : integer range 0 to 10;-- Waiting count variable
          variable Buffer_RX     : std_logic_vector(7 downto 0);-- Rx data Buffer variable
          variable acc_var       : std_logic_vector(3 downto 0);-- Rx Accumulator variable
    variable RX_flag       : std_logic;

          variable  TX_Data_Count : integer range 0 to 7;       -- TX data count variable
```

```
        variable  DATA_TX     : std_logic_vector(7 downto 0);-- TX data input variable
        variable  TX_flag     : std_logic_vector(1 downto 0);-- Tx flag variable
-----------------------------

----- DWT variables ------
 variable delay_register_a, delay_register_b : std_logic_vector( N downto 0);
 variable delay_register_c              : std_logic_vector( N downto 0);
 variable delay_register_d, delay_register_e : std_logic_vector( N downto 0);
 variable delay_register_f              : std_logic_vector( N downto 0);

 variable pipeline_register_a           : std_logic_vector( N downto 0);
 variable pipeline_register_b           : std_logic_vector( N downto 0);
 variable pipeline_register_c           : std_logic_vector( N downto 0);
 variable pipeline_register_d           : std_logic_vector( N downto 0);
 variable pipeline_register_e           : std_logic_vector( N downto 0);
 variable pipeline_register_f           : std_logic_vector( N downto 0);
 variable pipeline_register_g           : std_logic_vector( N downto 0);
 variable pipeline_register_h           : std_logic_vector( N downto 0);

 Variable temp_a, temp_b, temp_bar      : std_logic_vector( N downto 0);

              Variable dwt_flag                 : std_logic;
-----------------------------

       Begin
              if clk'Event and clk = '1' then
                     CASE state is

     WHEN 0 =>  ----- Initialize 1 -----
                state <= 1;
                state_index_RX<=6;
                 state_index_TX<=1;
               data_temp<=(others=>'0');
               RX_flag:='0';
```

```
                TX_flag:="00";

  WHEN 1 =>  --Idel line
                wait_count:=0; --Clear wait_count
                acc_var:=(others=>'0');  --Clear acc_var
                 Buffer_RX:=(others=>'0');--Clear Buffer_RX
                RX_Data_Count := 0;-- Clear RX_Data_Count
                 if RX = '0' then -- If RX=0  set RX_Data_Count=0,
                      state <= 2; -- goto state 1
                end if;
WHEN 2 =>  --Start receive data
              if  wait_count = 9 then -- Waiting for data
                 wait_count := 0;--Clear wait_count
                  state <= 3;--goto state 2
           else
               wait_count:=wait_count+1;-- Accumulatc wait_count
           end if;

WHEN 3 =>  --Data Collection
              if wait_count = 9 then -- Accumulate data until wait_count = 9,
               state <= 4;        -- goto state 3;
               wait_count := 1;    -- Set wait_count = 1
              acc_var:=acc_var+RX;-- Accumulate data at last time


               if acc_var >= "0111" then -- if Accumulator>6 then
                Buffer_RX(RX_Data_Count) := '1';-- data = '1'
                acc_var:=(others=>'0');--Clear acc_var
              else
                Buffer_RX(RX_Data_Count) := '0';
                acc_var:=(others=>'0');--Clear acc_var
              end if;
            else
                wait_count:=wait_count+1; -- Accumulate wait_count
                acc_var:=acc_var+RX; -- Accumulate d
```

```
            end if;

WHEN 4 =>  -- Checking for Collection pass
            if RX_Data_Count = 7 then -- If data = 8 bit then
              state <= 5;          -- goto state = 4
            else
              state <= 3;
              RX_Data_Count:=RX_Data_Count+1; -- Accumulate RX_Data_Count
            end if;

 WHEN 5 =>  --stop
              data_led<=Buffer_RX;
            if wait_count = 9 then
               if RX_flag='0' then
                  state <=1;
                     data_temp(7 downto 0)<=Buffer_RX;
                  RX_flag:='1';
               else
                  data_temp(15 downto 8)<=Buffer_RX;
                  state <= state_index_RX;
                  RX_flag:='0';
               end if;
               wait_count := 0;
            else
                wait_count:=wait_count+1;
            end if;


      ---------------------- TX -------------------------------
   WHEN 6 => state <= 7;
              xe(7 downto 0)<=data_temp(7 downto 0);
              xo(7 downto 0)<=data_temp(7 downto 0);
              xe(15 downto 8)<="00000000";
```

```
            xo(15 downto 8)<="00000000";

WHEN 7 => state <= 8;
            ----------------
            --Scaling Step--
            ----------------
        --1/K
        ye <= (pipeline_register_g+ --[(2^(0))]
           (not((pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N downto 3)))+'1')+ -- [-(2^(-3))]
           (not((pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N downto 7)))+'1')+ -- [-(2^(-7))]
              (pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N)
              &pipeline_register_g(N downto 9))); -- [(2^(-9))]

        --K
        yo <= (pipeline_register_h+ --[(2^(0))]
            (pipeline_register_h(N)
```

```
      &pipeline_register_h(N)
      &pipeline_register_h(N)
      &pipeline_register_h(N downto 6))+ -- [(2^(-3))]
      (pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N downto 6))+ -- [(2^(-6))]
      (pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N)
        &pipeline_register_g(N downto 7))); -- [(2^(-7)
      ---------------------
      --Second Lifting Step--
      ---------------------
--updater (delta)--

temp_a := (delay_register_f + pipeline_register_f);
temp_b := (temp_a(N) &temp_a(N downto 1))+ -- [(2^(-1))]
        (not((temp_a(N) &temp_a(N) &temp_a(N)
        &temp_a(N) &temp_a(N downto 4)))+'1'); -- [-(2^(-4))]

                        pipeline_register_g := pipeline_register_e + temp_b;
pipeline_register_h := pipeline_register_f;

delay_register_f := pipeline_register_f;

--predictor (gamma)--
```

```
temp_a := (delay_register_d + pipeline_register_c);
temp_b :=  temp_a + -- [(2^(0))]
      (temp_a(N) &temp_a(N) &temp_a(N)&temp_a(N)
      &temp_a(N) &temp_a(N) &temp_a(N)&temp_a(N downto 7))+ -- [(2^(-7))]
   (not((temp_a(N) &temp_a(N) &temp_a(N)&temp_a(N downto 3)))+'1'); -- [(2^(-3))]

pipeline_register_f := delay_register_e  + temp_b;
pipeline_register_e := delay_register_d;

delay_register_d := pipeline_register_c;
                    delay_register_e := pipeline_register_d;

          ----------------------
         --First Lifting Step--
          ----------------------

--updater (beta)--

temp_a := (delay_register_c + pipeline_register_b);
temp_bar := not(temp_a)+'1';
temp_b := (temp_bar(N) &temp_bar(N) &temp_bar(N) &temp_bar(N)
      &temp_bar(N) &temp_bar(N) &temp_bar(N) &temp_bar(N)
      &temp_bar(N) &temp_bar(N) &temp_bar(N) &temp_bar(N)
      &temp_bar(N downto 12)) + -- [-(2^(-12))]

      (temp_a(N) &temp_a(N) &temp_a(N) &temp_a(N)
      &temp_a(N) &temp_a(N) &temp_a(N) &temp_a(N)
      &temp_a(N) &temp_a(N downto 9))+ -- [(2^(-9))]

      (temp_bar(N) &temp_bar(N) &temp_bar(N)
      &temp_bar(N) &temp_bar(N downto 4))+ -- [-(2^(-4))]

      (temp_a(N) &temp_a(N) &temp_a(N) &temp_a(N)
```

```
                &temp_a(N) &temp_a(N) &temp_a(N) &temp_a(N downto 5)); -- [(2^(-7))]

        pipeline_register_c := pipeline_register_a  + temp_b;
        pipeline_register_d := pipeline_register_b;

        delay_register_c := pipeline_register_b;

        --predictor (alpha)--

        temp_a := (delay_register_a + xe);
        temp_bar := not(temp_a)+'1';
        temp_b :=  temp_bar + -- [-(2^(0))]
              (temp_a(N) &temp_a(N) &temp_a(N) &temp_a(N)
              &temp_a(N) &temp_a(N downto 5))+ -- [(2^(-5)))]
              (temp_a(N) &temp_a(N) &temp_a(N) &temp_a(N)
              &temp_a(N) &temp_a(N) &temp_a(N) &temp_a(N downto 7))+ -- [(2^(-7))]
              (temp_bar(N) &temp_bar(N downto 1))+ -- [-(2^(-1))]
              (temp_bar(N) &temp_bar(N) &temp_bar(N)& temp_bar(N downto 3)); -- [-(2^(-3))]

                pipeline_register_b := delay_register_b  +  temp_b;
                pipeline_register_a := delay_register_a;

        delay_register_a := xe;
                delay_register_b := xo;


WHEN 8 => state <= 9;
                DATA_TX:=ye(7 downto 0);

WHEN 9 => TX <= '1';
                TX_Data_Count := 0;

                  if wait_count = 9 then
                      wait_count := 0;
```

```
                state <= 10;
                else
                        wait_count:=wait_count+1;
               end if;

WHEN 10 =>  TX <= '0'; -- start
                 if wait_count = 9 then
                  wait_count := 0;
                state <= 11;
                   else
                wait_count:=wait_count+1;
                  end if;

WHEN 11 => --Transfer
               if TX_Data_Count = 7 then
                TX <= DATA_TX(TX_Data_Count);
               if wait_count = 9 then
               wait_count := 0;
               state <=12;
                  else
               wait_count:=wait_count+1;
                end if;

                  else
               TX <= DATA_TX(TX_Data_Count);
              if wait_count = 9 then
               wait_count := 0;
                    TX_Data_Count := TX_Data_Count
             else
             wait_count:=wait_count+1;
            end if;
                  end if;
```

```
WHEN 12 => TX_Data_Count := 0; --stop
                    TX <= '1';
                  if wait_count = 9 then
                  wait_count := 0;

                  if   TX_flag="00" then
                       state <= 9;
                     TX_flag:="01";
                     DATA_TX:= ye(15 downto 8);
                    data_led<=DATA_TX;
                elsif TX_flag="01" then
                       state <= 9;
                    DATA_TX:= yo(7 downto 0);
                    TX_flag:="10";
                    data_led<=DATA_TX;
               elsif TX_flag="10" then
                       state <= 9;
                    DATA_TX:= yo(15 downto 8);
                    TX_flag:="11";
                    data_led<=DATA_TX;
               else
                    state <= state_index_TX;
                   TX_flag:="00";
              end if;
            else
             wait_count:=wait_count+1;
         end if;
    ---------- TX ----------
              WHEN others =>

              END CASE;
               end if;
      end process;
end;
```