โครงการพัฒนาโปรแกรมบีบอัดข้อมูลภาพ ตามมาตรฐาน JPEG2000 บนบอร์ดทดลองแบบฝังตัว

THE DEVELOPMENT OF JPEG2000 IMAGE COMPRESSION STANDARD ON EMBEDDED SYSTEM BOARD

นายไชยสิทธิ์ ยิ้มเสมอ
นายพิศาล เต็มจิตต์ภักดี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2549

โครงการพัฒนาโปรแกรมบีบอัดข้อมูลภาพ ตามมาตรฐาน JPEG2000 บนบอร์ดทดลองแบบฝังตัว

THE DEVELOPMENT OF JPEG2000 IMAGE COMPRESSION STANDARD ON EMBEDDED SYSTEM BOARD

โดย

นายไชยสิทธิ์ ยิ้มเสมอ

นายพิศาล เต็มจิตต์ภักดี

อาจารย์ที่ปรึกษา

ผศ.ดร.อรฉัตร จิตต์โสภักดร์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โครงการพัฒนาโปรแกรมบีบอัดข้อมูลภาพตามมาตรฐาน JPEG2000 บนบอร์ดทดลองแบบฝังตัว
THE DEVELOPMENT OF JPEG2000 IMAGE COMPRESSION STANDARD ON EMBEDDED SYSTEM BOARD

ผู้จัดทำ

1. นายไชยสิทธิ์ ยิ้มเสมอ 47015679
2. นายพิศาล เต็มจิตต์ภักดี 47015686

อาจารย์ที่ปรึกษา

(ผศ.ดร.อรฉัตร จิตต์โสภักตร์)

โครงการพัฒนาโปรแกรมบีบอัดข้อมูลภาพ ตามมาตรฐาน JPEG2000 บนบอร์ดทดลองแบบฝังตัว

นายไชยสิทธิ์ ยิ้มเสมอ 47015679
นายพิศาล เต็มจิตต์ภักดี 47015686
ผศ.ดร.อรฉัตร จิตต์โสภักตร์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

โครงงานนี้จัดทำขึ้นเพื่อการพัฒนาโปรแกรมบีบอัดข้อมูลภาพตามมาตรฐาน JPEG2000 บนบอร์ดทดลองแบบฝังตัวโดยมีวิธีดำเนินงานคือการพัฒนา Source code ให้เหมาะสมกับบอร์ดทดลอง โครงงานนี้เจาะจงใช้กับบอร์ด Altair(X-scale) และได้ source code ที่เป็น JPEG-2000 Part-1 standard (นั่นคือ ISO/IEC 15444-1) พัฒนาขึ้นมาโดย *Michael D. Adams* มหาวิทยาลัย Victoria ในชื่อ JasPer Software (version 1.700.2) ภายใต้ compiler Microsoft Visual C++ 6.0 ในแบบภาษา C นำมาพัฒนาปรับปรุงเพิ่มเติมแก้ไขให้สามารถคอมไพล์และเข้ารหัสภาพ JPEG2000 บนบอร์ด Altair(X-scale)ได้อย่างเหมาะสมและสามารถบีบอัดข้อมูลภาพตามมาตรฐาน JPEG2000 ได้

THE DEVELOPMENT OF JPEG2000 IMAGE COMPRESSION STANDARD ON EMBEDDED SYSTEM BOARD

Mr.Chaiyasit Yimsamoe 47015679

Mr.Pisan Themjitpakdee 47015686

Asst. Prof. Dr.Orachat Chitsobhuk Advisor

Academic Year 2006

ABSTRACT

This project attempts to implement jpeg2000 image compression standard in the embedded system board. The Altair (X-Scale) board is chosen as the development embedded board. The jpeg2000 source code is developed from Jasper Software (version 1.700.2), which is JPEG2000 Part-1 Standard (i.e. ISO/IEC 15444-1), originated from Michael D. Adams, the University of Victoria. Microsoft Visual C++ 6.0 is used as the compiler. The developed JPEG2000 source code can be compiled and encoded successfully. The experimental results show that the JPEG2000 image compression obtained from the embedded board completely compile to the JPEG2000 standard.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จได้อย่างสมบูรณ์ด้วยคำแนะนำจาก ผศ.ดร.อรฉัตร จิตต์โสภักตร์ ซึ่งเป็นอาจารย์ผู้ควบคุมปริญญานิพนธ์ ข้าพเจ้ามีความรู้สึกซาบซึ้งและเคารพนับถือเนื่องด้วยความอนุเคราะห์จากอาจารย์เป็นอย่างสูง

ขอกราบขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า

ขอขอบคุณรุ่นพี่ เพื่อนๆ รุ่นน้อง สาขาวิศวกรรมคอมพิวเตอร์ และคณะอื่นๆ ทุกคนที่มีส่วนร่วมให้คำปรึกษาแนะนำข้าพเจ้าในหลายเรื่องและคอยให้กำลังใจเสมอมา

ขอขอบพระคุณบริษัท ThaiGerTec ที่ได้ให้ความอนุเคราะห์ให้คำปรึกษาเรื่องการใช้งานเกี่ยวกับบอร์ด X-Scale แด่ข้าพเจ้า

กราบขอบพระคุณ บิดา มารดา ที่เป็นกำลังใจ เป็นที่ยึดเหนี่ยวจิตใจให้กับข้าพเจ้าเมื่อยามท้อแท้และหมดกำลังใจ

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์นี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

นายไชยสิทธิ์ ยิ้มเสมอ
นายพิศาล เต็มจิตต์ภักดี

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

ในปัจจุบันการใช้งาน Embedded System Board เข้ามามีบทบาทในงานอุตสาหกรรมเป็นอย่างมาก จนมามีบทบาททางด้านการประมวลผลภาพ (Image processing) ดังในหัวข้อของโครงงานนี้คือการที่ทำให้ Embedded System Board สามารถที่จะรองรับ JPEG2000 สามารถเข้ารหัสและถอดรหัสภาพได้จึงต้องมีการพัฒนา Source Code ที่จะมาทำงานบน Embedded System Board เพื่อที่จะถอดรหัสภาพดังกล่าวและสามารถนำไปใช้ได้จริงในงานด้านการประมวลผลภาพ (Image Processing)

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 เพื่อศึกษาและทำความเข้าใจกระบวนการในการบีบอัดข้อมูลภาพ ตามมาตรฐาน JPEG2000

1.2.2 เพื่อศึกษาและทำความเข้าใจ Embedded System Board

1.2.3 เพื่อนำความรู้และความเข้าใจในกระบวนการบีบอัดข้อมูลภาพ ตามมาตรฐาน JPEG2000 มาสร้างเป็นไลบรารี่(Library) เพื่อใช้งานบน Embedded System Board

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1.3.1 มีความรู้ความเข้าใจกระบวนการในการบีบอัดข้อมูลภาพ ตามมาตรฐาน JPEG2000

1.3.2 มีความรู้ความเข้าใจ Embedded System Board

1.3.3 ไลบรารี่(Library) โปรแกรมสำหรับการบีบอัดข้อมูลและคลายข้อมูล(Encode/ Decode) ภาพตามมาตรฐาน JPEG2000

1.3.4 ตัวอย่างโปรแกรมที่ใช้งานไลบรารี่โปรแกรม JPEG2000 (JPEG2000 Program Library) ที่สามารถบีบอัดข้อมูลและคลายข้อมูลภาพบน Embedded System Board ได้

1.4 ขอบเขตของโครงงาน

ศึกษาเรียนรู้และเข้าใจกระบวนการทำงานของการบีบอัดข้อมูลในมาตรฐาน JPEG2000 พัฒนาโดยการ Coding หรือสร้างไลบรารี่เพื่อเข้ารหัสภาพในรูปแบบ JPEG2000 ได้

1.5 ส่วนประกอบของรายงาน

รายงานฉบับนี้มีประกอบไปด้วยเนื้อหา 5 บท ได้แก่ บทนำ ทฤษฎีที่เกี่ยวข้อง การออกแบบและพัฒนา การทดลองและผลการทดลอง และสรุปผลการทดลอง โดยแต่ละบทประกอบไปด้วยเนื้อหาดังต่อไปนี้

1.5.1 บทนำ ประกอบด้วย ความเป็นมาของปัญหา วัตถุประสงค์ของโครงงาน ประโยชน์ที่คาดว่าจะได้รับ ขอบเขตของโครงงาน และส่วนประกอบของรายงาน

1.5.2 ทฤษฎีที่เกี่ยวข้อง ประกอบด้วยเรื่อง Header format of image Processing, Forward Intercomponent Transform, Forward Intracomponent Transform, Quantization, Tier-1, Tier-2, Intel XScale PXA255 Architecture, และวิธีตรวจวัดคุณภาพของภาพ

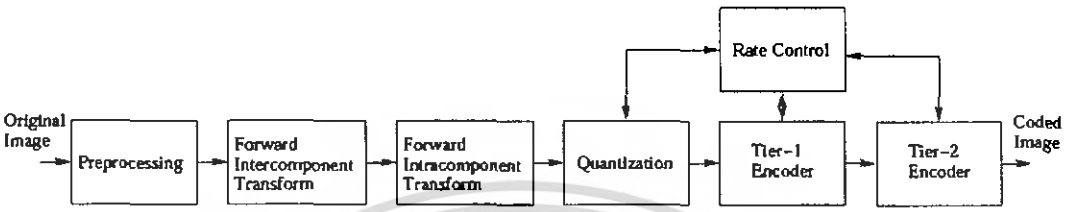
1.5.3 การออกแบบและพัฒนา ซึ่งได้กล่าวถึงการเริ่มการ Coding การประยุกต์ใช้ Source Code ที่เป็นมาตรฐานบน PC ให้ใช้งานบนบอร์ด X-Scale ได้

1.5.4 การทดลองและผลการทดลอง ประกอบด้วย การทดลอง เข้ารหัสและถอดรหัสภาพในรูปแบบของ JPEG2000 เปรียบเทียบทั้งบน PC และบนบอร์ด X-scale

1.5.5 สรุปผลการทดลองและแนวทางการพัฒนาต่อ

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องในการทำโครงการพัฒนาโปรแกรมบีบอัดข้อมูลภาพตามมาตรฐาน JPEG2000 บนบอร์ดทดลองแบบฝังตัว โดยมีบล็อกการทำงานดังรูปที่ 2.1



รูปที่ 2.1 ภาพ Block diagram ของ การเข้ารหัสภาพ JPEG2000

ทฤษฎีที่นำมาประยุกต์ใช้ในโครงการนี้แบ่งออกเป็น 9 หัวข้อสำคัญดังนี้

2.1 Header Format of Image

2.2 Preprocessing

2.3 Forward Intercomponent Transform

2.4 Forward Intracomponent Transform

2.5 Quantization

2.6 Tier-1 Algorithms in JPEG2000

2.7 Tier-2 Coding

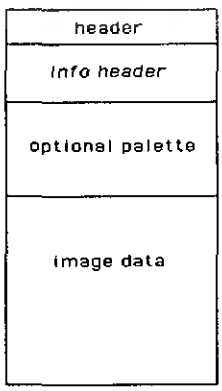
2.8 Intel XScale PXA255 Architecture

2.9 วิธีตรวจวัดคุณภาพของภาพ

2.1 Header Format of Image

2.1.1 BMP File Format

เป็นภาพที่ง่ายที่สุดต่อการทำความเข้าใจมีส่วนประกอบที่ไม่ซับซ้อนมากนัก DATA หลังจาก Header ไม่ได้มีการเข้ารหัสไว้โดยมีการแสดงถึงเนื้อข้อมูลจริง ๆ ในชนิดข้อมูลแบบ Integer จึงทำให้ไฟล์ภาพชนิดนี้ใช้ Memory ในการจัดเก็บมากที่สุดในรูปแบบไฟล์ภาพชนิดต่างๆ



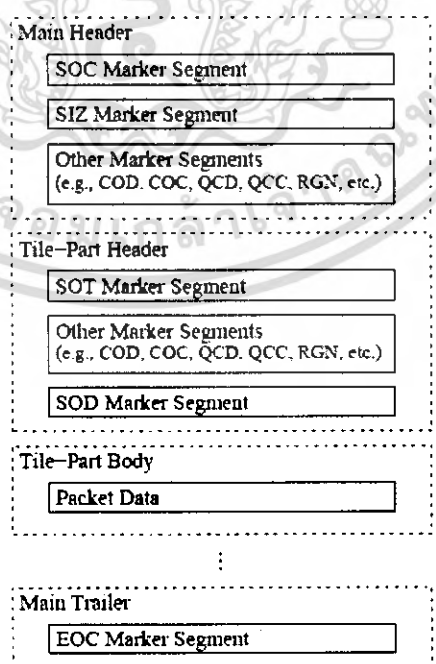
รูปที่ 2.2 BMP Segment Format

ตารางที่ 2.1 File Format BMP

Field Name	Size in Bytes	Description
bfType	2	เป็นตัวบอกว่าเป็น file BMP มีค่าเท่ากับ "BM"
bfSize	4	ขนาดของรูปภาพ
bfReserved1	2	Unused - must be zero
bfReserved2	2	Unused - must be zero
bfOffBits	4	ตำแหน่งเริ่มต้นการอ่าน
biSize	4	ขนาดของ header
biWidth	4	ความกว้างของรูป
biHeight	4	ความสูงของรูป
biPlanes	2	Must be 1
biBitCount	2	จำนวน bit ต่อ 1 ตำแหน่ง - 1, 2, 4, 8, 16, 24, หรือ 32
iCompression	4	รูปแบบการบีบอัด (0 = ไม่มีการบีบอัด)
biSizeImage	4	ขนาดของรูปที่มีการบีบอัด ถ้าไม่มีเป็น 0
biXPelsPerMeter	4	X Preferred resolution in pixels per meter
biYPelsPerMeter	4	Y Preferred resolution in pixels per meter
biClrUsed	4	Number Color Map entries that are actually used
biClrImportant	4	หมายเลขของสีที่สำคัญ

2.1.2 JP2 File Format

ไฟล์ภาพชนิดนี้เป็นไฟล์ที่ได้มีการเข้ารหัสเรียบร้อยแล้วหากต้องการทราบถึงเนื้อข้อมูลจริงๆ ต้องทำการ De-compress หรือแปลงภาพไปเป็นแบบที่เรียบง่ายอย่างเช่น BMP ที่ได้กล่าวมาแล้วในหัวข้อที่ 2.1.1



รูปที่ 2.3 Format Segment ของไฟล์ภาพแบบ Jpeg2000

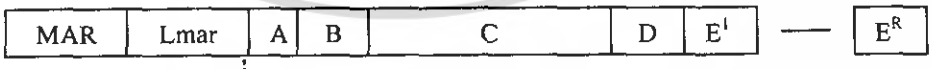
Syntax และ Code-stream Rules

1. Main header จะใส่ไว้ในส่วนเริ่มต้นของ Code-stream และใช้กั้น Code-stream พร้อมกับเตรียมข้อมูลทั่วไปเกี่ยวกับการบีบอัดไฟล์

2. Tile-part header จะใส่ไว้ในส่วนเริ่มต้นของ Bit stream ที่ถูกบีบอัดสำหรับแต่ละ Tile-part และใช้กั้นแต่ละ Tile-part พร้อมกับเตรียมข้อมูลของแต่ละ Tile ทุกๆ Marker ใน Code-stream มีขนาดความยาว 2 ไบท์โดยไบท์แรกเป็น 0 x FF เสมอและไบท์ที่สองมีค่าระหว่าง 0 x 01 ถึง 0 x FE มาตรฐาน JPEG2000 Part1 ได้นิยาม Marker segment ทั้ง 6 ประเภทดังต่อไปนี้

- Delimiting: เป็นตัวกั้นระหว่าง Header กับ Data
- Fixed information marker segments: จะประกอบด้วยข้อมูลของรูปภาพเฉพาะใน JPEG2000 Part1 เท่านั้นที่ส่วนของ Fixed information marker segments ต้องระบุ Image marker และ Tile size (SIZ) ไว้ใน Main header
- Functional maker segments: ใช้นิยาม Code function ที่ใช้เช่น Coding style default (COD) marker segment ถูกใช้นิยาม Coding style ซึ่งจะใช้เป็นค่าเริ่มต้นสำหรับการบีบอัดรูปหรือส่วนต่างๆของรูปภาพ
- Bitstream marker segments: ใช้สำหรับระบุความผิดพลาดที่เกิดขึ้นอย่างไม่แน่นอนซึ่ง marker เหล่านี้จะถูกพบใน Bitstream แต่จะไม่พบใน Main header หรือ Tile-part header
- Pointer marker segments: segment นี้จะระบุค่าชดเชยใน Bitstream โดย Marker นี้เป็นออปชั่นเสริมซึ่งเตรียมไว้สำหรับข้อมูลเรื่องความยาวและ Pointer ภายใน Bitstream
- Informational segments: เตรียมไว้สำหรับข้อมูลเพิ่มเติมเกี่ยวกับรูปภาพเช่น คอมเมนต์ต่างๆ สามารถใส่ไว้ในส่วนของ Main header หรือ Tile-part header ด้วย Comment and extension marker segment (*CME*)

ในส่วนของ 2 ไบท์แรกก่อน Marker เป็นจะ Unsigned big-endian integer value เช่น Lmar ที่ได้แสดงในรูป 2.4 ซึ่งแสดงความยาวของ Marker segment ทั้งหมด โดยไม่รวมอีก 2 ไบท์ที่เป็นตัว Marker มันเอง ความยาวของ Marker parameter อาจเป็น 1, 2, 4 ไบท์หรือจำนวนที่เปลี่ยนแปลงได้



รูปที่ 2.4 Sample marker segment description

2.1.2.1 กฎเบื้องต้น

ทั้งเจ็ดกฎดังต่อไปนี้เป็นกฎใน Annex A ของ JPEG2000 Part 1 standard [2] ทุกๆ JPEG2000 Compliance Codec System ต้องเป็นไปตามกฎเหล่านี้

1. ทุก Marker และ Marker segment และ Header ต้องเป็นทวีคูณของ 8 bits (1 ไบท์) ดังนั้น ถ้าจำนวนของบิตใน Bitstream data ระหว่างแต่ละ Header มีจำนวนไม่เท่ากับ 8 ก็จะถูกเพิ่มจนครบ ซึ่งเรียกว่าการทำ Zero-padding (การเพิ่มบิต 0 เข้าไป)

2. ทุก Marker และ Marker segment ภายใน Tile-part header สามารถใช้การได้ (หรือมีความถูกต้อง) เฉพาะ Tile นั้นๆ ส่วน Tile อื่นๆ อาจจะใช้ Marker เดียวกันในพารามิเตอร์ที่แตกต่างกัน
3. ทุก Marker และ Marker segment ใน Main header สามารถใช้การได้กับทั้งภาพ เว้นแต่ว่าถูกข้ามโดย Marker segment ในส่วนของ Tile-part header
4. Delimiting marker segments และ Fixed information marker segment อาจปรากฏในตำแหน่งที่เฉพาะเจาะจงของ Code-stream ตัวอย่างเช่น SOC (start of code-steam) marker ต้องปรากฏในฐานะที่เป็น Marker แรกใน Main header
5. Marker segment เหล่านี้จะนิยามภาพด้วย Code-stream ได้อย่างถูกต้องถ้าการเปลี่ยนแปลงใดๆ ที่เกิดขึ้นกับ Mode-stream ตัว Marker segment เหล่านั้นก็ต้องอัพเดทด้วย
6. ทุกค่าของพารามิเดอร์ใน Marker segment เป็น Big-endean (ไบท์(byte) ที่มีนัยสำคัญสูงสุดมาก่อน) ทุก Marker ที่มีค่าระหว่าง 0 x FF30 และ 0 x FF3F จะไม่ใช่ Marker parameter โดยถูกสงวนด้วยมาตรฐานที่กำหนด

2.1.2.2 Marker และ Marker Segment Definition

รูปที่ 2.4 แสดงตัวอย่างของ Marker segment definition โดยสองไบต์แรกถูกแทนด้วยอักษรย่อ 3 ตัว ของ Marker segment และสองไบต์ถัดไป (Lmar) ระบุความยาวทั้งหมดของส่วน Marker segment โดยไม่รวมสองไบท์ที่เป็น Marker มันเอง ตารางที่ 2.2 แสดง Marker name และ Code value ของ Marker segment รายละเอียดและวากยสัมพันธ์สำหรับ Marker และ Marker segment ต่างๆ สามารถดูได้ใน Annex A ของ JPEG2000 Part 1

ตารางที่ 2.2 Marker Segments: Marker, Name, and Value

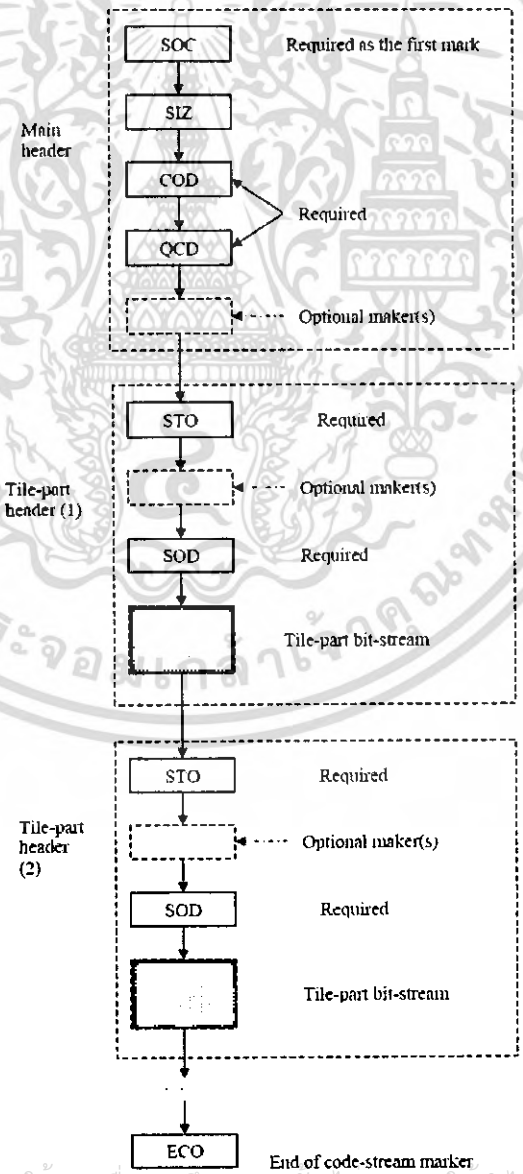
	Name	Code Value
Delimiting Marker Segments		
Start of Code-Stream	SOC	0xFF4F
Start of Tile-part	SOT	0xFF90
Start of data	SOD	0xFF93
End of Code-Stream	EOC	0xFFD3
Fixed Info Marker Segments		
Image and tile size	SIZ	0xFF51
Functional Marker Segments		
Coding style default	COD	0xFF52
Coding style component	COC	0xFF53
Region of interest	RGN	0xFF5E
Quantization default	QCD	0xFF5C
Quantization component	QCC	0xFF5D
Progression order default	POD	0xFF5F
Pointer Marker Segments		
Tile-part lengths main header	TLM	0xFF55
Packet length main header	PLM	0xFF57
Packet length Tile-part header	PLT	0xFF58
Packet packet header main header	PPM	0xFF60

ตารางที่ 2.2 Marker Segments: Marker, Name, and Value (ต่อ)

	Name	Code Value
Packet packet header Tile-part header	PPT	0xFF61
In Bit-stream Marker Segments		
Start of packet	SOP	0xFF91
End of packet header	EPH	0xFF92
Informational Marker Segments		
Comment and extension	CME	0xFF64

2.1.2.3 Headers Definition

Main header ที่แสดงในรูปที่ 2.5 ประกอบด้วยส่วนต่างๆที่รู้จักในนาม Main header marker segment บาง Marker จำเป็นที่ต้องแสดงแต่บางตัวอาจจะแสดงหรือไม่แสดงก็ได้ เช่น SOC และ SIZ ต้องระบุเป็น Marker segment อันดับแรกและอันดับที่สองในส่วนของ Main header อย่างไรก็ตาม PLT (packet-length) marker ไม่อนุญาตให้ปรากฏใน Main header และเป็น Marker ทางเลือกใน Tile-part header รูปที่ 2.5 แสดง Marker segment ต่างๆ โดยสังเขป



รูปที่ 2.5 Main Header Marker Segments

2.1.2.4 Code-stream Organization

ตารางที่ 2.3 File Format JP2

Type	ความหมาย
Start of codestream (SOC)	ตำแหน่งเริ่มของข้อมูล
End of codestream (EOC)	ตำแหน่งสุดท้ายของข้อมูล
Start of tile-part (SOT)	แสดงตำแหน่งของ Tiling
Start of data (SOD)	แสดงตำแหน่งสุดท้ายของ Tiling
Image and tile size (SIZ)	บอกขนาดของรูปภาพ
Coding style default (*COD*)	บอกรูปแบบการ code (e.g., multicomponent transform, wavelet/ subband transform, tier-1/tier-2coding parameters, etc.).
Coding style component (COC)	ระบุ subset ของ code สำหรับ 1component.
Quantization default (QCD)	ระบุค่า(i.e., quantizer type, quantizer parameters).
Quantization component (QCC)	ระบุค่า สำหรับ 1 component
Region of interest (RGN)	ระบุค่า Coding parameters.
Region of interest (RGN)	Optional
Progression order change(POD)	ถ้าใช้ POD จะถูกใช้ใน main หรือ tile-part header
Tile-part length(TLM)	Optional
Packed length(PLM)	Optional
Comments and extension(CME)	Optional

ส่วนของ Tile-part header ที่แสดงในรูป 2.5 ประกอบด้วย Segment ต่างๆที่รู้จักในนาม Tile-part header marker segment บาง Marker จำเป็นที่ต้องแสดงแต่บางตัวอาจจะแสดงหรือไม่แสดงก็ได้ ตารางที่ 2.4 แสดงรายละเอียดคร่าวๆ ของ Marker segment เหล่านั้น

ตารางที่ 2.4 Tile-part Header Marker Segments

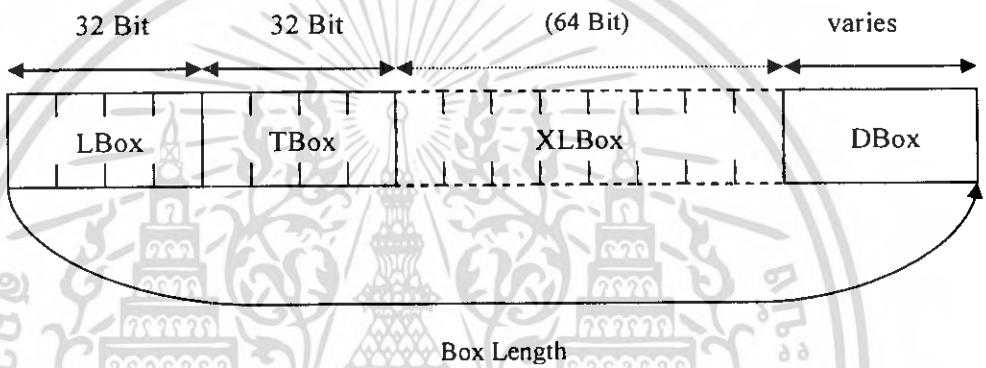
Name	Description	Required/Optional
SOT	*Start of tile*	*Required as the first marker segment of the tile-part header*
COD	Coding style default	Optional and no more than one COD per component
COC	Coding style component	Optional and no more than one COC per component
QCD	Quantization default	Optional and no more than one QCD per component
QCC	Quantization component	Optional and no more than one QCD per component
RGN	Region of interest	*Optional and no more than one QCD per component*
POD	Progression order change	Required if any progression order changes from main POD
PPT	Packed packet headers	Optional, either PPM or PPT or code-stream packed header are required
PLT	Packed length, tile-part length	Optional
CME	Comments and extension	Optional
SOD	Start of data	*Required, marks the beginning of the current tile-part data*

2.1.2.5 File format สำหรับ JPEG2000 Part 1: JP2 format

จากรูปที่ 2.5 จาก SOC (start-of-code-stream) ไปจนถึง EOC(end-of-code-stream)จะเห็นว่า Code-stream ของ JPEG2000 นั้นจะมีพร้อมอยู่แล้ว ส่วนประกอบต่างๆ ของรูปภาพขึ้นอยู่กับกฎต่างๆ ของวากยสัมพันธ์(Syntax) และ Code-stream ที่ได้กล่าวไว้แล้วก่อนหน้านี้ อย่างไรก็ตาม Annex I ของ JPEG2000 Standard Part 1[2] ได้กำหนดรูปแบบไฟล์ทางเลือก(Optional file format) ซึ่งก็คือ JP2 File Format ว่าสามารถใช้ในการบีบอัดข้อมูลภาพ JPEG2000 ได้ เพื่อการจำแนกไฟล์ได้สะดวก ลักษณะ

ไฟล์แบบ ".jp2"หรือ ".JP2" นั้นควรนำมาใช้ในการจัดระบบไฟล์ดังเช่นในระบบไฟล์ของ Macintosh นั้น โค้ดชนิด "jp2" ก็ควรกำหนดให้เป็นไฟล์ JP2

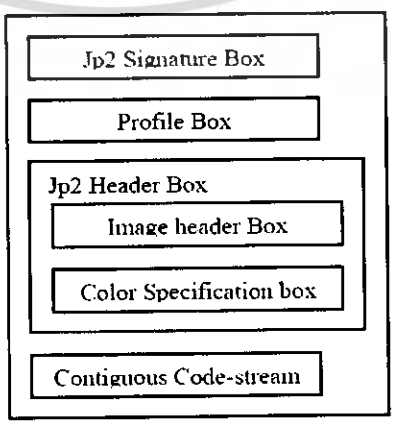
บล็อกที่สร้างขึ้นมาของรูปแบบไฟล์ JP2 จะถูกเรียกว่า Box ซึ่งใช้บรรจุ JPEG2000 code-stream หรือข้อมูลอื่นๆ เช่น คุณสมบัติรูป ลิขสิทธิ์ทรัพย์สินทางปัญญา ข้อมูลการค้า เป็นต้น รูปที่ 2.6 แสดงนิยามของ JP2 Box ซึ่งอาจประกอบด้วย 4 เขตข้อมูล(Field)โดยเขตข้อมูลแรกคือ LBox ระบุความยาวของ Box ซึ่งค่าในแต่ละเขตข้อมูลเก็บไว้ในรูปแบบ 32-bit big-endian unsigned integer ถ้าค่า 0 ถูกระบุในส่วนเขตข้อมูลของ LBox แสดงว่า Box นี้เป็น Box สุดท้ายของไฟล์ ถ้าค่า 1 ถูกระบุในเขตข้อมูลของ LBox แสดงว่าความยาวที่แท้จริงของ Box จะระบุในเขตข้อมูลที่ 3 ซึ่งเป็นเขตข้อมูลเสริมหรือ XLBox ซึ่งเก็บในรูปแบบ 64-bit big-endian unsigned integer เขตข้อมูลที่สองหรือ TBox จะระบุชนิดของข้อมูลที่เก็บอยู่ในเขตข้อมูลสุดท้ายหรือ DBox โดยขนาดของเขตข้อมูล DBox จะแปรผันตามชนิดของ Box



รูปที่ 2.6 นิยามของ JP2 Box

a) File Format Organization

ไฟล์ JP2 ถูกจัดการคล้ายกับเป็นลำดับที่ติดกันไปของ Box ซึ่งบาง Box ต้องใช้งาน แต่บาง Box เป็นทางเลือกเสริมโดยผู้สร้างไฟล์ที่ถูกบีบอัดนั้น รูปที่ 2.7 แสดงโครงสร้างของไฟล์ JP2 กับ Box ที่ต้องใช้เท่านั้น ดังรูป JP2 Signature box ควรจะเป็น Box แรกในไฟล์และควรตามมาด้วย Profile box ในส่วนของ JP2 Header box เป็น Superbox ซึ่งอาจบรรจุไปด้วย Box อื่นๆ จำนวนมาก เช่น Image Header box และ Color Specification box ในส่วนของ Contiguous Code-stream box ไม่ควรปรากฏก่อน JP2 Header box ส่วน box อื่นๆ ที่เป็นทางเลือกเสริมอาจจะไม่พบในไฟล์ JP2 ยกเว้นจะเจาะจงมา อย่างไรก็ตามทุกๆ ข้อมูลควรอยู่ในรูปแบบของ Box



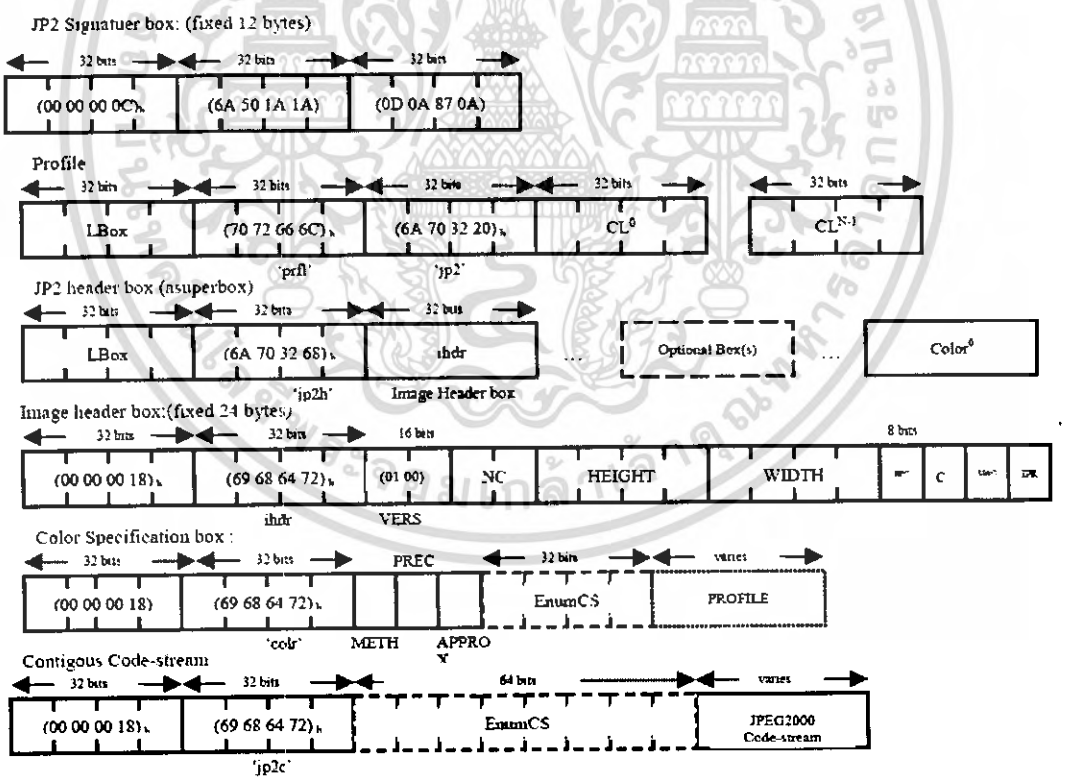
รูปที่ 2.7 โครงสร้างของไฟล์ JP2 กับ Box ที่ต้องใช้เท่านั้น

b) JP2 Required Boxed

ในส่วนนี้ เราจะกล่าวถึงรายละเอียดของ Box ที่ต้องใช้ใน JP2 เท่านั้น สำหรับรายละเอียดของ box ที่เป็นทางเลือกเสริมสามารถอ่านเพิ่มเติมได้จากเอกสาร JPEG2000 standard Part 1 [2] ตารางที่ 2.5 แสดงชื่อและชนิดของ Box ที่จำเป็นต้องใช้ และรูปที่ 2.8 จะแสดงโครงสร้างแบบไบนารี่ของ Box ที่จำเป็นต้องใช้ 16 บิตแรกของแต่ละ Box ประกอบด้วย 2 เขตข้อมูล (LBox และ TBox) ขึ้นอยู่กับความยาวและชนิดของ Box รายละเอียดต่างๆ สามารถอธิบายได้ดังนี้

ตารางที่ 2.5 JP2 Required Boxes

Name	Type (TBox field in the box)
JP2 Signature box	'jp\032\032'(6A 50 1A 1A)$_h$
Profile box	'prfl'(70 72 66 6c)$_h$
JP2 Header box	'jp2h'(6A 70 32 68)$_h$
Image Header box	'ihdr'(69 68 64 72)$_h$
Color Specification box	'colr'(63 6F 6C 72)$_h$
Contiguous Code-stream Box	'jp2c'(6A70 32 63)$_h$



รูปที่ 2.8 JP2 Required Boxes Definition

JP2 Signature box: เป็น Box ซึ่งใช้ยืนยันไฟล์โดยไม่ซ้ำกัน และจะเป็น Box แรกในไฟล์ซึ่งอยู่ในรูป Fixed-length 12-byte string โดยมีค่าเป็น (00 00 00 0C 6A 50 1A 1A 0D 0A 87 0A)$_h$

Profile box: Box นี้ควรอยู่ต่อจาก JP2 Signature box ชนิดของ Box นี้ในเขตข้อมูล TBox เป็น 'prfl' = $(70\ 72\ 66\ 6C)_h$ และในเขตข้อมูล DBox ควรประกอบด้วย Brand (BR) และ N compatibility list information (CL^i เมื่อ $0 <= i <= N-1$) ทั้ง BR และ CL^i ถูกเข้ารหัสแบบ ASCII จำนวน 4 byte string สำหรับไฟล์ JP2 ค่าของ BR ต้องเท่ากับ 'jp2\040' = $(6A\ 70\ 32\ 20)_h$ และต้องมี CL^i อย่างน้อยหนึ่งค่า ซึ่งค่า N บอกจำนวนของเขตข้อมูล CL^i ซึ่งดูได้จากความยาวของ box นี้

JP2 Header box: box นี้เป็น Superbox อันประกอบด้วย Box ต่างๆมากมาย ซึ่งอาจจะเป็น Image Header box และอย่างน้อยที่สุด คือ Color Specification box บรรจุอยู่ภายใน JP2 Header box ชนิดของ Box นี้คือ 'jp2h' = $(6A\ 70\ 32\ 68)_h$ และควรอยู่ต่อจาก JP2 Signature box แต่อยู่ก่อน Contiguous Code-stream box

Image Header box: Box นี้มี fixed-length 24 byte เก็บข้อมูลทั่วไปของรูปภาพ ตัวอย่างเช่น ความกว้าง ความยาว และจำนวนของส่วนประกอบ โดยมีชนิดของ Box เป็น 'ihdr' = $(69\ 68\ 64\ 72)_h$ และเขตข้อมูลของ DBox จะถูกแบ่งออกเป็นเขตข้อมูลย่อยๆ 8 เขตข้อมูลดังที่ได้อธิบายไว้ในตารางที่ 2.6

Color Specification box: Box นี้จะระบุพื้นที่สีเพื่อใช้ในการขยายข้อมูลที่ถูกบีบไว้นั่นเองโดยมีชนิดของ Box เป็น 'colr' = $(63\ 6F\ 6C\ 72)_h$ และในเขตข้อมูลของ DBox จะเพิ่มขึ้นเป็น 5 เขตข้อมูลดังที่ได้อธิบายไว้ในตารางที่ 2.7

Contiguous Code-stream box: Box นี้บรรจุ JPEG2000 code-stream ที่ถูกต้องและสมบูรณ์ และชนิดของ Box นี้ คือ 'j2pc' = $(6A\ 70\ 32\ 63)_h$

ตารางที่ 2.6 Format of the Contents of the Image Header Box

Field name	Description	Size (bytes)	Value
VERS	Major/minor version number	2	$(01\ 00)_h$
NC	Number of components	2	$1-(2^{16}-1)$
HEIGHT	Image height	4	$1-(2^{32}-1)$
WIDTH	Image width	4	$1-(2^{32}-1)$
BPC	Bits per component	1	-127 – 127
C	Compression type	1	7
UnkC	Color space unknown	1	0 – known 1 - unknown
IPR	Intellectual property	1	0 or 1

ตารางที่ 2.7 Format of the Contents of the Color Specification Box

Field name	Description	Size (bytes)	Value
METH	Specification method	1	1-Enumerated 2-Reatricted ICC profile
PREC	Precedence	1	0
APPROX	Color space approximation	1	0
EnumCs	Enumerated color space	4(MTH=1) 0(MTH=2)	$0(2^{32}-1)$ (16/17 for a RGB /grayscale[4]) nonexistent
PROFILE	ICC profile[3]	varies	varies

2.1.2.6 ตัวอย่างของ JPEG2000

ในส่วนนี้จะแสดงตัวอย่างของ JPEG2000 compressed code-stream ด้วยรูปแบบ JP2 ตัวอย่างนี้ใช้ภาพ 24-bit RGB โดยมีความกว้างและความยาวเท่ากับ 40X30 = $(28)_h$ x $(1E)_h$ ดังที่ได้แสดงไปแล้วจะเห็นว่าไฟล์ JP2 มีทั้งหมด 6 Box ที่จำเป็นต้องใช้และอีก 2 Box ที่เป็นทางเลือกเสริม (Resolution และ Capture Resolution Box) ซึ่งจะใช้เพื่อการอ้างอิง JP2 box เริ่มด้วย 4-byte box length และตามด้วย 4-byte box type เสมอ ตัวอย่างเช่น Resolution box มี 26 ไบท์(byte)ซึ่งแสดงโดย 4 ไบท์แรกในตัวอย่างนี้ตามด้วย 4-byte box type 'res' = $(72\ 65\ 73\ 20)_h$ โดย Box นี้เป็น Superbox ด้วยเช่นกัน ซึ่งประกอบไปด้วย Capture Resolution box 18 ไบท์ความยาวทั้งหมดของ Resolution box นี้มีค่าเท่ากับ 4+4+18 = 26 ไบท์ ในส่วนของ Contiguous Code-stream box ซึ่งค่าทั้งหมดเป็น 0 นั่นหมายความว่า Box นี้เป็น Box สุดท้ายของไฟล์ ค่า $(FF\ 4F)_h$ แสดง Start of code-stream (SOC marker) และ $(FF\ D9)_h$ แสดง End of code-stream (EOC marker) เหล่านี้เป็นเพียงส่วนหนึ่งใน Code-stream เท่านั้นดังที่ได้แสดงไว้ใน ไบท์สุดท้ายของ SOT marker ขนาดของภาพ 40X30 = $(28)_h$ x $(1E)_h$ สามารถพบได้ใน Image Header box หรือ SIZ marker segment

```
00 00 00 0C => JP2 Sigature box (12 bytes)
6A 50 1A 1A 0D 0A 87 0A

00 00 00 14 => Profile box (20 bytes)
70 72 66 6C 6A 70 32 20
00 00 00 00 6A 70 32 20

00 00 00 47 => JP2 Header box (73 bytes)
6A 70 32 68

00 00 00 18 ==>>Image Header box (24 bytes)
69 68 64 72  01 00 00 03
00 00 00 1E 00 00 00 28
07 07 00 00

00 00 00 0F == >> Color Specification box (15 bytes)
663 6F 6C 72 01 00 00 00 00 00 10
**************Optional boxes**********************
00 00 00 1A ==>> Resolution box (26 bytes)
72 65 73 20

00 00 00 12 ==>>Capture Resolution box (18 bytes)
72 65 73 63 00 48 00 FE 00 48 00 FE 04 04
************************************************
00 00 00 00 ==> Contiguous Color-stream box (last box)
6A 70 32 63

FF 4F (SOC)---> start of code-stream

FF 51 (SIZ)
00 2F 00 00 00 00 00 28 00 00 00 1E 00 00 00 00
00 00 00 00 00 00 00 28 00 00 00 1E 00 00 00 00
00 00 00 00 00 03 07 01 01 07 01 01 07 01 01
FF 5C (QCD)
00 0D 40 40 48 48 50 48 48 50 48 48 50

FF 52 (COD)
00 0C 00 00 00 01 01 03 04 04 00 01

FF 64 (CME)
00 0C 00 01 41 56 4C 54 5F 31 31 34

FF 90 (SOT)
00 0A 00 00 00 00 09 49 00 01

FF 93 (SOD)--->Start of tile-part

.....
FF D9 (EOC)---> End of code-stream
```

รูปที่ 2.9 JPEG2000 compressed code-stream โดยใช้ภาพขนาด 24-bit RGB

2.2 Preprocessing

เป็นขั้นตอนของการเตรียมรูปภาพให้เหมาะสมกับการแปลงรูปภาพไปในอีกรูปแบบหนึ่งดังตัวอย่างนี้คือให้เหมาะกับขนาดของหน่วยความจำของเราแล้วปรับค่าของขนาด(Magnitude) ให้อยู่ในช่วงที่สามารถคำนวณได้

2.2.1 Tiling

คือการแบ่งขนาดของภาพให้เหมาะสมกับการคำนวณหรืออาจเหมาะกับขนาดของหน่วยความจำที่มีอยู่อย่างจำกัดหรือเพื่อความเร็วบางอย่างที่ทำแบบภาพเล็กๆ จะเร็วกว่า ในขั้นตอนนี้จะแบ่งภาพให้มีขนาดเท่าๆ กัน โดยภาพจะไม่มีการทับซ้อนกันและไม่ขึ้นต่อกันโดยถ้ากำหนดขนาดของการแบ่งใหญ่กว่าภาพจะต้องมีการ Path ในส่วนที่เกินออกจากภาพ โดยแสดงดังรูปที่ 2.10

รูปที่ 2.10 การทำ Tiling Partition

2.2.2 Level Offset

เป็นการแปลงค่าจากค่าที่เป็นแบบไม่มีเครื่องหมาย (Unsigned) ไปเป็นค่าที่มีเครื่องหมาย (Signed) เพื่อให้เหมาะสมและสามารถคำนวณทางคณิตศาสตร์ได้

ดังเช่น Values -> $-2^{B-1} <= x[n] < 2^{B-1}$

เมื่อ B คือค่ายกกำลัง

$x[n]$ คือค่าของจำนวนใดๆในแต่ละตำแหน่งของภาพ

2.3 Forward Intercomponent Transform

เป็นการแปลงค่าของสีใน Special Domain แบบ R, G, B ไปเป็นการแสดงค่าของแสงส่องกระทบ (Y-Luminance) กับส่วนของสี (Cb,Cr – Color component) คือ R->Y, G->Cb, B->Cr หรือ YCbCr

Y-Luminance

Y,Cb,Cr

Cb,Cr – Color component

2.3.1 Irreversible Color Transform (ICT)

การกระทำนี้มีสูตรในการทำแบบตายตัวคือคูณด้วยเมตริกซ์(Matrix) ขนาด [3*3] โดยค่าในเมตริกซ์นั้นได้มีการพิสูจน์และกำหนดมาให้เป็นค่าที่เหมาะสมแล้วซึ่งค่าที่อยู่ในเมตริกซ์นั้นจะต้องเป็นคู่กันกับการแปลงกลับ การแปลงไปโดยจากรูปที่ 2.11 และการแปลงกลับจากรูปที่ 2.12

$$\begin{bmatrix} V_0(x,y) \\ V_1(x,y) \\ V_2(x,y) \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} U_0(x,y) \\ U_1(x,y) \\ U_2(x,y) \end{bmatrix}$$

เมื่อ $V_0 = Y$ $U_0 = R$

$V_1 = Cb$ $U_1 = G$

$V_2 = Cr$ $U_2 = B$

รูปที่ 2.11 การแปลง RGB-to-YCbCr

$$\begin{bmatrix} U_0(x,y) \\ U_1(x,y) \\ U_2(x,y) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34413 & -0.71414 \\ 1 & -1.772 & 0 \end{bmatrix} \begin{bmatrix} V_0(x,y) \\ V_1(x,y) \\ V_2(x,y) \end{bmatrix}$$

เมื่อ $U_0 = R$ $V_0 = Y$

$U_1 = G$ $V_1 = Cb$

$U_2 = B$ $V_2 = Cr$

รูปที่ 2.12 การแปลง YCbCr-to- RGB

2.4 Forward Intracomponent Transform

เป็นการแปลงภาพจาก Special Domain ไปเป็นในรูปแบบของความถี่ (Frequency) โดยแบ่งช่วงความถี่เป็น Subband ที่รู้จักกันดีคือการทำเวฟเล็ต (Wavelet) ซึ่งจะสามารถทำได้ด้วยกันหลายแบบ ในที่นี้ขอเสนอในแบบของ Haar Wavelet [2.4.4] และ Daubechies4 Wavelet [2.4.5]

2.4.1 ประวัติ Discrete Wavelet Transform (DWT)

การวิเคราะห์เวฟเล็ตได้ถูกคิดค้นและพัฒนาจากนักคณิตศาสตร์ในปี ค.ศ. 1980 หลังจากนั้นได้ถูกนำมาใช้งานกันอย่างแพร่หลายในปี 1990 เพราะเวฟเล็ตสามารถที่ใช้ในการวิเคราะห์สัญญาณต่างๆ หรือแม้กระทั่งการวิเคราะห์ภาพหรือนำไปใช้ในการบีบอัดข้อมูล ข้อมูลที่ได้จากการแปลงเวฟเล็ตจะถูกจัดเรียงใหม่ โดยทำการแยกค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่สูงและความถี่ต่ำออกจากกัน ค่าสัมประสิทธิ์ที่ได้จากการแปลงเวฟเล็ตทั้งหมดจะมีค่าเท่ากับข้อมูลอินพุท ตัวอย่างการนำเวฟเล็ตไปประยุกต์ใช้งานเช่น การเก็บข้อมูลลายนิ้วมือของ FBI

2.4.2 การเลือกเวฟเล็ตเพื่อนำไปใช้งาน

การเลือกเวฟเล็ต (Wavelet)เพื่อนำไปใช้งานจะต้องพิจารณาจากหลายๆ ด้าน เพื่อให้เหมาะกับงานที่จะนำไปใช้ไม่ว่าจะเป็นฟังก์ชั่นที่ใช้กับการแปลงไป (Decomposition) และฟังก์ชั่นที่ใช้กับแปลงกลับ (Reconstruction) ค่า PSNR หรือแม้แต่ค่า Zero Moment ถ้าหากพิจารณาจากฟังก์ชั่นที่ใช้ในการแปลงเวฟเล็ตจะต้องคำนึงถึงความซับซ้อนในการคำนวณ และอัตราการสูญเสียของข้อมูล ถ้าหากต้องการความถูกต้องของข้อมูลจะทำให้มีการคำนวณที่ซับซ้อน เนื่องจากค่าสัมประสิทธิ์ที่ได้จากการคำนวณจะอยู่ในรูปของจำนวนจริง (Floating Point) แต่หากต้องการลดความซับซ้อนในการคำนวณลง โดยยอมให้ข้อมูลเกิดการสูญเสียได้ ก็สามารถใช้การคำนวณหาค่าสัมประสิทธิ์แบบเลขจำนวนเต็มได้ (Integer Wavelet)

ค่า PSNR (Peek Signal to Noise Ratio) เป็นค่าที่ใช้ในขั้นตอนการวัดประสิทธิภาพของอัลกอริธึม จะดูจากคุณภาพของภาพที่ได้เป็นหลัก ค่าที่ PSNR หาได้จากสมการที่ (2.1)

$$PSNR = 20\log_{10}\left(\frac{A}{RMSE}\right) \tag{2. 1}$$

โดยที่ ค่า RMSE หาได้จากสมการที่ (2.2)

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

เมื่อ A คือ จำนวน Gray level

N คือ จำนวน Pixel ในภาพ

$p(i)$ คือ ค่าจุดในภาพต้นฉบับ

$p'(i)$ คือ ค่าจุดในภาพที่ผ่านการเข้ารหัส

ค่า Zero Moment เป็นการนับจำนวนของข้อมูลที่มีค่าเป็นศูนย์ หรือข้อมูลที่ต่ำกว่าระดับที่ตั้งเอาไว้ (Threshold) ของข้อมูลทั้งหมดที่ได้จากการแปลงเวฟเล็ต (Wavelet Transform) ค่า Zero Moment จะใช้ในการวัดประสิทธิภาพของการบีบอัดข้อมูล ค่า Zero Moment หาได้จากสมการ (2.3)

$$Zero\ \ Moment = \sum_{x=1}^{M}\sum_{y=1}^{N} Z_{coef}(x,y) \tag{2.3}$$

โดยที่ Z_{coef} หาได้จากสมการที่ (2. 4)

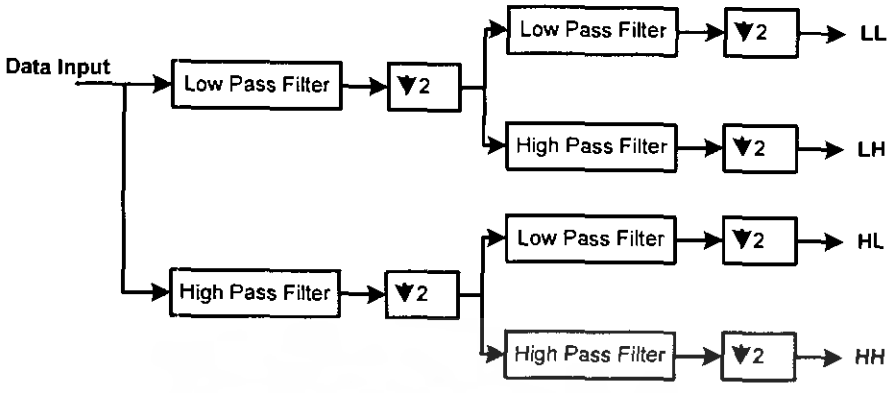
$$Z_{coef}(x,y) = \begin{cases} 1 & |w(x,y)| \le Threshold \\ 0 & Otherwise \end{cases} \tag{2.4}$$

เมื่อ $w(x,y)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงเวฟเล็ตที่ตำแหน่งต่างๆ

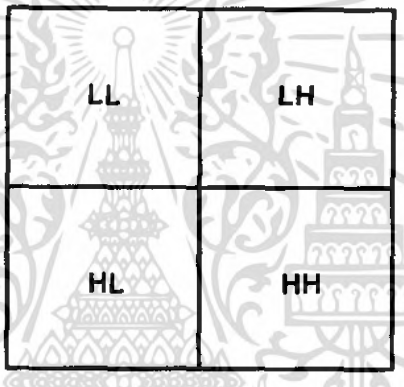
$Threshold$ คือ ค่าระดับของข้อมูลที่ตั้งเอาไว้ (ในที่นี้ใช้ 1.5)

2.4.3 การแปลงเวฟเล็ต (Wavelet) กับข้อมูลภาพ

ข้อมูลภาพมีลักษณะเป็นข้อมูล 2 มิติ ดังนั้นการทำการแปลงเวฟเล็ตกับภาพจะเป็นการทำการแปลงเวฟเล็ตในทางแนวแกน x และแกน y สลับกันในแต่ละครั้งสามารถเขียนเป็นแผนผังได้ดังรูปที่ 2.13



รูปที่ 2.13 แสดงแผนผังของการแปลงเวฟเล็ต



รูปที่ 2.14 แสดงลักษณะของการแบ่งแบนด์ย่อยของภาพ

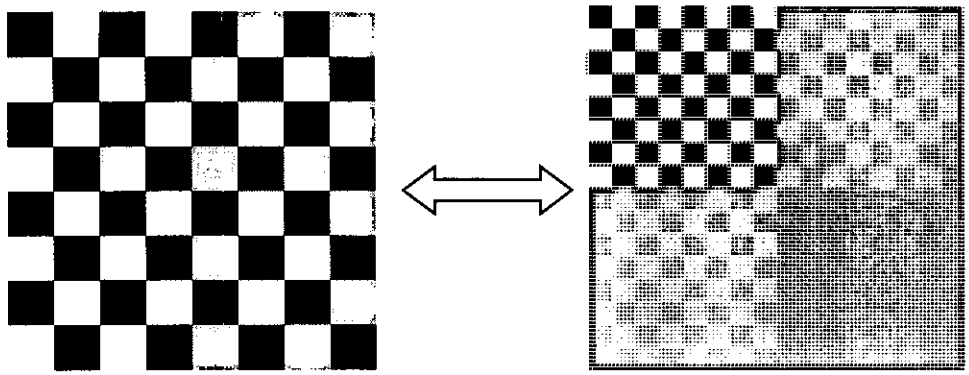
ลักษณะของข้อมูลที่ได้จากการแปลงเวฟเล็ต แบบ 2 มิติ จะถูกแบ่งออกเป็น 4 แบนด์ย่อย ภาพที่ได้จะมีลักษณะดังรูปที่ 2.14 โดยแต่ละแบนด์ย่อยจะมีคุณสมบัติดังนี้

LL: คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากการผ่านตัวกรองความถี่ต่ำสองครั้งและรายละเอียดของข้อมูลภาพส่วนใหญ่จะอยู่บริเวณนี้

LH: คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่ต่ำในแนวตั้งแล้วนำผลลัพธ์ที่ได้ไปผ่านตัวกรองความถี่สูงในแนวนอนส่วนนี้จะทำการเก็บข้อมูลในแนวตั้ง

HL: คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่สูงในแนวตั้งแล้วนำผลลัพธ์ที่ได้ไปผ่านตัวกรองความถี่ต่ำในแนวนอนส่วนนี้จะทำการเก็บข้อมูลในแนวนอน

HH: คือ ส่วนที่เก็บค่าสัมประสิทธิ์ที่ได้จากการผ่านตัวกรองความถี่สูงสองครั้งในส่วนนี้จะเก็บข้อมูลในแนวทแยงมุม และส่วนนี้จะมีความสำคัญน้อยที่สุด



(a) (b)

รูปที่ 2.15 แสดงภาพตัวอย่างการแปลงเวฟเล็ต (a) ภาพต้นฉบับ (b) ภาพที่ได้จากแปลงเวฟเล็ต

2.4.4 Haar Wavelet

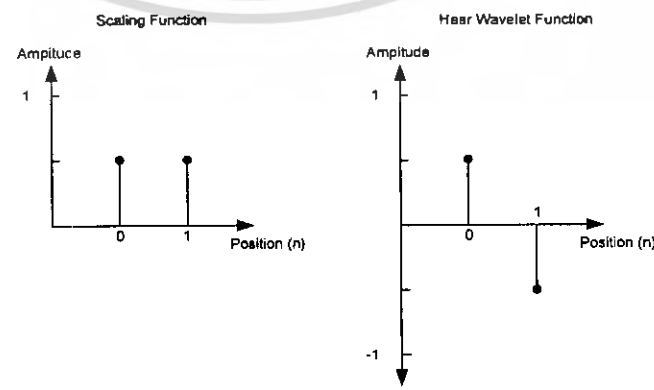
การแปลงทำวิธีนี้ง่ายต่อการทำความเข้าใจมากแต่ไม่ได้เป็นวิธีที่นิยมกัน การแปลงเวฟเล็ตแบบ Haar เป็นวิธีที่สามารถทำความเข้าใจได้ง่าย เพราะว่ามีการคำนวณที่ไม่ซับซ้อนมากนัก

สมการที่ใช้ในการแปลงเวฟเล็ตโดยทั่วไปสามารถที่จะแบ่งออกเป็น 2 ส่วน คือ Scaling Function และ Wavelet Function ดังสมการที่ 2.5 และ 2.6 ตามลำดับ

$$h_n = \begin{cases} \frac{1}{2} & n = 0,1 \\ 0 & Otherwise \end{cases} \tag{2.5}$$

$$g_n = \begin{cases} \frac{1}{2} & n = 0 \\ -\frac{1}{2} & n = 1 \\ 0 & Otherwise \end{cases} \tag{2.6}$$

เมื่อ h_n คือ Scaling Function และ g_n คือ Wavelet Function แบบ Haar ถ้านำสมการทั้งสองไปเขียนเป็นกราฟจะมีลักษณะดังแสดงในรูปที่ 2.16

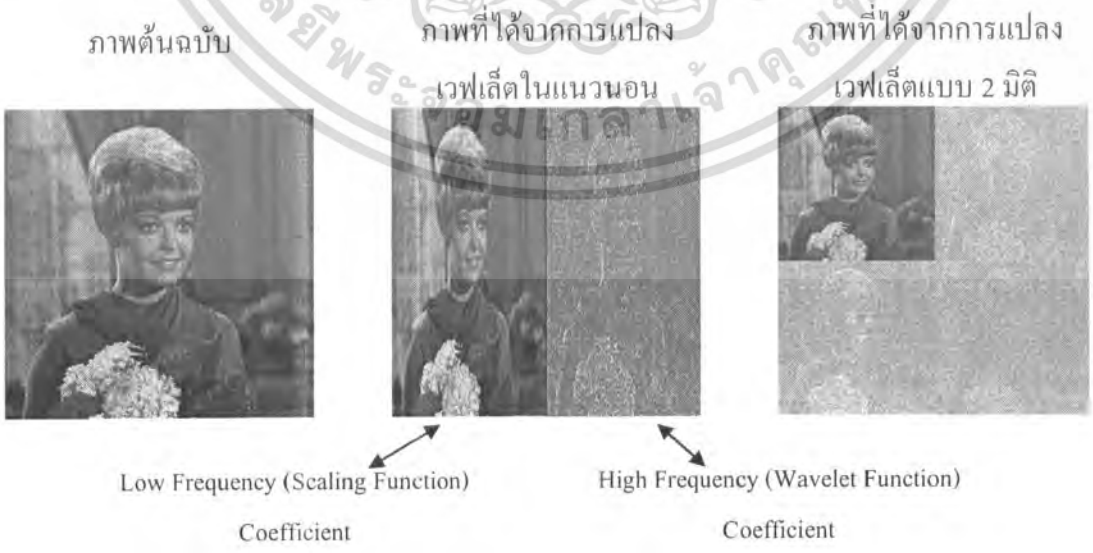


รูปที่ 2.16 แสดงรูปสัญญาณที่ใช้ในการแปลงเวฟเล็ตแบบ Haar

Original Pixel Values								Scaling Function Coefficients				Wavelet Function Coefficients			
0	1	2	3	4	5	6	7	$\frac{(0+1)}{2}$	$\frac{(2+3)}{2}$	$\frac{(4+5)}{2}$	$\frac{(6+7)}{2}$	$\frac{(0-1)}{2}$	$\frac{(2-3)}{2}$	$\frac{(4-5)}{2}$	$\frac{(6-7)}{2}$

รูปที่ 2.17 แสดงการนำ Haar เวฟเล็ตไปประยุกต์ใช้งานกับข้อมูลที่เป็นภาพ

การประยุกต์ใช้ Haar เวฟเล็ตฟังก์ชั่นกับข้อมูลที่เป็นภาพ สามารถแสดงในรูปที่ 2.17 ซึ่งเป็นการแปลงเวฟเล็ตในแนวนอน (Horizontal) ค่าสัมประสิทธิ์ที่ได้จากการแปลงจะประกอบด้วย ค่าสัมประสิทธิ์ที่ได้จาก Scaling Function ซึ่งก็คือ ค่าสัมประสิทธิ์ที่ได้จากตัวกรองความถี่ต่ำ (Low-Frequency Coefficient) และค่าสัมประสิทธิ์ของ Wavelet Function คือ ค่าสัมประสิทธิ์ที่ได้ตัวกรองความถี่สูง (High-Frequency Coefficient) การหาค่าสัมประสิทธิ์ของการแปลงเวฟเล็ตแบบ Haar ค่าที่ได้จาก Scaling Function คือ ค่าเฉลี่ยของผลรวมของสองพิกเซล (Pixel) ส่วนสัมประสิทธิ์ที่ได้ของเวฟเล็ตฟังก์ชั่นจะเป็นการหาค่าเฉลี่ยของความแตกต่างของสองพิกเซล เมื่อทำการแปลงเวฟเล็ตไปประยุกต์ใช้กับข้อมูลภาพจะทำการแปลงเวฟเล็ตในแนวนอนและแนวตั้ง โดยแสดงผลลัพธ์ที่ได้ในรูปที่ 2.18



รูปที่ 2.18 ภาพผลลัพธ์ที่ได้จากการแปลงเวฟเล็ตของข้อมูลภาพ

เมื่อทำความเข้าใจเกี่ยวกับการแปลงเวฟเล็ตโดยใช้ฟังก์ชั่น Haar เป็นที่เรียบร้อยแล้วในหัวข้อถัดไปจะกล่าวถึงการแปลงเวฟเล็ตที่มีความซับซ้อนมากขึ้นคือ Daubechies4 (D4) ซึ่งเป็นวิธีที่นิยมใช้ในการบีบอัดข้อมูลหรือใช้ในการวิเคราะห์ภาพ

2.4.5 Daubechies4 Wavelet (D4)

การแปลงเวฟเล็ตแบบ Daubechies4 (D4) จะมีฟิวเตอร์ทั้งหมด 4 แท็ป (4 Tap Filters) ซึ่งประกอบไปด้วยค่า Scaling Function และ Wavelet Function จำนวน 4 ค่า และค่าสัมประสิทธิ์ของ Scaling Function ประกอบไปด้วยค่าตามสมการ 2.7 ถึง 2.10

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}} \tag{2.7}$$

$$h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}} \tag{2.8}$$

$$h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}} \tag{2.9}$$

$$h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}} \tag{2.10}$$

Wavelet Function จะประกอบไปด้วยค่าตามสมการ 2.11 ถึง 2.14

$$g_0 = h_3 \tag{2.11}$$

$$g_1 = -h_2 \tag{2.12}$$

$$g_2 = h_1 \tag{2.13}$$

$$g_3 = -h_0 \tag{2.14}$$

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

ในการคำนวณหาค่าสัมประสิทธิ์ของ Scaling และ Wavelet Function แต่ละครั้งจะต้องการข้อมูลอินพุต 4 จำนวน ซึ่งสามารถเขียนเป็นสมการได้ดังต่อไปนี้

Daubechies4 Scaling Function

$$a[i] = h_0 S[2i] + h_1 S[2i+1] + h_2 S[2i+2] + h_3 S[2i+3] \tag{2.15}$$

Daubechies4 Wavelet Function

$$c[i] = g_0 S[2i] + g_1 S[2i+1] + g_2 S[2i+2] + g_3 S[2i+3] \quad (2.16)$$

ในการแปลงเวฟเล็ตของข้อมูลที่มีขนาดจำกัดจะทำการคำนวณได้จนถึงตำแหน่งที่ $N-4$ เพราะเมื่อทำการคำนวณต่อไปจะมีปัญหาเนื่องจากข้อมูลที่ใช้ในการคำนวณประกอบด้วย $S[N-2], S[N-1], S[N], S[N+1]$ แต่ $S[N]$ และ $S[N+1]$ หาค่าไม่ได้ (ทั้งสองตำแหน่งนี้เป็นตำแหน่งที่มีค่ามากกว่าอาร์เรย์) ปัญหาที่เกิดขึ้นนี้สามารถแสดงให้อยู่ในรูปของเมตริกซ์ (2.17) ได้ดังนี้

Daubechies D4 forward transform matrix for an 8 element signal

$$\begin{bmatrix} a_0 \\ c_0 \\ a_1 \\ c_1 \\ a_2 \\ c_2 \\ a_3 \\ c_3 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 \end{bmatrix} \bullet \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \end{bmatrix} \quad (2.17)$$

การแปลงข้อมูลกลับคืนมา(Inverse Transform) จะเกิดปัญหาคล้ายกับการแปลงเวฟเล็ตแต่ปัญหาดังกล่าวจะเกิดกับค่าเริ่มต้นสองค่าคือ ในการคำนวณค่าสัมประสิทธิ์ครั้งแรกจะใช้ค่าข้อมูล $a[-1], c[-1], a[0]$ และ $c[0]$ ดังแสดงในเมตริกซ์ (2.18) ดังต่อไปนี้

Daubechies D4 inverse transform matrix for an 8 element transform result

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \end{bmatrix} = \begin{bmatrix} h_2 & g_1 & h_0 & g_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_3 & g_3 & h_1 & g_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_2 & g_2 & h_0 & g_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_3 & g_3 & h_1 & g_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_2 & g_2 & h_0 & g_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_3 & g_3 & h_1 & g_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_2 & g_2 & h_0 & g_0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_3 & g_3 & h_1 & g_1 \end{bmatrix} \bullet \begin{bmatrix} a_0 \\ c_0 \\ a_1 \\ c_1 \\ a_2 \\ c_2 \\ a_3 \\ c_3 \end{bmatrix} \quad (2.18)$$

ปัญหาดังกล่าวนี้สามารถแก้ไขได้ 3 วิธี

1. แทนที่ข้อมูลที่หายไปด้วยค่ามูลตัวสุดท้ายหรือข้อมูลตัวก่อนหน้า

2. ทำการแทนข้อมูลที่หายไปในลักษณะของทรงกระบอก ตัวอย่างเช่น ข้อมูลลำดับที่ -1 ก็จะนำข้อมูลตัวที่ N มาใช้ในการคำนวณ

3. ทำการออกแบบ Scaling Function และ Wavelet Function พิเศษที่ใช้ในการคำนวณบริเวณขอบของชุดข้อมูล

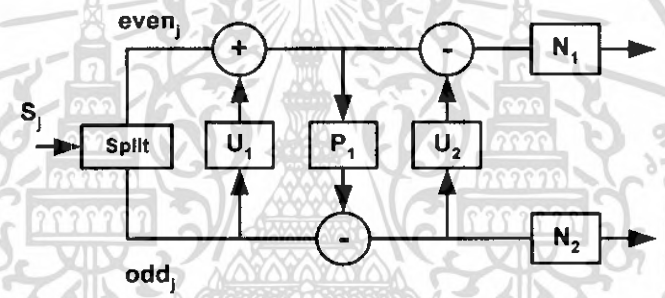
หมายเหตุ: การนำข้อมูลที่มีค่าเป็นศูนย์มาแทนที่ข้อมูลที่หายไปก็สามารถทำได้แต่ว่าอาจทำให้เกิดการสูญเสียมากกว่าวิธีอื่น

2.4.6 การแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธีการของ Lifting

การแปลงเวฟเล็ตแบบ Lifting ได้พัฒนาโดย Wim Sweldens แนวคิดของการแปลงเวฟเล็ตแบบ Lifting มีประโยชน์เกี่ยวกับการจัดการหน่วยความจำเพราะว่าการแปลงเวฟเล็ตในลักษณะนี้จะไม่ต้องการหน่วยความจำสำหรับพักข้อมูล การแปลงเวฟเล็ตแบบ Daubechies4 (D4) ที่จะนำเสนอต่อไปจะมีลักษณะการทำงานดังแสดงในบล็อกไดอะแกรมรูปที่ 2.19 และขบวนการแปลงข้อมูลกลับคืนมา จะใช้การกลับสถานะของขบวนการแปลงเวฟเล็ตทำได้โดยการสลับเครื่องหมายรูปที่ 2.20 รายละเอียดในส่วนนี้จะได้กล่าวถึงในหัวข้อถัดไป

2.4.6.1 การแปลงเวฟเล็ต (Forward Transform)

การแปลงเวฟเล็ตแบบ Daubechies4 (D4) โดยใช้วิธีการของ Lifting จะประกอบไปด้วยขั้นตอนการ Update และ Predict ในขั้นตอนของการแปลงเวฟเล็ตจะต้องมีการปรับค่าผลลัพธ์ (Normalization) ของการแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธีการของ Lifting ดังที่ได้แสดงดังรูปที่ 2.19



รูปที่ 2.19 การแปลงเวฟเล็ตแบบ Daubechies4 (D4) ด้วยวิธีการของ Lifting

ขั้นตอน Split จะทำหน้าที่แยกข้อมูลอินพุทที่เข้ามา ลำดับที่เป็นเลขคู่ (even) จะถูกเก็บไว้ในอาร์เรย์ส่วนแรก (S_0 ถึง S_{half-1}) และลำดับที่เป็นเลขคี่ (odd) จะถูกเก็บไว้ในอาร์เรย์ครึ่งหลัง (S_{half} ถึง S_{N-1}) จากสมการที่ใช้ในการแปลงเวฟเล็ต (2.19) ถึง (2.25) โดยที่ $S[half+n]$คือ ลำดับที่เป็นเลขคี่ ส่วน $S[n]$หมายถึง ลำดับที่เป็นเลขคู่ ถึงแม้ว่าในไดอะแกรมรูปที่ 2.19 จะใช้ฟังก์ชั่นในการปรับค่า 2 ฟังก์ชั่น แต่ทั้งสองฟังก์ชั่นจะเป็นส่วนกลับซึ่งกันและกัน ฟังก์ชั่นที่ใช้ในการแปลงคือ

Update 1 (U1):

For n = 0 to half - 1

$$S[n] = S[n] + \sqrt{3}S[half+n] \tag{2.19}$$

Predict (P):

$$S[half] = s[half] - \frac{\sqrt{3}}{4}S[0] - \frac{\sqrt{3}-2}{4}S[half-1] \tag{2.20}$$

For n = 1 to half -1

$$S[half+n]=s[half+n]-\frac{\sqrt{3}}{4}S[n]-\frac{\sqrt{3}-2}{4}S[n-1] \quad (2.21)$$

Update 2 (U2):

For n = 0 to half-2

$$S[n]=S[n]-S[half+n+1] \quad (2.22)$$

$$S[half-1]=S[half-1]-S[half] \quad (2.23)$$

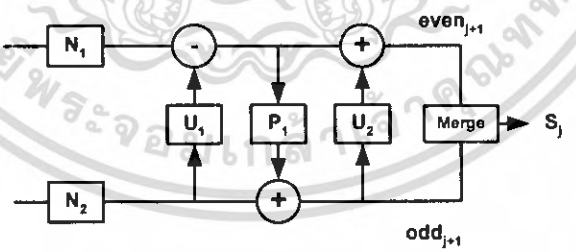
Normalize (N_1 และ N_2):

$$S[n]=\left[\frac{\sqrt{3}-1}{\sqrt{2}}\right]S[n] \quad (2.24)$$

$$S[n+half]=\left[\frac{\sqrt{3}+1}{\sqrt{2}}\right]S[n+half] \quad \text{หรือ} \quad \frac{S[n+half]}{\left[\frac{\sqrt{3}-1}{\sqrt{2}}\right]} \quad 2.25)$$

2.4.6.2 การแปลงกลับเวฟเล็ตแบบ Lifting (Inverse Transform)

ขบวนการในการแปลงข้อมูลกลับคืนมา จะมีลักษณะเดียวกับการแปลงเวฟเล็ตแต่จะทำการกลับสถานะของเครื่องหมายจากบวกเป็นลบและจากลบเป็นบวก ดังแสดงในรูปที่ 2.20



รูปที่ 2.20 แสดงการแปลงกลับของ Daubechies4 (D4) ด้วยวิธีการของ Lifting

หลังจากนั้นจะทำการรวมข้อมูลเข้าด้วยกัน (Merge) ระหว่างข้อมูลที่ได้จากลำดับที่เป็นคู่และลำดับที่เป็นคี่

ฟังก์ชั่นที่ใช้ในการแปลงกลับแบบ Lifting ดังแสดงในสมการ (2.26) ถึง (2.32)

Update 1':

For n = 0 to half - 1

$$S[n]=S[n]-\sqrt{3}S[half+n] \quad (2.26)$$

Predict':

$$S[half] = s[half] + \frac{\sqrt{3}}{4}S[0] + \frac{\sqrt{3}-2}{4}S[half-1] \tag{2.27}$$

For n = 1 to half -1

$$S[half+n] = s[half+n] + \frac{\sqrt{3}}{4}S[n] + \frac{\sqrt{3}-2}{4}S[n-1] \tag{2.28}$$

Update 2':

For n = 0 to half-2

$$S[n] = S[n] + S[half+n+1] \tag{2.29}$$

$$S[half-1] = S[half-1] + S[half] \tag{2.30}$$

Normalize':

$$S[n] = \left[\frac{\sqrt{3}+1}{\sqrt{2}}\right]S[n] \text{ หรือ } \frac{S[n]}{\left[\frac{\sqrt{3}+1}{\sqrt{2}}\right]} \tag{2.31}$$

$$S[n+half] = \left[\frac{\sqrt{3}-1}{\sqrt{2}}\right]S[n+half] \tag{2.32}$$

การแปลงเวฟเล็ตที่นำเสนอข้างต้นจะมีรูปแบบการคำนวณในลักษณะของเลขจำนวนจริงทั้งหมดทำให้สามารถกู้ข้อมูลกลับคืนกลับมาได้เหมือนข้อมูลต้นฉบับ แต่ถ้าหากต้องการลดความซับซ้อนในการคำนวณลงและยอมให้เกิดการสูญเสียของข้อมูลได้ก็สามารถใช้การแปลงเวฟเล็ตแบบเลขจำนวนเต็มได้ (Integer Wavelet Transform)

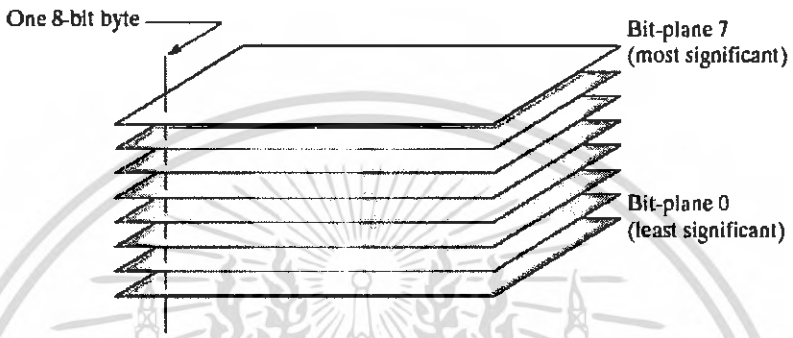
2.5 Quantization

เป็นการลดขนาดของจำนวนบิตในการแสดงรูปภาพซึ่งมีที่รู้จักด้วยกันสองแบบคือ 1.Quantization แบบการตัดบิตนัยสำคัญต่ำๆ ทิ้งไป 2.การ Uniform Quantization เป็นที่แน่นอนว่า การ Quantization นั้นทำให้ข้อมูลหายไปและไม่สามารถกู้กลับคืนมาได้ แต่มันจะให้ผลของการลดจำนวนข้อมูลได้สูงมาก 1:100 หรือมากกว่าแล้วแต่ระดับ Threshold ที่เราตั้งไว้

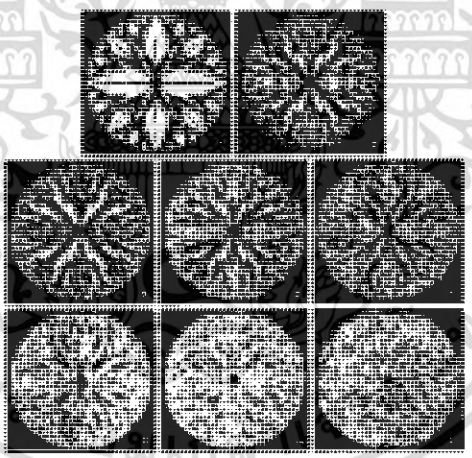
2.5.1 Quantization แบบการตัดบิตนัยสำคัญต่ำๆ ทิ้งไป

วิธีนี้ไม่ได้นำมาใช้ในมาตรฐาน JPEG2000 นี้แต่มันเป็นวิธีที่ง่ายต่อการทำความเข้าใจมาก เพราะมันมีรูปแบบที่ง่าย สามารถทำในภาพที่เป็น Special Domain ได้เท่านั้นโดยจะให้ตัวอย่างในภาพ

แบบ Gray Level ที่มีการแสดงผลในแต่ละจุดพิกเซลขนาดระดับที่ 255 ระดับคือใช้ข้อมูล 8 บิต(bit)ในการแสดงผลแต่ละจุด โดยถ้าหากเราทำการสไลด์ (Slide) ภาพโดยกำหนดแต่ละเลเยอร์ (Layer) โดยเรียงจาก Bit-plane layer 0ไปถึง Bit-plane layer 7 ดังรูปที่ 2.21 จะเห็นว่าภาพนั้นถูกแยกอีกเป็นบิตเพลนซึ่งในแต่ละเลเยอร์จะประกอบไปด้วยค่าสองระดับเรียงปะปนกันคือ 0 และ 1 เท่านั้น และจากรูปที่ 2.22 พบว่าภาพที่มีความสำคัญต่อการมองเห็นนั้นจะอยู่ในช่วงบิตที่สูงๆและจะคงเหลือรายละเอียดอยู่ที่บิตต่ำๆลงไปถ้าเราตัดในส่วนของบิตที่ 0 ออกไปภาพที่ได้ จะไม่มีผลต่อการมองเห็นด้วยตามนุษย์มากนักแต่มีผลต่อการลดข้อมูลได้มาก



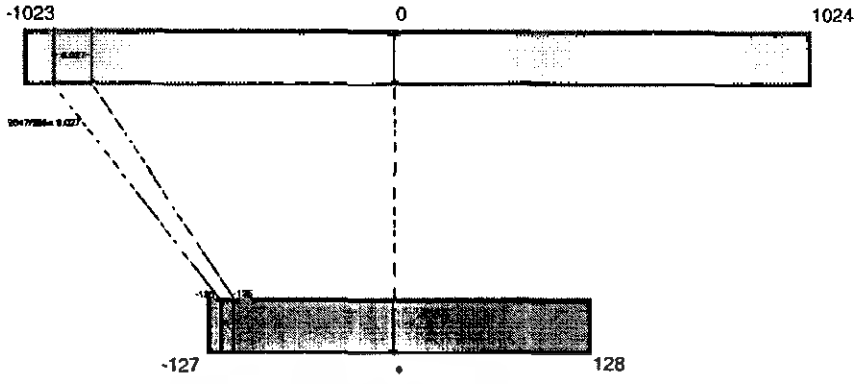
รูปที่ 2.21 สไลด์ (Slide) ภาพ Gray level ขนาด 8 บิต



รูปที่2.22 ภาพจากด้านบนของแต่ละ Slide Layer

2.5.2 การ Uniform Quantization

วิธีนี้ใช้ในมาตรฐาน JPEG2000 โดยวิธีการ Mapping อธิบายได้ดังนี้คือถ้าหากข้อมูลที่เรามีนั้นในแต่ละจุดของพิกเซลมีค่าของข้อมูลอยู่ในระยะห่างที่กว้างมากๆ อาจอยู่ในช่วง (-1023) ไปจนถึงช่วง 1024 เราจำเป็นจะต้องใช้จำนวนของบิตข้อมูลขนาด 2^{10} คือ 10 บิตต่อ 1 จุดภาพแต่เราต้องการที่จะลดให้เหลือแค่เพียง 8 บิตคือ 2^{8} ดังนั้นค่าที่เราจะนำมาจับคู่ (Match) กันนั้นจะต้องบังคับให้อยู่ในระยะห่าง -127 ถึง 128 คือใน 1 ค่าของช่วงที่เล็กว่าจะแทนช่วงในส่วนของค่าที่ใหญ่กว่านั่นคือทำให้เกิดการ Quantization ดังแสดงในรูปที่ 2.23



รูปที่ 2.23 การ Mapping Quantization

2.6 Tier-1 Algorithms ใน JPEG2000

จะอาศัยอัลกอลิทึม **EBCOT (Embedded Block Coding with Optimized Truncation)** ของ David S. Taubman โดยหลังจากที่ได้ทำขั้นตอน DWT (Discrete Wavelet Transform) และผ่านขั้นตอนการ Quantization, Region of Interest Coding, Rate Control แล้วจึงนำข้อมูลมาตัดแบ่งบล็อก (Block) แล้วแยกเป็นบิตเพลน (Bit plane) โดยข้อมูลที่ได้จะนำมาทำตามขั้นตอนโดยจะมีการแบ่งกลุ่มเพลน (Plane) ดังนี้และที่สำคัญคือบิตเพลนชั้นบนสุดนั้นจะถูกนำไปทำขั้นตอนที่ Cleanup Pass เท่านั้นส่วนที่เหลือจะต้องไปเข้าเรียงลำดับ

1. Significance Propagation Pass (SPP) ขั้นตอนนี้จะต้องทำเป็นขั้นตอนแรก
2. Magnitude Refinement Pass (MRP) จะทำเป็นขั้นตอนที่สอง
3. Cleanup Pass (CUP) จะทำเป็นขั้นตอนสุดท้าย

2.6.1 การทำ Fractional Bit-Plane Coding

2.6.1.1 นิยามศัพท์

Code-Block (γ): คือ Array สองมิติประกอบไปด้วยค่าจำนวนเต็ม (wavelet *coefficients* จะทำ *quantization* หรือไม่ก็ได้) แต่ละโค้ดบล็อก (Code-Block) จะมีความกว้างและความสูงเป็นตัวกำหนดขนาด จำนวนเต็มแต่ละค่าอาจจะเป็น ค่าบวก ศูนย์ หรือลบ แต่ละองค์ประกอบของโค้ดบล็อกสร้างขึ้นด้วย σ, σ' และ η เพื่อแสดงสถานะต่างๆ ของการเข้ารหัส (Coded States) ขององค์ประกอบเหล่านั้น

Sign Array (χ): χ คือ Array สองมิติที่แสดงสัญลักษณ์ต่างๆ ของส่วนประกอบของโค้ดบล็อกและมีมิติที่มีลักษณะตรงกันกับ โค้ดบล็อกแต่ละส่วนประกอบ $\chi[m, n]$ แสดงถึงข้อมูลสัญลักษณ์ของส่วนประกอบที่สอดคล้องกันของ $\gamma[m, n]$ ในโค้ดบล็อกดังนี้

$$\chi[m, n] = \begin{cases} 1 \text{ if } \gamma[m, n] < 0 \\ 0 \text{ otherwise} \end{cases}$$

χ คือ Sign Array

γ คือ โค้ดบล็อก (Code-Block)

m คือ แถว

n คือ คอลัมน์

เมื่อมีการอ้างถึง χ[m, n] ในระหว่างการเข้ารหัสหรือการถอดรหัส m หรือ n อาจอยู่นอกอาณาเขตของโค้ดบล็อก เช่น m = -1 กรณีนี้อาจจะเกิดขึ้นหากว่าเราดำเนินการอยู่ที่บริเวณขอบของโค้ดบล็อกซึ่งในกรณีแบบนั้น χ[m, n] จะถูกตั้งค่าให้เท่ากับ 0 เสมอ

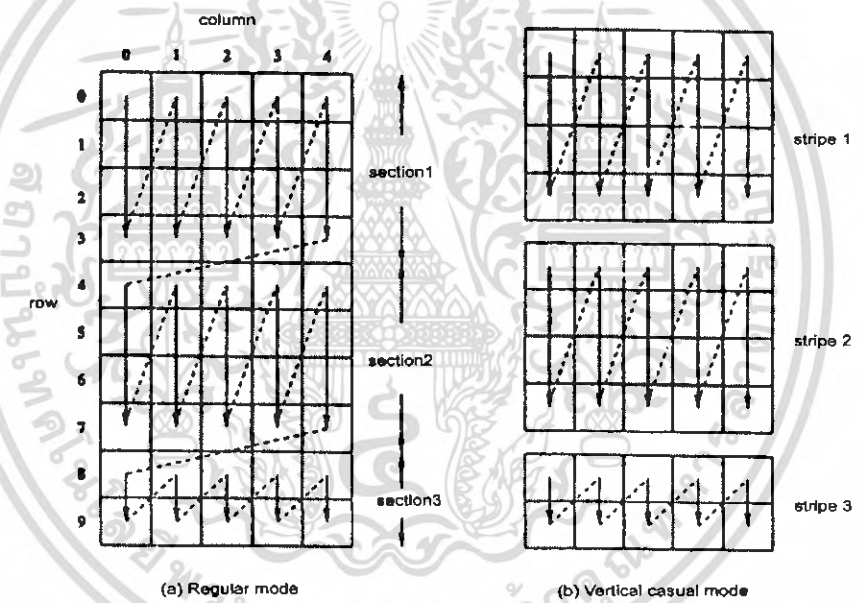
Magnitude Array (υ): υ คือ Array สองมิติของจำนวนเต็มที่ไม่ได้กำหนดไว้ (Unsigned Integers) มิติของ Array นี้มีลักษณะที่ตรงกันกับมิติของโค้ดบล็อกแต่ละองค์ประกอบของ υ แสดงถึงค่าสมบูรณ์ของเลขจำนวนเต็มในตำแหน่งที่สอดคล้องกันในโค้ดบล็อกนั่นคือ υ[m, n] = [γ [m, n]] เมื่อ γ [m, n] คือ ส่วนประกอบเลขจำนวนเต็มในตำแหน่งพิเศษ (m, n) ใน code-block ส่วนเครื่องหมาย υ^{P}[m, n] ใช้เพื่อแสดง bit P^{th} ของ υ[m, n]

Bit-Plane: ขนาดของ Array υ ที่สามารถดูข้อมูลได้เป็นแบบ Array สามมิติที่มีลำดับบิตของเลขจำนวนเต็มอยู่ในมิติที่สาม แต่ละลำดับของบิตต่าง ๆ ของทุก ๆ ส่วนประกอบของ υ ตัวสร้างบิดเพลนหนึ่งๆ ขึ้นและเรายังสามารถดูข้อมูลของ υ ในรูปแบบของ Array มิติเดียวที่ประกอบไปด้วยจำนวนของบิตเพลนเช่น สมมติว่า (2, 0) คือ Array υ 2x1 ที่ประกอบไปด้วย 2 elements ฉะนั้นบิตเพลนของ Array นี้ก็คือ (1, 0) และ (0, 0)

เราอาจจะกล่าวได้ว่าบิตเพลนหนึ่งมีนัยสำคัญมากกว่าบิดเพลนหนึ่งหากว่าบิดของบิตเพลนนั้นมีนัยสำคัญมากกว่าบิดของบิตเพลนอื่นๆ จะมีบิตเพลนเพียงบางตัวของ υ เท่านั้นที่จะถูกเข้ารหัสเพราะว่าลำดับของบิดหนึ่ง ๆ นั้นอาจจะเป็นไปได้ว่ามี 0 นำเป็นจำนวนมากในค่าใดค่าหนึ่งที่เป็นบวก ซึ่งเราจะทำการเข้ารหัสเฉพาะบิตเพลนที่มีนัยสำคัญคือประกอบด้วย บิต 1 อย่างน้อย 1 อันและทุก ๆ บิดเพลนอันต่อมาที่มีนัยสำคัญน้อยกว่าโดยไม่คำนึงถึงว่าบิตเพลนเหล่านั้นจะมีบิต 1 หรือไม่หรืออาจจะกล่าวได้ว่าบิดเพลนที่มีประกอบด้วยบิต 0 ทั้งหมดนั้นจะไม่ถูกทำการเข้ารหัสเว้นเสียแต่ว่าบิตเพลนนั้นมีนัยสำคัญมากกว่าคือประกอบด้วยบิด 1 จำนวน 1 ตัวเป็นอย่างน้อย ซึ่งเราเรียกบิตเพลนที่ไม่ได้ทำการเข้ารหัสว่า "Leading-zero bit-planes" ส่วนเลขจำนวนเต็มลบ P นั้นมักจะใช้สำหรับการอ้างถึงบิตเพลนที่ถูกทำการเข้ารหัส โดยเราใช้ υ^{P}[m, n] แทนบิดที่อยู่ในตำแหน่ง Spatial location(m, n) ของบิตเพลน P ของโค้ดบล็อก

Scan Pattern: เป็นตัวกำหนดลำดับของการเข้ารหัสหรือการถอดรหัสบิตเพลนของโค้ดบล็อกซึ่งพื้นที่สแกน(Scan Pattern) ของโค้ดบล็อกหนึ่งๆ นั้นสามารถแบ่งออกได้เป็นส่วน ๆ หรือเป็นตอน ๆ โดยแต่ละส่วนนั้นประกอบไปด้วยแถว 4 แถวติดกันโดยเริ่มต้นจากแถวแรกของโค้ดบล็อก ถ้าหากจำนวนแถวทั้งหมดของโค้ดบล็อกนั้นมีไม่สามารถแบ่งออกเป็นส่วนละ 4 แถวได้พอดี ทุก ๆ ส่วนนั้นจะต้องประกอบไปด้วย 4 แถวยกเว้นส่วนสุดท้าย การสแกนจะเริ่มจากส่วนแรกและไล่ลงไปเรื่อย ๆ จนถึงส่วนสุดท้ายจนกระทั่งทุกๆส่วนประกอบของโค้ดบล็อกนั้นได้ถูกทำการเข้ารหัสหรือถอดรหัส

ทั้งหมด แล้วในแต่ละส่วนจะถูกสแกนจากแถวแรกของคอลัมน์แรก ส่วนตำแหน่งถัดไปที่จะถูกสแกนก็จะเป็นแถวถัดไปของคอลัมน์เดียวกัน หลังจากที่คอลัมน์หนึ่งในส่วนนั้นได้ถูกทำการเข้ารหัสเสร็จสมบูรณ์แล้วก็จะเริ่มการสแกนที่แถวแรกของคอลัมน์ถัดไปในส่วนเดียวกันและดำเนินการเข้ารหัสอย่างต่อเนื่องจนกระทั่งทุกคอลัมน์ในส่วนนั้นได้ทำการเข้ารหัส กระบวนการเดียวกันนี้ก็จะถูกนำไปใช้กับส่วนที่อยู่ถัดไป (Adjacent Section) จนกระทั่งทุก ๆ ส่วนนั้นได้ทำการเข้ารหัสทั้งหมด รูปที่ 2.24 แสดงตัวอย่างรูปแบบการสแกนโค้ดบล็อกขนาด 5x10 รูปที่ 2.24 (a) แสดงรูปแบบการสแกนในโหมดที่เป็นไปตามกฎ(Regular Mode) ซึ่งในส่วนแรกของ JPEG2000 ยังเป็นตัวกำหนดโหมดของการสแกนแบบ Vertical Causal Mode ด้วย ใน Vertical Causal Mode นี้ ทุกๆ ส่วนของพื้นที่สแกน(Scan Pattern) ซึ่งก็คือ 4 แถวใน 1 คอลัมน์จะถูกนำไปพิจารณาเป็น Standalone Module ในอีกมุมหนึ่งก็คือข้อมูลทุกอย่างของ Neighbors ในโค้ดบล็อกเดียวกันแต่ต่างส่วนกันที่อยู่ในโหมด Vertical Causal Mode จะไม่ถูกใช้ในส่วนปัจจุบัน (Current Section) ส่วนรูปที่ 2.24 (b) แสดงการสแกนของตัวอย่างเดียวกันกับใน รูปที่ 2.24(a)ในโหมด Vertical Causal Mode



รูปที่ 2.24 Scan Pattern with 5x10 code-block: (a) Regular mode; (b) Vertical casual mode

State Variable σ, σ´ และ η: Array ทั้งสามชนิดนี้เป็นแบบไบนารี่(Binary) สองมิติถูกสร้างขึ้นมาเพื่อแสดงถึงสถานะของการเข้ารหัสในขณะที่ดำเนินการเข้ารหัสของแต่ละส่วนประกอบของโค้ดบล็อกมิติของ Array เหล่านี้จะมีลักษณะที่ตรงกันกับมิติของโค้ดบล็อกส่วนประกอบแต่ละส่วนของ Array เหล่านี้จะมีค่าเริ่มต้นเป็น 0 (นั่นคือ σ[m, n] = 0, σ´[m, n] = 0, และ η[m, n] = 0, สำหรับ m และ n ทุก ๆ ตัว) เมื่อกระบวนการเข้ารหัสเริ่มขึ้น ค่าของตัวแปรทั้งสอง σ[m, n] และ σ´[m, n] อาจเปลี่ยนเป็น 1 ขึ้นอยู่กับเงื่อนไขต่าง ๆ แต่จะไม่เปลี่ยนกลับไปเป็น 0 จากว่าทั้งโค้ดบล็อกจะทำการเข้ารหัสเสร็จ หรืออาจจะกล่าวได้ว่าค่าของ η[m, n] จะกลับมาเป็น 0 ทันทีหลังจากที่ทำการเข้ารหัสบิตเพลน แต่ละอันเสร็จสิ้นลง หลักการนำตัวแปรทั้งสามตัวนี้ไปใช้มีดังต่อไปนี้:

- เมื่อ σ[m, n] = 1 แสดงว่าบิตตัวแรกที่ไม่เป็น 0 ของ υ[m, n] ที่แถว m และคอลัมน์ n ได้ถูกเข้ารหัสหรือไม่เช่นนั้นมันจะมีค่าเท่ากับ 0 และเมื่อมีการอ้างถึง σ[m, n] ในขณะที่ทำการเข้ารหัสหรือถอดรหัส m หรือ n อาจจะอยู่นอกบริเวณหรือมีค่าที่ใช้ไม่ได้ เช่น m = -1 ซึ่งในกรณีนี้ σ[m, n] มีค่าเท่ากับ 0
- เมื่อ σ´[m, n] = 1 แสดงว่าปฏิบัติการ Magnitude Refinement Coding ได้ถูกนำมาใช้ร่วมกับ υ[m, n] หรือไม่เช่นนั้นมันจะมีค่าเท่ากับ 0
- เมื่อ η[m, n] = 1 แสดงว่าปฏิบัติการ Zero Coding ได้ถูกนำมาใช้ใน υ^{p}[m, n] ใน Significant Propagation Pass (SPP) หรือไม่เช่นนั้นมันจะมีค่าเท่ากับ 0

Preferred Neighborhood: ส่วนประกอบ γ[m, n] ใน Code-block นั้นเรียกได้ว่าเป็น Preferred Neighborhood ถ้า adjacent neighbor อย่างน้อย 1 ในแปด 8 ตัวมีค่า σ เท่ากับ 0

Zero Coding Tables: การทำ Zero coding นั้นประกอบไปด้วย Zero Coding Tables จำนวน 3 ตาราง (ตารางที่ 2.8 – 2.10) ข้อมูลของ Context ที่เกิดขึ้นจากการทำ Zero coding ขึ้นอยู่กับค่า Significant state(σ) ของ Neighbor ทั้ง 8 ของส่วนย่อย(Element) ซึ่งก็คือบิตถูกทำการเข้ารหัส ในรูปที่ 2.25 ได้แสดง Neighbor ทั้ง 8 ของบิต ๆ หนึ่งคือ X ที่ถูกนำไปใช้ในทั้ง 3 ตารางเพื่อสร้าง "บริบท (Context)" ตัวอย่างเช่น X คือบิตใน Subband LL และ Neighbor ในแนวนอนของมันสองตัวมีค่า σ = 1 (นั่นคือ $\sum H = 2$) ฉะนั้น Context value 8 จะถูกใช้ดังแสดงไว้ในตารางที่ 2.8

D_0	V_0	D_1
H_0	X	H_1
D_3	V_1	D_2

รูปที่ 2.25 Neighborhood for Zero Coding Context Generation

จากรูปที่ 2.25 เป็นการกำหนดความสัมพันธ์ของบิตรอบข้างบิต X โดยจะมีผลต่อการเข้ากฎการเข้ารหัสในตารางต่อๆ ไป

ตารางที่ 2.8 Zero Coding Context Table for Code-Block from LL and LH Subbands

LL and LH Subbands			Context Label
∑H	∑V	∑D	CX
2	x	x	8
1	≥1	x	7
1	0	≥1	6
1	0	0	5
0	2	x	4
0	1	x	3
0	0	≥2	2
0	0	1	1
0	0	0	0

ตารางที่ 2.9 Zero Coding Context Table for Code-Block from HL Subbands

HL Subbands			Context Label
∑H	∑V	∑D	CX
x	2	x	8
≥1	1	x	7
0	1	≥1	6
0	1	0	5
2	0	x	4
1	0	x	3
0	0	≥2	2
0	0	1	1
0	0	0	0

จากตารางที่ 2.8 และ 2.9 อธิบายได้ว่าเมื่อดูจากตารางแรกจะได้ผลรวมของบิตต่างๆ ได้ดังตารางจะมีการนับสามตัว คือ ∑H (ผลรวมของ H), ∑V (ผลรวมของ V), ∑D (ผลรวมของ D) โดยข้อมูลของแต่ละตัวอย่างที่รู้คือจะเป็นเพียงแค่ 0 หรือ 1 เท่านั้นส่วน x คือไม่สนใจจะเป็นข้อมูล 0 หรือ 1 ก็ได้ และ CX คือบริบทของการเข้ารหัส Zero Coding

ตารางที่ 2.10 Zero Coding Context Table for Code-Block from HH Subbands

HH Subbands		Context Label
∑(H+V)	∑D	CX
x	≥3	8
≥1	2	7
0	2	6
≥2	1	5
1	1	4
0	1	3
≥2	0	2
1	0	1
0	0	0

จากตารางที่ 2.10 จะนับคล้ายๆ กับตารางที่ 2.8 และ 2.9 แต่ต่างตรงที่นำการนับบิตของ H และ V รวมกัน

2.6.1.2 การเข้ารหัส (Coding Operation)

การทำการเข้ารหัสที่ใช้ใน EBCOT เพื่อสร้างค่าต่าง ๆ ของบริบท(CX) หรือ (Context(CX)) และ คำสั่งตัดสินใจ (*D*) (หรือ Decision (*D*))ให้เป็นข้อมูลที่ใช้ได้ทันทีก่อนการทำ BAC นั้นมีอยู่ 4 อย่าง CX คือเลขจำนวนเต็มที่มีค่าเป็นบวกส่วน *D* คือค่าของ Binary ซึ่งมีค่าเป็น 0 หรือ 1 ในการทำ Coding operation ทั้ง 4 ขั้นตอนนี้มีค่าบริบทอยู่ 19 ค่าคือ 0 – 18 ส่วนดรรชนีของบิตเพลน (Bit-plane Index)ปัจจุบันนั้นกำหนดให้เป็น *P* ไม่ว่าจะเป็นที่ไหนหรือเมื่อไหร่ก็ตามที่ปฏิบัติการเหล่านี้ถูกนำไปใช้มันจะเป็นตัวกำหนด ในการทำ Coding pass ในขณะนั้นกำหนดตำแหน่งของบิตปัจจุบัน รวมทั้งกำหนดสถานะของตัวแปร

การทำ Zero Coding (ZC): ในการทำ Zero coding นั้น Decision bit *D* จะมีค่าเท่ากับ $\upsilon^{P}[m, n]$ และค่า CX ก็จะเลือกมาจากตาราง "Zero coding context tables" ทั้งสามตาราง ซึ่งจะ

เกี่ยวเนื่องกับ Subband ที่สัมพันธ์กัน(LH, HL หรือ HH)ที่โค้ดบล็อกเป็นส่วนหนึ่งของ Subband นั้น ในแต่ละตารางบริบทประกอบไปด้วย 9 รายการซึ่งเกิดขึ้นจากค่าต่าง ๆ ของสถานภาพของ Neighbor ทั้ง 8 ของค่าสัมประสิทธิ์ของ bit $\upsilon^{P}[m,\ n]$ จากตารางที่ 2.8 – 2.10 จะเห็นว่าแต่ละรายการนั้นขึ้นอยู่กับว่ามี Neighbor ของ $\upsilon^{P}[m, n]$ จำนวนเท่าใดหรืออันไหนที่มีนัยสำคัญ

การทำ Sign Coding (SC): D และ CX สำหรับ Sign Coding เป็นการตัดสินใจโดยอ้างอิงค่าแนวนอน H และค่าแนวตั้ง V ให้สมมติว่าตำแหน่งปัจจุบันเป็นตำแหน่ง (m, n) ค่าของ H และ V จะเป็นไปตามสมการนี้

$$H = \min[1, \max(-1,\ \sigma[m, n-1] \times (1-2\chi[m, n-1]) + \sigma[m, n+1] \times (1-2\chi[m, n+1]))] \quad (2.33)$$

$$V = \min[1, \max(-1,\ \sigma[m-1, n] \times (1-2\chi[m-1, n]) + \sigma[m+1, n] \times (1-2\chi[m+1, n]))] \quad (2.34)$$

จากค่าของ H และ V แสดงได้ 3 สิ่งที่เป็นไปได้ดังนี้

1) 0 แสดงว่าทั้งคู่ (Neighbors) เป็นเพื่อนบ้านที่ไม่สำคัญต่อกันหรือทั้งคู่เป็นเพื่อนบ้านที่สำคัญต่อกันแต่เครื่องหมายตรงข้ามกัน
2) 1 แสดงว่าต้องมีหนึ่งหรือทั้งคู่ที่เป็นเพื่อนบ้านที่มีเครื่องหมาย บวก
3) -1 แสดงว่าต้องมีหนึ่งหรือทั้งคู่ที่เป็นเพื่อนบ้านที่มีเครื่องหมาย ติดลบ

การเป็น Neighbor หมายความว่าทั้งสองตำแหน่งนั้นอยู่ประชิดกันทางด้านแนวนอนของพื้นที่สแกนปัจจุบันสำหรับ H และทางด้านแนวตั้งของพื้นที่สแกนปัจจุบันสำหรับ V ความสำคัญที่ตำแหน่งหนึ่งหมายความว่า ค่าของตัวแปรสถานะ σ ที่ตำแหน่งนั้นคือ 1 ถ้าไม่มีความสำคัญหมายความว่าค่าของ σ นั้นเป็น 0

จากตารางที่ 2.11 แสดงให้เห็นถึงค่า H และ V ถูกใช้พร้อมกันเพื่อตัดสินบริบท (CX) และ binary value $\hat{\chi}$, ซึ่งในที่กำหนดใช้เพื่อการคำนวณค่าของ D ขณะที่ $D = \chi \otimes \hat{\chi}[m, n]$, ซึ่ง $\otimes$ แทนเครื่องหมาย Exclusive-OR

ตารางที่ 2.11 ตารางอ้างอิงสำหรับการทำ Sign Coding

H	V	$\hat{\chi}$	CX
1	1	0	13
1	0	0	12
1	-1	0	11
0	1	0	10
0	0	0	9
0	-1	1	10
-1	1	1	11
-1	0	1	12
-1	-1	1	13

การทำ Magnitude Refinement Coding (MRC): สำหรับการเข้ารหัสนี้ D ที่จุด [m, n] ใน P^{th} bit-plane ที่ดูเรียบง่ายเท่ากันกับ ค่าของ υ^{p}[m, n] ค่าของ CX ถูกกำหนดโดย σ[m, n] และผลรวมของมัน ของแปดตัวที่อยู่ติดกับมัน ค่าของการกำหนดตัวแปร σ มีดังนี้คือ

- ถ้า σ' = 1 ที่ตำแหน่งปัจจุบันซึ่งแสดงว่ามันไม่ได้เป็น Magnitude Refinement สำหรับบิตนี้ ฉะนั้น CX=16
- เมื่อ σ' = 0 ที่ตำแหน่งปัจจุบันและผลรวมของค่าของแปดตัวรอบๆ σ เป็น 0 อีกด้วย ฉะนั้น CX=14
- เมื่อ σ' = 0 ที่ตำแหน่งปัจจุบันและผลรวมของค่าของแปดตัวรอบๆ σ มีค่ามากกว่า 0 ฉะนั้น CX=15

ในตารางที่ 2.12 เราย่อตรรกะ (Logic) สำหรับการก่อให้เกิดค่าของบริบท (CX) ได้ตามนี้

ตารางที่ 2.12 ตารางอ้างอิงสำหรับการทำ Magnitude Refinement Coding

σ' [m,n]	σ[m-1,n] + σ[m+1,n] + σ[m-1,n-1] + σ[m-1,n+1] + σ[m+1,n-1] + σ[m+1,n+1]	CX
1	x	16
0	≥1	15
0	0	14

Run-Length Coding (RLC): ไม่เหมือนกับสามกระบวนการที่กล่าวมาแล้ว Run-Length อาจจะเข้ารหัสได้ทีเดียว 1 ถึง 4 บิตในการแสกน จำนวนบิตที่แท้จริงที่ถูกเข้ารหัสขึ้นอยู่กับบิตที่เป็น 1 บิตแรกที่เกิดของทั้ง 4 บิต ถ้าทั้ง 4 บิตเป็น 0 ทั้ง 4 บิตถูกเข้ารหัส ถ้ามีหนึ่งหรือมากกว่าของทั้ง 4 บิตมีค่าเป็น 1 แล้วบิต 1 แรกในรูปแบบการแสกนและ 0 ตัวก่อนหน้าระหว่างพื้นที่แสกนปัจจุบันถูกเข้ารหัส เช่น สมมติมี 0101 เป็น 4 บิตติดกันตามรูปแบบการแสกนบิตเพลน ถ้าปัจจุบันเป็นที่ตัวแรกคือ 0 แล้วเราใช้ Run-Length Coding จะได้ 2 บิตที่ถูกเข้ารหัสคือ 01 และต่อมาก็จะเป็น 0 ตัวที่สองถัดมา

กระบวนการ Run-Length Coding อาจจะก่อให้เกิด หนึ่ง *D* หรือสาม *D* ขึ้นอยู่กับว่าทั้ง 4 บิตที่ติดกันเป็นบิต 0 ทั้งหมดหรือไม่ สำหรับ *D* ตัวแรกมีค่าเท่ากับ 0 ถ้าทั้ง 4 บิตมีค่าเป็นบิต 0 ถ้าไม่เป็น 0 ทั้งหมด *D* จะมีค่าเท่ากับ 1 สำหรับทั้งสองกรณี *CX* มีค่าเท่ากับค่าของ Run-length context ที่ 17 หรืออาจจะกล่าวได้ว่า ค่า (CX, D) ที่ (17, 0) แสดงให้เห็นถึงบิต 0 ทั้งสี่ที่อยู่ติดกัน และค่า (CX, D) ที่ (17, 1) แสดงให้เห็นว่ามีบิต 1 อยู่หนึ่งอันในตำแหน่งรูปแบบพื้นที่สแกนปัจจุบัน (Current scan pattern)

ในกรณีที่มีบิตอย่างน้อยหนึ่งในสี่บิตในรูปแบบพื้นที่สแกนปัจจุบันนั้นเป็น 1 จะมี *D* อยู่สองตัวขึ้นไปที่จะถูกใช้ร่วมกับ UNIFORM ที่ค่าบริบทเป็น *18* เพื่อแสดงตำแหน่งของบิต 1 ตัวแรกในจำนวน 4 บิตในพื้นที่สแกน เมื่อความสูงของพื้นที่สแกนมีค่าเท่ากับ 4 ส่วน Zero-based index ที่มีบิตสองบิตก็สามารถบ่งชี้ได้ตำแหน่งของบิต 1 ตัวแรกนั้นอยู่ที่จุดสูงสุด *D* ตัวแรกกับ *D* ตัวที่สองที่ใช้ร่วมกับบริบท UNIFORM แสดงให้เห็นถึงบิตที่มีนัยสำคัญมากที่สุดและบิตที่มีนัยสำคัญน้อยที่สุดของทั้งสองบิตนี้ที่เป็นเครื่องหมายแสดงถึงระยะทาง

ต่อจากตัวอย่างการเข้ารหัส 01 ค่าของ *D* ตัวแรกกับ *D* ตัวที่สองมีค่าเป็น 0 และ 1 ตามลำดับ และค่า (CX, D) ก็จะเป็น (18, 0) และ (18, 1)

ตารางที่ 2.13 แสดงหัวข้อย่อยๆ ของทั้งหมด 19 บริบทใช้ในสี่กระบวนการเข้ารหัส และมันยังมีค่าลักษณะเดียวกันในดรรชนีเริ่มต้น (Initial Index) สำหรับค่าตารางประมาณความเป็นไปได้ใน BAC (จะกล่าวถึงภายหลัง)

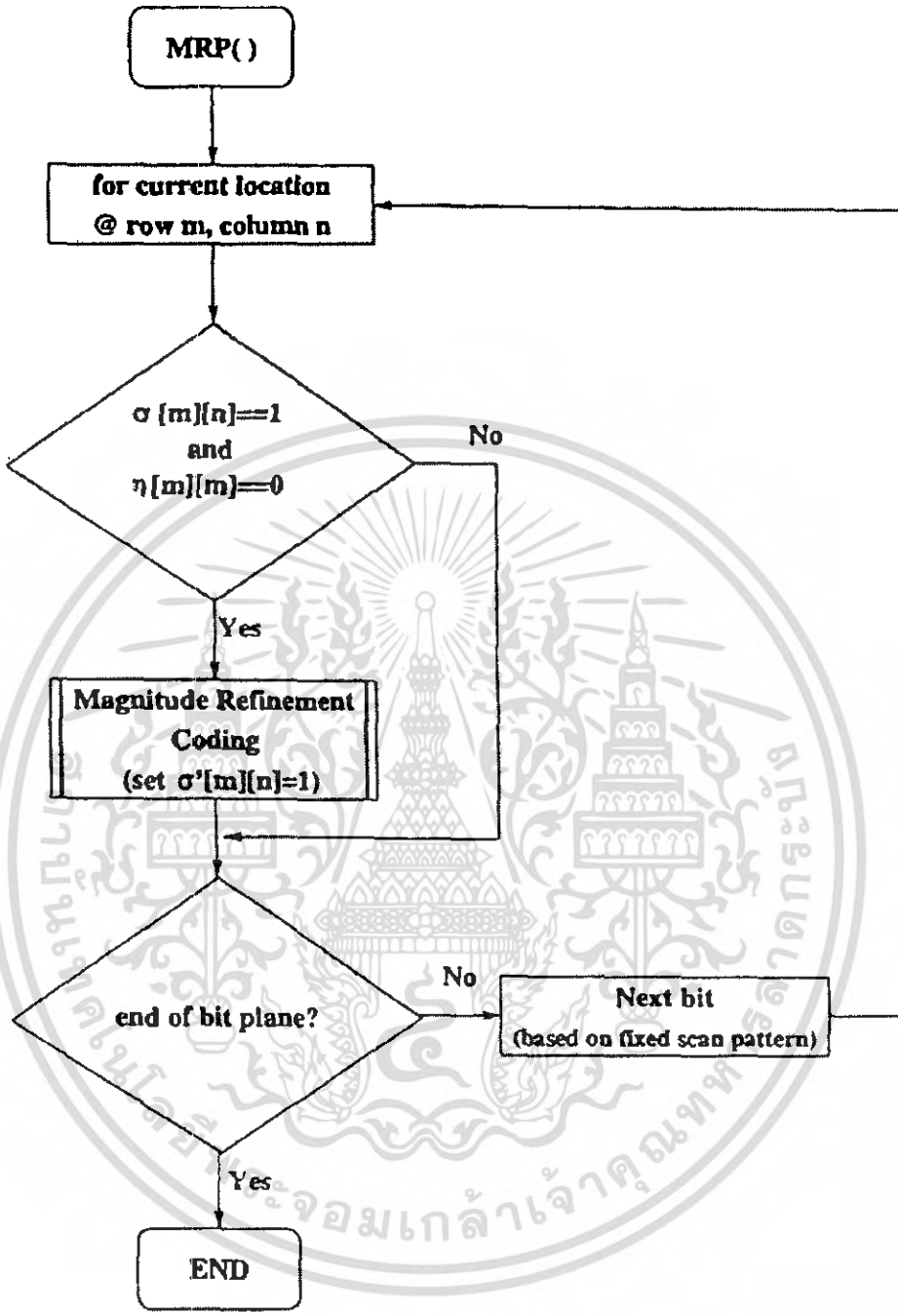
ตารางที่ 2.13 19 บริบทและดรรชนีเริ่มต้นสำหรับตารางประมาณความเป็นไปได้ BAC

Operation	Context CX	Initial Index I(CX)
Zero Coding	0	4
	1	0
	2	0
	3	0
	4	0
	5	0
	6	0
	7	0
	8	0
Sign Coding	9	0
	10	0
	11	0
	12	0
	13	0
Magnitude Refinement Coding	14	0
	15	0
	16	0
Run-Length Coding	17	3
UNIFORM	18	46

2.6.1.3 การทำ Coding Pass

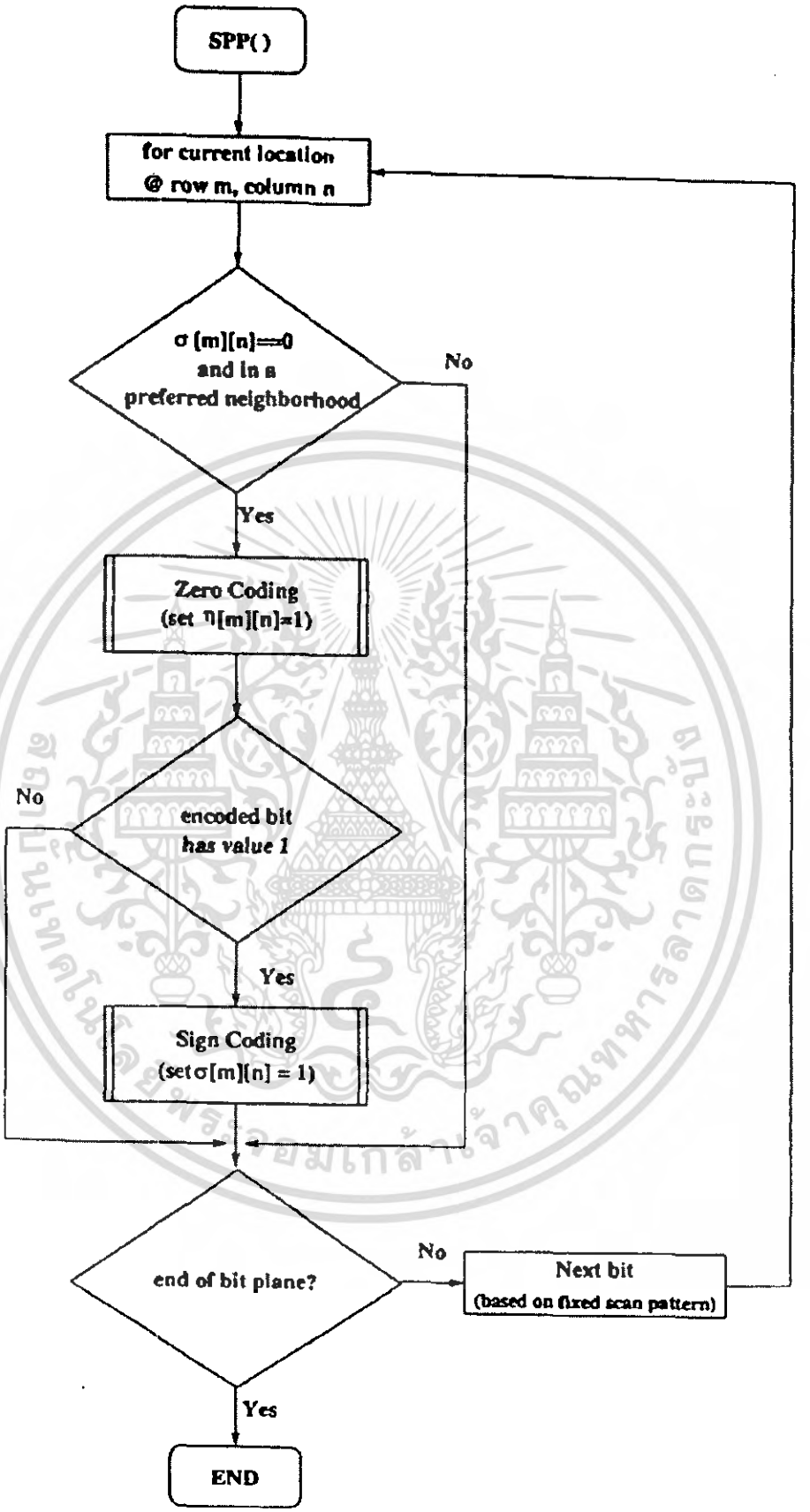
การทำ Coding Pass มีอยู่ 3 ขั้นตอน คือ Significance Propagation Pass (SPP), Magnitude Refinement Pass (MRP), และ Cleanup Pass (CUP) ทั้งสามขั้นตอนนี้จะถูกนำไปใช้กับทุกๆ บิตของแต่ละบิตเพลนในแต่ละโค้ดบล็อกยกเว้นบิตเพลนแรกซึ่งจะถูกทำ CUP เท่านั้นส่วนบิตเพลนที่เหลือจะถูกเข้ารหัสด้วยขั้นตอน SPP, MSP และ CUP ตามลำดับ

Significance Propagation Pass (SPP) เป็น Pass แรกที่นำไปใช้กับทุก ๆบิตเพลนในโค้ดบล็อกยกเว้นบิตเพลนแรก อันดับแรก SPP ใช้การทำ Zero coding ถ้าตำแหน่งที่กำลังสแกน (m, n) นั้นอยู่ท่ามกลาง Neighbor ที่เหมาะสม และ $\sigma[m, n] = 0$ ถ้าไม่สามารถทำ Zero coding ได้ $\eta[m, n]$ จะถูกตั้งค่าเป็น 1 ภายหลังจากทำ Zero coding เสร็จเรียบร้อยแล้ว เราจำเป็นต้องตรวจสอบว่าจำเป็นต้องทำ Sign coding ในตำแหน่งบิตปัจจุบัน (m, n) นั้นหรือไม่ ถ้าตรวจสอบแล้วพบว่า $v^p[m, n] = 1$ จำเป็นจะต้องทำ Sign coding และเราต้องตั้งค่า $\sigma[m, n] = 1$ จากนั้นให้ทำการเข้ารหัสพื้นที่สแกนอย่างต่อเนื่องจนครบทุกบิตในบิตเพลน รูปที่ 2.26 แสดงความสัมพันธ์ของขั้นตอนต่าง ๆ ในการทำ SPP



รูปที่ 2.26 Significance Propagation Pass (SPP)

Magnitude Refinement Pass (MRP): เป็น Pass ที่สองที่นำไปใช้กับทุก ๆ บิตเพลนในโค้ดบล็อกยกเว้นบิตเพลนแรกซึ่งไม่จำเป็นต้องทำ MRP ถ้าสภาพของตัวแปร $\sigma[m, n] = 1$ และ $\eta[m, n] = 0$ แล้วให้ทำ MRC ในตำแหน่งที่สแกน (m, n) และกำหนดค่า $\sigma'[m, n] = 1$ จากนั้นให้ดำเนินการเข้ารหัสบิดต่าง ๆ ไปตามพื้นที่สแกนจนเสร็จทุกบิตในบิตเพลนนั้น ๆ รูปที่ 2.27 แสดงความสัมพันธ์ของขั้นตอนต่าง ๆ ในการทำ MRP



รูปที่ 2.27 Magnitude Refinement Pass (MRP)

Cleanup Pass (CUP): การทำ CUP จะทำกับทุก ๆ บิตเพลนของโค้ดบล็อกภายหลังจากที่ทำ MRP เสร็จเรียบร้อยแล้วยกเว้นบิตเพลนแรกที่ไม่ต้องทำ MRP

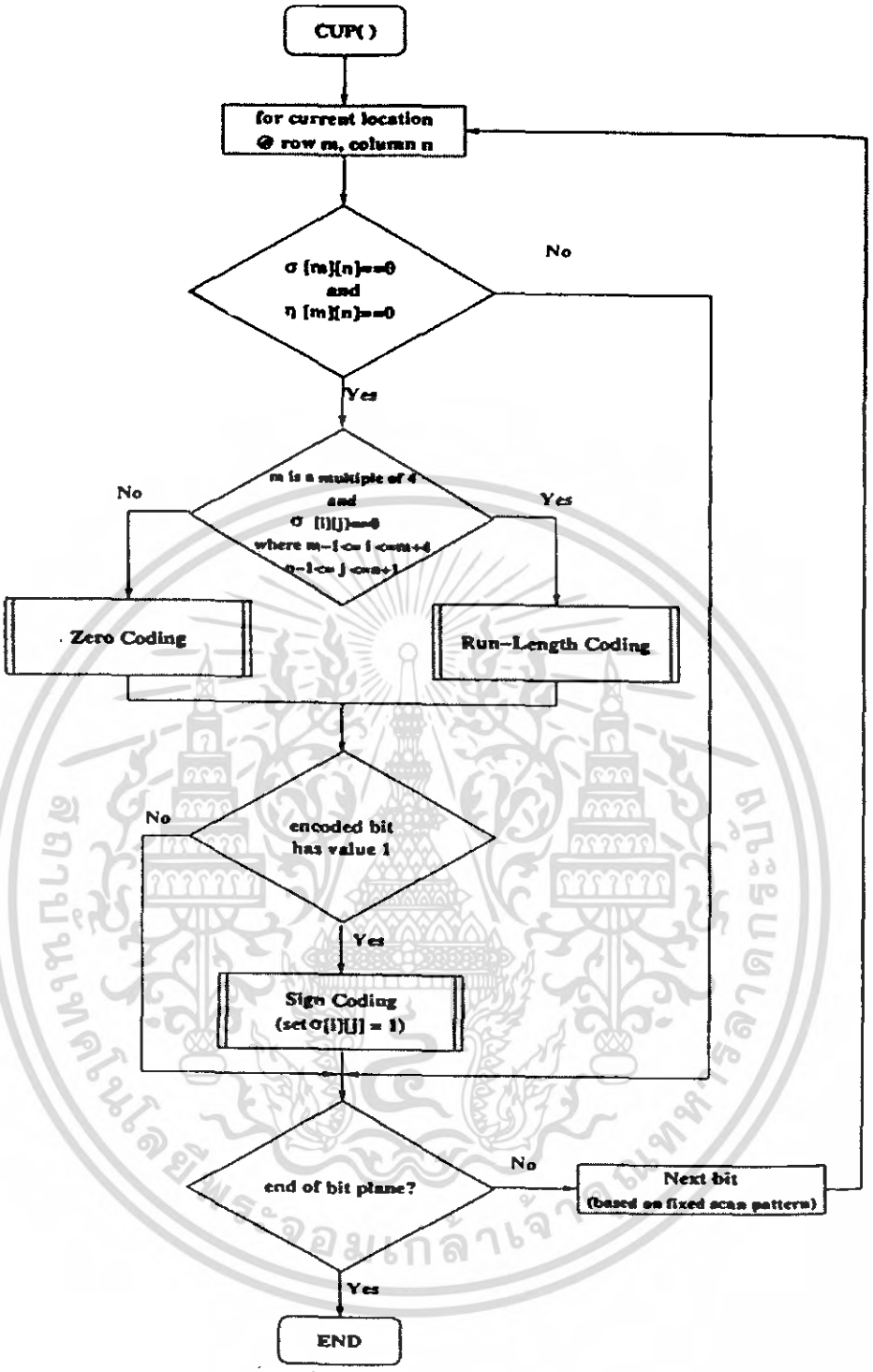
ในแต่ละตำแหน่ง (m, n) ที่อยู่ในพื้นที่สแกน CUP จะทำหน้าที่ตรวจสอบก่อนเป็นอันดับแรกว่า $\sigma[m, n]$ และ $\eta[m, n]$ ใดที่เป็น 0 ทั้งคู่ ถ้าหากไม่มีตำแหน่งใดเลยที่เป็น 0 กระบวนการทำ CUP ก็จะเลื่อนไปที่ตำแหน่งบิตถัดไปของบิตเพลนนั้น แต่ถ้าหากพบว่ามันเป็น 0 ทั้งคู่ต้องตรวจสอบต่อไปว่าต้องทำ Run-length coding (RLC) หรือต้องทำ Zero coding (ZC) อย่างใดอย่างหนึ่ง แต่ไม่ใช่ทำทั้งสองอย่างเพราะ RLC จะใช้ก็ต่อเมื่อเงื่อนไขทั้งสามอย่างต่อไปนี้เป็นจริง

1. m คือ ค่าต่าง ๆ ของ bit ทั้ง 4 รวมถึง m = 0
2. $\sigma = 0$ สำหรับตำแหน่งที่อยู่ติด ๆ กันในคอลัมน์เดียวกันโดยเริ่มจากตำแหน่งสแกนขณะนั้น
3. $\sigma = 0$ สำหรับ neighbor ที่อยู่ติด ๆ กันของ bit ทั้ง 4 ที่อยู่ติดกันในคอลัมน์นั้นๆ

หากมีเพียงหนึ่งเงื่อนไขที่ไม่ตรงกับเงื่อนไปทั้งสามข้อ ให้ทำ Zero coding ในตำแหน่งนั้น จำนวนของบิตที่เข้ารหัสอาจมีหลากหลายขึ้นอยู่กับว่าในตำแหน่งนั้นได้ทำ RLC หรือ ZC และ bit ถัดไปที่ต้องทำการเข้ารหัสจะเป็นบิตที่ต่อจากบิตที่ถูกเข้ารหัสเป็นอันดับสุดท้าย แต่ควรจำไว้ว่าการทำ RLC นั้นไม่ควรทำกับส่วนสุดท้ายของพื้นที่สแกนที่มีแถวน้อยกว่า 4 แถวเพราะว่าในคอลัมน์เดียวกันนั้นจะมีบิตไม่ครบจำนวน 4 บิต

เมื่อทำ RLC หรือ ZC เสร็จเรียบร้อยแล้วเราจำเป็นต้องตรวจสอบว่าต้องทำ Sign coding (SC) ก่อนที่จะข้ามไปยังบิดถัดไปหรือไม่ สมมติว่าตำแหน่งที่เข้ารหัสตำแหน่งสุดท้ายคือ (i, j) ถ้า $v^p[i, j] = 1$ ซึ่งแสดงให้เห็นว่าบิตนี้เป็นบิตที่มีนัยสำคัญมากที่สุดของพื้นที่สแกนปัจจุบันแล้ว CUP จะใช้การทำ SC และกำหนดให้ค่าของ σ ของตำแหน่งสุดท้ายที่ทำการเข้ารหัสนั้นมีค่าเป็น 1 (นั่นคือ $\sigma [i, j] = 1$) ทันทีที่ทำ RLC และ ZC เสร็จเรียบร้อย หรือไม่เช่นนั้นก็เกิดกรณีที่ไม่จำเป็นต้องทำ SC

จากนั้นให้ดำเนินการเข้ารหัสบิดต่าง ๆ ไปตามพื้นที่สแกนจนเสร็จทุกบิตในบิตเพลน นั้น ๆ เมื่อเสร็จสิ้นการทำ CUP ในบิตเพลนนั้นแล้ว ให้คืนค่า $\eta[m, n]$ เป็น 0 สำหรับทุก ๆ m และ n ในบิตเพลนก่อนที่จะย้ายไปยังบิตเพลนถัดไป รูปที่ 2.28 แสดงความสัมพันธ์ของขั้นตอนต่าง ๆ ในการทำ Cleanup Pass(CUP)



รูปที่ 2.28 Cleanup Pass (CUP)

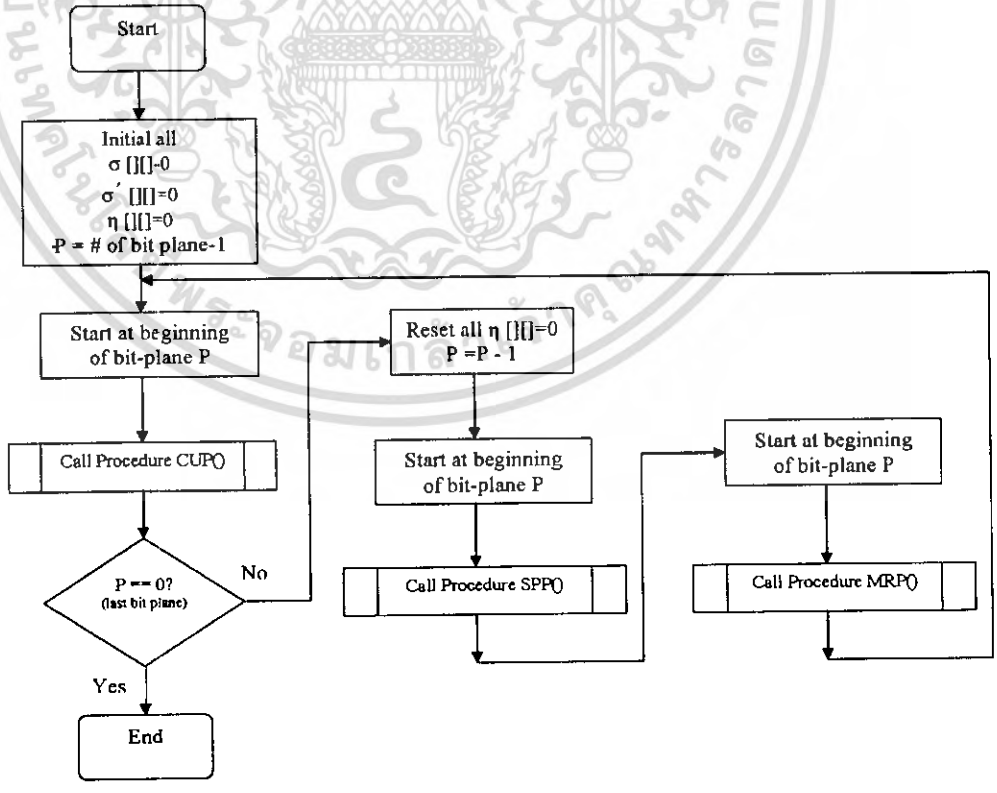
Selective binary arithmetic coding-bypass mode: แทนที่จะใช้ binary arithmetic coding (MQ-coder[2]) ในสัญลักษณ์ หรือ Symbol (บริบทและบิตสำหรับการตัดสินใจ) ที่เกิดขึ้นในระหว่างการทำ SPP, MRP และ CUP การทำ Bypass mode อนุญาตให้ข้ามการทำ MQ-coder ใน SPP และ MRP ภายหลังจากที่ MSB-plane(most significant bit-plane) ทั้ง 4 ถูกเข้ารหัส หรือกล่าวได้ว่าเฉพาะสัญลักษณ์ ที่เกิดขึ้นใน CUP ใน 4 บิตเพลน SLB-plane ที่จะถูกเข้ารหัสด้วย MQ-coder ดังนั้นถ้าเราเลือก Bypass mode ก็จะทำให้ Raw decision bits และ Sign bits ถูกเข้ารหัสระหว่าง SPP และ MRP

2.6.1.4 JPEG2000 Bit-Plane Coding: Encoder and Decoder Algorithms

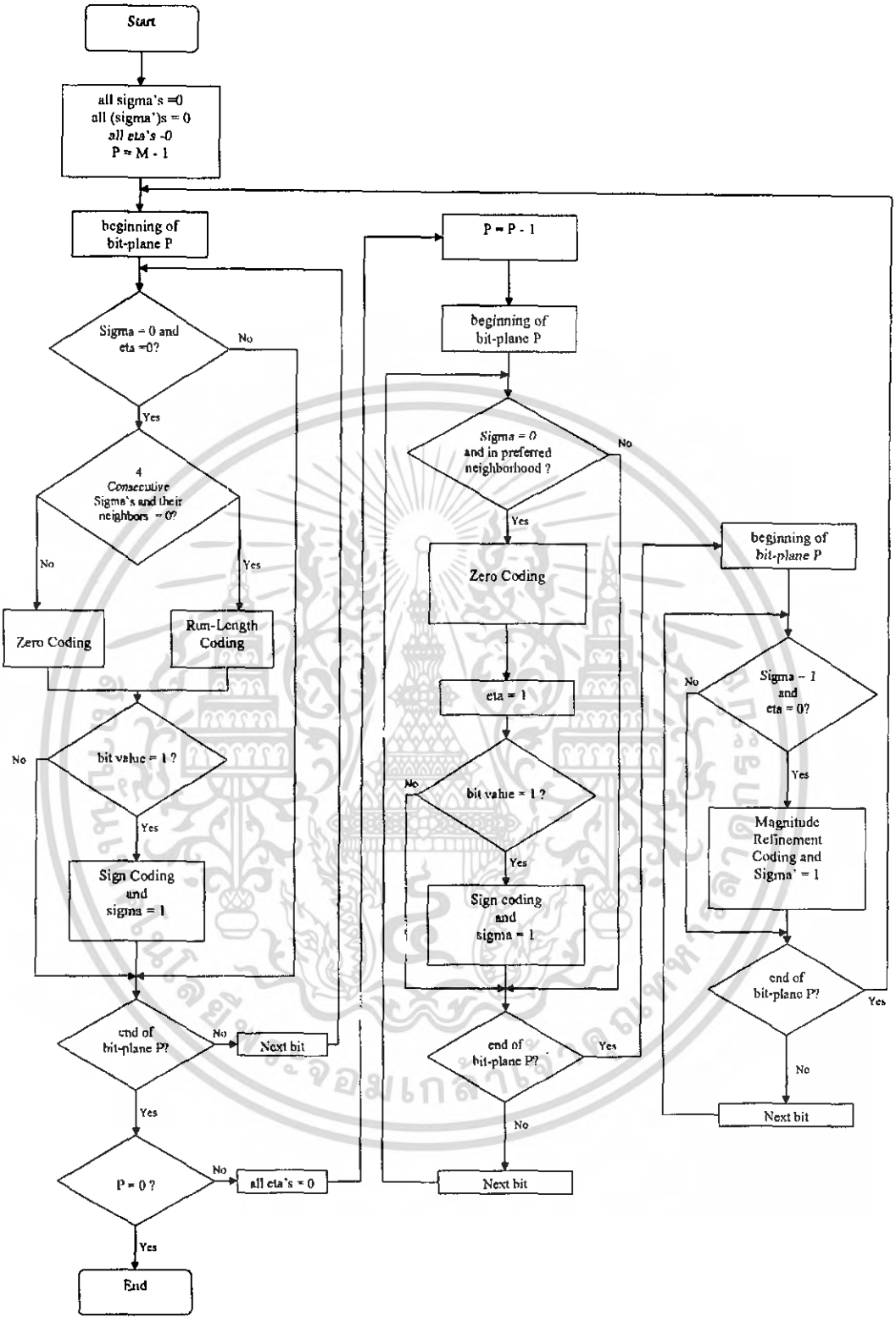
ดังที่กล่าวมาแต่ต้นว่า Quantized wavelet coefficients ในแต่ละ Subband จะถูกแปลงไปเป็น Sign-magnitude แสดงได้ก่อนที่จะเริ่มการเขารหัสเอ็นโทรพี (Entropy coding) สำหรับแต่ล่ะโค้ดบล็อกที่ใส่เข้าไป เราสามารถที่จะเริ่ม Array สองมิติ υ และ χ ซึ่งค่าของ υ[m, n] คือขนาด (Magnitude) และ χ[m, n] คือสัญลักษณ์(Sign) ที่บอกถึงองค์ประกอบที่ตำแหน่ง [m, n] ในโค้ดบล็อก จำนวนของบิตเพลนในโค้ดบล็อกจะถูกเข้ารหัส (P) ตัดสินโดยการค้นหาค่าที่มากที่สุดใน Array υ ในขั้นแรก Array สองมิติ σ, σ′ and η จะถูกจัดตำแหน่งเป็น 0

ในบิตเพลนแรกที่จะถูกเข้ารหัสคือ MSB-plane(Most Significant Bit-plane) ที่กล่าวถึงนิยามของบิตเพลนใน Leading-zero bit-plane ประกอบไปด้วย 0 ทั้งหมดจะถูกข้ามไปบิตเพลนที่มีนัยสำคัญสูงกว่าจะได้เขารหัสก่อนบิตเพลนที่มีนัยสำคัญต่ำกว่า ถ้า $P = 0$ จะไม่ต้องมีการเข้าโค้ดใดๆ ผลลัพธ์ (output) จะว่างเปล่า ถ้า $P \geq 1$ จึงเข้า Cleanup pass เฉพาะบิตเพลนแรก สำหรับบิตเพลนที่เหลือจะนำไปเข้า Significance propagation pass, Magnitude refinement pass และ Cleanup pass ตามลำดับ รูปที่ 2.29 แสดงความสัมพันธ์ขององค์ประกอบของ bit-plane ปลีกย่อย

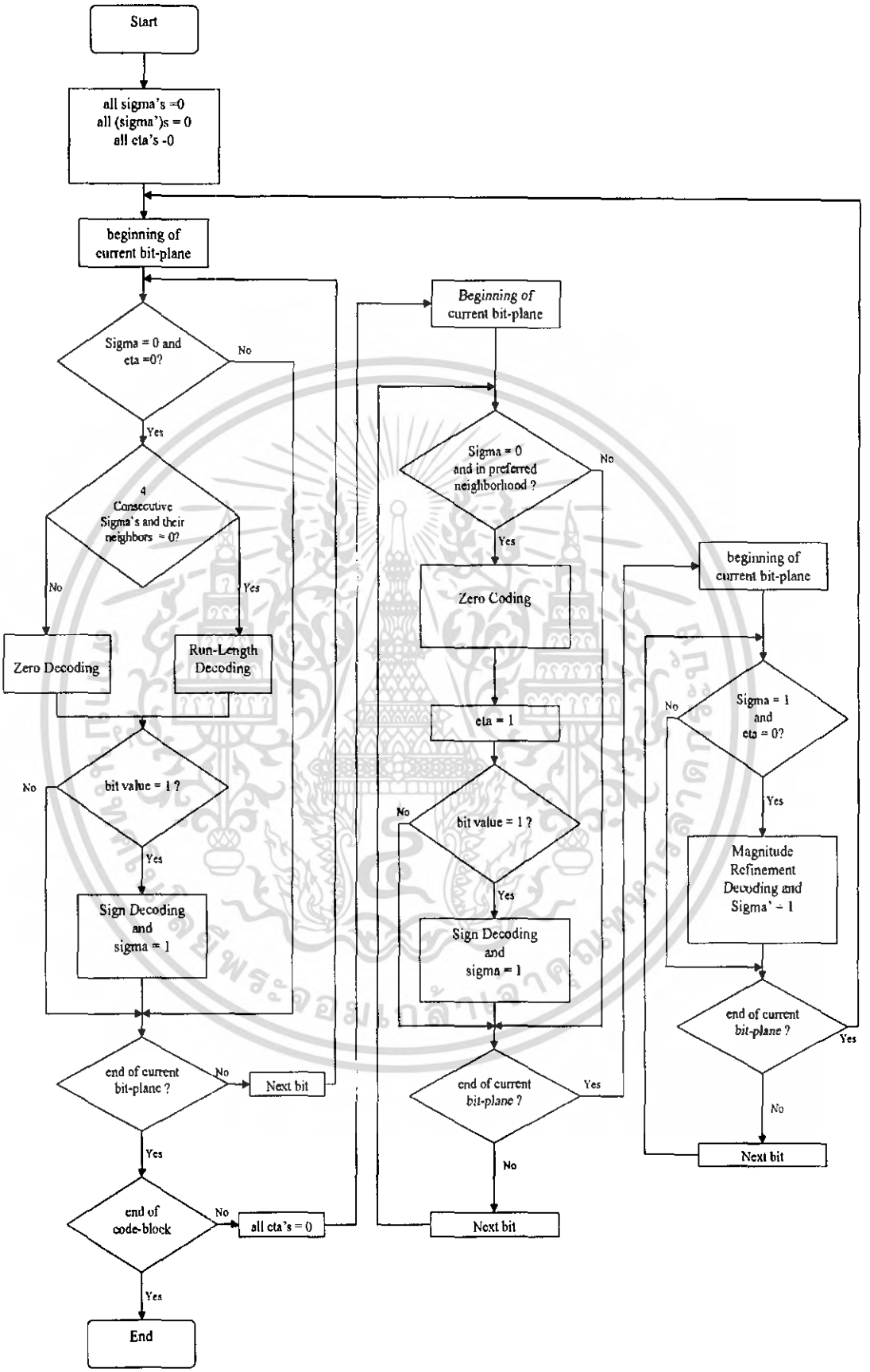
กระบวนการในการถอดรหัสนั้นมีความสำคัญพอๆกับการเข้ารหัส เพื่อประโยชน์ที่สมบูรณ์ เราได้แสดงให้เห็นถึงทั้งกระบวนการในการถอดรหัสและการเข้ารหัสดังรูปที่ 2.30 และ 2.31 ตามลำดับสำหรับกรณีที่ $P \geq 1$



รูปที่ 2.29 ความสัมพันธ์ขององค์ประกอบของ bit-plane ปลีกย่อย การเข้ารหัสสำหรับ $P > 0$



รูปที่ 2.30 ความสัมพันธ์ขององค์ประกอบต่างๆ ของ *Functional bit-plane Encoder* สำหรับ $P > 0$



รูปที่ 2.31 ความสัมพันธ์ขององค์ประกอบต่างๆ ของ Functional bit-plane Decoder สำหรับ $P > 0$

2.6.2 ตัวอย่างของ BPC Encoder

ในหัวข้อนี้เราจะแสดงตัวอย่างการดำเนินการไปทีละขั้นและแสดงผลลัพธ์ที่ได้จากการเข้ารหัสโค้ดบล็อกขนาด 4 x 4

- แสดงใส่ข้อมูลโค้ดบล็อกขนาด 4 x 4 นั่นคือ +7 ใช้ 4 บิตในการแสดง (0 111)

3	0	0	5
-3	7	2	1
-4	-1	-2	3
0	6	0	2

- *The magnitude array* (υ)

3	0	0	5
3	7	2	1
4	1	2	3
0	6	0	2

- The sign array (χ)

0	0	0	0
1	0	0	0
1	1	1	0
0	0	0	0

- บิตเพลนทั้งสามที่แสดงให้ดูดังต่อไปนี้เป็นบิตเพลนของขนาด Array υ^2, υ^1, และ υ^0 ตามลำดับ

υ^2

0	0	0	1
0	1	0	0
1	0	0	0
0	1	0	0

υ^1

1	0	0	0
1	1	1	0
0	0	1	1
0	1	0	1

υ^0

1	0	0	1
1	1	0	1
0	1	0	1
0	0	0	0

ลำดับของการเข้ารหัสสำหรับโค้ดบล็อกดังกล่าวและผลลัพธ์(CX, *D*)ที่ได้แสดงให้เห็นดังต่อไปนี้ ลำดับของการดำเนินการเข้ารหัส (ZC, RLC, SC, หรือ MRC) ที่ตำแหน่งบิต (row, column) สำหรับบิตเพลน *P* ผลลัพธ์ CX และ *D* แสดงให้เห็นที่หลังเครื่องหมาย Colon (:) ในแต่ละขั้นตอน ยกตัวอย่างเช่น การดำเนินการแรกในลำดับข้างล่างแสดงให้เห็นถึง Run-length coding (RLC) ที่ทำใน CUP ณ ตำแหน่ง (0, 0) ในบิตเพลนแรก ผลลัพธ์ที่ได้คือ CX = 17 และ *D* = 1 ผลลัพธ์ (CX, *D*) ที่ได้จะอยู่ในรูป (17, 1) แสดงให้เห็นว่ามีบิต 1 แถวละหนึ่งตัวในแต่ละแถวทั้งสี่แถวของตัวอย่างปัจจุบันที่แสกนมา ดังนั้น (CX, *D*) สองคู่มีค่าเป็น (18, 1) และ (18, 0) ซึ่งบิตที่ใช้ในการตัดสินใจ (Decision bit) ทั้งสองนี้แสดงให้เห็นว่าบิต 1 ตัวแรกที่ตำแหน่ง Zero based location $(10)_2 = 2$ ดังแสดงในคอลัมน์แรกของ υ^2 *Sign Coding* (SC) อยู่ต่อจาก RLC และเมื่อสถานะของตัวแปรต่างๆ มีค่าเริ่มต้นเป็นศูนย์บริบท 9 และ $\chi\hat{} = 0$ ถูกเลือกมาจากตาราง 2.11 บิตที่ใช้ในการตัดสินใจที่ $D = \chi(0, 0) \otimes \chi\hat{} = 1 \otimes 0 = 1$ (ซึ่ง $\otimes$ แทนเครื่องหมาย *Exclusive-OR*) ถูกนำไปใช้ดังแสดงในขั้นตอนที่ 4 ของการทำ CUP สำหรับบิตเพลนที่ 2 ข้างล่าง ภายหลังจากการทำ Pass แต่ละครั้งรายละเอียดต่าง ๆ ของสภาพของตัวแปร σ, η และ σ' ในบิตเพลนปัจจุบันก็จะถูกแสดงรายการออกมาเช่นกัน

CUP สำหรับบิตเพลน 2 (Bit-Plane 2)

1. RLC for (0, 0) : CX=17, *D*=1
2. RLC for (2, 0) :18, 1
3. RLC for (2, 0) :18, 0
4. SC for (2, 0) :9, 1
5. ZC for (3, 0) :3, 0
6. ZC for (0, 1) :0, 0
7. ZC for (1, 1) :1, 1
8. ZC for (1, 1) :9, 0
9. ZC for (2, 1) :7, 0
10. ZC for (3, 1) :1, 1
11. ZC for (3, 1) :9, 0
12. ZC for (0, 2) :1, 0
13. ZC for (1, 2) :5, 0
14. ZC for (2, 2) :2, 0
15. ZC for (3, 2) :5, 0
16. RLC for (0, 3) :17, 1
17. RLC for (0, 3) :18, 0
18. RLC for (0, 3) :18, 0
19. ZC for (0, 3) :9, 0
20. ZC for (1, 3) :3, 0
21. ZC for (2, 3) :0, 0
22. ZC for (3, 3) :0, 0

υ^2

0	0	0	1
0	1	0	0
1	0	0	0
0	1	0	0

σ

0	0	0	1
0	1	0	0
1	0	0	0
0	1	0	0

η

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

σ'

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

ตารางแสดง CUP สำหรับ Bit-Plane 2

SPP สำหรับบิตเพลน 1 (Bit-Plane 1)

23. ZC for (0, 0):CX=1, *D*=1
24. ZC for (0, 0):9, 0
25. ZC for (1, 0):7, 1
26. ZC for (1, 0):12, 1
27. ZC for (3, 0):7, 0
28. ZC for (0, 1):7, 0
29. ZC for (2, 1):7, 0
30. ZC for (0, 2):6, 0
31. ZC for (1, 2):6, 1
32. ZC for (1, 2):12, 0
33. ZC for (2, 2):3, 1
34. ZC for (2, 2):10, 1
35. ZC for (3, 2):7, 0
36. ZC for (1, 3):7, 0
37. ZC for (2, 3):6, 1
38. ZC for (2, 3):12, 1
39. ZC for (3, 3):3, 1
40. ZC for (3, 3):10, 0

υ^1			
1	0	0	0
1	1	1	0
0	0	1	1
0	1	0	1

σ			
1	0	0	1
1	1	1	0
1	0	1	1
0	1	0	1

η			
1	1	1	0
1	0	1	1
0	1	1	1
1	0	1	1

σ'			
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

ตารางแสดง SPP สำหรับ Bit-Plane 1

MRP สำหรับบิตเพลน 1 (Bit-Plane 1)

41. MRC for (2, 0) :CX=15, *D*=0
42. MRC for (1, 1) :15, 1
43. MRC for (3, 1) :14, 1
44. MRC for (0, 3) :14, 0

υ^1			
1	0	0	0
1	1	1	0
0	0	1	1
0	1	0	1

σ			
1	0	0	1
1	1	1	0
1	0	1	1
0	1	0	1

η			
1	1	1	0
1	0	1	1
0	1	1	1
1	0	1	1

σ'			
0	0	0	1
0	1	0	0
1	0	0	0
0	1	0	0

ตารางแสดง MRP สำหรับ Bit-plane 1

***CUP for* สำหรับบิตเพลนที่ 1 (*Bit-Plane 1*)**

(หมายเหตุ: Pass นี้จะไม่ทำให้เกิด CX or *D* ใดๆ)

υ^1			
1	0	0	0
1	1	1	0
0	0	1	1
0	**1**	**0**	**1**

σ			
1	0	0	1
1	1	1	0
1	0	1	1
0	**1**	**0**	**1**

η			
1	1	1	0
1	0	1	1
0	1	1	1
1	**0**	**1**	**1**

σ'			
0	0	0	1
0	1	0	0
1	0	0	0
0	**1**	**0**	**0**

ตารางแสดง CUP สำหรับ Bit-Plane 1

SPP สำหรับบิตเพลน 0 (*Bit-Plane 0*)

45. CZ for (3, 0) :CX=7, *D*=0
46. CZ for (0, 1) :7, 0
47. CZ for (2, 1) :8, 1
48. CZ for (2, 1) :11, 0
49. CZ for (0, 2) :7, 0
50. CZ for (3, 2) :8, 0
51. CZ for (1, 3) :7, 1
52. CZ for (1, 3) :13, 0

υ^0			
1	0	0	1
1	1	0	1
0	1	0	1
0	0	0	0

σ			
1	0	0	1
1	1	1	1
1	1	1	1
0	1	0	1

η			
0	1	1	0
0	0	0	1
0	1	0	0
1	0	1	0

σ'			
0	0	0	1
0	1	0	0
1	0	0	0
0	1	0	0

ตาราง SPP สำหรับ *Bit-Plane 0*

MRP สำหรับบิตเพลน 0 (Bit-Plane 0)

53. MRC for (0, 0) : CX=15, *D*=1
54. MRC for (1, 0) :15, 1
55. MRC for (2, 0) :16, 0
56. MRC for (1, 1) :16, 1
57. MRC for (3, 1) :16, 0
58. MRC for (1, 2) :15, 0
59. MRC for (2, 2) :15, 0
60. MRC for (0, 3) :16, 1
61. MRC for (2, 3) :15, 1
62. MRC for (3, 3) :15, 0

υ^0

1	0	0	1
1	1	0	1
0	1	0	1
0	0	0	0

σ

1	0	0	1
1	1	1	1
1	1	1	1
0	1	0	1

η

0	1	1	0
0	0	0	1
0	1	0	0
1	0	1	0

σ'

1	0	0	1
1	1	1	0
1	0	1	1
0	1	0	1

ตารางแสดง MRP สำหรับ Bit-Plane 0

CUP สำหรับบิตเพลน 0 (Bit-Plane 0)

(หมายเหตุ: Pass นี้จะไม่ทำให้เกิด CX or *D* ใดๆ)

υ^0

1	0	0	1
1	1	0	1
0	1	0	1
0	0	0	0

σ

1	0	0	1
1	1	1	1
1	1	1	1
0	1	0	1

η

0	1	1	0
0	0	0	1
0	1	0	0
1	0	1	0

σ'

1	0	0	1
1	1	1	0
1	0	1	1
0	1	0	1

ตารางแสดง CUP สำหรับ Bit-Plane 0

2.6.3 Binary Arithmetic Coding – MQ-Coder

การเข้ารหัสบิตเพลนเป็นส่วนๆ (EBCOT) เป็นตัวสร้างลำดับของสัญลักษณ์ (Symbol) ต่างๆ กับคู่ของบริบทและคำสั่งตัดสินใจ (CX, *D*) ขึ้นในแต่ละ Coding Pass ส่วน Binary Arithmetic MQ-coder ที่สามารถดัดแปลงใช้ได้โดยอยู่บนพื้นฐานของบริบทที่ใช้ใน JBIG2 นั้นก็ถูกนำมาดัดแปลงใช้ในการเข้ารหัสสัญลักษณ์ดังกล่าวในมาตรฐาน JPEG2000 ด้วย ค่าที่เป็นไปได้ต่าง ๆ (Qe) และประมาณการที่เป็นไปได้/กระบวนการที่ได้ทางตัวเลขนั้นได้มาจากมาตรฐาน JPEG2000 ดังตารางอ้างอิงแสดงค่าความน่าจะเป็นได้เกี่ยวกับฟังก์ชั่นทั้ง 4 (ตารางที่ 2.14) QM-coder เป็นตัวเข้ารหัส (Coder)หนึ่งใน Binary arithmetic coding (BAC) ที่ใช้ใน JPEG และเป็นตัวเข้ารหัสที่ถูกนำมาดัดแปลงใช้ใน JPEG2000 ซึ่งเรียกว่า MQ-coder ซึ่งจะต่างออกไปจาก *QM-coder* ในกระบวนการใช้ *MQ-coder* บนพื้นฐานของมาตรฐาน JPEG2000 นอกเหนือไปจากตารางที่แสดงค่าความน่าจะเป็นกับตาราง Qe (Qe-table) แล้ว ตารางเทียบค่า I(CX) และค่า MPS(CX) ก็มีความจำเป็นต่อการใช้ MQ-coder เหมือนกันเนื่องจากการเข้ารหัสของตัวเข้ารหัสบิตเพลนนั้นก่อให้เกิดบริบทที่แตกต่างกันขึ้นถึง 19 บริบทและเราจำเป็นต้องบันทึกร่อง(track) ของสถานะและดรรชนีของ Qe-table ของแต่ละบริบทด้วย สำหรับตารางเทียบ I(CX)

นั้นนำไปใช้ในการบันทึกร่อง(track) ของสถานะของ Qe-table และค่าเริ่มต้นที่มาตรฐาน JPEG2000 เป็นตัวกำหนดดังในตารางที่ 2.13 ที่แสดงบริบทที่แตกต่างกัน 19 บริบทและตารางดรรชนีเริ่มต้นสำหรับค่าประมาณการ BAC ส่วนตารางเทียบ MPS(CX) นั้นจะเป็นตัวกำหนดค่า 0 หรือ 1 ของค่าของสัญลักษณ์ (Symbol) ที่เป็นไปได้ของบริบท CX และ ค่า MPS(CX) ทุกค่านั้นถูกเริ่มต้นที่ค่า 0 ในตารางที่ 2.14 นั้นแสดงให้เห็นถึงตารางเทียบ 4 ตารางคือ ค่า Qe(I(CX)), NMPX(I(CX)), NLPS(I(CX)) และ SWITCH(I(CX)) ตามลำดับโดย

I(CX) คือดรรชนีปัจจุบันของบริบท CX

Qe(I(CX)) แสดงค่าที่เป็นไปได้

NMPX(I(CX))/ NLPS(I(CX)) แสดงดรรนีถัดไปสำหรับการคืนค่าปกติ MPS/LPS

SWITCH(I(CX)) คือตัวบ่งชี้ว่าค่า 0 และ 1 ของ MPS(CX) นั้นจำเป็นต้องเปลี่ยนหรือไม่

ตารางที่ 2.14 BAC Qe-value และตารางเทียบค่าความน่าจะเป็นโดยประมาณ

Index	Qe	NMPS	NLPS	SWITCH
0	0x5601	1	1	1
1	0x3401	2	6	0
2	0x1801	3	9	0
3	0x0AC1	4	12	0
4	0x0521	5	29	0
5	0x0221	38	33	0
6	0x5601	7	6	1
7	0x5401	8	14	0
8	0x4801	9	14	0
9	0x3801	10	14	0
10	0x3001	11	17	0
11	0x2401	12	18	0
12	0x1C01	13	20	0
13	0x1601	29	21	0
14	0x5601	15	14	1
15	0x5401	16	14	0
16	0x5101	17	15	0
17	0x4801	18	16	0
18	0x3801	19	17	0
19	0x3401	20	18	0
20	0x3001	21	19	0
21	0x2801	22	19	0
22	0x2401	23	20	0
23	0x2201	24	21	0
24	0x1C01	25	22	0
25	0x1801	26	23	0
26	0x1601	27	24	0
27	0x1401	28	25	0
28	0x1201	29	26	0
29	0x1101	30	27	0
30	0x0AC1	31	28	0
31	0x09C1	32	29	0
32	0x08A1	33	30	0
33	0x0521	34	31	0

ตารางที่ 2.14 BAC Qe-value และตารางเทียบค่าความน่าจะเป็นโดยประมาณ (ต่อ)

Index	Qe	NMPS	NLPS	SWITCH
34	0x0441	35	32	0
35	0x02A1	36	33	0
36	0x0221	37	34	0
37	0x0141	38	35	0
38	0x0111	39	36	0
39	0x0085	40	37	0
40	0x0049	41	38	0
41	0x0025	42	39	0
42	0x0015	43	40	0
43	0x0009	44	41	0
44	0x0005	45	42	0
45	0x0001	45	43	0
46	0x5601	46	46	0

ตารางเทียบดังกล่าวข้างต้นนี้จะถูกนำไปใช้ทั้งเป็นตัวเข้ารหัส(Encoder) และตัวถอดรหัส (Decoder) ซึ่งวิธีการใช้งานมีดังนี้

2.6.3.1 การใช้ MQ – Encoder

การใช้ MQ-encoder นั้นต้องใช้กับ Register A และRegister C ขนาด 32 bit ซึ่งมีโครงสร้างดังต่อไปนี้

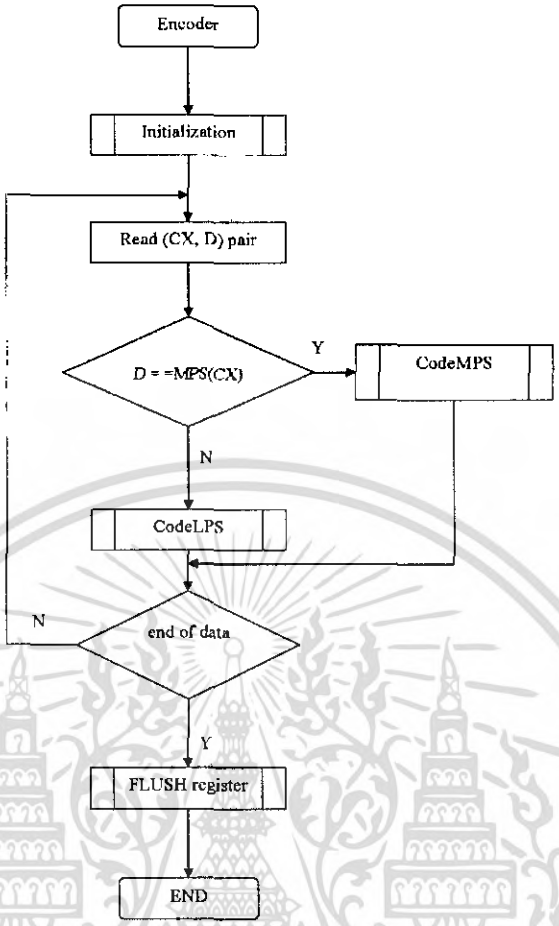
ตารางที่ 2.15 BAC Encoder Register Structures

32-Bit Register	MSB	LSB
C (Code Register)	0000 cbbb bbbb b*sss*	xxxx xxxx xxxx xxxx
A (Current Interval Value)	0000 0000 0000 0000	aaaa aaaa aaaa aaaa

โดย

- "a" คือ กลุ่ม bit ในรีจีสเตอร์ A
- "x" คือ กลุ่ม bit ในรีจีสเตอร์ C
- "s" คือ bit ว่าง, ซึ่งเตรียมการการจำกัดบน carryover
- "b" คือ bit สำหรับ ByteOut
- "c" คือ carry bit

Register A คือ Register แบบอันตรภาคที่ประกอบด้วยค่าอันตรภาคปัจจุบันที่ต้องใช้ใน MQ-encoder ส่วน Register C คือ Register ของรหัส ที่ประกอบไปด้วยส่วนต่าง ๆ ของ Codeword ในแต่ละขั้นของการเข้ารหัส ค่าเริ่มต้นของ Register A คือ 0 x 00008000 ซึ่งแสดงอันตรภาค (Interval) เริ่มต้นที่เป็นไปได้ และค่าเริ่มต้นของ Register C คือ 0 x 00000000 ซึ่งหมายความว่ายังไม่มีการสร้าง Codeword ใด ๆ ขึ้น



รูปที่ 2.32 BAC Encoder Flowchart

รูปที่ 2.32 แสดงความสัมพันธ์ของส่วนประกอบต่าง ๆ ของการทำ MQ-coder จะเห็นได้ว่ากระบวนการของ CodeMPS หรือ CodeLPS นั้นขึ้นอยู่กับค่าของ Decision bit (D) และ ค่า More probable symbol of context (MPS(CX)) ภายหลังจากที่ทุกสัญลักษณ์ (Symbol) ถูกดำเนินการแล้ว กระบวนการ FLUSH Register จะถูกนำมาใช้เพื่อบรรจุบิต 1 เข้าไปใน Register C ให้มากที่สุดเท่าที่จะเป็นไปได้ก่อนที่จะส่งไบท์สุดท้ายออกไปในรูปแบบของ Compressed codewords ส่วนรหัส (Code) ต่างๆ ของกระบวนการ MQ-coder Algorithm นั้นมีดังนี้

Initialization( ): กระบวนการ Initialization เป็นกระบวนการเริ่มต้น Register และตัวแปรต่าง ๆ ในการทำ MQ-encoder ตัวแปร B คือไบท์ที่บ่งชี้โดย Buffer pointer BC ที่ถูกบีบอัดมา ส่วน BPST คือ Pointer ที่ชี้ให้เห็นตำแหน่งที่ไบท์แรกจะเข้าไปติดตั้ง แล้วส่วน CT คือตัวนับที่ใช้ในการนับจำนวนของการเลื่อน (Shift) ที่ใช้ใน Register A และ Register C

```
Initialization()
{
      A = 0x00008000;
      C = 0x00000000;
      BP = BPST - 1;
      CT = 12;
      If (B == 0xFF) CT = 13;
      Reset ICX) and MPS(CX) with their initial values.
}
```

CodeMPS( ): กระบวนการ CodeMPS เป็นการบวนการที่โดยพื้นฐานแล้วใช้เพิ่มค่าความน่าจะเป็นของ current context, qe = Qe(I(CX)) เข้าไปใน Register C และปรับค่าอันตรภาค A ไปที่ A – qe เงื่อนไขการแลกเปลี่ยนระหว่าง MPS/LPS อาจเกิดขึ้นได้และทั้ง Register A และ Register C จะถูกตั้งค่าตามเดิมนั้น ขึ้นอยู่กับค่าอันตรภาคย่อยอันใหม่สำหรับ MPS ส่วน NMPS(I(CX)) นั้นเป็นตัวเลือกดรรชนีถัดไปให้กับบริบทปัจจุบัน CX

```
CodeMPS()
{
  qe = Qe(I(CX));
  A = A - qe;              /* new subinterval for MPS */

  if ( A < 0x8000 ){
      if ( A < qe )        /* condition exchange */
          A = qe;
      Else
          C = C + qe;
          / * choose next index for MPS */
          I(CX)= NMPS (I(CX));

        Call RenormalizationENC();
  }
  else
      C = C + qe;
}
```

CodeLPS( ): กระบวนการ CodeLPS เป็นกระบวนการแรกในการปรับค่าอันตรภาค A ไปยัง A – qe เมื่อ qe – Qe(I(CX)) ถ้าค่าอันตรภาคย่อยอันใหม่สำหรับ MPS มีขนาดใหญ่กว่าค่าของ qe แล้ว A จะถูกปรับค่าไปที่ qe ส่วน Register C นั้นยังคงไม่มีการเปลี่ยนแปลง นอกเสียจากว่า C จะถูกปรับค่าไปที่ C + qe ถ้า SWTCH flag สำหรับ index I(CX) เป็น 1 ดังตารางที่ 2.14 ค่า 0 และ 1 ของ MPS(CX) อาจมีการเปลี่ยนแปลงขึ้นอยู่กับ index ของ context (I(CX)) กระบวนการ Renormalization นั้นมักถูกนำมาใช้ในกระบวนการ CodeLPS( ) อยู่เสมอ ส่วนตัวแปร NLPS(I(CX)) เป็นตัวเลือกดรรชนีถัดไปของดรรนีปัจจุบัน CX

```
CodeLPS()
{

  qe = Qe(I(CX));
  A = A - qe;              /* new subinterval for MPS */

  if ( A >= qe )
      A = qe;              /* C is left unchanged */
  Else                     /* conditional exchange */
      C = C + qe;

  If ( switch(I(CX)) == 1)
      /* change the sense of MPS(CX) */
      MPS(CX) = 1 - MPS(CX);

   /* choose next index for LPS */
   I(CX) = NLPS(I(CX));
   Call RenormalizationENC();
}
```

RenormalizationENC( ): ฟังก์ชั่นนี้จะถูกใช้หลังจากการทำ LPS coding อยู่เสมอ และจะถูกใช้เมื่อไหร่ก็ตามที่หลังจากการทำ MPS coding แล้วค่าอันตรภาคใน Register A มีค่าน้อยกว่า 0 x 8000 กระบวนการ Normalization ถูกนำมาใช้เพื่อให้แน่ใจว่าค่าอันตรภาค A นั้นต้องมีค่ามากกว่า 0 x 8000 อยู่เสมอโดยการย้อนกลับกระบวนการเลื่อนไปทางซ้าย (Left-shifting) ทั้ง Register A และ Register C ที่บิต 1 ทุก ๆ ครั้งจนกว่า A จะมีค่ามากกว่าหรือเท่ากับ 0 x 8000 แล้วถ้ามีความจำเป็นที่จะต้องส่งข้อมูลบีบอัดออกไปให้เรียกใช้กระบวนการ ByteOut( )

```
RenormalizationENC()
{
  Do
  {
     A = A << 1; /* left shift 1 bit */
     C = C << 1; /* left shift 1 bit */
     CT = CT -1;
    if (CT == 0) call ByteOut();
  } while (A < 0 x 8000)
}
```

ByteOut( ): เป็นกระบวนการส่งออกข้อมูลบับอัดขนาด 1 ไบท์ในแต่ละครั้ง ในกระบวนการนี้ประกอบไปด้วยสองกระบวนการสำคัญคือ bit_Stuffing( ) หรือ no_bit_Stuffing( ) เพื่อใช้ในการจำกัดจำนวน carry bit เข้าไปในไบท์สมบูรณ์และ bit-stuffing หลังจากการเกิดไบท์ 0xFF

```
ByteOut()
{
  if ( B == 0xFF ) call bit_Stuffing();
  else{
         if ( C < 0x08000000 ) /* no carry bit */
                call no_bit_Stuffing();
         else {
                B = B + 1; /* add carry bit to B */
                if ( B == 0xFF ) {
                    C = C & 0x07FFFFFF;
                    call bit_Stuffing();
                }
                else
                    call no_bit_Stuffing();
         }
  }
}
```

Bit_Stuffing( ): ในกระบวนการนี้ carry bit c และ upper 7 ByteOut bits bs (ดังตาราง 7.8) จะถูกย้ายเข้าไปใน byte B

```
bit_Stuffing()
{
  BP= BP + 1;          /* output B                */
  B >> 20;             /* "cbbb bbb" bit of C     */
  C = C & 0x000FFFFF;
  CT = 7;
}
```

no_bit_Stuffing( ): ในกระบวนการนี้ 8 ByteOut bits bs (ดังตาราง 7.8) จะถูกย้ายเข้าไปใน byte B

```
no_bit_Stuffing()
{
    BP = BP + 1;         /* output B                */
    B = C >> 19;         /* "bbb bbb b" bit of C    */
    C = C & 0x0007FFFF;
    CT = 8;
}
```

FLUSHregister( ): เมื่อ symbol ทุกตัวสำหรับแต่ละ code-block ที่สร้างขึ้นโดย EBCOT ได้ทำการเข้ารหัสแล้ว กระบวนการ FLUSHregister จะเริ่มดำเนินการเพื่อบรรจุบิต 1 เข้าไปใน Register C ให้มากที่สุดเท่าที่จะเป็นไปได้ก่อนที่มันจะส่งออกไบท์สุดท้ายที่อยู่ในรูปแบบ Codewords ที่บีบอัดแล้ว

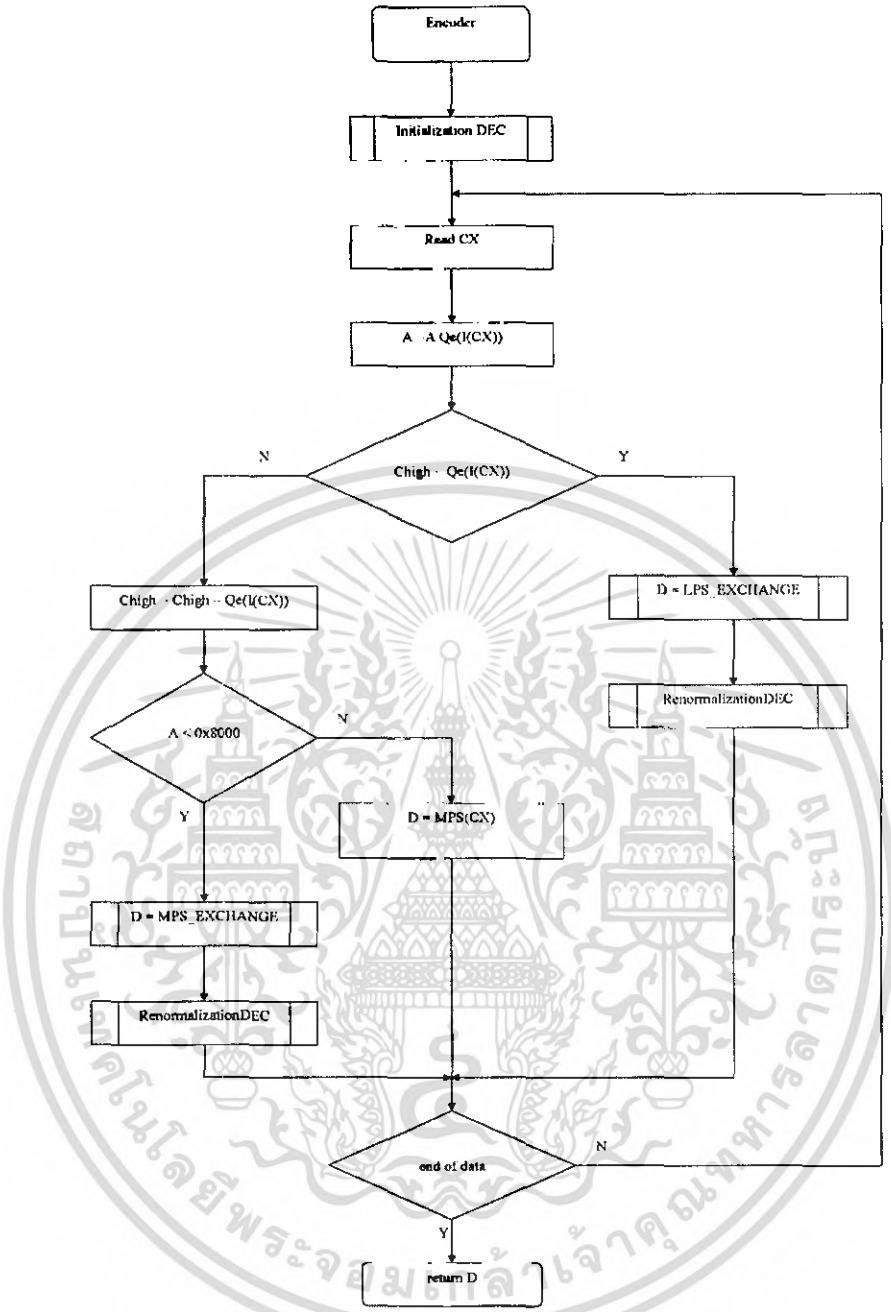
```
FLUSHregister()
{
    TempC = C + A;
    C = C | 0x0000FFFF;
    If ( C >= TempC) C = C - 0x00008000;

    C = C << CT;
    call ByteOut();
    C =C << CT;
    call ByteOut();

    if ( B == 0xFF)
        discard B;
    else
        BP = BP + 1    /* output B */
}
```

2.6.3.2 การใช้ MQ – Decoder

การทำ MQ-decoder ต้องการ Register ขนาด 16 บิตจำนวน 3 อย่างได้แก่ Register Chigh, Register Clow, และ Register A โครงสร้างของ Register ทั้งสามนี้ได้แสดงไว้ในตาราง 7.9 โดยหากรวมเอา Register Chigh และ Register Clow เข้าด้วยกันก็จะได้เป็น Register C ขนาด 32 บิตในระหว่างที่ทำการถอดรหัส(Decoding) นั้นข้อมูลใหม่จะถูกใส่เข้าไปที่ Register Clow ในตำแหน่ง Upper 8 bits (b บิตดังตาราง 2.16) ในขนาด 1 ไบท์ต่อครั้งส่วน Register A ก็มีค่าเริ่มต้นที่ 0x8000 เหมือนในการทำ MQ-encoder

ตารางที่ 2.16 BAC Decoder Register Structures

Register	MSB	LSB
Chigh	xxxx xxxx	xxxx xxxx
Clow	bbbb bbbb	0000 0000
A	*aaaa aaaa*	*aaaa aaaa*



รูปที่ 2.33 BAC Decoder Flowchart

รูปที่ 2.33 แสดงความสัมพันธ์ของโฟลชาร์ตส่วนประกอบต่าง ๆ ของการทำ MQ-decoder ส่วนรหัสcode) ต่างๆ ของกระบวนการ MQ-decoder Algorithm นั้นมีดังนี้

InitializationDEC(): อันดับแรกให้ใส่ไบท์ที่บีบอัดแล้วเข้าไปที่ตำแหน่ง Lower 8 bits ของ Register Chigh แล้วจากนั้น byte ใหม่จะถูกอ่านโดยกระบวนการ ByteIn()เพื่อกำหนดให้ค่าของ Register C มีค่าเริ่มต้นเท่ากับค่าเริ่มต้นของ Register A, Register C จะถูกเลื่อนไปทางซ้ายจำนวน 7 บิด และตัวนับการเลื่อน CT ก็จะถูกปรับตามไปด้วย

```
InitializationDEC()
    {
      /* BP is pointing to the first compressed byte  */
      /* B is the byte pointed to by the pointer BP   */
      BP = BPST;
      C = B << 16;

      call ByteIn();

      C = C << 7;
      CT = CT - 7;
      A = 0x8000;
      reset I(IX) and MPS(CX) with their initial values.
    }
```

ByteIN(): กระบวนการ ByteIn ทำหน้าที่อ่านไบท์ 1 ของบิตสตรีมที่บีบอัดแล้ว (Compressed bitstream) ในทุก ๆ ครั้งที่มันถูกเรียกและทำการชดเชยบิตที่บรรจุเข้าไปแล้วที่ 0xFF ไบท์ที่ถูกใส่เข้าไปในขั้นตอนการเข้ารหัส ถ้าหากพบว่าไบท์ขนาด 0xFF นั้นมีขนาดใหญ่กว่า 0x8F แล้วบิต 1 หลายๆ บิตจะถูกป้อนเข้าไปที่กระบวนการ ถอดรหัส

```
ByteIn()
{
  If (B == 0xFF){
         /* B1 is the byte pointer to by BP+1 */
         If ( B1 > 0x8F ){
           /* feed '1' bits to the decoder */
           C = C + 0xFF00;
           CT = 8;
         }
         else {
          BP = BP + 1;
            C = C + ( B << 9 );
           CT = 8;
         }
  }
  else {
         BP = BP + 1;
         C = C + (B << 8);
         CT = 8;
  }
}
```

LPS_EXCHANGE(): เงื่อนไขการแลกเปลี่ยน (Exchange) นั้นจะเกิดขึ้นหรือไม่เกิดขึ้นก็ได้ขึ้นอยู่กับค่า LPS Subinterval Value Qe(I(CX)) และค่า MPS Subinterval Value ของ Register A ในกรณีเดียวกันค่า MPS Subinterval Value ของ Register A จะถูกอัพเดตด้วย Qe(I(CX)) ส่วน Decision bit D จะถูกถอดรหัสหรือไม่ขึ้นอยู่กับเงื่อนไขนั้น ๆ

```
LPS_EXCHANGE()
{
     if ( A < Qe(I(CX)) ){ /* conditional exchange */
          A = Qe(I(CX));
          D = MPS(CX);
          I(CX) = NMPS(I(CX));
     }
     else {
          A = Qe(I(CX));
          D = 1 - MPS(CX);
          if ( SWITCH(I(CX)) == 1 )
          I(CX) = NLPS(I(CX));
     }

     return D;
}
```

MPS_EXCHANGE(): คล้ายกับขั้นของการทำ LPS_EXCHANGE() เงื่อนไขการเปลี่ยนแปลงใดๆ (Conditional exchange) อาจเกิดขึ้นได้โดยขึ้นอยู่กับค่าต่าง ๆ ของ Register A และ Qe(I(CX)) ส่วน Decision bit D จะถูกถอดรหัสหรือไม่ขึ้นอยู่กับเงื่อนไขนั้นๆ แต่อย่างไรก็ตาม MPS Subinterval Value ของ Register A จะไม่ถูกอัพเดตในขั้น MPS_EXCHANGE()

```
MPS_EXCHANGE()
{
      if ( A < Qe(I(CX)) ){ /* conditional exchange */
            D = 1 - MPS(CX);
            if ( SWITCH(I(CX)) == 1 ) MPS(CX) = 1 - MPS(CX);
            I(CX) = NLPS(I(CX));
      }
      else {
            D = MPS(CX);
            I(CX) = NMPS(I(CX));
      }

      return D;
}
```

RenormalizationDEC():ขั้นตอนของการคืนค่าเดิมของการถอดรหัส (Decoder Renomalization)นั้นจำเป็นต้องใช้หลังจากที่มีการเรียกใช้กระบวนการ MPX_EXCHANGE() หรือ LPS_EXCHANGE() ตัวตัวนับ CT จะทำหน้าที่บันทึกร่อง(track) ของจำนวนของบิตต่างๆที่ถูกบีบอัดที่ยังเหลืออยู่ในส่วนของ Clow ของ Register C ถ้าหากค่าของ CT นั้นลดลงมาอยู่ที่ศูนย์ (0) ไบท์ที่ถูกบีบอัดอันใหม่จะถูกนำเข้าไปโดยใช้กระบวนการ ByteIn()

```
RenormalizationDEC()
{
      do
            {
            If ( CT == 0 ) call ByteIn();
            A = A << 1;
            C + C << 1;
            CT = CT - 1;
      } while ( A < 0x8000 )
}
```

2.7 Tier – 2 Coding

ในการเข้ารหัสภาพ การทำ Tier – 2 Coding จะเกิดขึ้นภายหลังจากเสร็จสิ้นการทำ Tier – 1 Coding แล้ว ส่วนอินพุทที่จะใส่เข้าไปในกระบวนการ Tier – 2 Coding นั้นคือกลุ่มของ Bit-plane coding passes ที่เกิดขึ้นระหว่างการทำ Tier – 1 Coding ข้อมูลของ Coding Pass ที่จัดรวมเป็นชุดข้อมูลและถูกส่งเข้าไปใน Tier – 2 Coding นั้นเรียกว่าแพคเกจ (Packet) ซึ่งเมื่ออยู่ในกระบวนการหนึ่ง ๆ นั้นจะเรียกว่า Packetization และผลของแพคเกจที่ได้นั้นก็จะถูกส่งไปที่ Final Code Stream ในกระบวนการทำ Packetization กำหนดให้เกิดขึ้นในโครงสร้างเฉพาะของ Coding pass ที่อยู่ใน Output Code Stream ซึ่งโครงสร้างนี้จะเป็นประโยชน์ให้ต่อการกำหนดรูปแบบของรหัส (Code) ที่ต้องการซึ่งรวมไปถึงระดับอัตราความสามารถ(Rate Scalability) และการวัดระดับความก้าวหน้า(Progressive Recovery) ด้วยความเที่ยงตรงและชัดเจน

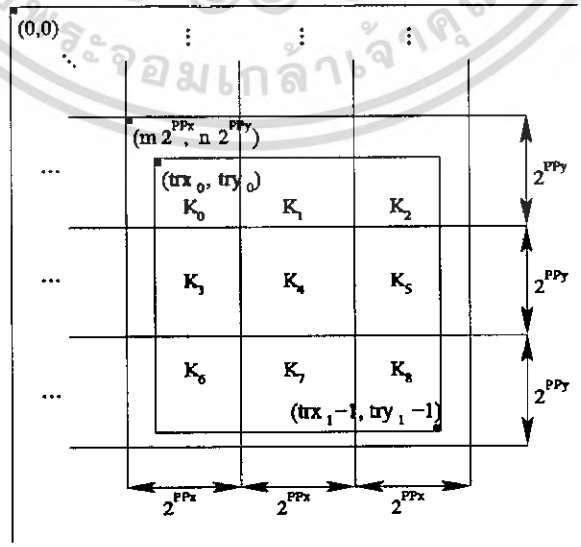
Packet ก็คือกลุ่มของชุดข้อมูล Coding pass data ซึ่งแต่ละแพคเกจนั้นจะประกอบไปด้วยสองส่วนคือ ส่วนที่เป็น Header และส่วนที่เป็น Body ส่วนแรกนั้นเป็นตัวชี้ให้เห็นว่า Coding pass ใดที่รวมอยู่ในแพคเกจในขณะที่ส่วนที่สองนั้นคือส่วนของ Coding pass data จริง ๆ แต่เมื่อเวลาที่มันอยู่ใน Code Stream นั้นส่วนของ Header และ Body อาจจะอยู่ด้วยกันหรือแยกกันก็ได้ขึ้นอยู่กับผลสำเร็จของแนวทางของการเข้ารหัส

ระดับอัตราความสามารถ(Rate Scalability) นั้นได้มาจาก (quality) Layers ข้อมูลที่ได้รับการเข้ารหัสในแต่ละไทล์ (Tile) นั้นจะถูกรวบรวมเข้าไปที่ L Layers จากหมายเลข 0 ถึง $L-1$ เมื่อ $L \geq 1$ แต่ละ Coding pass นั้นจะมีทั้งที่ถูกกำหนดให้เข้าไปเป็นส่วนหนึ่งใน L Layer และเป็นส่วนที่ถูกละทิ้งไป Coding pass ที่ประกอบไปด้วยข้อมูลที่สำคัญที่สุดจะถูกรวมไว้ใน Layer ที่ต่ำกว่าในขณะที่ Coding pass ที่ประกอบไปด้วยรายละเอียดประกอบนั้นจะถูกรวบรวมไว้ใน Layer ที่สูงกว่า ส่วนในระหว่างทำการ

ถอดรหัสนั้นภาพที่สร้างขึ้นมาใหม่จะมีคุณภาพเพิ่มขึ้นตามแต่ละ Layer ที่ทำสำเร็จในการเข้ากระบวนการ ในกรณีที่การบีบอัดเกิด Loss ขึ้นบาง Coding pass อาจจะถูกทิ้งไป (ไม่รวมอยู่ใน Layer ใด ๆ) ในกรณีที่ Rate control ต้องตัดสินว่า Coding pass ควรไปอยู่ใน Final code stream ใด ส่วนในกรณีที่การบีบอัดเกิด Lossless ขึ้นทุก Coding pass ต้องถูกรวมไว้ใน Layer แต่ถ้าหากมีการใช้ Layer มากขึ้น (นั่นคือ $L > 1$) Rate control จะต้องเป็นตัวกำหนดว่าแต่ละ Coding pass นั้นจะถูกใส่ไว้ใน Layer ใด เนื่องจากว่าบาง Coding pass นั้นอาจถูกละทิ้งไป การทำ Tier-2 coding จึงเป็น Primary source อันดับที่สองของข้อมูล Loss ที่เกิดขึ้นในเส้นทางการเข้ารหัสภาพ(Coding path)

เป็นที่ทราบกันแล้วว่าแต่ละ Coding pass นั้นประกอบด้วยส่วนประกอบเฉพาะ (Particular component), ระดับความคมชัด (Resolution level), Subband และ Code block ในการทำ Tier-2 coding นั้นแพคเกจหนึ่งก็จะก่อให้เกิดส่วนประกอบเฉพาะ (particular component), ระดับความคมชัด (resolution level), Layer และเกิดบริเวณของ 4-tuple แต่แพคเกจหนึ่ง ๆ นั้นไม่จำเป็นต้องประกอบไปด้วยข้อมูล Coding pass ทั้งหมดซึ่งหมายความว่าอาจมีบางแพคเกจที่ว่างเปล่าก็ได้ซึ่งแพคเกจว่างนี้บางทีก็มีความจำเป็นเนื่องจากแพคเกจหนึ่งนั้นต้องก่อให้เกิดการเชื่อมกันของทุก ๆส่วนประกอบ (Component) ความคมชัด (resolution), Layer และ อาณาเขต (Precinct) ถึงแม้ว่าผลลัพธ์ของแพคเกจที่ได้นั้นไม่ได้นำส่งข้อมูลใหม่ใด ๆ เลยก็ตาม

อาณาเขต(Precinct) มีความสำคัญต่อการจัดกลุ่ม Code block ต่าง ๆ ภายใน Subband หนึ่ง ๆ การแบ่งพาทิชั่น(Partition) ของอาณาเขตสำหรับ Subband ใด ๆ นั้นเกิดขึ้นจากการแบ่งพาทิชั่นของแบนด์ LL ที่เป็นพ่อแม่ของมันหรือ Parent LL Band (นั่นคือ LL Band ถัดไปที่มีระดับความคมชัดที่สูงกว่า) ระดับความคมชัดแต่ละระดับนั้นประกอบด้วยขนาดของอาณาเขตที่เป็นตัวเลข ความกว้างและความสูงของอาณาเขตที่เป็นตัวเลขนั้นต้องเป็นเลข 2 ยกกำลังและอยู่ในข้อจำกัดที่กำหนด(เช่น ความกว้างและความสูงสูงสุดคือ 2^{15}) LL Band สร้างขึ้นด้วยระดับความคมชัดแต่ละระดับที่แบ่งออกเป็นสองอาณาเขต (2 precincts) ซึ่งสามารถทำได้โดยการซ้อนแทน(Overlaying) LL Band ด้วยกริด (Grid) ที่มีพื้นที่ตามแนวนอนและแนวตั้งเป็น 2^{PPx} และ 2^{PPy} ตามลำดับดังรูปที่ 2.34



รูปที่ 2.34 Partitioning of Resolution into Precincts

จากรูปที่ 2.34 จะเห็นว่ากริดจะอยู่ในจุดเริ่มต้นเดียวกันกับระบบของ LL Band แนวขอบของ อาณาเขต (Precinct) ที่อยู่บริเวณขอบขอบ Subband อาจจะมีมิติที่เล็กกว่าขนาดที่เป็นตัวเลข ขอบเขตของอาณาเขต(Precinct) ที่ได้นั้นก็จะถูกกำหนดพื้นที่ลงใน Subband ลูก (Child subband) ของมัน(ถ้ามี) ใน Band ถัดไปที่มีระดับความคมชัดต่ำกว่าซึ่งทำได้โดยการใช้สูตรการเปลี่ยนรูป$(u, v) = (\lceil x/2 \rceil, \lceil y/2 \rceil)$ เมื่อ (x, y) และ (u, v) คือจุดร่วมของ LL Band และ Child Subband ตามลำดับ ชนิดของ Precinct partitioning ที่ประกอบขึ้นมานั้นเขตแดนของอาณาเขตนั้นจะกำหนดให้อยู่ในขอบเขตของโค้ดบล็อกอยู่เสมอ แต่ Precinct บางอันอาจเป็นไปได้เหมือนกันที่มันจะว่างเปล่า สมมติว่าขนาดของโค้ดบล็อกที่เป็นตัวเลขคือ $2^{xcb'} \times 2ycb'$ โดยสิ่งนี้เป็นผลให้ได้กลุ่มของโค้ดบล็อกต่าง ๆ เป็นตัวเลขนั้นคือ $2^{PPx'} - xcb' \times 2^{PPy' - ycb'}$ เมื่อ

$$PPx' = \begin{cases} PPx & \text{for } r = 0 \\ PPx - 1 & \text{for } r > 0 \end{cases}$$

$$PPy' = \begin{cases} PPy & \text{for } r = 0 \\ PPy - 1 & \text{for } r > 0 \end{cases}$$

และ r คือระดับความคมชัด

เมื่อ Coding pass data จากอาณาเขตที่ต่างกันถูกเข้ารหัสในแพคเกจที่แยกออกจากกันต่างหากการใช้อาณาเขตที่มีขนาดเล็กกว่าจะช่วยลดจำนวนของข้อมูลที่อยู่ภายในแต่ละแพคเกจและถ้าข้อมูลที่อยู่ในแพคเกจหนึ่งๆ นั้นมีน้อย การเกิดความผิดพลาดของบิต (Bit error) ก็มีแนวโน้มว่ามีน้อย (เพราะว่าในบางครั้งนั้นผิดพลาดของบิตในหนึ่งแพคเกจจะไม่มีผลกระทบต่อการถอดรหัสของแพคเกจอื่น) เพราะฉะนั้นการใช้อาณาเขตขนาดเล็กกว่าจะช่วยปรับปรุงโดยการลดจำนวนความผิดพลาดลงในขณะที่ประสิทธิภาพการเข้ารหัสจะแย่ลงหรือไม่นั้นขึ้นอยู่กับการเพิ่มจำนวนของแพคเกจที่มากเกินไป

การจัดลำดับข้อมูลแพคเกจใน Code stream นั้นมีมากกว่าหนึ่งวิธีซึ่งการจัดลำดับเหล่านั้นเรียกว่า "การเพิ่มลำดับมากขึ้น (Progression)" ซึ่งมีอยู่ 5 ลำดับข้อมูลคือ

1. การเรียงลำดับแบบ Layer – Resolution – Component – Position
2. การเรียงลำดับแบบ Resolution – Layer – Component – Position
3. การเรียงลำดับแบบ Resolution – Position – Component – Layer
4. การเรียงลำดับแบบ Position – Component – Resolution – Layer
5. การเรียงลำดับแบบ Component – Position – Resolution – Layer

ลำดับของแพคเกจต่าง ๆ นั้นจะมีการเรียงตามลำดับชื่อโดย Position คือตัวบ่งบอกถึงจำนวนของอาณาเขต (Precinct) และคีย์ของการเรียงลำดับจะจัดอันดับตามความมีนัยสำคัญของแพคเกจนั้นๆ ยกตัวอย่างเช่น ในกรณีที่การจัดลำดับแบบที่ 1 จะเห็นว่าแพคเกจดังกล่าวนั้นจะจัดให้ Layer ขึ้นก่อนตามด้วย Resolution ตามด้วย Component และ Precinct (position) เป็นลำดับสุดท้าย ซึ่งการเรียงลำดับแบบนี้จะสอดคล้องกับความคืบหน้าของการกู้คืนที่ตรงกับต้นฉบับ ส่วนการจัดลำดับแบบที่ 2 นั้นเรียงลำดับจากความคืบหน้าของการกู้คืนโดยดูจากความคมชัด(Resolution) และสามอย่างที่เหลือนั้นเป็นข้อมูลที่ค่อนข้างจำกัดและเป็นไปได้ที่จะการกำหนด Progression โดยผู้ใช้เพิ่มเติมเข้าไปโดยดูจากความสิ้นเปลืองที่เพิ่มขึ้นในการเข้ารหัสภาพ

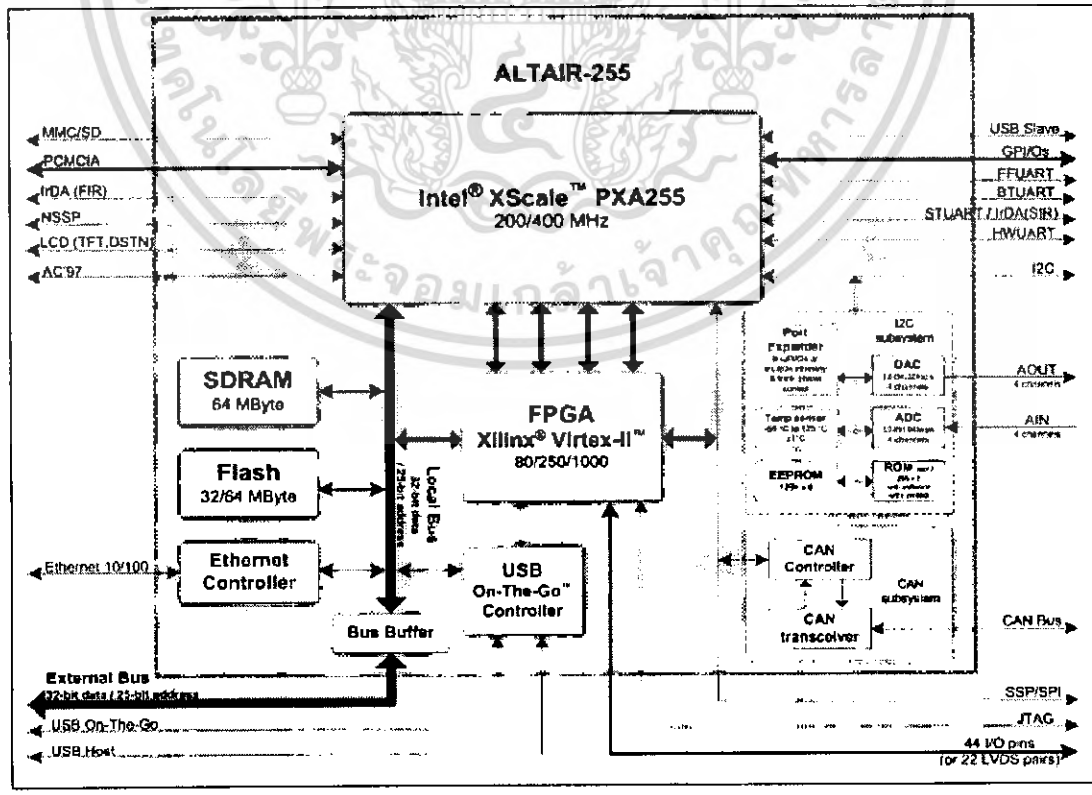
ในมุมมองที่ง่ายที่สุดก็คือทุกๆ แพคเกจจากแบบแผ่เรียง (Tile) หนึ่งใดปรากฏขึ้นพร้อมๆ กันใน Code stream แต่อย่างไรก็ตามข้อกำหนดที่มีอยู่ในการแทรกแพคเกจที่อยู่ต่างแบบแผ่เรียงกันเข้าไปนั้นก็ยังก่อให้เกิดการจัดเรียงข้อมูลแบบอื่นสามารถเกิดขึ้นได้ ยกตัวอย่างเช่น ถ้าความก้าวหน้าการกู้คืนของ Tiled image ถูกกำหนดเอาไว้ กรณีการเรียงแพคเกจแบบหนึ่งที่อาจเกิดขึ้นได้ ก็คือเอาทุกๆ แพคเกจที่เรียงข้อมูลโดยเอา Layer ขึ้นก่อนจากแบบแผ่เรียงต่างๆ มารวมไว้ด้วยกัน แล้วตามด้วยแพคเกจที่เรียงข้อมูลโดยเอา Layer ไว้เป็นอันดับที่สอง เป็นต้น

สำหรับการถอดรหัสภาพใน Tier – 2 นั้นจะทำได้โดยการตัดทอนข้อมูล Coding pass ที่มีอยู่หลากหลายออกไปจาก Code stream ซึ่งก็คือการทำ Depacketization และประกอบแต่ละ Coding pass ขึ้นมาโดยดูจากความสอดคล้องกับโค้ดบล็อกในกรณีที่เกิด Loss ขึ้นจะมี Coding pass เพียงบางส่วนเท่านั้นที่ที่จะได้รับการรับรองในการเข้ารหัสเพราะบางตัวอาจถูกละทิ้งไปโดยตัวเข้ารหัส (Encoder) ส่วนในกรณีที่เกิด Lossless ขึ้นทุกๆ Coding pass ต้องถูกส่งเข้าไปใน Code stream

2.8 Intel X-Scale PXA255 Architecture

ALTAIR-255ประกอบด้วย Intel® XScale™ PXA255 Xilinx® Virtex-II™ FPGA ลักษณะเด่นของ ALTAIR เป็นการรวมกันของ High-speed controller สามารถต่อ I/O ได้130 จุด และ Various I/O-standards (1.2-3.3Volt) สัญญาณเพิ่มได้ 840+ Mb/s ดังรูปที่ 2.35

มี I2C ควบคุมอุปกรณ์ต่อเชื่อมที่เก็บข้อมูล EEPROM non-volatile, มี Temperature sensor, Digital to Analog Converter (DAC) และ Analog Digital Converter (ADC) สำหรับการหาเอกลักษณ์บอร์ดไม่เหมือนใครหรือลักษณะเด่นความปลอดภัย Read-only และเก็บข้อมูลได้ยาวนาน

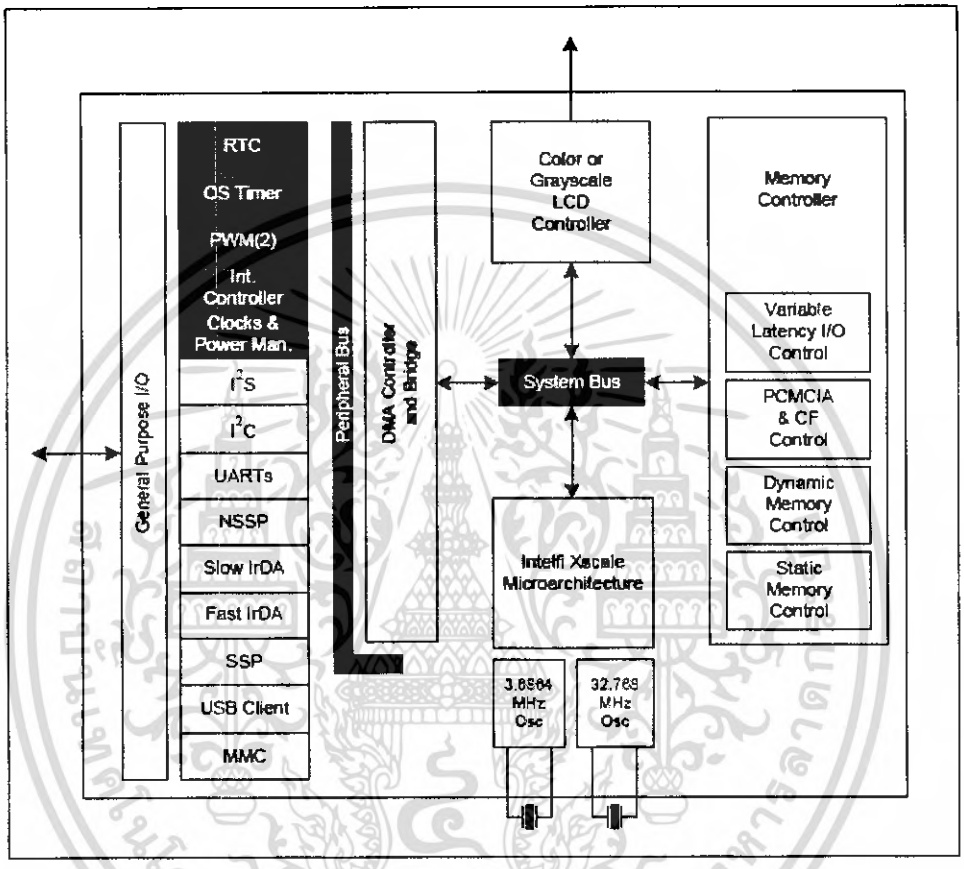


รูปที่ 2.35 Architecture

สามารถใช้ USB ในการเชื่อมต่อได้ และควบคุมด้วยชิพเดี่ยว

ความสามารถอีกอย่างคือสามารถ สั่งปิดด้วยซอฟแวร์ได้ ใช้กำลังในสถาน hole 3.3V 180mA

PXA255 มีการรวมระบบลงบนชิพเพื่อให้เกิดประสิทธิภาพสูงสุด กินพลังงานน้อย และมีขนาดเล็ก PXA255 สามารถใช้คำสั่งเหมือนกับ ARM Architecture Version 5TE ได้ และมีการคำนวณแบบ Floating point instructions และการเขียนโปรแกรมนั้นเขียนตามรูปแบบ ARM



รูปที่ 2.36 I/O Bus Controller

ตัว Processor มีการรองรับหน่วยความจำหลากหลายขนาด มีส่วนควบคุมหน้าจอแบบ LCD ขนาด 640x480 pixels

ยอมรับภาพระบบ Grayscale 1-, 2-, 4- and 8-bit และภาพสี 8- or 6-bit 256 รูปแบบ/512 ไบท์(byte) โดยการ Mapping

ALTAIR-255 Feature

Core

CPU	Intel® XScale™ PXA255 200 up to 400 MHz
SDRAM	64 MByte
Flash	32 MByte (64 MByte opt.)
FPGA	Xilinx® Virtex-II™

Interface

Address Bus	High-speed 32-Bit data and 25-Bit address bus, buffred
Ethernet	10/100Base-TX, full/half duplex
USB	USB 2.0, On-The-Go/host/device
CAN	Version 2.0B with high-speed transceiver 1Mb/s
LCD	Maximum 640x480x16 Bit/pixel, active (TFT) passive (DSTN)
I2C	400kBits/sec
Infrared	4Mbps, FIFO & DMA
MMC / SD card	
GPIO	130 GPI/Os, 840+ Mb/s
JTAG	Integrated chain

I2C Subsystem

EEPROM	128K x 8
Temp. Sensor	-55°C to 125°C, ±1°C
ADC	12-Bit, 94ksps, 4 channels
DAC	12-Bit, 22ksps, 4 channels
Port Expander	9 GPI/Os or 9 LEDs intensity and blink phase control
ROM	256 x 8 with software write protect

ELECTRICAL

Supply Voltage	Single 3.3Volt
Power	Min 0.6W, typical 1.2W, Comsumption max. 5.0W

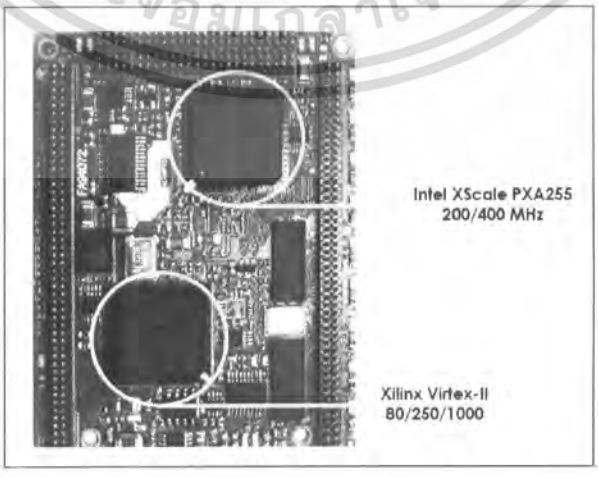
Mechanical

Dimensions	53.0 x 70.0 x 8.5 mm
Operating Temperature	0°C to 80°C

รูปที่ 2.37 Feature

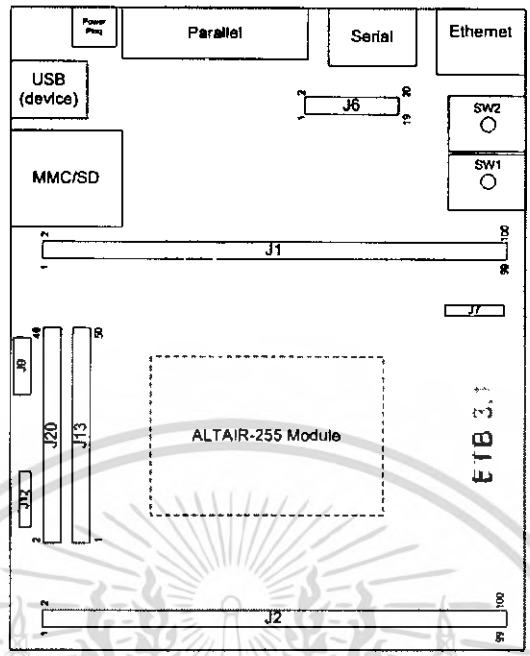
ตำแหน่งของอุปกรณ์

CPU



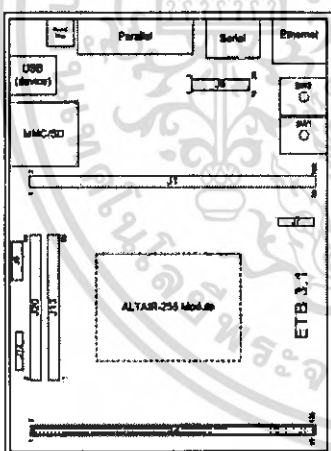
รูปที่ 2.38 ตำแหน่ง CPU

Connector



รูปที่ 2.39 Connector

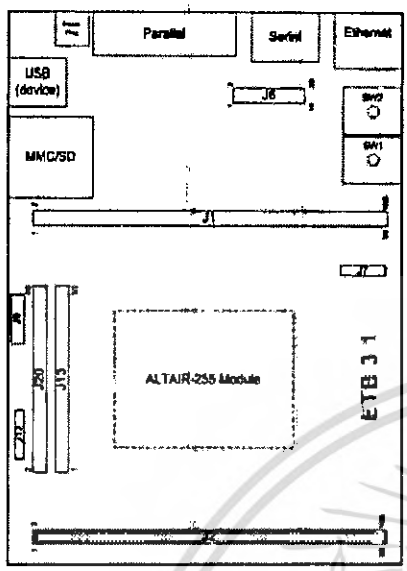
Pin Header J2 (J2:1 - J2:50)



Left			Right
Description	Pin		Description
GND	1	2	GND
NSSPRXD (NSSP)	3	4	NSSPSCLK (NSSP)
FFRXD (FFUART)	5	6	FFTXD (FFUART)
FFCTS (FFUART)	7	8	FFDCD (FFUART)
FFDSR (FFUART)	9	10	FFRI (FFUART)
FFDTR (FFUART)	11	12	FFRTS (FFUART)
NSSPSFRM (NSSP)	13	14	NSSPTXD (NSSP)
TDI (JTAG)	15	16	TDO (JTAG)
TMS (JTAG)	17	18	TCK (JTAG)
PWM0	19	20	PWM1
#TRST (JTAG)	21	22	GPIO22
SSPCLK (SSP/SPI)	23	24	SSPSFRM (SSP/SPI)
SSPTXD (SSP/SPI)	25	26	SSPRXD (SSP/SPI)
SSPEXTCLK (SSP/SPI)	27	28	BITCLK (AC'97)
SDATA_IN0 (AC'97)	29	30	SDATA_IN1 (AC'97)
SDATA_OUT (AC'97)	31	32	SYNC (AC'97)
#ACRESET (AC'97)	33	34	48MHz (PXA_USB)
CANL (CAN)	35	36	GND
CANH (CAN)	37	38	VCC
RTCCLK	39	40	3.6MHz
32KHz	41	42	GPIO1
3_SIGN - #BATT_ERR - #VDD_ERR - PWR_EN	43	44	SCL (I2C)
SCA (I2C)	45	46	DREQ0 (DMA)
DREQ1 (DMA)	47	48	GPIO0
FPGA_3_IO21	49	50	GCLK25

รูปที่ 2.40 Pin Connector J2 (J2:1 - J2:50)

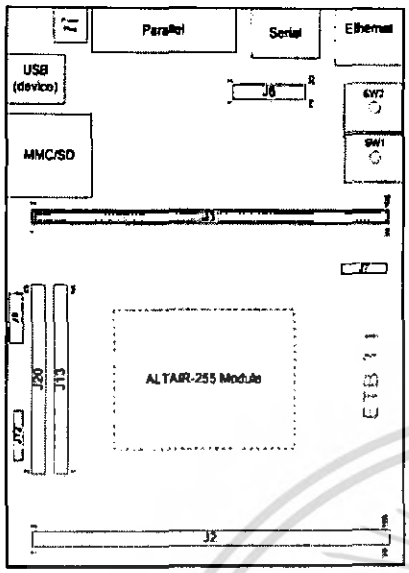
Pin Header J2 (J2:51 - J2:100)



Left			Right
Description	Pin		Description
FPGA_3_IO20	51	52	FPGA_3_IO17
FPGA_3_IO19	53	54	FPGA_3_IO16
FPGA_3_IO18	55	56	FPGA_3_IO15
FPGA_3_IO13	57	58	FPGA_3_IO14
FPGA_3_IO12	59	60	FPGA_3_IO08
FPGA_3_IO11	61	62	FPGA_3_IO09
FPGA_3_IO10	63	64	FPGA_3_IO04
FPGA_3_IO06	65	66	FPGA_3_IO05
FPGA_3_IO07	67	68	FPGA_3_IO00
FPGA_3_IO02	69	70	FPGA_3_IO01
FPGA_3_IO03	71	72	VCCO_3
FPGA_7_IO02	73	74	VCCO_7
FPGA_7_IO03	75	76	FPGA_7_IO00
FPGA_7_IO06	77	78	FPGA_7_IO01
FPGA_7_IO07	79	80	FPGA_7_IO04
FPGA_7_IO11	81	82	FPGA_7_IO05
FPGA_7_IO10	83	84	FPGA_7_IO08
FPGA_7_IO13	85	86	FPGA_7_IO09
FPGA_7_IO12	87	88	FPGA_7_IO15
FPGA_7_IO18	89	90	FPGA_7_IO14
FPGA_7_IO19	91	92	FPGA_7_IO16
FPGA_7_IO20	93	94	FPGA_7_IO17
FPGA_7_IO21	95	96	[redacted]
GND	97	98	GCLK0S
GND	99	100	GCLK5S

รูปที่ 2.41 Pin Connector J2 (J2:51 - J2:100)

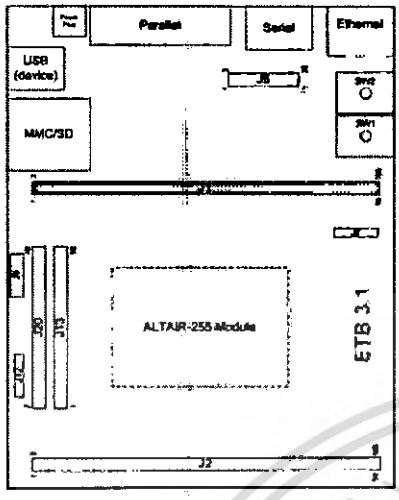
Pin Header J1 (J1:1 – J1:50)



Left			Right
Description	Pin		Description
USB_N (PXA_USB)	1	2	IRTXD (IrDA)
USB_P (PXA_USB)	3	4	IRRXD (IrDA)
GND	5	6	
GND	7	8	
GCLK3P (FPGA)	9	10	DQM3 (memory)
DQM2 (memory)	11	12	DQM1 (memory)
DQM0 (memory)	13	14	GPIO21
MBREQ (memory)	15	16	MBGNT (memory)
SDCLK2 (memory)	17	18	SDCLK1 (memory)
#RESET_OUT	19	20	SDCKE1 (memory)
#M_RESET (memory)	21	22	#SDCAS (memory)
#SDRAS (memory)	23	24	#SDCS2 (memory)
#SDCS1 (memory)	25	26	#CS4 (I/O-memory)
#CS3 (I/O-memory)	27	28	#CS2 (I/O-memory)
#CS1 (I/O-memory)	29	30	RDY
DATA31 (data bus)	31	32	DATA30 (data bus)
DATA29 (data bus)	33	34	DATA28 (data bus)
DATA27 (data bus)	35	36	DATA26 (data bus)
DATA25 (data bus)	37	38	DATA24 (data bus)
DATA23 (data bus)	39	40	DATA22 (data bus)
DATA21 (data bus)	41	42	DATA20 (data bus)
DATA19 (data bus)	43	44	DATA18 (data bus)
DATA17 (data bus)	45	46	DATA16 (data bus)
DATA15 (data bus)	47	48	DATA14 (data bus)
DATA13 (data bus)	49	50	DATA12 (data bus)

รูปที่ 2.42 Pin Connector J1 (J1:1 – J1:50)

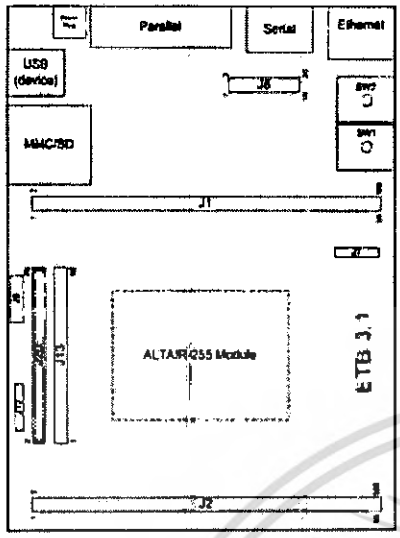
Pin Header J1 (J1:51 – J1:100)



Left			Right
Description	Pin		Description
DATA11 (data bus)	51	52	DATA10 (data bus)
DATA9 (data bus)	53	54	DATA8 (data bus)
DATA7 (data bus)	55	56	DATA6 (data bus)
DATA5 (data bus)	57	58	DATA4 (data bus)
DATA3 (data bus)	59	60	DATA2 (data bus)
DATA1 (data bus)	61	62	DATA0 (data bus)
#WR_BUF (I/O memory)	63	64	#WR_BUF (I/O memory)
#OE_BUF (I/O memory)	65	66	#CS_BUF (I/O memory)
SDCLK0_BUF (memory)	67	68	SDCKE0_BUF (memory)
ADDR25 (address bus)	69	70	ADDR24 (address bus)
ADDR23 (address bus)	71	72	ADDR22 (address bus)
ADDR21 (address bus)	73	74	ADDR20 (address bus)
ADDR19 (address bus)	75	76	ADDR18 (address bus)
ADDR17 (address bus)	77	78	ADDR16 (address bus)
ADDR15 (address bus)	79	80	ADDR14 (address bus)
ADDR13 (address bus)	81	82	ADDR12 (address bus)
ADDR11 (address bus)	83	84	ADDR10 (address bus)
ADDR09 (address bus)	85	86	ADDR08 (address bus)
ADDR07 (address bus)	87	88	ADDR06 (address bus)
ADDR05 (address bus)	89	90	ADDR04 (address bus)
ADDR03 (address bus)	91	92	ADDR02 (address bus)
ADDR01 (address bus)	93	94	ADDR00 (address bus)
	95	96	GND
EHT_RX- (Ethernet)	97	98	EHT_TX- (Ethernet)
EHT_RX+ (Ethernet)	99	100	EHT_TX+ (Ethernet)

รูปที่ 2.43 Pin Connector J1 (J1:51 – J1:100)

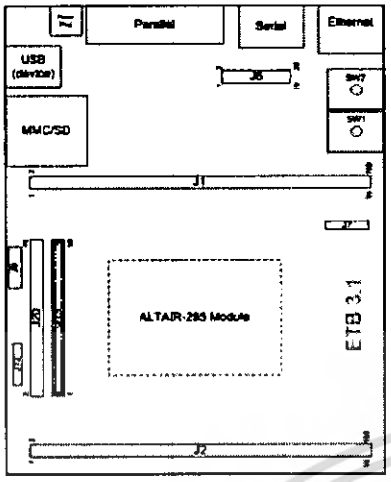
Pin Header J20



Left			Right
Description	Pin		Description
RS232_DCD_BTTXD	1	2	--not assigned--
RS242_DSR_BTRXD	3	4	--not assigned--
RS232_RXD	5	6	--not assigned--
RS232_RTS	7	8	--not assigned--
RS232_TXD	9	10	--not assigned--
RS232_CTS_IRRXD	11	12	--not assigned--
RS232_DTR	13	14	--not assigned--
RS232_RI_IRTXD	15	16	--not assigned--
USB_VBUS (OTG_USB)	17	18	--not assigned--
USB_ID (OTG_USB)	19	20	--not assigned--
#H_OC2 (OTG_USB)	21	22	--not assigned--
#H_PSW1 (OTG_USB)	23	24	--not assigned--
#H_OC1 (OTG_USB)	25	26	--not assigned--
#H_PSW2 (OTG_USB)	27	28	--not assigned--
USB_VDD_5V (OTG_USB)	29	30	--not assigned--
OTG_DP1 (OTG_USB)	31	32	--not assigned--
OTG_DM1 (OTG_USB)	33	34	GND_EXT
H_DP2 (OTG_USB)	35	36	AIN3 (I2C_MAX1237)
H_DM2 (OTG_USB)	37	38	AIN4 (I2C_MAX1237)
USB_#OTGMODE | VMONITOR	39	40	AOUT3 (I2C_MAX5842)
AIN1 (I2C_MAX1237)	41	42	AOUT4 (I2C_MAX5842)
AIN2 (I2C_MAX1237)	43	44	--not assigned--
AOUT1 (I2C_MAX5842)	45	46	--not assigned--
AOUT2 (I2C_MAX5842)	47	48	--not assigned--
AIN_REF (I2C_MAX5842) |VMONITOR	49	50	

รูปที่ 2.44 Pin Connector J20

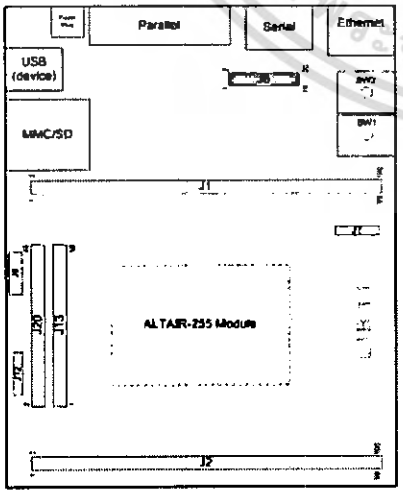
Pin Header J13



Left			Right
Description	Pin		Description
	1	2	
	3	4	
GCLK1P (FPGA)	5	6	MMCMD (MMC)
L_DD15 (LCD)	7	8	MMCCS1 (MMC)
L_DD14 (LCD)	9	10	MMCCLK (MMC)
L_DD12 (LCD)	11	12	L_DD13 (LCD)
L_DD10 (LCD)	13	14	L_DD11 (LCD)
L_DD08 (LCD)	15	16	L_DD09 (LCD)
L_DD06 (LCD)	17	18	L_DD07 (LCD)
L_DD04 (LCD)	19	20	L_DD05 (LCD)
L_DD02 (LCD)	21	22	L_DD03 (LCD)
BTRXD (BTUART)	23	24	L_DD01 (LCD)
MMDAT (MMC)	25	26	L_DD00 (LCD)
BTCTS (MMC)	27	28	L_BIAS (LCD)
L_PCLK (LCD)	29	30	L_LCLK (LCD)
L_FCL (LCD)K	31	32	BTRTS (BTUART)
MMCCS0	33	34	BTTXD (BTUART)
#POE (PCMCIA)	35	36	GPIO3
#PWAIT (PCMCIA)	37	38	#PIOS16 (PCMCIA)
PSKTSEL (PCMCIA)	39	40	#PREG (PCMCIA)
#PCE1 (PCMCIA)	41	42	#PCE2 (PCMCIA)
#PIOW (PCMCIA)	43	44	#PIOR (PCMCIA)
	45	46	#PWE (PCMCIA)
	47	48	
	49	50	

รูปที่ 2.45 Pin Connector J13

Pin Header J6 (JTAG)



Left			Right
Description	Pin		Description
	1	2	*--not assigned--*
#TRST	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
GND	11	12	GND
TDO	13	14	GND
#M_RESET	15	16	GND
--not assigned--	17	18	GND
--not assigned--	19	20	GND

รูปที่ 2.46 Pin Connector J6 (JTAG)

ตารางแสดงการเชื่อมต่อ GPIO บน PXA255 ระหว่าง GPIO pins กับ test board pins และมีบางขาต่อไปยัง PGA\

GPIO	Alternate Function Name	Signal Description	Pin	FPGA
GP1	GP_RST	Active low GP_reset	J2:12	
GP4	-	-	-	PROG_B
GP5	-	-	-	RDWR_B
GP6	MMCCLK	MMC Clock	J13:10	
GP7	48 MHz clock	48 MHz clock output	J2:34	P8
GP8	MMCCS0	MMC Chip Select 0	J13:33	
GP9	MMCCS1	MMC Chip Select 1	J13:8	
GP10	RTCCLK	real time clock (1 Hz)	J2:39	N9
GP11	3.6 MHz	3.6 MHz oscillator out	J2:40	N8
GP12	32 kHz	32 kHz out	J2:41	B8
GP13	MBGNT	memory controller grant	J1:16	
GP14	MBREQ	memory controller alternate bus master request	J1:15	
GP15	nCS_1	Active low chip select 1	J1:29	P1
GP16	PWM0	PWM0 output	J2:19	L3
GP17	PWM1	PWM1 output	J2:20	L4
GP18	RDY	Ext. Bus Ready	J1:30	M3
GP19	DREQ[1]	External DMA Request		H4
GP20	DREQ[0]	External DMA Request	J2:46	H3
GP21	ALTAIR_LED	LED on ALTAIR-255 Core Module	J1:14	R14
GP22	ALTAIR_LED	LED on ALTAIR-255 Core Module	J2:22	D2
GP23	SCLK	SSP clock	J2:23	G1
GP24	SFRM	SSP Frame	J2:24	G2
GP25	TXD	SSP transmit	J2:25	G3
GP26	RXD	SSP receive	J2:26	G4
GP27	EXTCLK	SSP ext_clk	J2:27	
GP28	BITCLK	AC97 bit_clk	J2:28	
	BITCLK	I2S bit_clk		
	BITCLK	I2S bit_clk		
GP29	SDATA_IN0	AC97 Sdata_in0	J2:29	
	SDATA_IN	I2S Sdata_in		

รูปที่ 2.47 GPIO บน PXA255

GPIO	Alternate Function Name	Signal Description	Pin	FPGA
GP30	SDATA_OUT	I2S Sdata_out	J2:31	
	SDATA_OUT	AC97 Sdata_out		
GP31	SYNC	I2S sync	J2:32	
	SYNC	AC97 sync		
GP32	SDATA_IN1,SYSCLK	AC97 Sdata_in1	J2:30	
	SYSCLK	I2S System Clock		
GP33	nCS[5]	Active low chip select 5		C7
GP34	FFRXD	FFUART receive	J2:5	C10
GP35	CTS	FFUAT Clear to send	J2:7	E10
GP36	DCD	FFUART Data carrier detect	J2:8	A11
GP37	DSR	FFUART data set ready	J2:9	B11
GP38	RI	FFUART Ring Indicator	J2:10	C11
GP39	FFTXD	FFUART transmit data	J2:6	D10
GP40	DTR	FFUART data terminal ready	J2:11	D11
GP41	RTS	FFUART request to send	J2:12	E11
GP42	BTRXD	BTUART receive data	J13:23	T14
	HWRXD	HWUART receive data		
GP43	BTTXD	BTUART transmit data	J13:34	T13
	HWTXD	HWUART transmit data		
GP44	BTCTS	BTUART clear to send	J13:27	
	HWCTS	HWUART clear to send		
GP45	BTRTS	BTUART request to send	J13:32	
	HWRTS	HWUART request to send		
GP46	ICP_RXD	ICP receive data	J1:4	J3
	RXD	STD_UART receive data		
GP47	TXD	STD_UART transmit data	J1:2	J4
	ICP_TXD	ICP transmit data		
GP48	nPOE	Output Enable for Card Space	J13:35	N2
	HWTXD	HWUART transmit data		
GP49	nPWE	Write Enable for Card Space	J13:46	M4
	HWRXD	HWUART receive data		
GP50	nPIOR	I/O Read for Card Space	J13:44	N6
	HWCTS	HWUART clear to send		
GP51	nPIOW	I/O Write for Card Space	J13:43	P6
	HWRTS	HWUART request to send		
GP52	nPCE[1]	Card Enable for Card Space	J13:41	
GP53	nPCE[2]	Card Enable for Card Space	J13:42	
GP54	nPSKTSEL	Socket Select for Card Space	J13:39	
GP55	nPREG	Card Address bit 26	J13:40	
GP56	nPWAIT	Wait signal for Card Space	J13:37	
GP57	nIOIS16	Bus width select for I/O Card Space	J13:38	
GP58	LDD[0]	LCD data pin 0	J13:26	
GP59	LDD[1]	LCD data pin 1	J13:24	

รูปที่ 2.47 GPIO บน PXA255 (ต่อ)

GPIO	Alternate Function Name	Signal Description	Pin	FPGA
GP60	LDD[2]	LCD data pin 2	J13:21	
GP61	LDD[3]	LCD data pin 3	J13:22	
GP62	LDD[4]	LCD data pin 4	J13:19	
GP63	LDD[5]	LCD data pin 5	J13:20	
GP64	LDD[6]	LCD data pin 6	J13:17	
GP65	LDD[7]	LCD data pin 7	J13:18	
GP66	LDD[8]	LCD data pin 8	J13:15	
	MBRREQ	memory controller alternate bus master req		
GP67	LDD[9]	LCD data pin 9	J13:16	
	MMCCS0	MMC Chip Select 0		
GP68	LDD[10]	LCD data pin 10	J13:13	
	MMCCS1	MMC Chip Select 1		
GP69	LDD[11]	LCD data pin 11	J13:14	
	MMCCLK	MMC_CLK		
GP70	LDD[12]	LCD data pin 12	J13:11	
	RTCCLK	Real Time clock (1Hz)		
GP71	LDD[13]	LCD data pin 13	J13:12	
	3.6 MHz	3.6 MHz oscillator clock		
GP72	LDD[14]	LCD data pin 14	J13:9	
	32 kHz	32 kHz clock		
GP73	LDD[15]	LCD data pin 14	J13:7	
	MBGNT	Memory controller grant		
GP74	LCD_FCLK	LCD Frame clock	J13:31	
GP75	LCD_LCLK	LCD line clock	J13:30	
GP76	LCD_PCLK	LCD Pixel clock	J13:29	
GP77	LCD_ACBIAS	LCD AC Bias	J13:28	
GP78	nCS[2]	Active low chip select 2	J1:28	T5
GP79	nCS[3]	Active low chip select 3	J1:27	R5
GP80	nCS[4]	Active low chip select 4	J1:26	N1
GP81	NSSPSCLK	NSSP Serial clock is input	J2:4	
	NSSPSCLK	NSSP Serial clock is output		
GP82	NSSPSFRM	NSSP frame is input	J2:13	
	NSSPSFRM	NSSP frame is output		
GP83	NSSPTXD	NSSP transmit	J2:14	
	NSSPRXD	NSSP receive		
GP84	NSSPTXD	NSSP transmit	J2:3	
	NSSPRXD	NSSP receive		

รูปที่ 2.47 GPIO บน PXA255 (ต่อ)

Universal Asynchronous Receiver - Transmitter (UART)

อุปกรณ์ที่ต่ออนุกรม (Serial devices) ทุกชนิดใช้อินเตอร์เฟสชิป UARTในการติดต่อสื่อสารกับเครื่องคอมพิวเตอร์ ชิป UARTเปลี่ยนข้อมูลจากแบบขนาน ให้เป็นแบบอนุกรม หรือจากแบบอนุกรม ให้เป็นแบบขนาน PXA255 มี UARTs อยู่ 4 อย่างคือ

1. Full Function UART (FFUART)

Signal Description	Pin
FFUART receive	J2:5
FFUAT Clear to send	J2:7
FFUART Data carrier detect	J2:8
FFUART data set ready	J2:9
FFUART Ring Indicator	J2:10
FFUART transmit data	J2:6
FFUART data terminal ready	J2:11
FFUART request to send	J2:12

รูปที่ 2.48 Full Function UART (FFUART)

2. Bluetooth UART (BTUART)

Signal Description	Pin
BTUART receive data	J13:23
BTUART transmit data	J13:34
BTUART clear to send	J13:27
BTUART request to send	J13:32

รูปที่ 2.49 Bluetooth UART (BTUART)

3. Standard UART (STUART)

Signal Description	Pin
STUART receive data	J1:4
STUART transmit data	J1:2

รูปที่ 2.50 Standard UART (STUART)

4. Hardware UART (HWUART)

Signal Description	Pin
HWUART receive data	J13:23
HWUART transmit data	J13:34
HWUART clear to send	J13:27
HWUART request to send	J13:32
HWUART transmit data	J13:35
HWUART receive data	J13:46
HWUART clear to send	J13:44
HWUART request to send	J13:43

รูปที่ 2.51 Hardware UART (HWUART)

FPGA แสดงขาFPGA กับ pin

Signal Description	Pin	FPGA
FPGA_3_IO_0	J2:68	J15
FGPA_3_IO_1	J2:70	J16
FGPA_3_IO_2	J2:69	J13
FGPA_3_IO_3	J2:71	J14
FGPA_3_IO_4	J2:64	M13
FGPA_3_IO_5	J2:66	M14
FGPA_3_IO_6	J2:65	N14
FGPA_3_IO_7	J2:67	N15
FGPA_3_IO_8	J2:60	P16
FGPA_3_IO_9	J2:62	N16
FGPA_3_IO_10	J2:63	K13
FGPA_3_IO_11	J2:61	K14
FGPA_3_IO_12	J2:59	K15
FGPA_3_IO_13	J2:57	K16
FGPA_3_IO_14	J2:58	M15
FGPA_3_IO_15	J2:56	M16
FGPA_3_IO_16	J2:54	L13
FGPA_3_IO_17	J2:52	L14
FGPA_3_IO_18	J2:55	K12
FGPA_3_IO_19	J2:53	L12
FGPA_3_IO_20	J2:51	L15
FPGA_3_IO_21	J2:49	L16
FPGA_7_IO_0	J2:76	H16
FPGA_7_IO_1	J2:78	H15
FPGA_7_IO_2	J2:73	H14
FPGA_7_IO_3	J2:75	H13
FPGA_7_IO_4	J2:80	E14
FPGA_7_IO_5	J2:82	E13
FPGA_7_IO_6	J2:77	D15
FPGA_7_IO_7	J2:79	D14
FPGA_7_IO_8	J2:84	D16
FPGA_7_IO_9	J2:86	C16
FPGA_7_IO_10	J2:83	G14
FPGA_7_IO_11	J2:81	G13
FPGA_7_IO_12	J2:87	G16
FPGA_7_IO_13	J2:85	G15
FPGA_7_IO_14	J2:90	F14
FPGA_7_IO_15	J2:88	F13
FPGA_7_IO_16	J2:92	E16
FPGA_7_IO_17	J2:94	E15
FPGA_7_IO_18	J2:89	G12
FPGA_7_IO_19	J2:91	F12
FPGA_7_IO_20	J2:93	F16
FPGA_7_IO_21	J2:95	F15

รูปที่ 2.52 FPGA

FPGA I/O Memory Map

Data Bus

Signal Description	Pin	FPGA
D00	J1:62	P13
D01	J1:61	R13
D02	J1:60	N12
D03	J1:59	P12
D04	J1:58	P5
D05	J1:57	N5
D06	J1:56	R4
D07	J1:55	P4
D08	J1:54	R8
D09	J1:53	R9
D10	J1:52	P9
D11	J1:51	T8
D12	J1:50	T7
D13	J1:49	T10
D14	J1:48	R7
D15	J1:47	R10
D16	J1:46	M7
D17	J1:45	M6
D18	J1:44	N7
D19	J1:43	P7
D20	J1:42	T6
D21	J1:41	R6
D22	J1:40	M10
D23	J1:39	N10
D24	J1:38	P10
D25	J1:37	T11
D26	J1:36	R11
D27	J1:35	P11
D28	J1:34	M11
D29	J1:33	R12
D30	J1:32	N11
D31	J1:31	T12

รูปที่ 2.53 FPGA I/O Memory Map: Data Bus

Address Bus

Signal Description	Pin	FPGA
A00	J1:94	B4
A01	J1:93	C4
A02	J1:92	A5
A03	J1:91	D5
A04	J1:90	A7
A05	J1:89	C13
A06	J1:88	B10
A07	J1:87	B7
A08	J1:86	A12
A09	J1:85	B5
A10	J1:84	A10
A11	J1:83	D12
A12	J1:82	B13
A13	J1:81	B12
A14	J1:80	C5
A15	J1:79	C12
A16	J1:78	E1
A17	J1:77	F4
A18	J1:76	E2
A19	J1:75	F3
A20	J1:74	C6
A21	J1:73	D6
A22	J1:72	E7
A23	J1:71	B6
A24	J1:70	E6
A25	J1:69	A6

รูปที่ 2.54 FPGA I/O Memory Map: Address Bus

2.9 วิธีตรวจวัดคุณภาพของภาพ

ในส่วนของการทดลองเพื่อวัดประสิทธิภาพของวิธีการที่ได้นำเสนอ เราได้ใช้ค่า Peak Signal to Noise Ratio (PSNR) ซึ่งเป็นค่ามาตรฐานที่บ่งบอกถึงคุณภาพที่เปลี่ยนระหว่างรูปภาพสองภาพมาใช้ในการเปรียบเทียบที่สภาวะต่างๆกัน ซึ่งสมการที่ใช้ในการคำนวณหาค่า PSNR มีดังนี้

$$PSNR = 20\log_{10}\left(\frac{b}{RMSE}\right) \tag{2.35}$$

เมื่อตัวแปร b คือค่าสูงสุดที่เป็นไปได้ของพิกเซลในภาพ ในที่นี้ก็คือค่าสูงสุดที่ข้อมูลขนาด 8 บิตสามารถแสดงได้ นั่นคือ 255 (คิดในกรณีของรูปภาพแบบ grayscale) และถ้าเป็นแบบ 24 บิตสามารถแสดงได้ 16,777,216 (คิดในกรณีของรูปภาพแบบ RGB)

สำหรับค่า Root Mean Square Error (RMSE) และค่า Mean Square Error (MSE) สามารถคำนวณได้จากสมการดังต่อไปนี้

$$RMSE = \sqrt{MSE} \tag{2.36}$$

$$MSE = \frac{1}{N} \times \left(SUM_{ij} \left| Org_{ij} - Wmk_{ij} \right|^2 \right) \tag{2.37}$$

เมื่อ Org_{ij} คือค่าพิกเซลของภาพต้นฉบับ Wmk_{ij} คือค่าพิกเซลของภาพที่เราจะนำมาเปรียบเทียบ และ N คือจำนวนพิกเซลทั้งหมดภายในภาพที่ได้มาจากผลคูณระหว่างขนาดพิกเซลทางกว้างและขนาดพิกเซลทางยาว

PSNR นี้จะมีค่าสูงเมื่อผลของการเปรียบเทียบระหว่างภาพทั้งสองมีความใกล้เคียงกันมาก ในทางกลับกัน หาก PSNR ที่ได้มีค่าต่ำหรือเข้าใกล้ศูนย์ แสดงว่าภาพที่นำมาเปรียบเทียบกันมีความแตกต่างกันมากนั่นเอง รูปภาพที่นำไปใช้จริงในทางปฏิบัติควรอยู่ระหว่าง 20 – 40 dB ค่า PSNR จะสะท้อนถึงภาพที่ได้จากการบีบอัดว่ามีความผิดเพี้ยนจากภาพต้นแบบมากน้อยเพียงใด นั้นคือถ้าค่า PSNR มีค่ามากแสดงว่าภาพที่ได้จากการบีบอัดมีความผิดเพี้ยนน้อย และถ้าค่า PSNR มีค่าน้อยแสดงว่าภาพที่ได้จากการบีบอัดมีความผิดเพี้ยนมาก และอัตราบิตเรท (Bit rate) คือค่าเฉลี่ยของจำนวนบิตต่อพิกเซล (Bits per pixel, bpp) ของภาพที่ผ่านการบีบอัดข้อมูล

$$\text{Bit rate (bpp)} = \frac{\text{bit compress}}{\text{pixel image}} \tag{2.38}$$

เมื่อ bit compress คือจำนวนบิตทั้งหมดของภาพที่ผ่านการบีบอัดข้อมูล

pixel image คือจำนวนพิกเซลทั้งหมดของภาพต้นแบบ

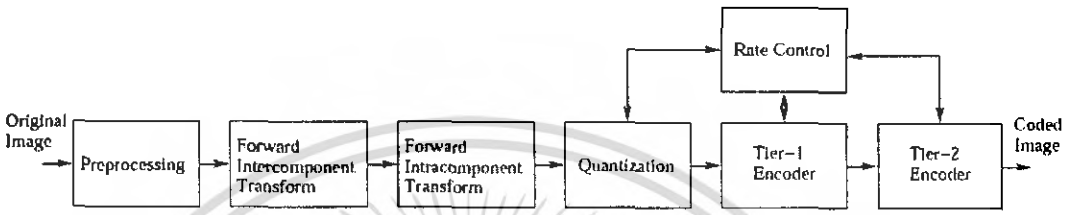
โดยอัตราบิต จะมีความสัมพันธ์กับขนาดของไฟล์ภาพที่ผ่านการบีบอัดข้อมูล ดังตารางที่ 2.17 และทำการทดสอบที่อัตราบิตเดียวกันคือ 0.5, 1, 1.5 และ 2 bpp

ตารางที่ 2.17 ความสัมพันธ์ระหว่างอัตราบิตและขนาดของไฟล์ภาพที่ผ่านการบีบอัดข้อมูล

อัตราบิต (bpp)	ขนาดของไฟล์ภาพที่ผ่านการบีบข้อมูล
0.25	1/32 ของไฟล์ภาพต้นแบบ
0.5	1/16 ของไฟล์ภาพต้นแบบ
1	1/8 ของไฟล์ภาพต้นแบบ
2	1/4 ของไฟล์ภาพต้นแบบ

บทที่ 3
การออกแบบและพัฒนา

โครงการนี้ได้ทำการศึกษาทฤษฎีของการเข้ารหัสภาพ JPEG และ JPEG2000 block diagram ต่างๆ และนำฐานความรู้ที่ได้ไปพัฒนา Code โปรแกรมซึ่งทำงานร่วมกับบอร์ตทดลองแบบฝังตัวโดยจะเจาะจงใช้บอร์ดชนิด X-Scale



รูปที่ 3.1 Standard JPEG 2000 compress

การออกแบบและพัฒนาโครงการประกอบไปด้วย 9 ขั้นตอน ดังนี้

3.1 การเตรียมพร้อม Compiler

3.2 การต่อไฟล์และแก้ไขไฟล์ให้ค้นหา Library ที่ไฟล์ Code แต่ละไฟล์ต้องการ

3.3 การค้นหาฟังก์ชั่น(Function) ที่ส่วนต่างๆได้เรียกใช้งาน

3.4 การเขียนฟังก์ชั่น(Function) เพิ่มในส่วนที่ Compiler ไม่มีให้

3.5 การ Compile ทีไฟล์ในแต่ละส่วนของการเตรียม Library Jasper

3.6 การพัฒนาส่วนการ Compile ไฟล์ให้สะดวกขึ้น

3.7 Executable File

3.8 การส่ง-รับไฟล์และภาพที่ต้องการไปสู่บอร์ด X-Scale

3.9 การเพิ่มฟังก์ชั่นจับเวลาให้ทั้งกับ Code บน PC และบน X-Scale

หลังจากได้ทดสอบเขียน Source Code เองขึ้นมาบางส่วนเช่นส่วนของการถอดข้อมูลภาพ FORMAT BMP และการทำเวฟเลต (Wavelet) Daubechies 4 แบบ C++ บน .net ซึ่งได้ทดสอบแล้วเข้าใจถึงส่วนต่างๆของไฟล์ภาพแบบ BMP และการทำเวฟเลตไปพอสมควรแต่เมื่อนำ Source Code ไปใช้งานจริงบน Board X-Scale พบปัญหาหลายอย่างเกี่ยวกับ Library ที่ใช้ของ C++ ดังนั้นโค้ดที่ได้พัฒนาได้ขั้นแรกนั้นจึงทำงานได้แต่เพียงบน PC และทำได้เฉพาะส่วนของการทำเวฟเลตได้เท่านั้น

หลังจากที่ได้ทำการสืบค้นข้อมูลจากงานวิจัยที่มีคนพัฒนาซอฟแวร์เกี่ยวกับการทำ JPEG2000 ได้พบกับ Free Source Code ของ *Adams* [2003] ชื่อ Software Jasper Version 1.700.2 เป็น Code ในภาษา C เน้นใช้งานกับวินโดวส์โดย Microsoft Visual C++ 6.0 การพัฒนาได้เริ่มขึ้นเมื่อนำ Source มาใช้งานตาม Manual Compile สามารถใช้งานบน ระบบปฏิบัติการวินโดวส์ได้เป็นอย่างดีโดยรับอินพุทได้ทั้ง bmp, jpc, jp2 และสามารถนำไปแปลงเป็นอีกรูปแบบ(Format) ที่เราต้องการได้โดยการป้อนคำสั่ง (key command) ที่

ต้องการได้เลย ยกตัวอย่างเช่น “jasper --input lena.bmp --output lena.jp2 --output-format jp2” และยังได้มีซอฟแวร์อีกตัวที่ใช้ในการทดสอบภาพที่ได้ทำการเข้ารหัสไปโดยเป็น Source ที่ใช้งานร่วมกับ jasper ได้ในชื่อของ imgcmp ใช้ในการวัดคุณภาพของภาพสองภาพว่ามีความผิดพลาด (Error) มากน้อยเพียงใดโดยได้ออกมาในรูปของ PSNR เป็นต้นโดยการ Compile เช่นเดียวกับ jasper และมีรูปแบบคำสั่ง(command) ดังนี้ “imgcmp -f original.pgm -F reconstructed.pgm -m psnr”

การพัฒนาขั้นแรกได้เริ่มขึ้นโดยจาก Source Cde เดิมที่มีส่วนที่ไม่จำเป็นหลายส่วน ได้ทำการตัดส่วนที่ไม่ต้องการออก เช่น ส่วนที่เป็นของ Microsoft visual c++ 6.0 และส่วนของการเข้าและถอดรหัสภาพอีกหลายๆ รูปแบบ ส่วนของ jpeg, pnm, pgx, ras, mif เพื่อทำให้ Source Code มีขนาดเล็กลงเพื่อให้ง่ายต่อการแปลงให้ใช้งานได้กับ X-Scale ที่มี Library อยู่จำกัด

การพัฒนาขั้นต่อมาคือการถอด Code ส่วนที่ใช้งานมาเพื่อให้รองรับต่อ Compiler ของ X-Scale โดยจะอธิบายเป็นลำดับขั้นตอนดังนี้

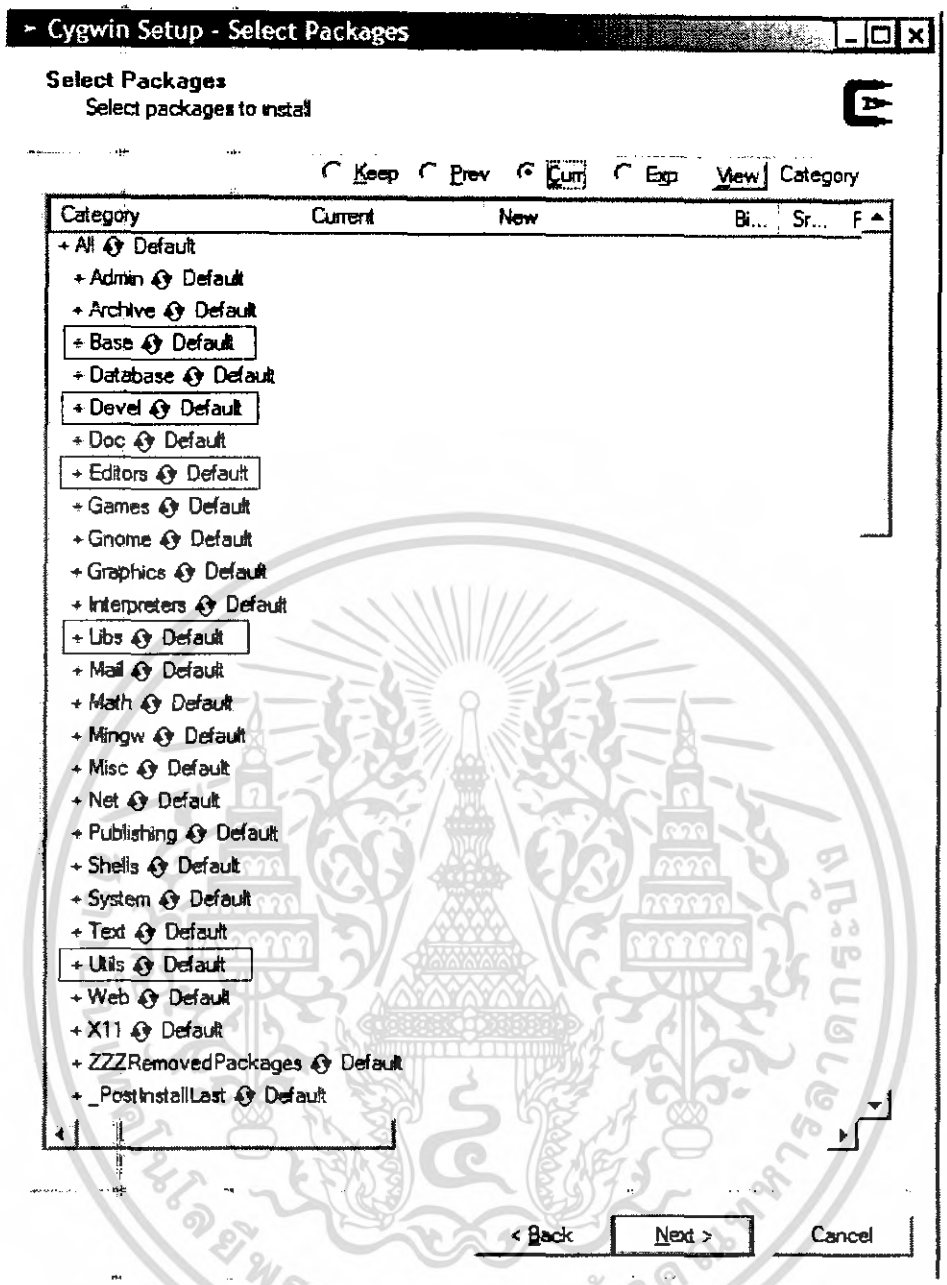
3.1 การเตรียมพร้อม Compiler (Cross-Compiler)

3.1.1 การติดตั้ง Cygwin

การติดตั้ง Cygwin สำหรับการพัฒนาแอพลิเคชั่นบนบอร์ด AltairCygwin เป็นโปรแกรมที่จำลองโครงสร้างของระบบปฏิบัติการ Linux/Unix โดยมีลักษณะตั้งแต่โครงสร้าง Directory ไปจนถึงระบบโปรแกรมที่ทำงานภายใน Cygwin ได้ โดยลักษณะดังกล่าวทำให้ Cygwin สามารถที่จะเลียนแบบการทำงานของ Linux/Unix ได้ในรูปแบบของ UserApplication และสามารถใช้งานโปรแกรมของ GNU หรือ Opensource ต่างๆ ได้ ซึ่งการทำงานทั้งหมดนี้ จะอยู่ใต้ระบบปฏิบัติการวินโดวส์อีกทีหนึ่ง ดังนั้น นักพัฒนาจึงมักใช้สำหรับการพัฒนาโปรแกรมบน Linux/Unix ได้ โดยไม่ต้องลง Linux ตัวเต็มๆ ให้เปลืองคอมพิวเตอร์ (PC) อีกเครื่องหนึ่ง

ในการพัฒนาแอพลิชั่นสำหรับบอร์ด Altair นั้น จำเป็นต้องมีการสร้าง Development Environment ซึ่งประกอบไปด้วย Compiler, Utility ต่างๆ โดยสำหรับ Compiler นั้น ได้มีนักพัฒนาผู้อื่น ได้ทำการสร้าง Cross-Compiler ซึ่งเป็น Compiler สำหรับการพัฒนางานบนบอร์ดที่มีการใช้งาน CPU 32-bit Intel PXA Processor และซอฟแวร์ ดังนั้นจึงเป็นการสะดวกมากขึ้น ถ้ามีการนำ Cygwin มาใช้เป็น Environment ในการพัฒนา ซึ่งจะทำให้สามารถทำการพัฒนาแอพลิเคชั่นบน PXA ได้เหมือนกับการพัฒนาบน Linux PC

ในการติดตั้ง Cygwin นั้นควรเตรียมที่ว่างประมาณ 1.2 GB ขึ้นไปเพื่อที่จะได้ติดตั้ง Library และเครื่องมือ (Tool) ต่างๆ ได้ครบสำหรับการพัฒนาโดยเลือกติดตั้งแพคเกจ (packet) ตามนี้



รูปที่ 3.2 Cygwin Setup – Select Packages

3.1.2 การติดตั้ง Compiler Tool-chain

การติดตั้ง Utilitiy ที่จำเป็น

ในการคอมไพล์เคอร์เนลของลีนุกซ์บน Cygwin นั้น เราต้องติดตั้ง libelf เป็นไลบรารี่สำหรับการจัดรูปแบบของการทำ Executable File (ซึ่งเป็นรูปแบบที่ใช้สำหรับลีนุกซ์เคอร์เนล) ซึ่งไม่มีให้ในซีดีรอมของ Cygwin ดังนั้น เราจึงต้องหาซอร์สโค้ดจากอินเตอร์เน็ตมาคอมไพล์และติดตั้งเอง (ซึ่งได้โหลดและจัดเก็บอยู่ในซีดีรอมของ Altair แล้ว) โดยขั้นตอนการติดตั้งมีดังนี้

1. เข้าไปใน Directory ของซีดีรอม ซึ่งอยู่ที่ /cygdrive/<ซีดีรอมไดร์ฟ> (สมมติไดร์ฟซีดีรอมเป็นไดร์ฟ E:) โดยใช้คำสั่ง change directory

```
$ cd /cygdrive/e
```

2. ทำการคัดลอกซอร์สโค้ดของ libelf ซึ่งอยู่ที่ /cygdrive/e/tools/cygwin_libelf/libelf-0.8.6.tar.gz ไปไว้ในฮาร์ดไดรฟ์ของเรา โดยใช้คำสั่ง "cp" โดยคัดลอกไปไว้ที่ /tmp ชั่วคราว
```
$ cp /cygdrive/e/tools/cygwin_libelf/libelf-0.8.6.tar.gz /tmp/
```
3. ย้าย Directory เข้าไปที่ /tmp โดยคำสั่ง
```
$ cd /tmp
```
4. ทำการ Uncompress ไฟล์ที่ถูกบีบอัดด้วยคำสั่ง
```
$ tar zxf libelf-0.8.6.tar.gz
```
5. ย้าย Directory เข้าไปในโฟล์เดอร์ที่ถูกสร้างขึ้นใหม่จากไฟล์บีบอัด แล้วทำการ Compile และติดตั้งตามลำดับ
```
$ cd libelf-0.8.6
$ ./configure
$ make
$ make install
```
จะได้โฟลเดอร์ arm-3.4.1

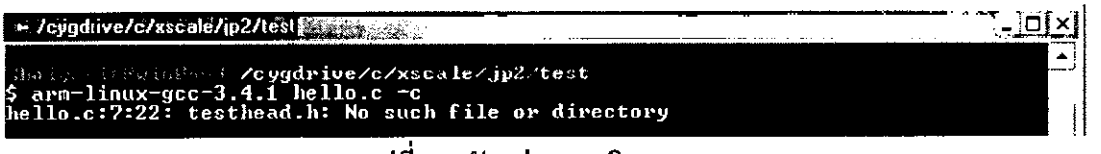
6. ทำการเพิ่ม Path ของ arm-3.4.1 เข้าไปให้กับตัวแปร Environment ในการเปิดโปรแกรม Cygwin ทุกครั้ง เพื่อสามารถใช้คำสั่งในการทำ Cross-compiler เช่น arm-linux-gcc ได้โดยอัตโนมัติ โดยให้ทำการเพิ่มบรรทัดในการเพิ่ม path เข้าไปต่อท้ายไฟล์ /etc/profile โดยใช้คำสั่งต่อไปนี้
```
$ echo 'export PATH=$PATH:/usr/local/arm-3.4.1/bin' >> /etc/profile
```
ออกจากโปรแกรม Cygwin แล้วเปิดใหม่อีกครั้ง

7. ทดสอบการใช้คำสั่ง โดยใช้คำสั่ง
```
$ arm-linux-gcc -version
```
จะได้ผลลัพธ์ดังนี้ แสดงว่าการติดตั้ง Tool-chain เสร็จสิ้น
```
arm-linux-gcc (GCC) 3.4.1
Copyright (C) 2004 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There
is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```
จนถึงขั้นตอนนี้ เราก็สามารถคอมไพล์โปรแกรมที่สามารถประมวลผลบนบอร์ด Altair ได้แล้ว

3.2 การต่อไฟล์และแก้ไขไฟล์ให้ค้นหา Library ที่ไฟล์ Code แต่ละไฟล์ต้องการ

3.2.1 Error ที่พบบ่อยๆ และวิธีแก้

เนื่องจาก Source Code ได้แยกกันไว้เป็นหลายๆ ตัวด้วยกัน โดยมีการจัดหมู่แยกกันแต่ละโฟลเดอร์ในการ Compile โดย Microsoft Visual C++ 6.0 นั้นได้มีการจัดลำดับและตำแหน่งไว้ซึ่งจะลิงค์ไฟล์ตามโฟลเดอร์ย่อย (Subfolder) ต่างๆไว้แล้วจึงทำให้ Compile แล้วไม่เกิดปัญหาและเมื่อนำมาใช้บน Compile arm-linux การลิงค์ไฟล์นั้นทำได้ยากจึงต้องมีการเรียก Path โดยตรงจากไฟล์ต้นที่ต้องการไฟล์อื่นๆเองดังนั้น Error ที่พบแรกๆเลยคือการไม่พบไฟล์ include ที่ต้องการยกตัวอย่างเช่น



รูปที่ 3.3 ตัวอย่างการเกิด Error

การแก้ปัญหานั้นทำได้โดยเพียงไปที่ไฟล์ที่ทำการ Compile แล้วแก้ไข Path ไฟล์ที่ต้องการให้พบ แต่ก่อนอื่นต้องรู้ว่าไฟล์ที่ต้องการนั้นอยู่ที่ใดเสียก่อนแล้วจึงทำการแก้ไข

3.2.2 Error ของการค้นหาไฟล์ไม่พบและวิธีแก้ไข

Error ลักษณะนี้มักจะพบเมื่อ Library ของ Compiler ที่ใช้ไม่มีให้โดยที่พบมักเกี่ยวกับฟังก์ชั่น Math, Float โดยการแก้ไขนี้ทำได้โดยการค้นหา Library ที่ต้องการนี้จาก Compiler ของภาษา C อื่นมาใช้ ซึ่งในการพัฒนานี้ได้นำมาจาก Code compiler studio ของ DSP (Digital Signal Process) มาใช้ร่วมเพราะถ้าใช้ของ C ทั่วไปมักเจาะจงใช้กับ win32 เป็นส่วนใหญ่จึงนำมาประยุกต์ใช้กับของเราไม่ได้หลักจากที่ได้พบก็ได้เพิ่ม Library นี้ไปให้กับ Code ที่เรียกใช้ฟังก์ชั่นนั้นๆ

3.3 การค้นหาฟังก์ชั่น (Function) ที่ส่วนต่างๆ ได้เรียกใช้งาน

เมื่อทำการ Compile ไฟล์ซึ่งมักเกิดปัญหาที่ไม่พบฟังก์ชั่นแล้วเกิดเป็น Error ขึ้นจะมีหลักการสังเกตง่ายๆคือถ้า ฟังก์ชั่นที่เรียกหานั้นขึ้นต้นด้วย jas_xxx แสดงว่าเป็นฟังก์ชั่นที่อยู่ใน Source ต้นฉบับแต่ถ้าฟังก์ชั่นที่เรียกหาไม่ขึ้นต้นแบบนี้แสดงว่าต้องการจากภายนอกคืออาจจะเป็นใน Library ของ Compiler เอง และหากเกิดกรณีนี้จะต้องทำการเขียนขึ้นมาเองหรือจะไปดูจาก Standard ของ VC6 แล้วเขียนขึ้นใหม่ให้รองรับ Compiler ของเราในที่นี้ได้กล่าวไว้แล้วว่าได้นำมาจากส่วนของ Library ใน DSP Compiler ซึ่งเป็นฟังก์ชั่นที่ทำงานหน้าที่แบบเดียวกัน

3.4 การเขียนฟังก์ชั่น (Function) เพิ่มในส่วนที่ Compiler ไม่มีให้

ที่ Libary ของ Xscaleไม่มีให้ ตัวอย่างบางส่วน เช่น ฟังก์ชั่นยกกำลัง pow(); ,floor(); ,cei();

```
double pow(double x,double gamma)
{
      double sum = x ;
      double c=1;
      for(c;c==gamma;c++)
      {
            sum=sum*x;
      }
      return sum;
/********************/

double floor(double x)
{
   double y;
   return (modf(x, &y) < 0 ? y - 1 : y);
}
/********************/
double ceil(double x)
{
   double y;
   return (modf(x, &y) > 0 ? y + 1 : y);
}
```

3.5 การ Compile ทีละไฟล์ในแต่ละส่วนของการเตรียม Library Jasper

3.5.1 ไฟล์ที่ต้องทำการเตรียมเริ่มแรก

จาก Source Code ต้นฉบับนั้นได้จัดหมวดหมู่ไว้เป็นโฟลเดอร์ (Folder) หลังจากที่ได้ตัดส่วนที่ไม่จำเป็นออกไปแล้วโดยส่วนนี้จะเป็นเพียง Library สำหรับโปรแกรมเพื่อให้โปรแกรมหลักเรียกใช้งานฟังก์ชั่นได้สะดวกซึ่งในส่วนนี้จะไม่มีฟังก์ชั่นหลักโดยแต่ไฟล์มีหน้าที่แจกแจงได้ดังนี้

- **BASE** เป็นที่รวมของ Source Code ที่ทำหน้าที่พื้นฐานต่างๆ โดยมีจุดสังเกตว่าจะขึ้นต้นไฟล์ว่า JAS_xxx แสดงดังนี้

```
jas_cm.c jas_debug.c jas_getopt.c jas_icc.c jas_iccdata.c jas_image.c
jas_init.c jas_malloc.c jas_seq.c  jas_stream.c  jas_string.c  jas_tvp.c
jas_version.c
```

- **INCLUDE** เป็นไฟล์ที่มีหน้าที่เกี่ยวข้องกับฟังก์ชันไฟล์ใน Baseโฟลเดอร์ และจะเป็นส่วนของการติดต่อกับไฟล์อื่นๆที่เรียกใช้ฟังก์ชั่นจากไฟล์ใน Base โฟลเดอร์

```
jas_cm.h  jas_config.h  jas_config2.h  jas_debug.h  jas_fix.h
jas_getopt.h  jas_icc.h jas_image.h  jas_init.h  jas_malloc.h  jas_math.h
jas_seq.h  jas_stream.h  jas_string.h jas_tvp.h  jas_types.h jas_version.h
jasper.h onlyjp2.h
```

ไฟล์ onlyjp2.h นี้สำคัญเป็นไฟล์ที่เราได้เพิ่มขึ้นเพื่อไม่ให้โปรแกรมเรียกฟังก์ชันที่นอกเหนือคือไม่เรียกส่วนที่เราตัดไปเช่น jpg, ras เพื่อให้ไฟล์อื่นๆรู้ว่าไม่มีสิ่งนี้

- **BMP** รวมโค้ดในส่วนที่เกี่ยวกับ bmp การถอดและเข้ารหัส bmp จริงแล้วเป็นการถอด Header และใส่ Header ของ bmp รวมถึงส่วนของการนำข้อมูล raw เข้าสู่หน่วยความจำด้วย

```
bmp_cod.c bmp_dec.c bmp_enc.c bmp_cod.h bmp_enc.h
```

- **JPC** รวมโค้ดในส่วนของการเข้าและถอดรหัส jpeg2000 ตั้งแต่เริ่มจนจบโดยข้อมูลที่รับต้องเป็น RAW แล้วและได้ผลลัพธ์เป็นไฟล์ JPC (jpeg code stream)

```
jpc_bs.c jpc_cs.c jpc_dec.c jpc_enc.c jpc_math.c jpc_mct.c jpc_mqcod.c
jpc_mqdec.c jpc_mqenc.c jpc_qmfb.c jpc_t1cod.c jpc_t1dec.c jpc_t1enc.c
jpc_t2cod.c jpc_t2dec.c jpc_t2enc.c jpc_tagtree.c jpc_tsfb.c jpc_util.c
jpc_bs.h jpc_cod.h jpc_cs.h jpc_dec.h jpc_enc.h jpc_fix.h jpc_flt.h
jpc_math.h jpc_mct.h jpc_mqcod.h jpc_mqdec.h jpc_mqenc.h jpc_qmfb.h
jpc_t1cod.h jpc_t1dec.h jpc_t1enc.h jpc_t2cod.h jpc_t2dec.h jpc_t2enc.h
jpc_tagtree.h jpc_tsfb.h jpc_util.h
```

- **JP2** เป็นการเข้าและถอดรหัส JPEG2000 เช่นกันแต่จะอาศัยฟังก์ชันของ jpc มาช่วยโดยส่วนใหญ่โดยจะมีความต่างที่ Format Header

```
jp2_cod.c jp2_dec.c jp2_enc.c jp2_cod.h jp2_dec.h
```

3.5.2 การเลือก Compile ไฟล์ และไฟล์หลัก (Main File)

การคอมไพล์จะเลือกคอมไพล์ใน Base ก่อนโดยจะคอมไพล์ทีละไฟล์ที่เป็นไฟล์ .c คำสั่งที่ใช้ดังนี้ "arm-linux-gcc-3.4.1 codefile.c –c" ถ้าสามารถคอมไพล์ได้ผ่านจะได้ไฟล์ที่เป็นไฟล์ obj เป็น .o ดังตัวอย่างจะได้ codefile.o ซึ่งเป็นไฟล์ที่ยังไม่มีการลิ้งโดยจะต้องทำการคอมไพล์ให้เป็น obj จนครบทุกไฟล์ ".c" รวมถึงไฟล์หลักที่มี Main ซึ่งเป็นไฟล์ชื่อ jasper.c ในไฟล์นี้จะเป็นไฟล์ที่ดำเนินการตั้งแต่การรับ Key Command แล้วจัดแจงถอดความแล้วเรียกฟังก์ชันที่เหมาะสมจาก Library ที่ได้คอมไพล์ไว้เป็น obj ทั้งหมด โดยไฟล์ obj ทีเป็น Library คือ

```
jas_debug.o jas_getopt.o  jas_icc.o jas_iccdata.o jas_image.o jas_init.o
jas_malloc.o jas_seq.o  jas_stream.o jas_string.o    jas_tvp.o
jas_version.o jas_cm.o bmp_cod.o  bmp_dec.o  bmp_enc.o jpc_bs.o jpc_cs.o
jpc_dec.o jpc_enc.o jpc_math.o jpc_mct.o jpc_mqcod.o jpc_mqdec.o
jpc_mqenc.o jpc_qmfb.o jpc_t1cod.o jpc_t1dec.o jpc_t1enc.o jpc_t2cod.o
jpc_t2dec.o jpc_t2enc.o jpc_tagtree.o jpc_tsfb.o jpc_util.o jp2_cod.o
jp2_dec.o jp2_enc.o
```

และไฟล์หลักที่มี Main

```
jasper.o
```

ขั้นสุดท้ายคือการ Link ไฟล์เพื่อให้ได้ Execute ไฟล์โดยใช้คำสั่งดังนี้

```
"arm-linux-gcc-3.4.1 jasper.o jas_debug.o jas_getopt.o  jas_icc.o
jas_iccdata.o jas_image.o jas_init.o jas_malloc.o jas_seq.o  jas_stream.o
jas_string.o    jas_tvp.o jas_version.o jas_cm.o bmp_cod.o  bmp_dec.o
bmp_enc.o jpc_bs.o jpc_cs.o jpc_dec.o jpc_enc.o jpc_math.o jpc_mct.o
jpc_mqcod.o jpc_mqdec.o jpc_mqenc.o jpc_qmfb.o jpc_t1cod.o jpc_t1dec.o
jpc_t1enc.o jpc_t2cod.o jpc_t2dec.o jpc_t2enc.o jpc_tagtree.o jpc_tsfb.o
jpc_util.o jp2_cod.o jp2_dec.o jp2_enc.o -o jasper.out"
```

โดยการลิ้งค์นี้จะใช้ option "-o" ตามด้วยชื่อไฟล์ที่ต้องการให้ได้ออกมาที่นี้คือ jasper.out เพื่อให้ใช้ได้กับ X-Scale การลิงค์นี้จะเชื่อมโยงโดยตรงจากไฟล์หลัก Main โดยจะรวมทั้ง Library เข้าไว้เป็นไฟล์ Output ไฟล์เดียวกัน

3.6 การพัฒนาส่วนการ Compile ไฟล์ให้สะดวกขึ้น

เป็นขั้นตอนโดยการเริ่มจาก .obj แล้วทำเป็น Execute โดยจะต้องทำการคอมไพล์ทีละตัวทำให้ใช้เวลานานจึงสร้างสคริปต์ไฟล์ โดยจะทำการเขียนคำสั่งการคอมไพล์ไว้ในสคริปต์ไฟล์

3.6.1 สคริปต์ไฟล์จะทำการเขียนเป็น 2 ส่วน

โดยส่วนแรกจะอยู่ในโฟลเดอร์แต่ละโฟลเดอร์ดังตารางที่ 3.1

ตารางที่ 3.1 การทำสคริปต์ไฟล์

ชื่อFolder	ชื่อสคริปต์ไฟล์	คำสั่งที่อยู่ในสคริปต์ไฟล์
base	base.txt	arm-linux-gcc jas_cm.c -c arm-linux-gcc jas_debug.c -c arm-linux-gcc jas_getopt.c -c arm-linux-gcc jas_icc.c -c arm-linux-gcc jas_iccdata.c -c arm-linux-gcc jas_image.c -c arm-linux-gcc jas_init.c -c arm-linux-gcc jas_malloc.c -c arm-linux-gcc jas_seq.c -c arm-linux-gcc jas_stream.c -c arm-linux-gcc jas_string.c -c arm-linux-gcc jas_tvp.c -c arm-linux-gcc jas_version.c -c
bmp	bmp.txt	arm-linux-gcc bmp_cod.c -c arm-linux-gcc bmp_dec.c -c arm-linux-gcc bmp_enc.c -c
jp2	jp2. txt	arm-linux-gcc jp2_cod.c -c arm-linux-gcc jp2_dec.c -c arm-linux-gcc jp2_enc.c -c
jpc	jpc.txt	arm-linux-gcc jpc_bs.c -c arm-linux-gcc jpc_cs.c -c arm-linux-gcc jpc_dec.c -c arm-linux-gcc jpc_enc.c -c arm-linux-gcc jpc_math.c -c arm-linux-gcc jpc_mct.c -c arm-linux-gcc jpc_mqcod.c -c arm-linux-gcc jpc_mqdec.c -c arm-linux-gcc jpc_mqenc.c -c arm-linux-gcc jpc_qmfb.c -c arm-linux-gcc jpc_t1cod.c -c arm-linux-gcc jpc_t1dec.c -c arm-linux-gcc jpc_t1enc.c -c arm-linux-gcc jpc_t2cod.c -c arm-linux-gcc jpc_t2dec.c -c arm-linux-gcc jpc_t2enc.c -c arm-linux-gcc jpc_tagtree.c -c arm-linux-gcc jpc_tsfb.c -c arm-linux-gcc jpc_util.c -c

คำอธิบาย arm-linux-gccg เป็นชื่อ Compiler ที่ใช้

– c เป็นตัวกำหนดการทำงานให้ออกเป็น object file

ตารางที่ 3.2 จะเป็นสคริปต์ไฟล์ ที่สั่งงานรวมทั้งหมดที่จะรับคำสั่งผ่าน All.txt

ชื่อสคริปต์ไฟล์	Folderที่จะเข้าไปทำงาน/ชื่อสคริปต์ไฟล์	คำสั่งในสคริปต์ไฟล์
base.txt	base/ base.txt	cd base ./ base.txt
bmp.txt	bmp/ bmp.txt	cd bmp ./ bmp.txt
jp2. txt	jp2/ jp2. txt	cd bmp ./ jp2.txt
jpc.txt	jpc/ jpc.txt	cd bmp ./ jpc.txt

3.6.2 การเปลี่ยน Permission

เมื่อทำการสร้างสคริปต์ไฟล์แล้วทำการเปลี่ยนโหมดการทำงาน ด้วยคำสั่ง chmod 777 แล้วตามด้วยชื่อสคริปต์ไฟล์เพื่อให้สคริปต์ไฟล์มีความสามารถในการ Execute

3.6.3 คำสั่งและการใช้งาน

เมื่อทำการ เปลี่ยน Permission ของสคริปต์ไฟล์ทุกตัวแล้วให้สั่ง Execute ไฟล์ All.txt ด้วยคำสั่ง $./All.txt ถ้าไม่มี Error ใดๆเกิดขึ้นก็จะได้ ไฟล์ .obj และทำการคอมไพล์ไฟล์ jasper.c ด้วยคำสั่ง

```
arm-linux-gcc jasper.c -c
```

3.7 Executable File

เมื่อเราได้ Object Fileมาแล้วจะได้ไฟล์

```
jasper.o jas_debug.o jas_getopt.o  jas_icc.o jas_iccdata.o jas_image.o
jas_init.o jas_malloc.o jas_seq.o  jas_stream.o jas_string.o    jas_tvp.o
jas_version.o jas_cm.o bmp_cod.o  bmp_dec.o  bmp_enc.o jpc_bs.o jpc_cs.o
jpc_dec.o jpc_enc.o jpc_math.o jpc_mct.o jpc_mqcod.o jpc_mqdec.o
jpc_mqenc.o jpc_qmfb.o jpc_t1cod.o jpc_t1dec.o jpc_t1enc.o jpc_t2cod.o
jpc_t2dec.o jpc_t2enc.o jpc_tagtree.o jpc_tsfb.o jpc_util.o jp2_cod.o
jp2_dec.o jp2_enc.o
```

นำไฟล์ทั้งหมดมารวมกันแล้วใช้คำสั่ง

```
arm-linux-gcc jasper.o jas_debug.o jas_getopt.o  jas_icc.o jas_iccdata.o
jas_image.o jas_init.o jas_malloc.o jas_seq.o  jas_stream.o jas_string.o
jas_tvp.o jas_version.o jas_cm.o bmp_cod.o  bmp_dec.o  bmp_enc.o jpc_bs.o jpc_cs.o
jpc_dec.o jpc_enc.o jpc_math.o jpc_mct.o jpc_mqcod.o jpc_mqdec.o jpc_mqenc.o
jpc_qmfb.o jpc_t1cod.o jpc_t1dec.o jpc_t1enc.o jpc_t2cod.o jpc_t2dec.o jpc_t2enc.o
jpc_tagtree.o jpc_tsfb.o jpc_util.o jp2_cod.o jp2_dec.o jp2_enc.o -o jasper.out
```

3.7.1 การเพิ่มไฟล์ Text

การรวมไฟล์ทั้งหมดนั้นมักเกิดข้อผิดพลาดในระหว่างการทดสอบทำให้ต้องใช้เวลาในการพิมพ์นานจึงสร้างไฟล์ link.txt โดยภายในไฟล์ จะมีคำสั่ง

```
arm-linux-gcc jasper.o jas_debug.o jas_getopt.o  jas_icc.o jas_iccdata.o
jas_image.o jas_init.o jas_malloc.o jas_seq.o  jas_stream.o jas_string.o
jas_tvp.o jas_version.o jas_cm.o bmp_cod.o  bmp_dec.o  bmp_enc.o jpc_bs.o
jpc_cs.o jpc_dec.o jpc_enc.o jpc_math.o jpc_mct.o jpc_mqcod.o jpc_mqdec.o
jpc_mqenc.o jpc_qmfb.o jpc_t1cod.o jpc_t1dec.o jpc_t1enc.o jpc_t2cod.o
jpc_t2dec.o jpc_t2enc.o jpc_tagtree.o jpc_tsfb.o jpc_util.o jp2_cod.o
jp2_dec.o jp2_enc.o -o jasper.out
```

3.7.2 การเปลี่ยน *Permission*

เมื่อทำการสร้างสคริปต์ไฟล์ link.txt แล้วทำการเปลี่ยนโหมดการทำงาน ด้วยคำสั่ง chmod 777 link.txt เพื่อให้สคริปต์ไฟล์มีความสามารถในการ Execute

3.7.3 คำสั่งและการใช้งาน

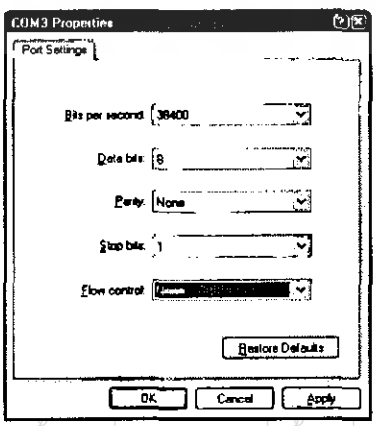
เมื่อทำการเปลี่ยน Permission ของสคริปต์ไฟล์ link.txt แล้วให้สั่ง Execute ไฟล์ link.txt ด้วยคำสั่ง $./ link.txt ถ้าไม่มี Error ใดๆเกิดขึ้นก็จะได้ ไฟล์ jasper.out ออกมา

3.8 การส่ง-รับไฟล์และภาพที่ต้องการไปสู่บอร์ต X-Scale

3.8.1 ช่องทางรับ-ส่ง

ช่องทางติดต่อกับ X-Scale เพื่อทำการรับส่งไฟล์นั้นมีได้หลายทางโดยเราเลือกใช้งานทางที่สะดวกที่สุดคือทาง LAN แต่ก่อนอื่นต้องมีการจัดเตรียมการติดต่อให้พร้อมโดยมีขั้นตอนดังนี้

1. คอนฟิกไอพีให้กับ Board ทางโดยใช้ Serial Port ติดต่อผ่านทาง Hyper-terminal และต้องเปิดเครื่องและต่อสาย Serial ระหว่าง Board กับคอมพิวเตอร์ให้เรียบร้อยและหลังจากนั้นเปิด Hyper-terminal ขึ้นมาตั้งค่าตามนี้ คือ Bit per second เป็น 38.4Kbps และ Flow control เป็น None



รูปที่ 3.4 การเซ็ตค่า Comport ของ Hyper-terminal

2. หลังจากติดต่อได้จะต้องพิมพ์ User ที่นี้ใช้ Root และ Pass ที่ตั้งไว้ที่นี้คือ Enter ผ่านได้ตาม Default หลังจากนั้นก็ตรวจสอบ ip address โดยคำสั่ง ifconfig บอร์ดจะแสดง ip address ขึ้นมาหลักจากนั้นเปลี่ยนให้อยู่ในคลาสให้ตรงกับ computer ของเราหรือจะเปลี่ยน ip ที่คอมพิวเตอร์ของเราให้อยู่ในคลาส subnet เดียวกันการเปลี่ยน ip ที่ X-Scale ได้โดยไปแก้ไขไฟล์ interface อยู่ที่ /dev/network/ โดยใช้คำสั่ง

```
$vi /dev/network/interface
```

หลังจากที่แก้ไขได้ตามที่ต้องการก็ save :wq!

และสั่งรีสตาร์ท X-scale

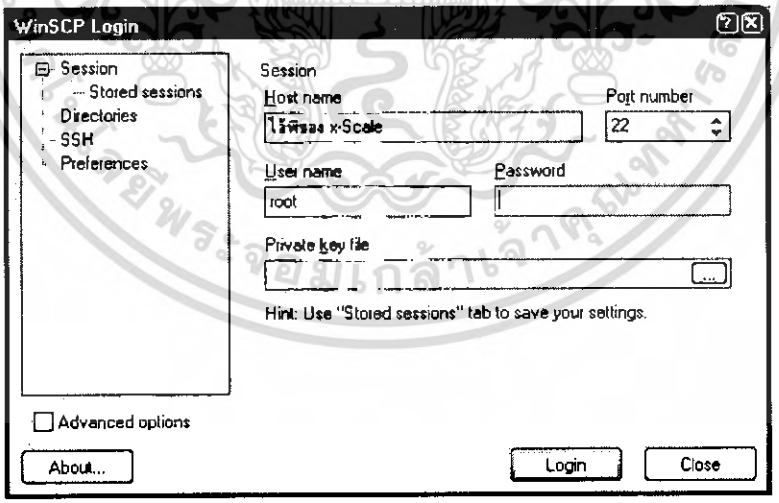
```
$restart
```

3. จากนั้นต่อสาย LAN เข้ากับทั้ง X-Scale และคอมพิวเตอร์ที่ต้องการติดต่อเข้าด้วยกันและหลังจากรีสตาร์ทเสร็จก็ทำการ ping ไปยัง ip ของตนเองที่ตั้งไว้ถ้าได้ก็ทดสอบ ping ไปยัง ip ของคอมพิวเตอร์ถ้าได้ก็มาถึงขั้นจัดเตรียมซอฟแวร์สำเร็จรูปบนคอมพิวเตอร์เพื่อเป็นการง่ายต่อการจะต้องติดตั้งโปรแกรมดังนี้คือ

Winscp ซึ่งมีให้ดาวน์โหลดฟรีทั่วไปโปรแกรมนี้เป็น SSH สำหรับติดต่อเพื่ออำนวยในการ อัพโหลดหรือดาวน์โหลดไฟล์จาก X-Scale และอีกโปรแกรมเพื่อความสะดวกในการใช้งาน Command คือ Putty ไปใช้งาน X-Scaleโดยทั้งสองติดต่อผ่าน port 22 ซึ่งมีการใช้งานดังนี้

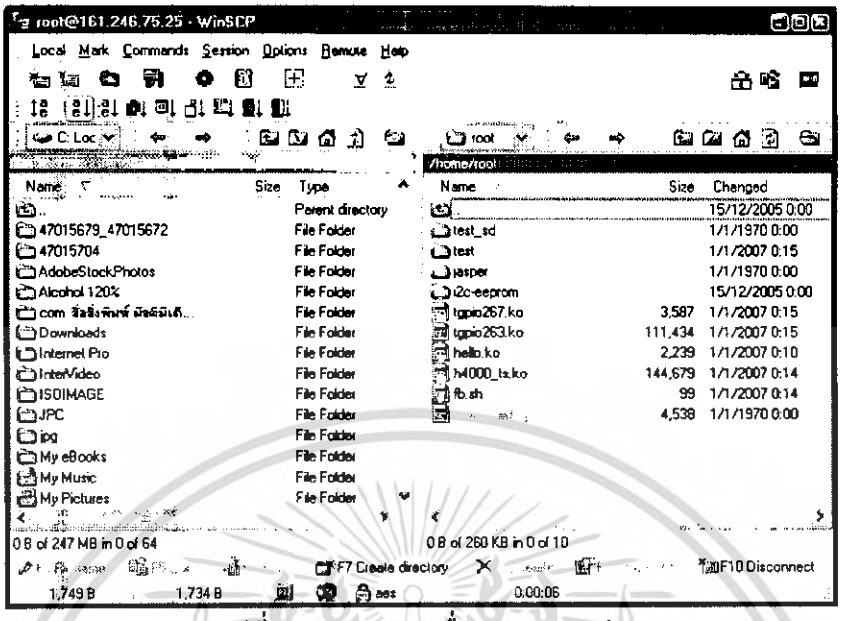
Winscp

เมื่อเปิดโปรแกรมขึ้นมาจะมีให้ใส่ Hostname ที่นี้ใส่ ip address ของ X-Scale Username: root



รูปที่ 3.5 WinSCP Login

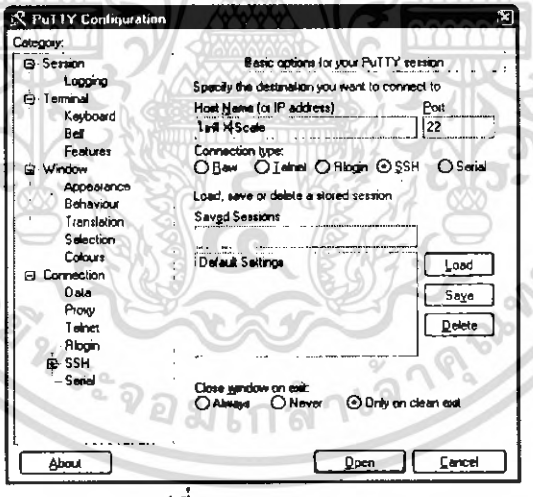
เมื่อติดต่อสำเร็จจะเข้าไปยังโฟลเดอร์ของ Root สามารถอัพโหลดและดาวน์โหลดไฟล์ได้โดยโปรแกรมนี้



รูปที่ 3.6 WinSCP เมื่อ Login สำเร็จ

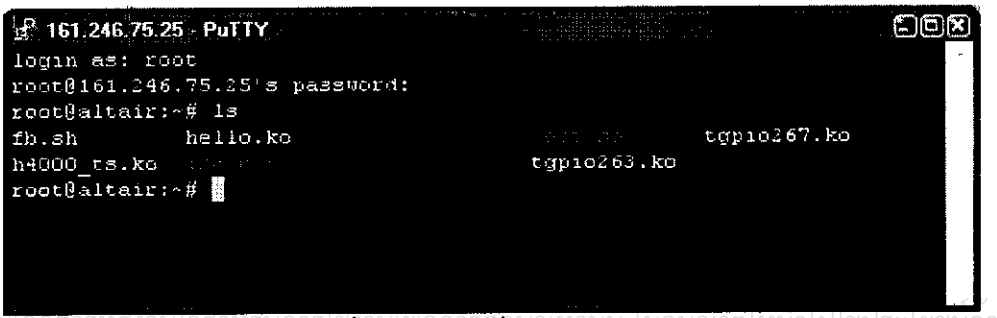
Putty

เมื่อเปิดโปรแกรมขึ้นมาจะมีให้ใส่ Hostname ที่นี้ใส่ ip address ของ X-Scale



รูปที่ 3.7 Putty Login

เมื่อเข้าไปจะมีให้ใส่ Username และ Password ที่นี้ใส่ root และไม่ต้องใส่ Password



รูปที่ 3.8 Putty เมื่อ Login สำเร็จ

3.8.2 การเตรียมตัว Execute

หลังจากได้ไฟล์ Execute ที่นี้คือ jasper.out ก็ทำการอัพโหลดไฟล์ไปยัง Directory ที่ต้องการและภาพที่ต้องการเข้ารหัสหลังจากที่เตรียมพร้อมแล้วต้องทำการเป็น Permission ให้กับไฟล์ Execute โดยใช้คำสั่ง

```
$chmod 777 jasper.out
```

3.8.3 ตัวอย่างคำสั่งต่างๆ ที่ใช้ได้

```
$./jasper.out --input (name1) --output (name2) --output-format (name3)
$./jasper.out -f  (name1) -F (name2) -T (name3) -O rate=0.01
```

โดยที่ name1: คือชื่อไฟล์ input จะเป็นได้ทั้งสามรูปแบบ (format) คือ bmp, jpc ,jp2 โดยต้องเป็นชื่อตามด้วย format เช่น lena.bmp

name2: คือชื่อไฟล์ output ที่ต้องการและชื่อต้องตามด้วย format ได้ทั้งสาม bmp, jpc ,jp2 ยกตัวอย่างเช่น lena_lossy.jpc

name3: คือ format output ชื่อนี้ต้องมีความสัมพันธ์กับ name2 คือถ้าตั้งไว้เป็น format ไดก็ต้องใส่ format ตรงตามนั้นที่ name2 เป็น"lena_lossy.jpc" ตัวอย่างนี้ก็จะเป็น jpc

rate: คือการกำหนดว่าจะบีบอัดที่อัตราเท่าใดโดยมีอัตราส่วนเป็น % เช่น 10% จะใส่ rate=0.10 โดยถ้าจะเข้ารหัสแบบไม่สูญเสียก็จะไม่ใส่ option นี้เหมือนดังเช่นคำสั่งแรก

3.9 การเพิ่มฟังก์ชั่นจับเวลาให้ทั้งกับ code บน PC และบน X-Scale

การเพิ่มนี้จะเพิ่มทั้งกับ Source Code บนวินโดวส์ Microsoft Visual C++ 6.0 และทั้งกับบน X-Scale การเพิ่มนั้นจะมีรูปแบบที่ไม่ยากคือจะมีการเก็บค่าที่ได้จากฟังก์ชั่น clock(); (ฟังก์ชันนี้มีอยู่ใน STD time.h)ไว้แล้วจึงทำการเรียกฟังก์ชัน Encoder หรือ Decoder และหลังจากทำการเข้าหรือถอดรหัสเสร็จแล้วก็จะเรียงฟังก์ชัน clock(); อีกครั้งแล้วเก็บค่าไว้ในตัวแปรอีกตัวหลังจากนั้นนำตัวแปรที่เก็บค่าหลังลบด้วยตัวแปรแรกจะได้ค่าจำนวน clock ที่ใช้ ซึ่งจากการศึกษาพบว่าใน win32 ของวินโดวส์จะใช้ค่า

```
#define CLOCKS_PER_SEC 1000/*WIN32 & MAC */
```

1000 เป็นตัวหารซึ่งจะได้ค่าเวลาที่แท้จริงออกมาเป็นวินาที และถ้าเป็นของ dsp จะใช้ค่าความเร็วของ cpu มาเป็นตัวหารคือถ้า DSP 200MHz ตัวอย่างเช่น

```
#define CLOCKS_PER_SEC 200000000/*200MHz DSP */
```

และจากการทดสอบ X-Scale ที่ความเร็ว 400 MHz นี้จะใช้

```
#define CLOCKS_PER_SEC 1000000 /*400MHz Altair */
```

ยกตัวอย่างการเขียนดังนี้

```
      startclk = clock();
      if (!(image = jas_image_decode(in, cmdopts->infmt, cmdopts-
>inopts))) {
            fprintf(stderr, "error: cannot load image data\n");
            exit(EXIT_FAILURE);
      }
      endclk = clock();
;
      dectime = endclk - startclk;
    fprintf(stderr, "decoding time = %f", dectime / (double)
    CLOCKS_PER_SEC);
      fprintf(stderr, " second\n");
```

บทที่ 4

การทดลองและผลการทดลอง

ในบทนี้จะได้กล่าวถึงผลการทดลองของการเข้ารหัสด้วยมาตรฐาน JPEG 2000 ตามขั้นตอนที่ได้อธิบายไว้แล้วในบทที่ 3 สำหรับการทดลองนี้ได้แบ่งออกเป็น 2 การทดลองได้แก่ การทดสอบการเข้ารหัสบนเครื่อง PC และการทดสอบการเข้ารหัสบนบอร์ด Altair ซึ่งบทนี้ประกอบไปด้วยเนื้อหาดังนี้

4.1 การทดสอบการเข้ารหัสบนเครื่อง PC

4.1.1 การทดลองแบบ Loss Less

4.1.2 การทดลองแบบมี Loss ~ 10 เท่า

4.1.3 การทดลองแบบมี Loss ~ 100 เท่า

4.2 การทดสอบการเข้ารหัสบนบอร์ด Altair

4.2.1 การทดลองแบบ Loss Less

4.2.2 การทดลองแบบมี Loss ~ 10 เท่า

4.2.3 การทดลองแบบมี Loss ~ 100 เท่า

4.3 การวิเคราะห์และสรุปผลที่ได้จากการทดลอง

4.3.1 การวิเคราะห์และสรุปผลความถูกต้องของการเข้ารหัส jpeg 2000 บน PC และบนบอร์ด Altair

4.3.2 การวิเคราะห์และสรุปผลเวลาของการเข้ารหัส jpeg 2000 บน PC และ บนบอร์ด Altair

4.1 การทดสอบการเข้ารหัสบนเครื่อง PC

ในการทดลองทั้งหมดนี้ได้ทำการวัดผลการทดลองด้วยค่ามาตรฐาน 2 ค่า ได้แก่

1. Peak Signal to Noise Ratio (PSNR)
2. Mean Squared Error (MSE)

ภาพตั้งต้นที่ใช้ในการทดลองจัดเป็นภาพมาตรฐานทั่วไปที่มักใช้ในการทดลองเกี่ยวกับรูปภาพ เช่น การทำเวฟเลต (Wavelet) การแปลงภาพ การบีบอัดภาพเป็นต้น คือ Baboon.bmp และ Lena.bmp ซึ่งมีความละเอียดของภาพ 512*512 พิกเซล และขนาดภาพ 768 KB

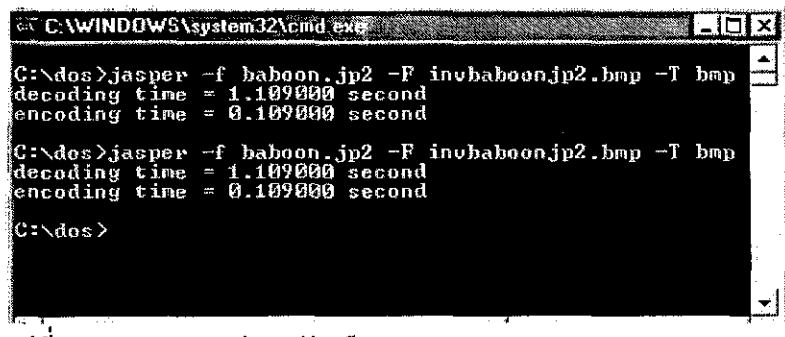
<u>รูปแบบคำสั่งและลักษณะการใช้งานการแปลงไฟล์ด้วย jasper</u>

./jasper –f *input filename* -F *output filename* –T *format file* –o rate=*value*

./jasper เป็นการสั่งให้โปรแกรมชื่อ Jasper ทำงานโดยมี Option ดังนี้

–f *input filename* เป็นการระบุชื่อไฟล์ที่ต้องการแปลง เช่น –f baboon.bmp

-F *output filename* เป็นการระบุชื่อไฟล์และนามสกุลที่ต้องการหลังการแปลง เช่น-F baboon.jp2



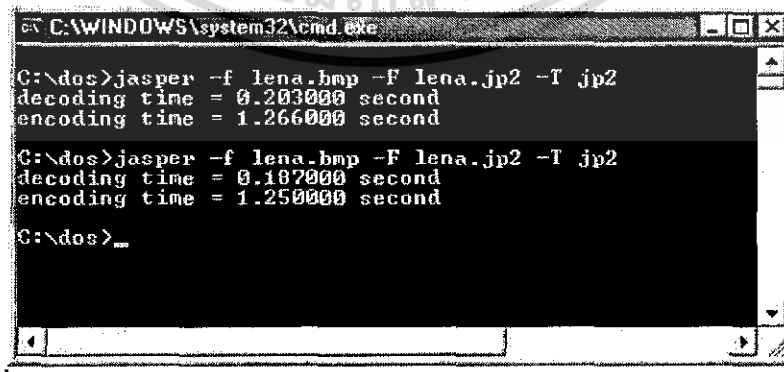
รูปที่ 4.5 แสดงการแปลงกลับเป็น bitmap (invbaboon.bmp Lossless)

- ทำการเปรียบเทียบค่าระหว่าง invbaboonjp2.bmp ซึ่งเป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ baboon.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือ ค่า PSNR = INF และค่า MSE = 0 ดังรูปที่ 4.6

รูปที่ 4.6 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (invbaboon.bmp Lossless)

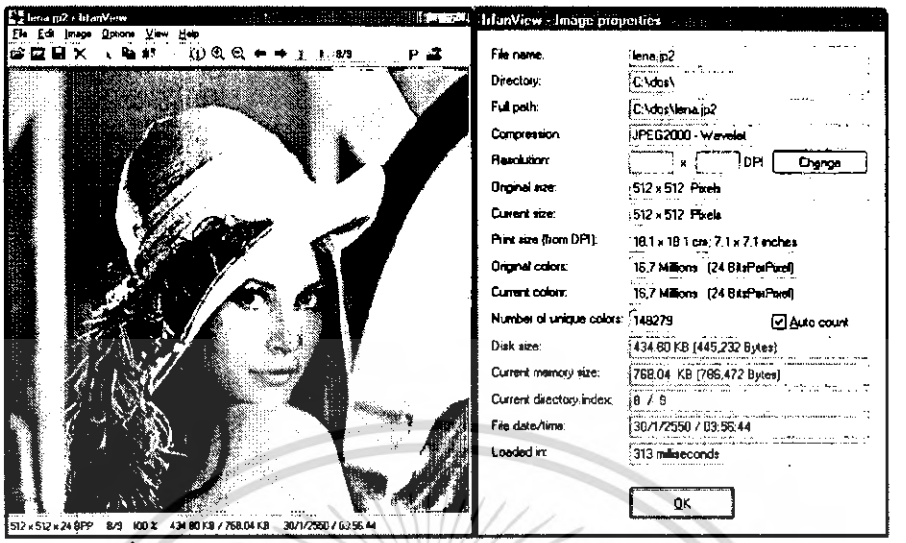
(2) รูป lena.bmp

- ทำการทดลองแปลงไฟล์ lena.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ JPEG2000 ชื่อ lena.jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.2030 วินาที และ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 1.2660 วินาที ดังรูปที่ 4.7



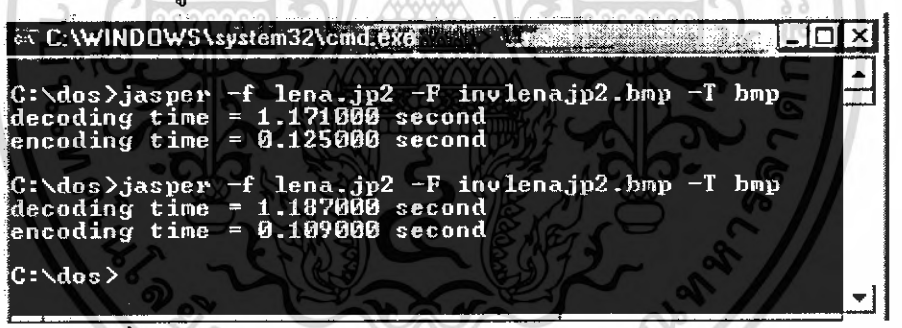
รูปที่ 4.7 แสดงการแปลง ภาพจาก bitmap เป็น JPEG 2000(lena.jp2 Lossless)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 361 KB (lena.jp2) ดังรูปที่ 4.8



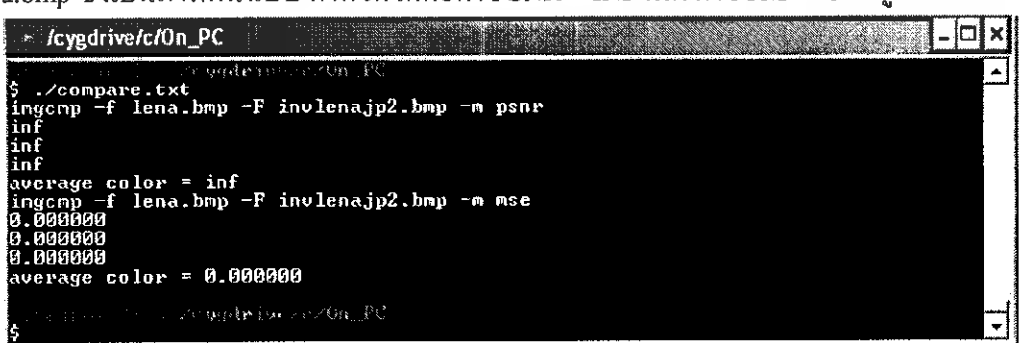
รูปที่ 4.8 แสดงขนาดและคุณสมบัติของภาพ (lena.jp2 Lossless)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 1.1710 วินาที และ Encoding Time หรือเวลาที่ใช้ใน การเข้ารหัสภาพเท่ากับ 0.1250 วินาที ดังรูปที่ 4.9



รูปที่ 4.9 แสดงการแปลงกลับเป็น bitmap(invlena.jp2 Lossless)

- ทำการเปรียบเทียบค่าระหว่าง invlenajp2.bmp ซึ่งเป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้ว กับ lena.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = INF และค่า MSE = 0 ดังรูปที่ 4.10

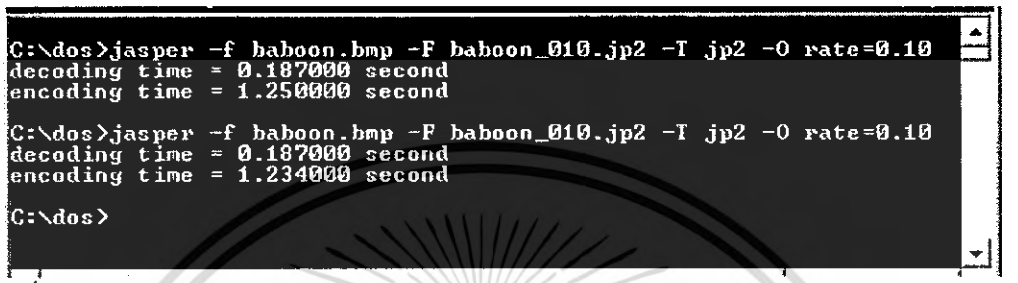


รูปที่ 4.10 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (invlena.jp2 Lossless)

4.1.2 แบบ Loss (มีการสูญเสีย) อัตราการบีบอัด ~ 10 เท่า

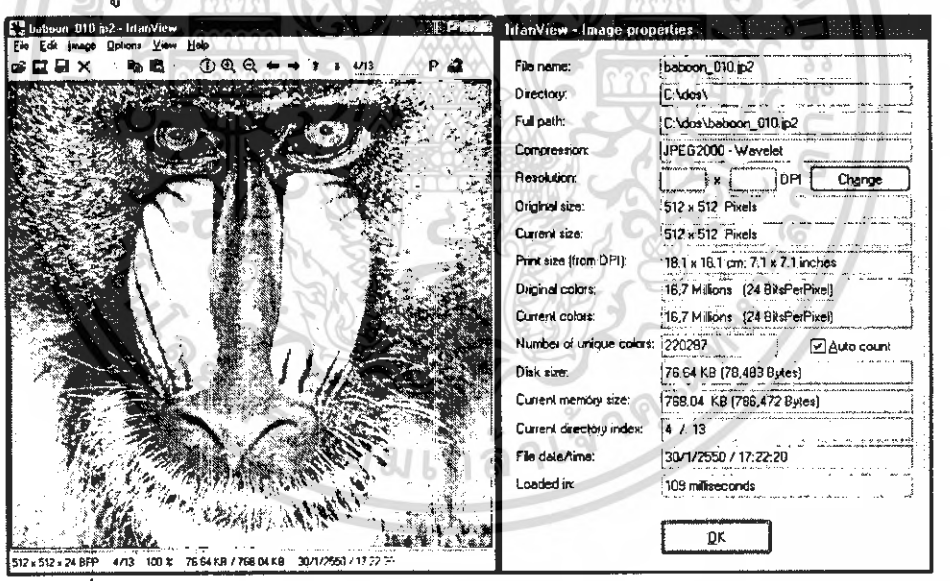
(1) รูป Baboon.bmp (rate=0.10)

- ทำการทดลองแปลงไฟล์ baboon.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ baboon_010.jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.1870 วินาที และ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 1.2500 วินาที ดังรูปที่ 4.11



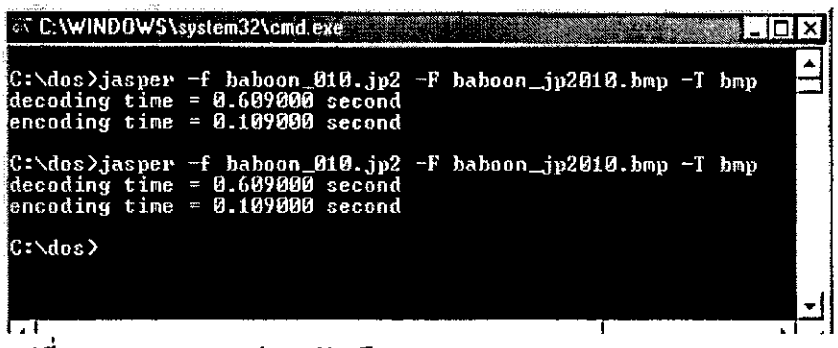
รูปที่ 4.11 แสดงการแปลง ภาพจาก bitmap เป็น JPEG 2000 (baboon_010.jp2 rate=0.10)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 361 KB (baboon_010.jp2) มองด้วยตาเปล่าไม่เห็นความเปลี่ยนแปลง ดังรูปที่ 4.12



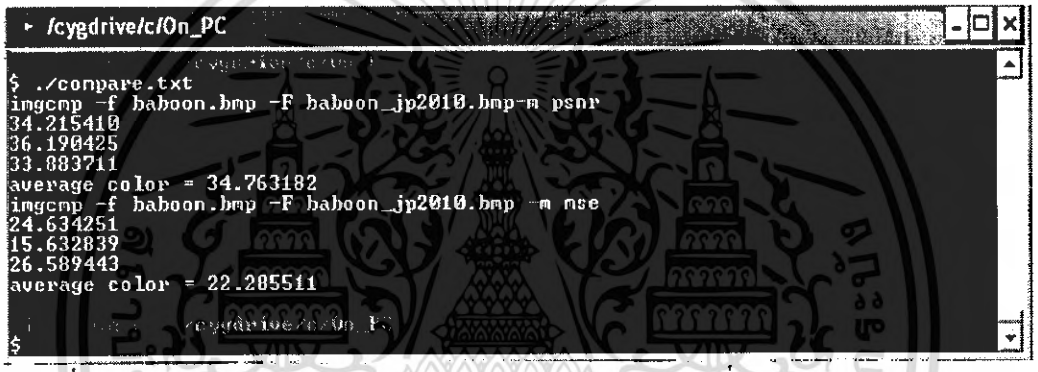
รูปที่ 4.12 แสดงขนาดและคุณสมบัติของภาพ (baboon_010.jp2 rate=0.10)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ(bitmap) จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.6090 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 0.1090 วินาที ดังรูปที่ 4.13



รูปที่ 4.13 แสดงการแปลงกลับเป็น bitmap (baboon_010.jp2 rate=0.10)

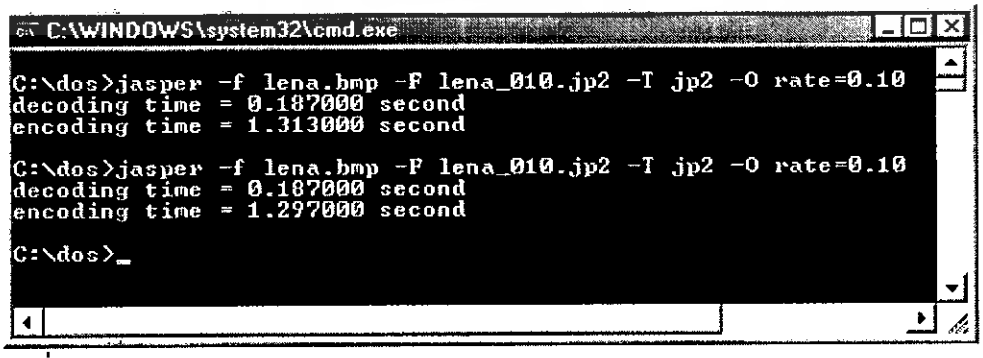
- ทำการเปรียบเทียบค่าระหว่าง baboon_jp2010.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ baboon.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = 34.763182 และค่า MSE = 22.285511 ดังรูปที่ 4.14



รูปที่ 4.14 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (baboon_010.jp2 rate=0.10)

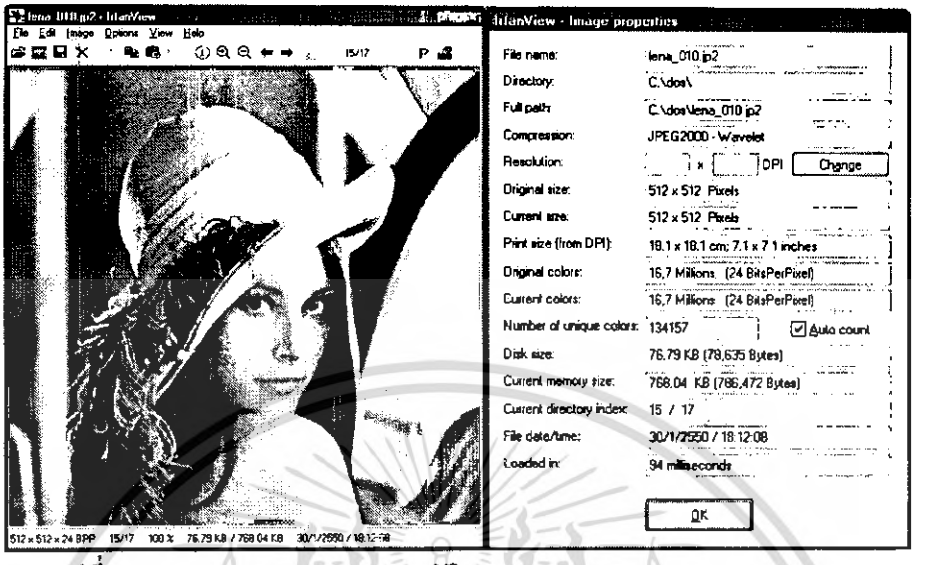
(2) รูป lena.bmp (rate=0.10)

- ทำการทดลองแปลงไฟล์ lena.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ lena_010.jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.1870 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 1.3130 วินาที ดังรูปที่ 4.15



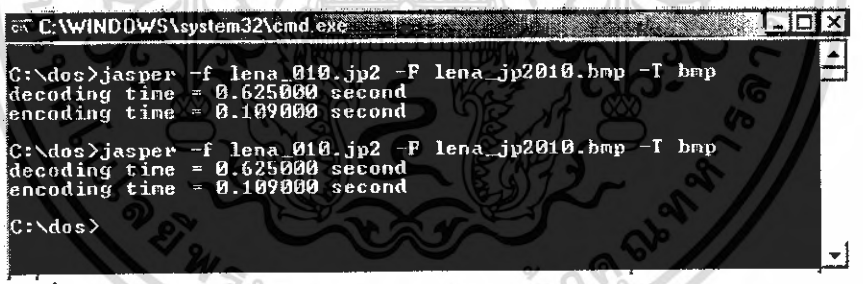
รูปที่ 4.15 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000(lena_010.jp2 rate=0.10)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 76 KB (lena_010.jp2) มองด้วยตาเปล่าไม่เห็นถึงความเปลี่ยนแปลง ดังรูปที่ 4.16



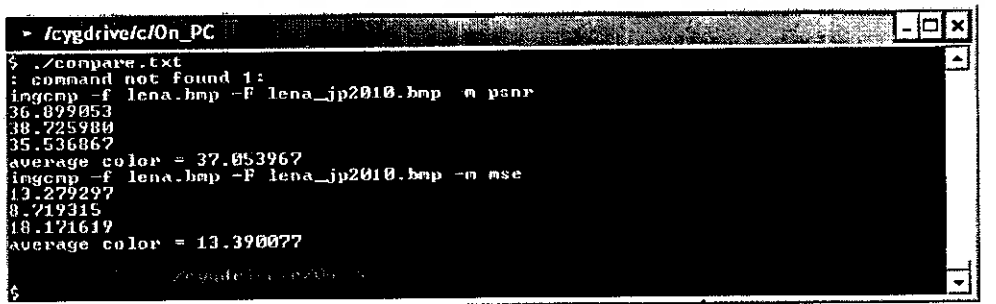
รูปที่ 4.16 แสดงขนาดและคุณสมบัติของภาพ (lena_010.jp2 rate=0.10)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ (bitmap) จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.6250 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 0.1090 วินาที ดังรูปที่4.17



รูปที่ 4.17 แสดงการแปลงกลับเป็น bitmap (lena_jp2010.bmp rate=0.10)

- ทำการเปรียบเทียบค่าระหว่าง lena_jp2010.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้ว กับ lena.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = 37.053967 ค่า MSE = 13.39007 ดังรูปที่ 4.18



รูปที่ 4.18 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (lena_jp2010.bmp rate=0.10)

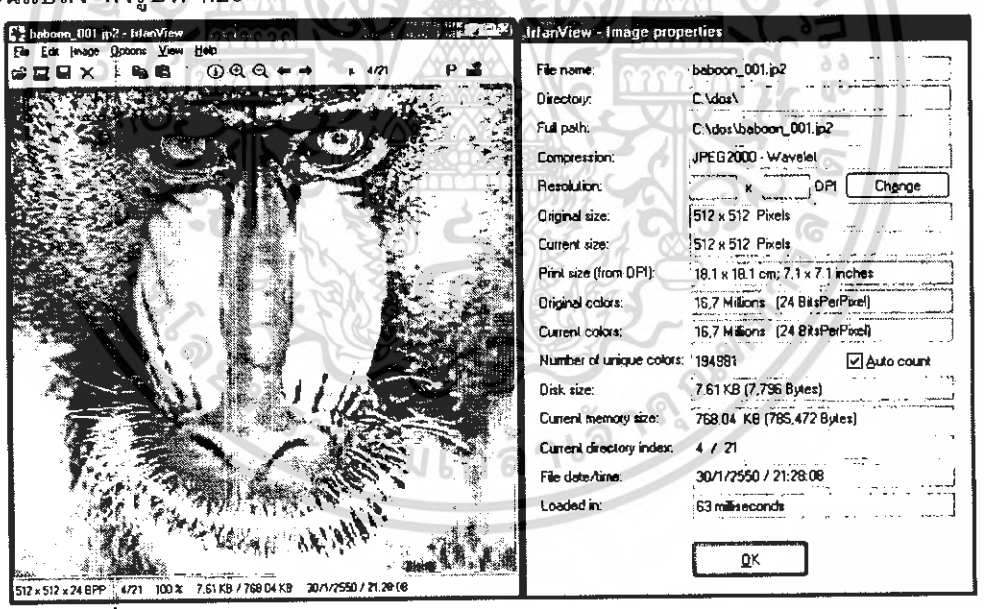
4.1.3 แบบ Loss (มีการสูญเสีย) อัตราการบีบอัด ~ 100เท่า

(1) รูป baboon.bmp (rate=0.01)

- ทำการทดลองแปลงไฟล์ baboon.bmp ความละเอียด 512*512 (color) ขนาด 768 KBแปลงเป็นไฟล์ jpeg 2000 ชื่อ baboon_001.jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.2030 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 1.1090 วินาที ดังรูปที่ 4.19

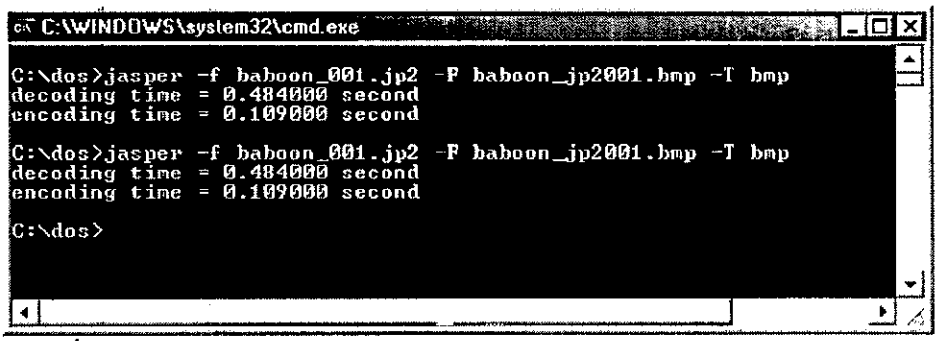
รูปที่ 4.19 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000(baboon_001.jp2 rate=0.01)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 7.61KB (baboon_001.jp2) มองด้วยตาเปล่าจะเห็นถึงความเปลี่ยนแปลง ดังรูปที่ 4.20



รูปที่ 4.20 แสดงขนาดและคุณสมบัติของภาพ (baboon_001.jp2 rate=0.01)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.4840 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 0.1090 วินาที ดังรูปที่ 4.21



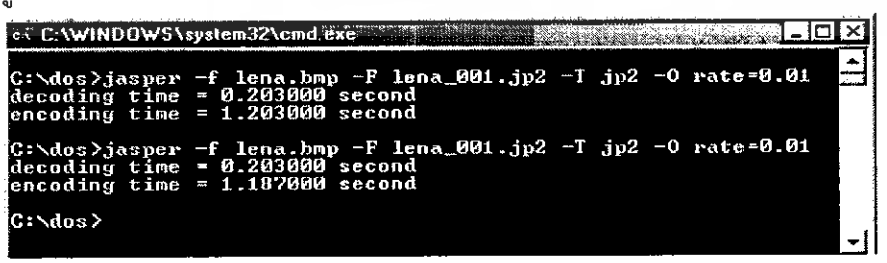
รูปที่ 4.21 แสดงการแปลงกลับเป็น bitmap (baboon_jp2001.bmp rate=0.01)

- ทำการเปรียบเทียบค่าระหว่าง baboon_jp2001.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ baboon.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR= 22.274108 และค่า MSE = 385.976060 ดังรูปที่ 4.22

รูปที่ 4.22 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (baboon_jp2001.bmp rate=0.01)

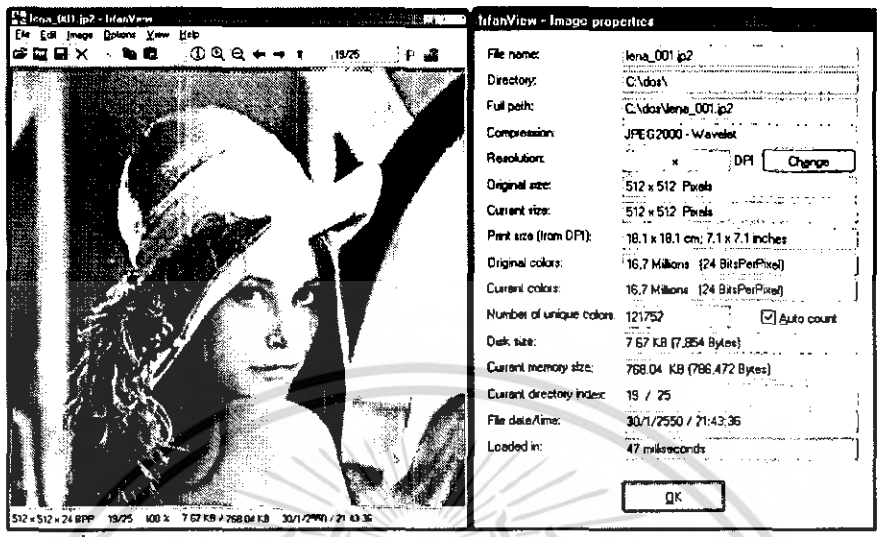
(2) รูป lena.bmp (rate=0.01)

- ทำการทดลองแปลงไฟล์ lena.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ lena_001.jp2 jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.2030 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 1.2030 วินาที ดังรูปที่ 4.23



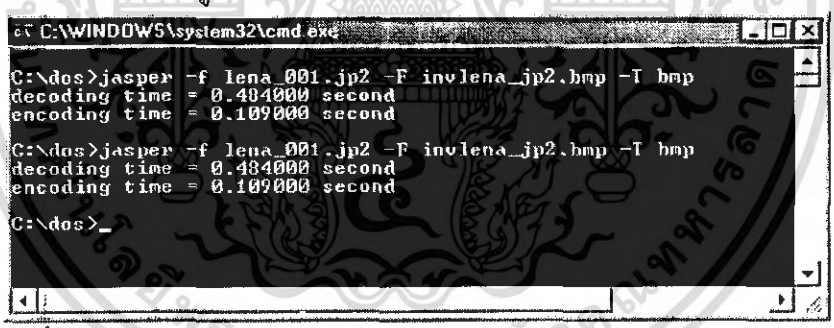
รูปที่ 4.23 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000 (lena_001.jp2 rate=0.01)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 7.67 KB (lena_001.jp2) มองด้วยตาเปล่าจะเห็นถึงความเปลี่ยนแปลง ดังรูปที่ 4.24



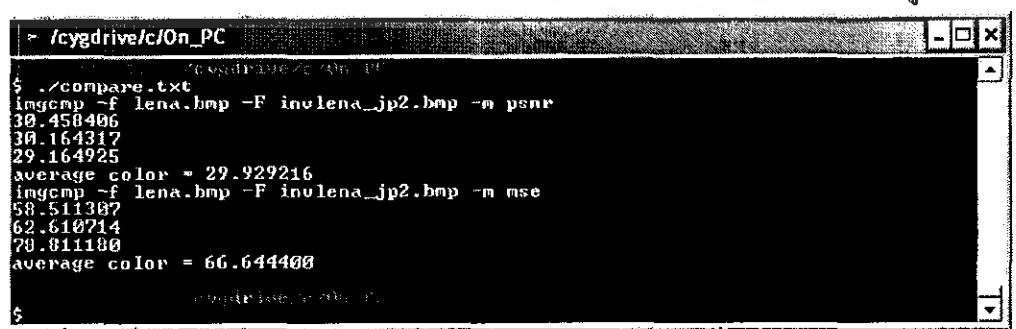
รูปที่ 4.24 แสดงขนาดและคุณสมบัติของภาพ (lena_001.jp2 rate=0.01)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 0.4840 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 0.1090 วินาที ดังรูปที่ 4.25



รูปที่ 4.25 แสดงการแปลงกลับเป็น bitmap (invlena_jp2.bmp rate=0.01)

- ทำการเปรียบเทียบค่าระหว่าง invlena_jp2.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ lena.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = 29.929216 และค่า MSE = 66.644400 ดังรูปที่ 4.26



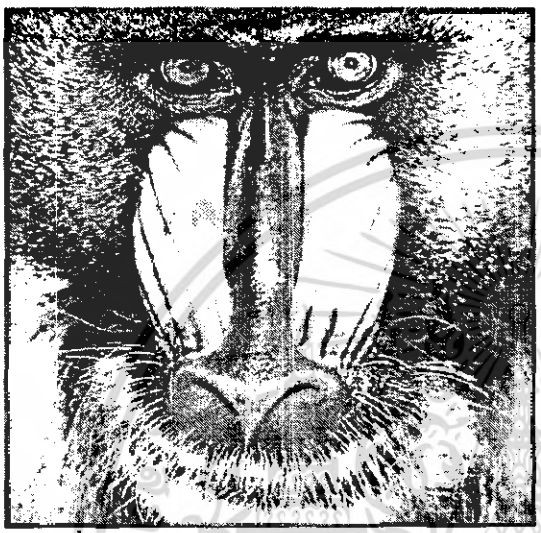
รูปที่ 4.26 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (invlena_jp2.bmp rate=0.01)

4.2 ทดสอบการเข้ารหัสบนบอร์ด Altair

คุณสมบัติของบอร์ด

ARM PXA255 400 MHz, Memory 32 MB

รูปที่ใช้ในการทดลองกับการทำงานกับ XSCALE 2 รูป คือ baboon.bmp และ Lena.bmp แสดงดังรูปที่ 4.27 และ 4.28



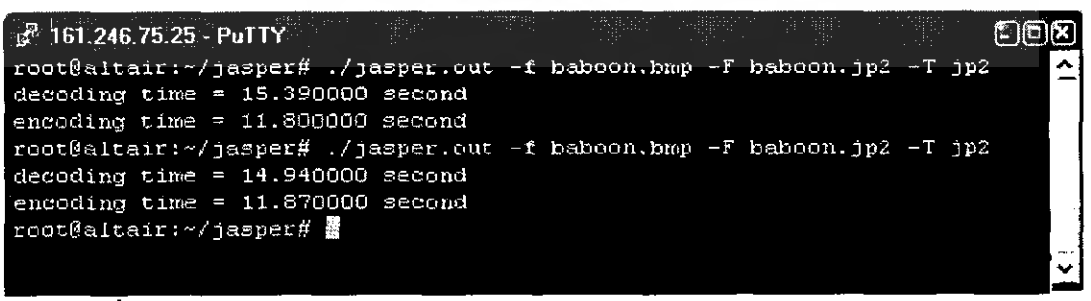
รูปที่ 4.27 Baboon.bmp 512*512 768KB

รูปที่ 4.28 Lena.bmp 512*512 768K

4.2.1 แบบ Lossless (ไม่มีการสูญเสีย)

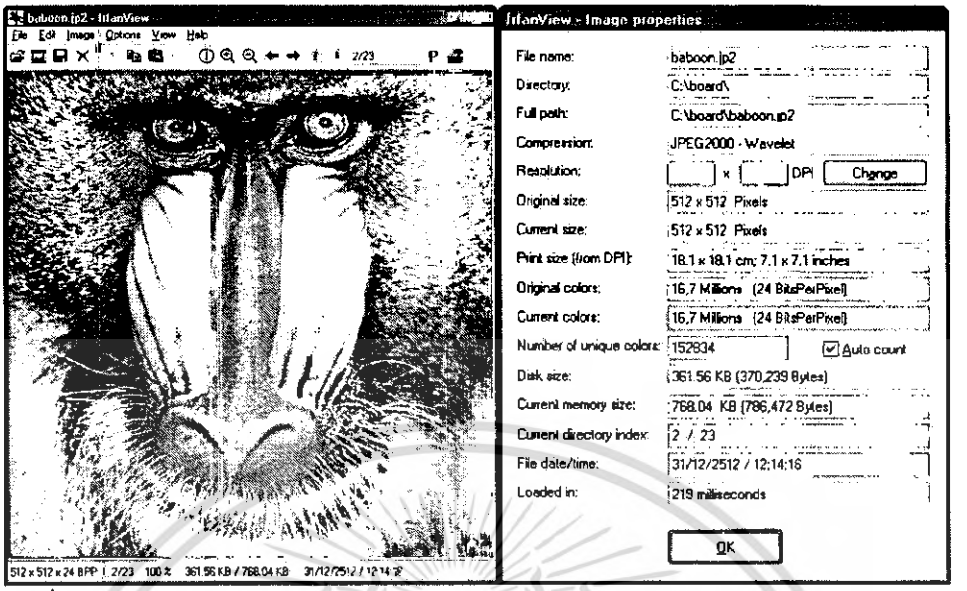
(1) รูป baboon.bmp

- ทำการทดลองแปลงไฟล์ baboon.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ baboon.jp2 jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 15.390 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 11.8000 วินาที ดังรูปที่ 4.29



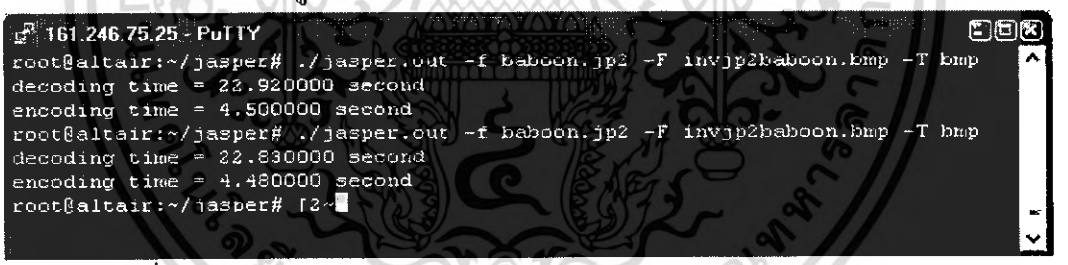
รูปที่ 4.29 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000 (baboon.jp2 Lossless)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 361.56 KB(baboon.jp2) ดังรูปที่ 4.30



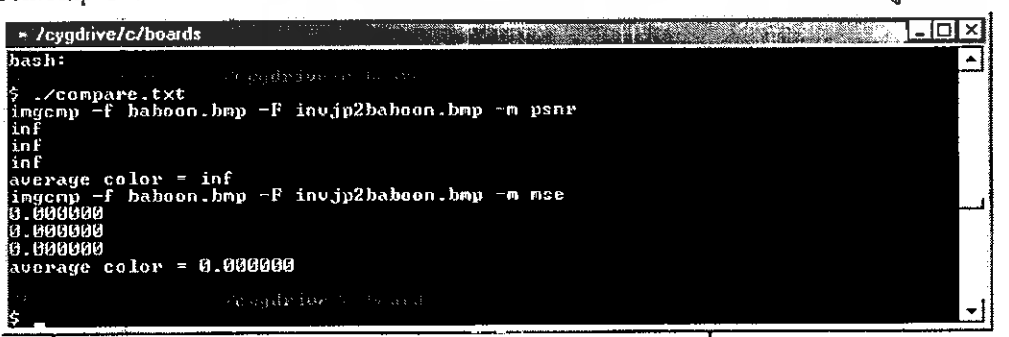
รูปที่ 4.30 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000 (baboon.jp2 Lossless)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 22.92000 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 4.500 วินาที ดังรูปที่ 4.31



รูปที่ 4.31 แสดงการแปลงกลับเป็น bitmap (invjp2baboon.bmp Lossless)

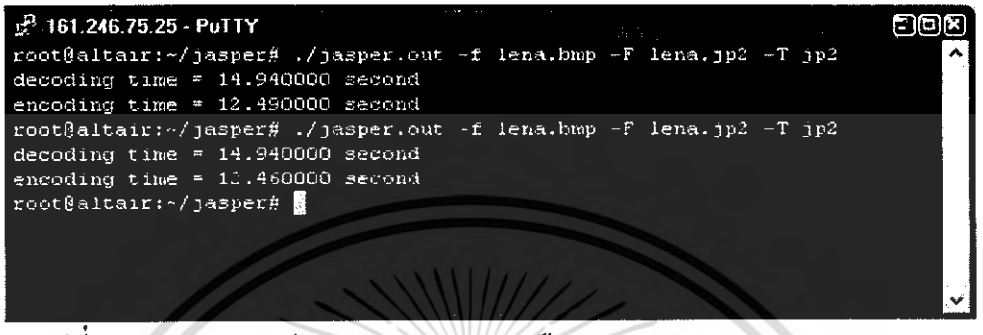
- ทำการเปรียบเทียบค่าระหว่าง invjp2baboon.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ baboon.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = INF และค่า MSE= 0 ดังรูปที่ 4.32



รูปที่ 4.32 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (invjp2baboon.bmp Lossless)

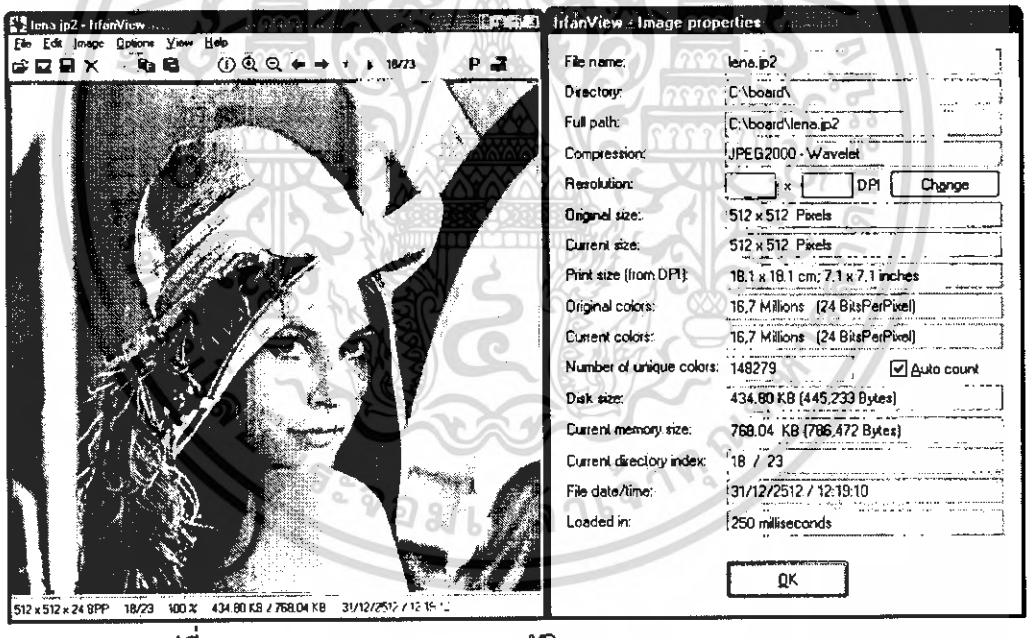
(2) รูป lena.bmp

- ทำการทดลองแปลงไฟล์ lena.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ lena.jp2 jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 14.9400 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 12.490 วินาที ดังรูปที่ 4.33



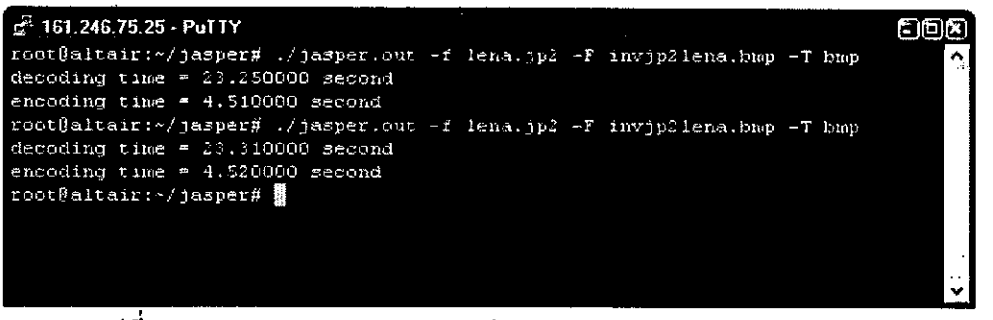
รูปที่ 4.33 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000 (lena.jp2 Lossless)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 434 KB(lena.jp2) ดังรูปที่ 4.34



รูปที่ 4.34 แสดงขนาดและคุณสมบัติของภาพ (lena.jp2 Lossless)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ jp2 จะได้ค่า Decoding Timeหรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 23.2500 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 4.5100 วินาที ดังรูปที่ 4.35



รูปที่ 4.35 แสดงขนาดและคุณสมบัติของภาพ (jp2lena.bmp Lossless)

- ทำการเปรียบเทียบค่าระหว่าง inv jp2lena.bmp ซึ่งเป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ lena.bmpซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = INF และค่า MSE = 0 ดังรูปที่ 4.36

รูปที่ 4.36 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (jp2lena.bmp Lossless)

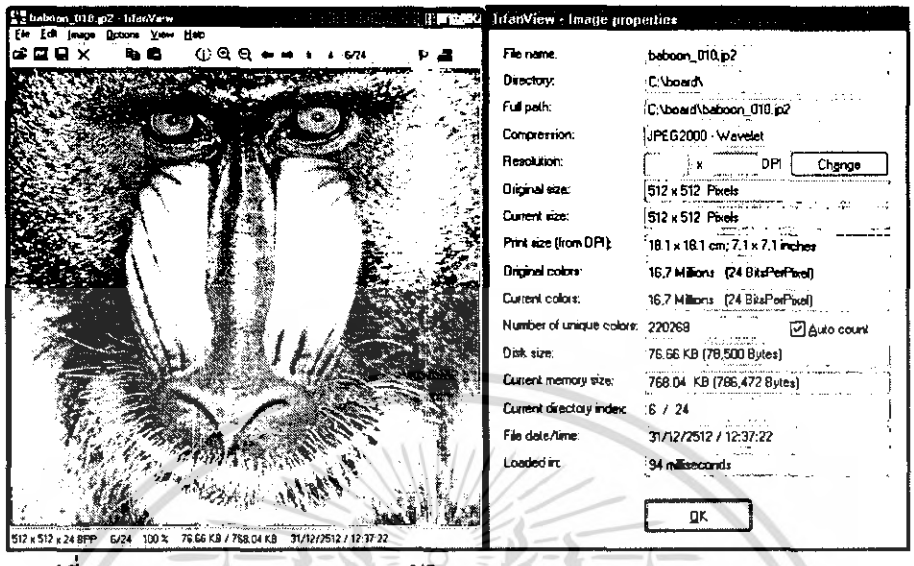
4.2.2 แบบ Loss (มีการสูญเสีย) อัตราการบีบอัด ~ 10 เท่า

(1) รูป baboon.bmp (rate=0.01)

- ทำการทดลองแปลงไฟล์ baboon.bmp ความละเอียด 512*512 (color) ขนาด 768 KBแปลงเป็นไฟล์ jpeg 2000 ชื่อ baboon_010.jp2 jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 14.940 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 11.0200 วินาที ดังรูปที่ 4.37

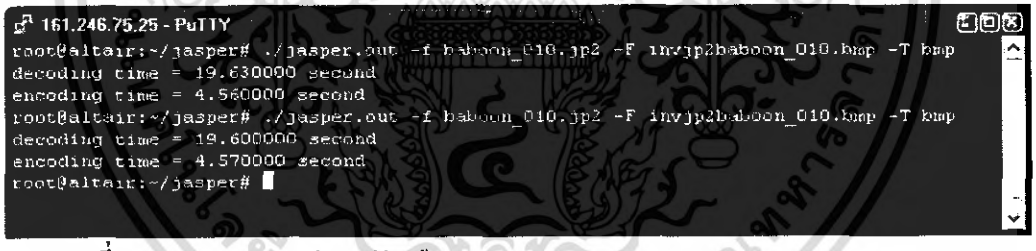
รูปที่ 4.37 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000 (baboon_010.jp2 rate=0.01)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 76.66 KB (baboon_010.jp2) ภาพที่ได้มองด้วยตาเปล่าแทบไม่เห็นการเปลี่ยนแปลง ดังรูปที่ 4.38



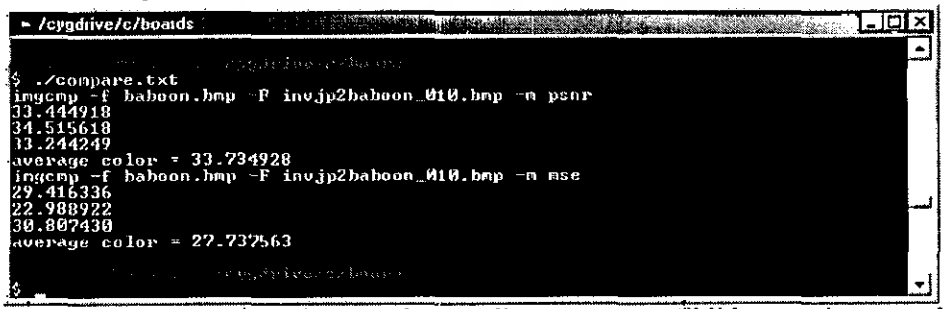
รูปที่ 4.38 แสดงขนาดและคุณสมบัติของภาพ (baboon_010.jp2 rate=0.01)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 19.6300 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 4.5600 วินาที ดังรูปที่ 4.39



รูปที่ 4.39 แสดงการแปลงกลับเป็น bitmap (invjp2baboon_010.bmp rate=0.01)

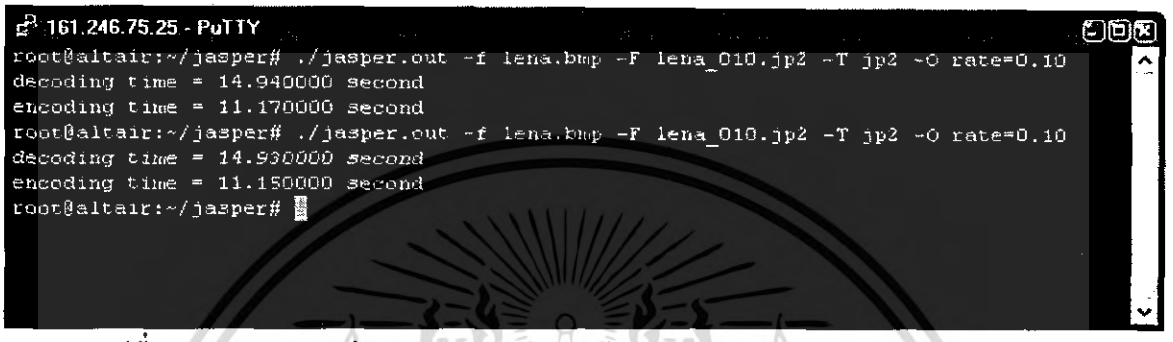
- ทำการเปรียบเทียบค่าระหว่าง invjp2baboon_010.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ baboon.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = 33.7349288ค่า MSE ได้ = 27.737563 ดังรูปที่ 4.40



รูปที่ 4.40 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (invjp2baboon_010.bmp rate=0.01)

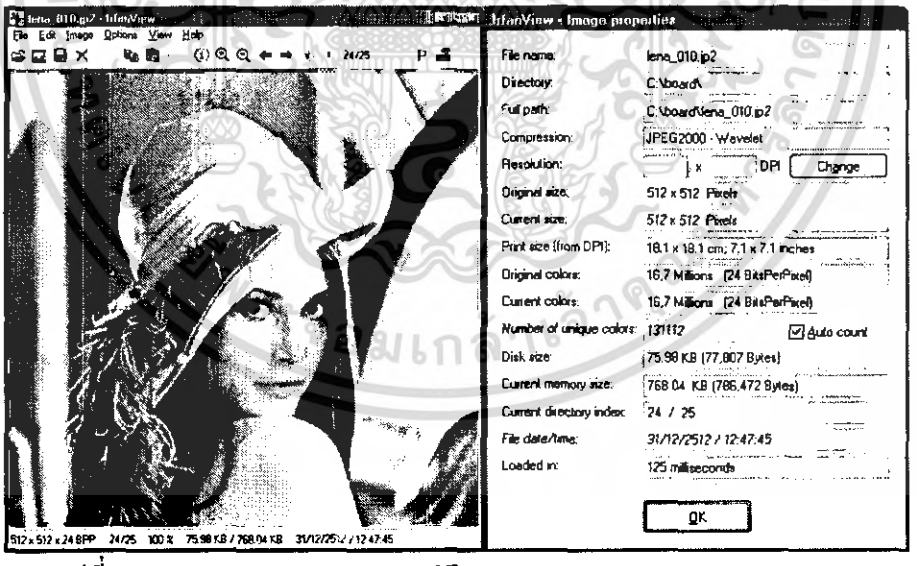
(2) รูป lena.bmp (rate=0.1)

- ทำการทดลองแปลงไฟล์ lena.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ lena_010.jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 14.9400 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 11.1700 วินาที ดังรูปที่ 4.41



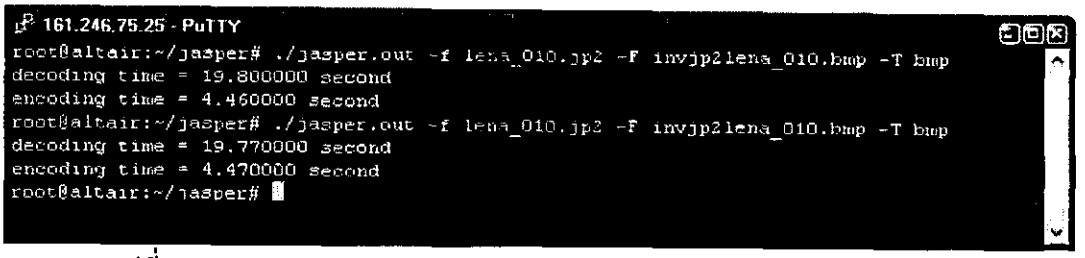
รูปที่ 4.41 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000 (lena_010.jp2 rate=0.1)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 75.98 KB (lena_010.jp2) มองด้วยตาเปล่าไม่เห็นความเปลี่ยนแปลง ดังรูปที่ 4.42



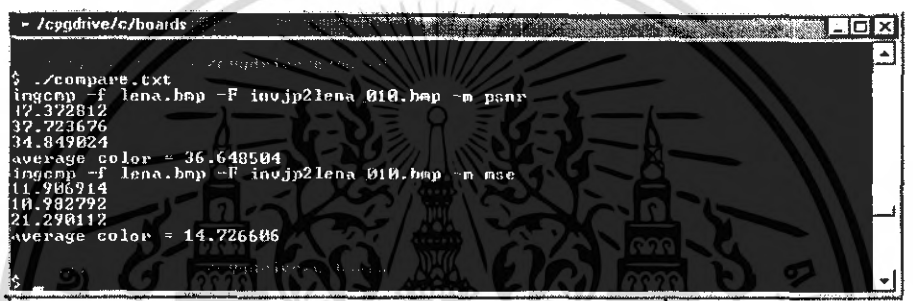
รูปที่ 4.42 แสดงขนาดและคุณสมบัติของภาพ (lena_010.jp2 rate=0.1)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 19.800 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 4.460 วินาที ดังรูปที่ 4.43



รูปที่ 4.43 แสดงการแปลงกลับเป็น bitmap (invjp2lena_010.bmp rate=0.1)

- ทำการเปรียบเทียบค่าระหว่าง invjp2lena_010.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ lena.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = 36.648504 และค่า MSE = 14.726606 ดังรูปที่ 4.44

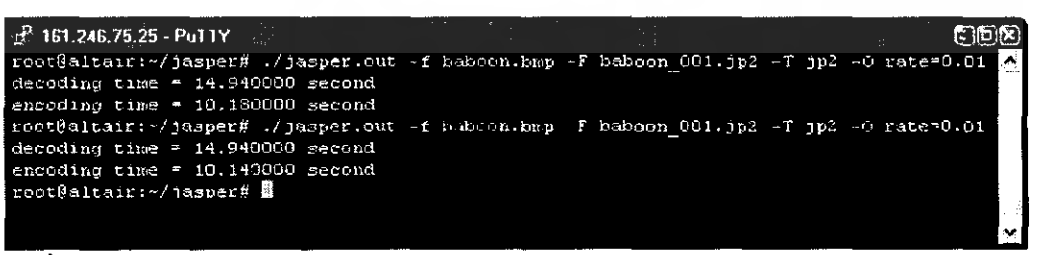


รูปที่ 4.44 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (invjp2lena_010.bmp rate=0.1)

4.2.3 แบบ Loss (มีการสูญเสีย) อัตราการบีบอัด ~ 100เท่า

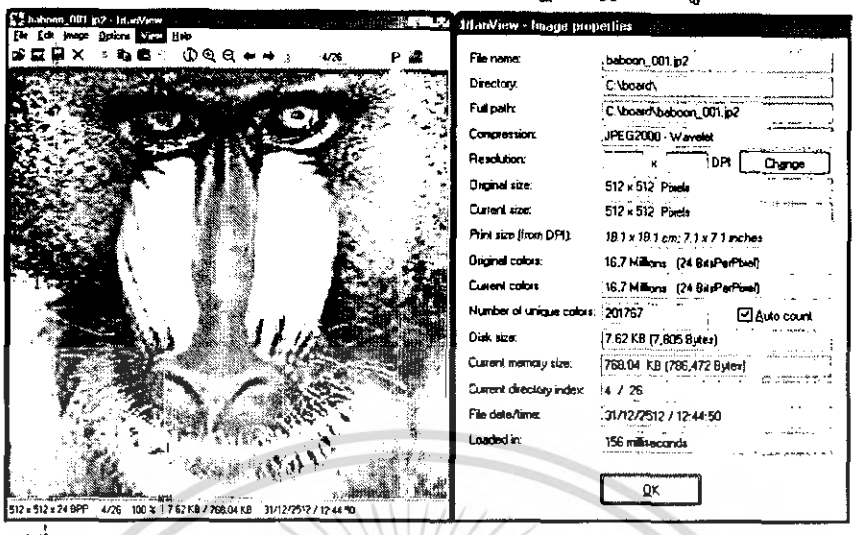
(1) รูป baboon.bmp (rate=0.01)

- ทำการทดลองแปลงไฟล์ baboon.bmp ความละเอียด 512*512 (color) ขนาด 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ baboon_001.jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 14.940 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 10.180 วินาที ดังรูปที่ 4.45



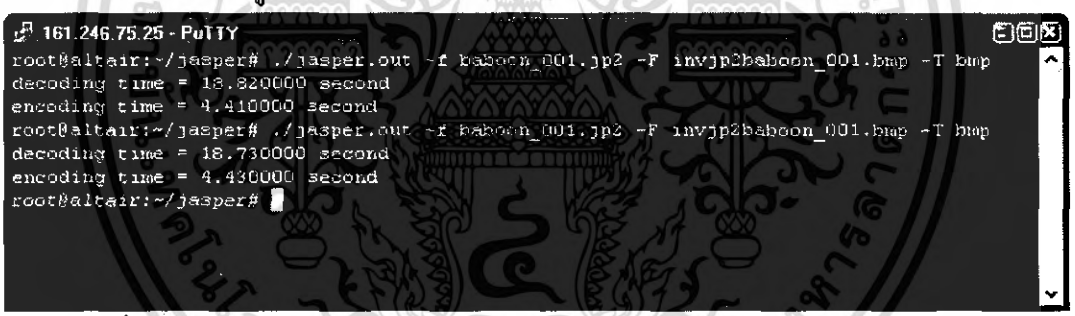
รูปที่ 4.45 แสดงการแปลง ภาพจาก bitmap เป็น JPEG 2000(baboon_001.jp2 rate=0.01)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 7.67 KB (baboon_001.jp2) ดังรูปที่ 4.6



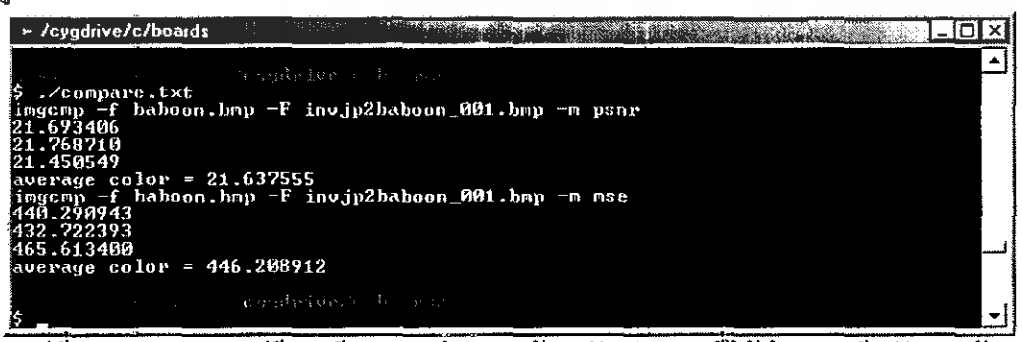
รูปที่ 4.46 แสดงขนาดและคุณสมบัติของภาพ (baboon_001.jp2 rate=0.01)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 18.820 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 4.410 วินาที ดังรูปที่ 4.47



รูปที่ 4.47 แสดงการแปลงกลับเป็น bitmap (invjp2baboon_001.bmp rate=0.01)

- ทำการเปรียบเทียบค่าระหว่าง invjp2baboon_001.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ baboon.bmp ซึ่งเป็นภาพต้นฉบับค่าที่วัดได้คือค่า PSNR = 21.637555 และค่า MSE = 446.208912 ดังรูปที่ 4.48



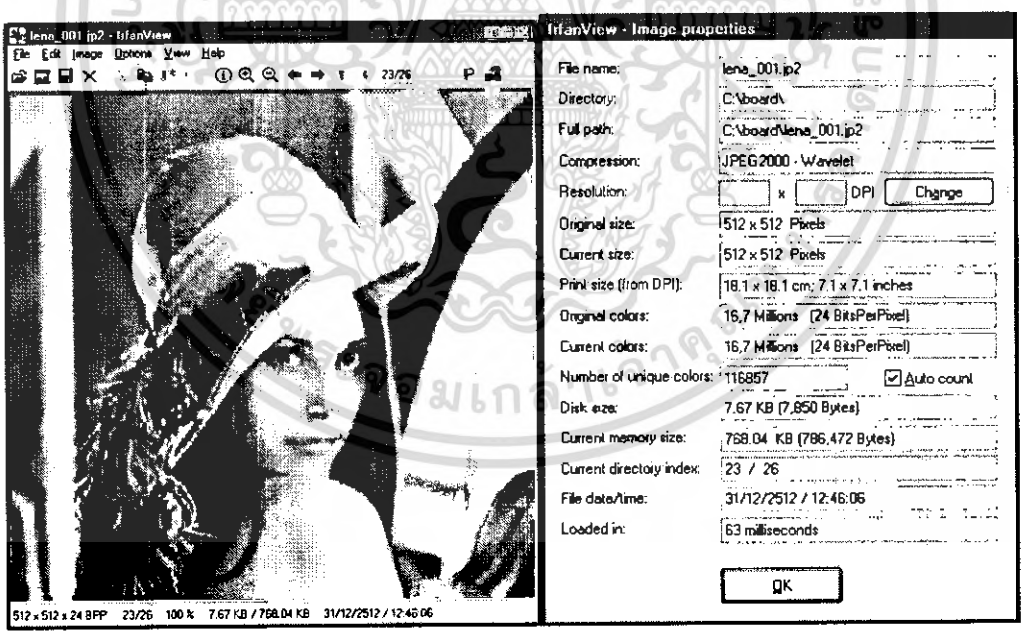
รูปที่ 4.48 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับภาพที่ได้ผ่านการบีบอัดมาแล้ว (invjp2baboon_001.bmp rate=0.01)

(2) รูป lena.bmp (rate=0.01)

- ทำการทดลองแปลงไฟล์ lena.bmp ขนาดภาพ 512*512 (color) ขนาดการเก็บข้อมูล 768 KB แปลงเป็นไฟล์ jpeg 2000 ชื่อ lena_001.jp2 จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ด้นแบบเท่ากับ 14.930 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 10.410 วินาที ดังรูป 4.49

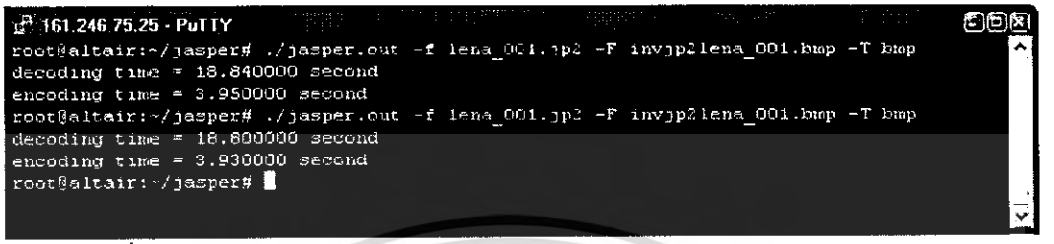
รูปที่ 4.49 แสดงการแปลงภาพจาก bitmap เป็น JPEG 2000(lena_001.jp2 rate=0.01)

- ขนาดภาพที่ได้หลังจากการทำการแปลง 7.67 KB (lena_001.jp2) ดังรูปที่ 4.50



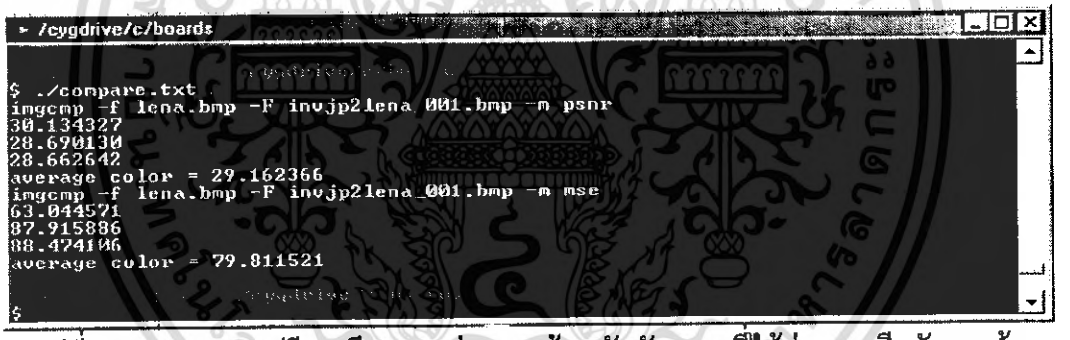
รูปที่ 4.50 แสดงขนาดและคุณสมบัติของภาพ (lena_001.jp2 rate=0.01)

- ทำการตรวจสอบความผิดเพี้ยนที่เกิดขึ้นกับภาพโดยการแปลงกลับให้อยู่ในรูปแบบเดียวกับภาพต้นแบบแล้วทำการเปรียบเทียบกับภาพต้นแบบ จะได้ค่า Decoding Time หรือเวลาที่ใช้ในการอ่านข้อมูลจากไฟล์ต้นแบบเท่ากับ 18.8400 วินาทีและ Encoding Time หรือเวลาที่ใช้ในการเข้ารหัสภาพเท่ากับ 3.9500 วินาที ดังรูปที่ 4.51



รูปที่ 4.51 แสดงการแปลงกลับเป็น bitmap (invjp2lena_001.bmp rate=0.01)

- ทำการเปรียบเทียบค่าระหว่าง invjp2lena_001.bmp ที่เป็นภาพที่ได้ผ่านการแปลงเรียบร้อยแล้วกับ lena.bmp ซึ่งเป็นภาพต้นฉบับ ค่าที่วัดได้คือค่า PSNR = 29.162366 และค่า MSE = 79.811521 ดังรูปที่ 4.52



รูปที่ 4.52 แสดงการเปรียบเทียบระหว่างภาพต้นฉบับกับ ภาพที่ได้ผ่านการบีบอัดมาแล้ว (invjp2lena_001.bmp rate=0.01)

4.3 การวิเคราะห์และสรุปผลที่ได้จากการทดลอง

4.3.1 การวิเคราะห์และสรุปผลความถูกต้องของการเข้ารหัส jpeg 2000 บน PC และบนบอร์ด Altair

ตารางที่ 4.1 การเปรียบเทียบผลความถูกต้องของการเข้ารหัส jpeg 2000 บน PC และ บนบอร์ด Altair

รูปภาพที่ใช้ในการบีบอัด	รูปแบบการบีบอัด	การเข้ารหัส jpeg 2000 บน PC			การเข้ารหัส jpeg 2000 บนบอร์ด Altair		
		ขนาด	PSNR	MSE	ขนาด	PSNR	MSE
 baboon.bmp 512*512 ขนาด 768KB	Loss Less	361.56KB	∞	0.0000000	361.56KB	∞	0.0000000
	rate=0.10	76.64KB	34.763182	22.285511	76.66KB	33.734928	27.737563
	rate=0.01	7.61KB	22.74108	385.976060	7.62KB	21.637555	446.208912
 lena.bmp 512*512 ขนาด 768KB	Loss Less	434.80KB	∞	0.000000	434.80KB	∞	0.0000000
	rate=0.10	76.79KB	37.053967	13.390077	75.98KB	36.648504	14.726606
	rate=0.01	7.67KB	29.929216	66.644400	7.67KB	29.162366	79.811521

จากตารางที่ 4.1 ค่า PSNR และค่า MSE ที่เกิดขึ้นบน PC และบน Altair มีค่าต่างกันเมื่ออยู่ในโหมด Lossy เนื่องจากได้มีการเขียนฟังก์ชั่น pow และได้มีการ floor และ ceil มาใส่เพิ่มเติมจึงทำให้มีผลต่อภาพผลลัพธ์ที่ออกมาเล็กน้อย

4.3.2 การวิเคราะห์และสรุปผลเวลาของการเข้ารหัส jpeg 2000 บน PC และบนบอร์ด Altair

ตารางที่ 4.2 การเปรียบเทียบเวลาเข้ารหัส jpeg 2000 บน PC และ บนบอร์ด Altair

รูปภาพที่ใช้ในการบีบอัด	รูปแบบการบีบอัด	การเข้ารหัส jpeg 2000	
		บน PC	บนบอร์ด Altair
 baboon.bmp	Loss Less	1.421 sec	27.19 sec
	rate=0.10	1.437 sec	25.96 sec
	rate=0.01	1.312 sec	25.12 sec
 lena.bmp	Loss Less	1.469 sec	27.43 sec
	rate=0.10	1.500sec	26.13 sec
	rate=0.01	1.406 sec	25.34 sec

จากตารางที่ 4.2 เนื่องจากการเข้ารหัสบน Altairใช้ CPU XScale PXA255 ซึ่งมีความเร็วสูงสุดที่ 400MHz และไม่มีคำสั่งที่รองรับการทำงานแบบ floating point ทำให้การเข้ารหัสบนAltair ช้ากว่าบน PC

บทที่ 5
สรุปผลการทดลอง

เนื้อหาที่กล่าวไว้ในบทที่ 5 แบ่งออกเป็น 2 ส่วน คือ การสรุปผลการทดลอง และแนวทางในการพัฒนาต่อ

5.1 สรุปผลการทดลอง

การทำการพัฒนาได้ทำการเขียนส่วนของการทำงาน Wavelet Source ด้วยภาษา C++ บน PC แล้วทำการแปลงลงบอร์ด Altair ที่มี Intel X-Scale เป็นระบบประมวลผลมีที่ความเร็ว 400 MHz ขนาดหน่วยความจำ 32 MB แต่ไม่สามารถนำลงไปได้เนื่องจากส่วนของ Library และความสามารถของตัวคอมไพล์ (Compiler) นั้นยังมีความแตกต่างกัน จึงได้ศึกษาและค้นคว้าข้อมูลเพิ่มเติมและได้พบ Source Code เกี่ยวกับการเข้ารหัส JPEG2000 ที่เป็น Open Source จึงได้ทำการทดลองนำลงบอร์ด Altair

ในส่วนการเริ่มการพัฒนาลงบอร์ดนั้น เริ่มจากการศึกษาและทำความเข้าใจ Source Code และตัดส่วนที่ไม่เกี่ยวข้องออก เช่น การรองรับรูปแบบไฟล์ต่างๆ ที่ไม่เกี่ยวข้องกับ JPEG 2000 การแปลงไฟล์ในสกุลต่างๆ นอกจากนามสกุล bitmap ซึ่งถือว่าเป็นขอบเขตที่ต้องการศึกษา ต่อมาจึงทำการคอมไพล์ (Compile) เป็นรูปแบบไฟล์หลักๆ ที่เป็นเสมือนเป็น Library ให้ไฟล์อื่นก่อนโดยคอมไพล์เป็น OBJ ทีละส่วนก่อน หลังจากที่ได้ OBJ ครบทุกส่วนแล้วจึงทำการเชื่อมโยงไฟล์ต่างๆ เพื่อให้ได้ Execute file ออกมาใช้งาน

จากนั้นจึงทำการทดสอบ Execute file ด้วยการนำภาพที่มีความละเอียด 512*512 พิกเซล ขนาด 768 KB มาทำการทดลองบนเครื่องคอมพิวเตอร์ PC และบอร์ด Altair เพื่อวัดความสามารถในการทำงานของ Execute file และได้ทำการเปรียบเทียบประสิทธิภาพระหว่างการเข้ารหัสบนเครื่องคอมพิวเตอร์ PC กับการเข้ารหัสบนบอร์ด Altair ได้ผลลัพธ์ไม่แตกต่างกัน สิ่งที่แตกต่างคือระยะเวลาที่ใช้

เมื่อปรับค่า rate ของการบีบอัดให้มากขึ้น ภาพที่ได้จะมีการผิดเพี้ยนไปจากต้นฉบับมากขึ้นเท่านั้น และในแง่ของประสิทธิภาพ เมื่อเกิด lossyสูญเสีย) ใน rate ที่เท่ากัน เราพบว่าการทำงานบนเครื่องคอมพิวเตอร์ PC สามารถบีบอัตได้เล็กกว่าบนบอร์ด Altair และใช้เวลาน้อยกว่าด้วยเนื่องจาก สูตรของการปรับอัตราส่วนนั้นไม่เท่ากันจากการที่ได้ปรับแก้ไขฟังก์ชั่นบางตัวไปทำให้การปรับ rate การบีบอัดนั้นเท่ากันจริงๆยังไม่ได้จะยังคงมีค่าที่ไม่ตรงกันอย่างเช่น เมื่อปรับการบีบอัดขนาด 10 เท่า(rate= 0.10) บน PC จะได้ค่า ~10 เท่า และบน Altair ก็ได้ ~10 เท่าดังนั้นผลของภาพที่ได้จะมีความแตกต่างกันเล็กน้อยเพราะrate มีค่าไม่เท่ากันจริง ส่วนในโหมด loss less นั้นค่า rate จะมีค่าเป็น 100 % (rate=1) ทั้งใน PC และบน Altair มีค่าเท่ากันผลลัพธ์ที่ได้จึงมีค่าเท่ากัน โดยวัดจากขนาดภาพ ,PSNR และ MSE ที่ได้ตรงกัน

5.2 แนวทางในการพัฒนาต่อ

5.2.1 เพิ่มคุณสมบัติไฟล์ภาพอื่นเช่น Jpeg, Gif, Tiff ฯลฯ

5.2.2 ทดลองพัฒนาการบีบอัดข้อมูลภาพบนซีพียู X-Scale ร่วมกับ FPGA เพื่อความประสิทธิภาพในการทำงานให้เร็วขึ้น

5.2.3 ขยายขอบเขตของการทำงานโดยนำโปรแกรมนี้ไปเป็น Library ให้ทำงานร่วมกับ Application สำเร็จรูปอื่นๆได้อย่างเหมาะสม

เอกสารอ้างอิง

[1]Acharya, T. and Tsai, Ping-Sing. 2005. **JPEG2000 Standard for Image Compression**. The United States of America. New Jersey: Wiley & Sons, Inc.

[2]Adams, M.D. 2002. "**The Jpeg-2000 Still Image Compression Standard**" (Last Revised: 2002-12-25) ISO/IEC jTC 1/SC 29/WG1 N2412. Paper Project, The University of Victoria.

[3]Gonzalez, R.C. and Wood, R.E. 2002. **Digital Image Processing**. 2nd Ed. Printic-Hall, Inc.

[4]_____. **A Really Friendly Guide to Wavelets**. [Online]. Available: http://perso.orange.fr/ polyvalens/clemens/wavelets/wavelets.html#section4

[5]_____. **BMP Image Format**. [Online]. Available: http://local.wasp.uwa.edu.au/~pbourke/ dataformats/bmp/

[6]_____. **JPEG2000 Jasper Software**. [Online]. Available: http://www.ece.uvic.ca/ ~mdadams/jasper/

[7]_____. **The Fast Lifting Wavelet Transform**. [Online]. Available: http://perso.orange.fr/ polyvalens/clemens/lifting/lifting.html