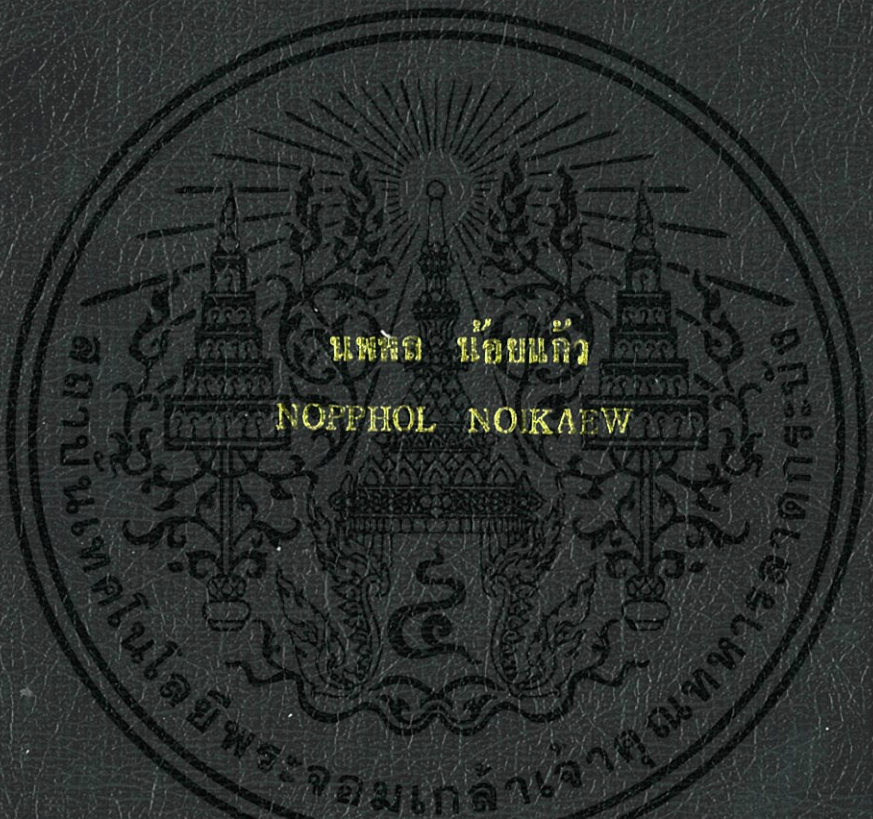
การประมวลผลสัญลักษณ์คู่สำหรับ MQ Arithmetic Coder ใน JPEG2000 บน FPGA

DUAL SYMBOL PROCESSING FOR MQ ARITHMETIC CODER IN JPEG2000 USING FPGA



นพพล น้อยแก้ว

NOPPHOL NOIKAEW

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2552

KMITL - 2009 - EN - M - 070 - 018

การประมวลผลสัญลักษณ์คู่สำหรับ MQ Arithmetic Coder ใน JPEG2000 บน FPGA

DUAL SYMBOL PROCESSING FOR MQ ARITHMETIC CODER IN JPEG2000 USING FPGA

นพพล น้อยแก้ว

NOPPHOL NOIKAEW

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2552

KMITL-2009-EN-M-070-018

DUAL SYMBOL PROCESSING FOR MQ ARITHMETIC CODER IN JPEG2000 USING FPGA

NOPPHOL NOIKAEW

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN COMPUTER ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2009

KMITL-2009-EN-M-070-018

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การประมวลผลสัญลักษณ์คู่สำหรับ MQ Arithmetic Coder ใน JPEG 2000 บน FPGA

Thesis Title Dual Symbol Processing for MQ Arithmetic Coder in JPEG 2000 using FPGA

นักศึกษา นายนพพล น้อยแก้ว

รหัสประจำตัว 47060819

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์ ผศ.ดร.อรฉัตร จิตต์โสภักตร์

หมายเลขวิทยานิพนธ์ KMITL-2009-EN-M-070-018

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.ปกรณ์	วัฒนจตุรพร	
ผศ.ดร.สุรินทร์	กิตติธรกุล	
ดร.ศุภกานต์	พิมลธเรศ	
รศ.ดร.สมศักดิ์	มิตะถา	
ผศ.ดร.อรฉัตร	จิตต์โสภักตร์	

วัน / เดือน/ ปี ที่สอบ วันอังคารที่ 3 กุมภาพันธ์ พ.ศ. 2552 เวลา 09.00-11.00 น.

สถานที่สอบ ณ อาคาร A ชั้น 3 ห้องประชุม 2

คณะวิศวกรรมศาสตร์ รับรองแล้ว

(รองศาสตราจารย์ ดร.กอบชัย เดชหาญ)

คณบดี คณะวิศวกรรมศาสตร์

วันที่ 3 กุมภาพันธ์ พ.ศ. 2552

หัวข้อวิทยานิพนธ์	การประมวลผลสัญลักษณ์คู่สำหรับ MQ Arithmetic Coder ใน JPEG2000 บน FPGA
นักศึกษา	นายนพพล น้อยแก้ว
รหัสนักศึกษา	47060819
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2551
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ.ดร.อรฉัตร จิตต์โสภักตร์

บทคัดย่อ

บทความนี้นำเสนอการออกแบบการประมวลผลสัญลักษณ์คู่สำหรับสถาปัตยกรรมเข้ารหัสเลขคณิต โดยทั่วไปการทำงานของส่วนเข้ารหัสเอ็มคิว จะทำงานแบบลำดับ ซึ่งสามารถประมวลผลได้ครั้งละหนึ่งสัญลักษณ์เท่านั้น อย่างไรก็ตามส่วนเข้ารหัสแบบระนาบบิต ซึ่งเป็นขั้นตอนการเข้ารหัสก่อนหน้าส่วนเข้ารหัสเอ็มคิว สามารถสร้างสัญลักษณ์ได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา ทำให้ส่วนเข้ารหัสเอ็มคิวไม่สามารถประมวลผลได้ทัน ส่งผลให้ความเร็วในการเข้ารหัสสำหรับ JPEG2000 ถูกจำกัดและกลายเป็นปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิว ดังนั้นบทความนี้จึงได้นำเสนอสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวที่ออกแบบให้สามารถประมวลผลได้สองสัญลักษณ์ต่อสัญญาณนาฬิกา ผลสำเร็จที่ได้เกิดจากการนำเสนอขั้นตอนการคาดการณ์และการเลือกค่าที่เหมาะสมของค่าขอบเขตบน และดัชนี ที่ทำให้สามารถประมวลผลสัญลักษณ์ที่สองโดยไม่จำเป็นต้องรอให้ประมวลผลสัญลักษณ์ที่หนึ่งเสร็จก่อน ส่งผลให้สถาปัตยกรรมเข้ารหัสเลขคณิตที่นำเสนอสามารถประมวลผลได้เร็ว และมีประสิทธิภาพการบีบอัดข้อมูลสูง

Thesis Title	Dual symbol processing for MQ Arithmetic Coder in JPEG2000 using FPGA
Student	Mr.Nopphol Noikaew
Student ID	47060819
Degree	Master of Engineering
Programme	Computer Engineering
Year	2008
Thesis Advisor	Asst. Prof. Dr. Orachat Chitsobhuk

ABSTRACT

In this paper, a design of a dual symbol processor for arithmetic coder architecture implemented on FPGA is proposed. Usually, the regular operation of the MQ-Coder is sequential, which can process only one symbol at a time. However, the Bit-Plane Coding can generate more than one symbol per clock cycle. Consequently, the coding speed will be limited and bottlenecked at the interface between the output of the Bit-Plane Coding and the input of the MQ-Coder. Therefore, the proposed MQ-Coder architecture is designed to process two symbols for each clock cycle. The success comes from the proposed prediction process of the upper bound, lower bound, byte out, and index values. This allows the second symbol to be processed without waiting for the end of the process of the first symbol. As a result, the proposed architecture of the arithmetic coder can effectively process dual symbols with faster speed and higher encoding performance.

กิตติกรรมประกาศ

คุณงามความดีและประโยชน์ที่ได้จากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่บิดามารดาอันเป็นที่รักและเคารพยิ่ง ตลอดจนผู้มีพระคุณทุกท่าน

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี ด้วยคำแนะนำจากอาจารย์ที่ปรึกษา ผศ.ดร. อรฉัตร จิตต์โสภักตร์ ที่ให้ความช่วยเหลือ อนุเคราะห์ และให้คำชี้แนะแนวทางในการแก้ปัญหาต่างๆ ตลอดจนให้ความรู้ ประสบการณ์ที่ดี และมีค่าอย่างยิ่งแก่ข้าพเจ้า ข้าพเจ้าจึงขอกราบขอบพระคุณท่านเป็นอย่างยิ่ง พร้อมกันนี้ข้าพเจ้าขอขอบพระคุณครูอาจารย์ที่เคารพทุกท่าน ที่ได้ประสิทธิประศาสน์วิชาความรู้ อีกทั้งถ่ายทอดประสบการณ์อันเป็นประโยชน์แก่ข้าพเจ้า

ขอขอบคุณ เพื่อนๆ พี่น้องนักศึกษาทุกคนที่ให้ความช่วยเหลือ คำแนะนำ และกำลังใจที่ดีตลอดมา

นพพล น้อยแก้ว

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

กิตติกรรมประกาศ III

สารบัญ IV

สารบัญตาราง VIII

สารบัญรูป X

บทที่ 1 บทนำ 1

1.1 ความเป็นมาและความสำคัญของปัญหา 1

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา 2

1.3 สมมุติฐานของการศึกษา 2

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย 3

1.5 ขอบเขตการวิจัย 4

1.6 ขั้นตอนของการศึกษา 4

1.7 เครื่องมือและอุปกรณ์ที่ใช้ในงานวิจัย 5

1.8 โครงสร้างของวิทยานิพนธ์ 5

บทที่ 2 งานวิจัยที่เกี่ยวข้องกับส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์ 6

2.1 มาตรฐาน JPEG2000 6

2.2 ส่วนเข้ารหัสไทเออร์วัน 10

2.3 ปัญหาการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์ 11

2.4 แนวทางแก้ไขการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์ 12

2.4.1 การประมวลผลสัญลักษณ์เดี่ยวสำหรับส่วนเข้ารหัสเอ็มคิวของงานวิจัย [1] ... 12

2.4.1.1 ส่วนที่ 1 13

2.4.1.2 ส่วนที่ 2 14

2.4.1.2.1 กรณีที่ 1 14

2.4.1.2.2 กรณีที่ 2 14

2.4.2 การประมวลผลสัญลักษณ์คู่สำหรับส่วนเข้ารหัสเอ็มคิวของงานวิจัย [2] 16

สารบัญ (ต่อ)

หน้า

บทที่ 3 ความรู้พื้นฐานในเรื่องที่เกี่ยวข้องกับงานวิจัย....21

3.1 การบีบอัดข้อมูล....21

3.1.1 แนวคิดของทฤษฎีข้อมูล....21

3.1.2 แบบจำลองไม่ต่อเนื่องที่ไม่ต้องใช้หน่วยความจำและเอ็นโทรปี....22

3.1.3 แบบจำลองการบีบอัดข้อมูล....23

3.2 การบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิต....24

3.2.1 อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิต....24

3.2.2 อัลกอริธึมสำหรับการถอดรหัสด้วยวิธีการเข้ารหัสเลขคณิต....27

3.3 การเข้ารหัสเลขคณิตแบบไบนารี....28

3.3.1 อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตแบบไบนารี....28

3.3.2 อัลกอริธึมสำหรับการถอดรหัส....31

3.4 การบีบอัดข้อมูลด้วยวิธีส่วนเข้ารหัสเอ็มคิว....32

3.4.1 อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีส่วนเข้ารหัสเอ็มคิว....33

3.4.2 อัลกอริธึมสำหรับการถอดรหัสด้วยวิธีส่วนเข้ารหัสเอ็มคิว....34

บทที่ 4 การออกแบบส่วนเข้ารหัสเอ็มคิวตามมาตรฐาน JPEG2000 สำหรับการใช้งานประเภทเรียลไทม์....36

4.1 การออกแบบส่วนตารางความน่าจะเป็นของส่วนเข้ารหัสเอ็มคิว....37

4.2 การออกแบบส่วนคำนวณค่าขอบเขตบน....40

4.3 การออกแบบส่วนคำนวณค่าขอบเขตล่าง....41

4.3.1 การคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ไม่ทำงาน....46

4.3.2 การคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ทำงาน....48

4.4 การออกแบบส่วนนำส่งผลลัพธ์....58

4.4.1 กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์....62

4.4.2 กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์....66

4.5 ตัวอย่างการคำนวณของสถาปัตยกรรมที่นำเสนอ....69

4.5.1 กรณีค่าดัชนีเปลี่ยนแปลง....69

4.5.2 กรณีนำส่งผลลัพธ์ 1 ไบต์....74

4.4.2.1 กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน....75

สารบัญ (ต่อ)

หน้า

4.5.2.2 กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน 78
4.5.3 กรณีการนำส่งผลลัพธ์ 2 ไบต์.....81

บทที่ 5 ผลการทดลองและการวิเคราะห์.....84
5.1 ขั้นตอนการเตรียมข้อมูลสำหรับการบีบอัดด้วยส่วนเข้ารหัสเอ็มคิว.....85
5.2 ขั้นตอนการทดลองการบีบอัดด้วยส่วนเข้ารหัสเอ็มคิว.....85
5.2.1 ส่วนติดต่ออินพุต.....86
5.2.2 ส่วนเข้ารหัสเอ็มคิว.....87
5.2.3 ส่วนติดต่อเอาต์พุต.....87
5.3 ผลการทดลอง.....87
5.3.1 ผลการทดลองส่วนติดต่ออินพุต.....88
5.3.2 ผลการทดลองส่วนเข้ารหัสเอ็มคิว.....89
5.3.3 ผลการทดลองส่วนส่วนติดต่อเอาต์พุต.....94
5.4 ผลการทดลองวัดประสิทธิภาพในการประมวลผลที่ใช้สำหรับส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว.....95
5.4.1 ผลการทดลองความเร็วในการทำงานของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว.....95
5.4.1.1 สำหรับงานวิจัย [2].....96
5.4.1.2 สำหรับงานวิจัยที่นำเสนอ.....97
5.4.2 ผลการทดลองความเร็วสูงสุดของสัญญาณนาฬิกาที่ส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิวสามารถทำงานได้.....99
5.4.3 ผลการทดลองวัดค่าปริมาณงานของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว.....99
5.4.4 ผลการทดลองทรัพยากรของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว.....100
5.4.5 การวิเคราะห์ประสิทธิภาพของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว.....101

บทที่ 6 สรุปผลการทดลอง.....102
6.1 สรุปและวิเคราะห์ผลการดำเนินงานวิจัย.....102
6.2 แนวทางในการพัฒนาต่อ.....103

สารบัญ (ต่อ)

หน้า

เอกสารอ้างอิง 104

ภาคผนวก 105

ภาคผนวก ก แสดงตารางที่ออกแบบเพิ่มเติมจากตารางมาตรฐาน.....106

ภาคผนวก ข งานวิจัยที่ได้รับการตีพิมพ์.....111

ประวัติผู้เขียน 116

สารบัญตาราง

ตารางที่ หน้า

3.1 แสดงตารางแบบจำลองค่าความน่าจะเป็น.....25

4.1 ตัวอย่างที่ 1 การคำนวณค่า C กรณีจำนวนลัพธ์จากการบีบอัดเท่ากับ 1 ไบต์.....49

4.2 ตัวอย่างที่ 2 การคำนวณค่า C กรณีจำนวนลัพธ์จากการบีบอัดเท่ากับ 1 ไบต์.....49

4.3 ตัวอย่างที่ 1 กรณีไม่เพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์.....54

4.4 ตัวอย่างที่ 2 กรณีไม่เพิ่มบิตผลลัพธ์และกรณีเพิ่มบิตผลลัพธ์.....54

4.5 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Q_E ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง.....71

4.6 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง.....71

4.7 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง.....71

4.8 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง.....71

4.9 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Q_E ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งกระทบผลต่อผลลัพธ์ปัจจุบัน.....75

4.10 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน 75

4.11 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน 75

4.12 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน 76

4.13 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Q_E ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งกระทบผลต่อผลลัพธ์ปัจจุบัน.....78

4.14 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน.....78

4.15 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน.....78

4.16 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน.....78

4.17 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Q_E ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผล กระทบต่อผลลัพธ์ไบต์ที่สอง.....81

4.18 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง.....81

4.19 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง.....81

4.20 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง.....81

สารบัญตาราง (ต่อ)

ตารางที่ หน้า

5.1 แสดงตารางทรัพยากรที่ใช้ในการออกแบบส่วนเข้ารหัสเอ็มคิวส่วนคำนวณ A และ C 93

5.2 แสดงจำนวนสัญญาณนาฬิกาที่ใช้ในการบีบอัดข้อมูล 1 สัญลักษณ์ 96

5.3 แสดงความเร็วสูงสุดของสัญญาณนาฬิกาที่ส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิวสามารถทำงานได้ 99

5.4 แสดงปริมาณงานของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว 100

5.5 แสดงทรัพยากรของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว 101

ก.1 แสดงค่าของตาราง NLPS_QE, ตาราง NMPS_QE, ตาราง LZ_QE, ตาราง NLPS_SWITCII และตาราง NMPS_SWITCH 107

ก.2 แสดงค่าของตาราง NLPS_NLPS, ตาราง NMPS_NLPS, ตาราง NLPS_NMPS และตาราง NMPS_NMPS 109

สารบัญรูป

รูปที่ หน้า

2.1 แสดงโครงสร้างตัวเข้ารหัสตามมาตรฐาน JPEG2000 7
2.2 แสดงตัวอย่างภาพภายหลังแบ่งออกเป็นส่วนเท่าๆ กัน 7
2.3 แสดงตัวอย่างภาพภายหลังการแปลงเวฟเล็ตสำหรับภาพตัดส่วนแต่ละไทล์ 9
2.4 แสดงตัวอย่างภาพภายหลังการแบ่งภาพออกเป็นโค้ดบล็อกย่อยๆ 10
2.5 แสดงโครงสร้างส่วนเข้ารหัสไทเออร์วันตามมาตรฐาน JPEG2000 11
2.6 แสดงสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวของงานวิจัย [1] 13
2.7 แสดงแกนต์ชาร์ตของสถาปัตยกรรมงานวิจัย [1] 15
2.8 แสดงโค้ดเทียมการปรับปรุงค่าขอบเขตบน 15
2.9 แสดงโค้ดเทียมการปรับปรุงค่าขอบเขตล่าง 16
2.10 แสดงสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวของงานวิจัย [2] 17
2.11 แสดงแกนต์ชาร์ตของสถาปัตยกรรมงานวิจัย [2] 19
2.12 แสดงแกนต์ชาร์ตของสถาปัตยกรรมงานวิจัย [2] (ต่อ) 20
3.1 แสดงแบบจำลองการบีบอัดข้อมูล 23
3.2 แสดงอัลกอริธึมสำหรับการบีบอัดข้อมูลของการเข้ารหัสเลขคณิต 25
3.3 แสดงตัวอย่างการบีบอัดข้อมูลของการเข้ารหัสเลขคณิต 27
3.4 แสดงกฎการเข้ารหัสเลขคณิตแบบไบนารี 29
3.5 แสดงตัวอย่างการเข้ารหัสเลขคณิตแบบไบนารี (1) 30
3.6 แสดงตัวอย่างการเข้ารหัสเลขคณิตแบบไบนารี (2) 31
3.7 แสดงแผนผังการทำงานการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิว 33
3.8 แสดงโค้ดเทียมการกำหนดค่าเริ่มต้นของส่วนเข้ารหัสเอ็มคิว 34
3.9 แสดงโค้ดเทียมการบีบอัดข้อมูลแบบเอ็มพีเอสของส่วนเข้ารหัสเอ็มคิว 34
3.10 แสดงโค้ดเทียมการบีบอัดข้อมูลแบบแอลพีเอสของส่วนเข้ารหัสเอ็มคิว 34
3.11 แสดงแผนผังการทำงานการถอดรหัสของส่วนเข้ารหัสเอ็มคิว 35
4.1 แสดงสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวที่นำเสนอ 38
4.2 แสดงแผนผังการทำงานของส่วนคำนวณค่าขอบเขตบน 42
4.3 แสดงสถาปัตยกรรมของส่วนคำนวณค่าขอบเขตบน 42
4.4 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่าขอบเขตบน 43
4.5 แสดงโค้ดเทียมของการทำรีนอมอลไลซ์ 44
4.6 แสดงโค้ดเทียมของส่วนนำส่งผลลัพธ์ 44

สารบัญรูป (ต่อ)

รูปที่ หน้า

4.7 แสดงโค้ดเทียมของฟังก์ชัน BIT_STUFFING 44
4.8 แสดงโค้ดเทียมของฟังก์ชัน NO_BIT_STUFFING....45
4.9 แสดงสถาปัตยกรรมของส่วนคำนวณค่าขอบเขตล่าง 45
4.10 แสดงแผนผังการทำงานของส่วนคำนวณค่าขอบเขตล่าง 47
4.11 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่าขอบเขตล่าง 47
4.12 แสดงโค้ดเทียมกรณีไม่เพิ่มบิตผลลัพธ์ของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์....53
4.13แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณีจำนวนผลลัพธ์เท่ากับ “2” ไบต์ 56
4.14 แสดงโค้ดเทียมการเลือกค่า C กรณีไม่เพิ่มบิตผลลัพธ์ของรูปที่ 4.11....57
4.15 แสดงโค้ดเทียมการเลือกค่า C กรณีกรณีเพิ่มบิตผลลัพธ์ของรูปที่ 4.13....57
4.16 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณี P1 และ $P3_1$....58
4.17 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณี P2....58
4.18 แสดงโค้ดเทียมใหม่ของส่วนนำส่งผลลัพธ์....59
4.19 แสดงโครงสร้างการทำงานของส่วนนำส่งผลลัพธ์....60
4.20 แสดงการเลื่อนบิตของผลลัพธ์ในค่า C....61
4.21 แสดงโค้ดเทียมการตรวจสอบผลลัพธ์ล่าสุด โดยใช้แฟล็กชื่อ “TWOBYTE”....62
4.22 แสดงโค้ดเทียมการเลือกผลลัพธ์ของฟังก์ชั่น BIT_STUFFING_1....64
4.23 แสดงโค้ดเทียมการเลือกค่าบิตทด....65
4.24 แสดงโค้ดเทียมการเลือกผลลัพธ์ไบต์ที่หนึ่งกรณีกรณีเพิ่มบิตผลลัพธ์และค่า B = 0xFF....67
4.25 แสดงโค้ดเทียมการเลือกผลลัพธ์ไบต์ที่สองกรณีกรณีเพิ่มบิตผลลัพธ์และค่า B = 0xFF....68
4.26 แสดงโค้ดเทียมการเลือกผลลัพธ์ไบต์ที่สองกรณีไม่เพิ่มบิตผลลัพธ์....70
4.27 แสดงการเปลี่ยนแปลงของค่า A จากตัวอย่างกรณีค่าดัชนีเปลี่ยนแปลง....73
4.28 แสดงการเปลี่ยนแปลงของค่า C จากตัวอย่างกรณีค่าดัชนีเปลี่ยนแปลง....74
4.29 แสดงการเปลี่ยนแปลงของค่า A จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าไม่ส่งกระทบผลต่อผลลัพธ์ปัจจุบัน....77
4.30 แสดงการเปลี่ยนแปลงของค่า C จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าไม่ส่งกระทบผลต่อผลลัพธ์ปัจจุบัน....77

สารบัญรูป (ต่อ)

รูปที่ หน้า

4.31 แสดงการเปลี่ยนแปลงของค่า A จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าส่งกระทบผลต่อผลลัพธ์ปัจจุบัน 80
4.32 แสดงการเปลี่ยนแปลงของค่า C จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าส่งกระทบผลต่อผลลัพธ์ปัจจุบัน 80
4.33 แสดงการเปลี่ยนแปลงของค่า A กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง 83
4.34 แสดงการเปลี่ยนแปลงของค่า C กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง.83
5.1 แสดงตัวอย่างการสร้างไฟล์อินพุตและไฟล์เอาต์พุตด้วยโปรแกรม JJ2000....85
5.2 แสดงสถาปัตยกรรมสำหรับการทดลองส่วนเข้ารหัสเอ็มคิว 86
5.3 แสดงสถาปัตยกรรมส่วนติดต่ออินพุต 86
5.4 แสดงตัวอย่างการทำงานของส่วนติดต่ออินพุต 89
5.5 แสดงผลลัพธ์การจำลองการทำงานของส่วนติดต่ออินพุต....89
5.6 แสดงวงจรเบื้องต้นสำหรับการอ่านค่าสัญลักษณ์จากส่วนติดต่ออินพุต 90
5.7 แสดงผลการทดลองการอ่านค่าสัญลักษณ์จากส่วนติดต่ออินพุต....91
5.8 แสดงผลลัพธ์การจำลองการทำงานส่วนคำนวณขอบเขตบนของส่วนเข้ารหัสเอ็มคิว....92
5.9 แสดงผลลัพธ์การจำลองการทำงานส่วนคำนวณขอบเขตล่างของส่วนเข้ารหัสเอ็มคิว 93
5.10 แสดงขั้นตอนการทดลองส่วนส่วนติดต่อเอาต์พุต....94
5.11 แสดงตัวอย่างไฟล์ตั้งต้นและไฟล์ทดสอบ....94
5.12 แสดงตัวอย่างข้อผิดพลาดจากการทดสอบ 94

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

จากความสำเร็จของมาตรฐาน JPEG ในปี ค.ศ. 1990 ทำให้แอพพลิเคชั่นต่างๆ นำมาตรฐาน JPEG ไปประยุกต์ใช้งานอย่างกว้างขวาง เช่น แอพพลิเคชั่นที่เกี่ยวข้องกับรูปภาพ มัลติมีเดีย และอินเตอร์เน็ต เป็นต้น ต่อมามีความต้องการเพิ่มความสามารถให้กับมาตรฐาน JPEG ซึ่งไม่สามารถตอบสนองความต้องการใหม่ๆ ได้ ทำให้เกิดการพัฒนามาตรฐาน JPEG ขึ้นใหม่เป็นมาตรฐาน JPEG2000

มาตรฐาน JPEG2000 ถูกนำเสนอในปี ค.ศ. 2001 มาตรฐานใหม่นี้สามารถใช้งานกับรูปภาพได้หลากหลายชนิด ได้แก่ ภาพสองระดับ ภาพระดับสีเทา ภาพองค์ประกอบหลายระดับ และภาพสี เป็นต้น หลากหลายคุณสมบัติ ได้แก่ ภาพธรรมชาติ ภาพทางวิทยาศาสตร์ ภาพทางการแพทย์ และภาพตัวอักษร เป็นต้น และหลากหลายการใช้งาน ได้แก่ ไคลเอนต์/เซิร์ฟเวอร์ การรับส่งแบบเรียลไทม์ เป็นต้น จุดเด่นของมาตรฐาน JPEG2000 เช่น สามารถบีบอัดข้อมูลภาพด้วยอัตราส่วนการบิดเบือน (Rate Distortion) ที่อัตราการบีบอัดต่ำๆ ได้ โดยยังคงคุณภาพใกล้เคียงกับต้นฉบับเดิม สามารถบีบอัดข้อมูลภาพได้ทั้งแบบสูญเสีย (Lossy) และแบบไม่สูญเสีย (Lossless) ทนทานต่อความผิดพลาดที่เกิดจากการสื่อสาร และมีความปลอดภัยสูง เป็นต้น

เนื่องจากมาตรฐาน JPEG2000 เป็นมาตรฐานใหม่ มีฟังก์ชั่นการทำงานที่หลากหลายและหลายฟังก์ชั่นมีหลักการทำงานที่ซับซ้อน ทำให้ใช้เวลาในการประมวลผลมากขึ้น เมื่อเทียบกับมาตรฐาน JPEG เดิม ดังนั้นการนำมาตรฐาน JPEG2000 ไปใช้ในงานประเภทเรียลไทม์ ยังคงมีข้อจำกัดในเรื่องของเวลา ส่วนเข้ารหัสไทเออร์วัน (Tier-1 Encoder) ซึ่งเป็นส่วนบีบอัดข้อมูลตามมาตรฐาน JPEG2000 เป็นส่วนที่ใช้เวลาในการประมวลผลมากที่สุด

ส่วนเข้ารหัสไทเออร์วันประกอบด้วยส่วนเข้ารหัสแบบระนาบบิต (Bit-Plane Coding) ทำหน้าที่สร้างสัญลักษณ์ (Symbol) และส่วนเข้ารหัสเอ็มคิว (MQ-Coder) ทำหน้าที่บีบอัดข้อมูล จากงานวิจัย [3] เป็นงานวิจัยส่วนเข้ารหัสแบบระนาบบิตสามารถสร้างสัญลักษณ์ได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา ทำให้เกิดปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิว

จากปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิวดังที่ได้กล่าวมา ความเร็วในการประมวลผลของส่วนเข้ารหัสเอ็มคิวถูกจำกัดเนื่องจากการทำงานของส่วนเข้ารหัสเอ็มคิวเป็นแบบลำดับขั้น (Sequential) ทำให้สามารถประมวลผลได้ครั้งละหนึ่งสัญลักษณ์เท่านั้น ดังนั้นการออกแบบให้ส่วนเข้ารหัสเอ็มคิวทำงานแบบ

ขนาน (Parallel) จึงทำให้สามารถประมวลผลได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา จะช่วยลดเวลาในการประมวลผลและลดปัญหาคอขวดลงได้ สำหรับการออกแบบต้องมีส่วนคำนวณหรือส่วนคาดเดาผลลัพธ์ล่วงหน้า เพื่อคงความถูกต้องให้กับส่วนเข้ารหัสเอ็มคิว

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1.2.1 เพื่อศึกษาการทำงานของมาตรฐาน JPEG2000

1.2.2 เพื่อศึกษาการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิว

1.2.3 เพื่อลดเวลาที่ใช้ในการประมวลผลของส่วนเข้ารหัสเอ็มคิว

1.2.4 เพื่อลดปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิว

1.2.5 เพื่อปรับการทำงานของส่วนเข้ารหัสเอ็มคิวจากลำดับขั้นเป็นขนาน

1.3 สมมุติฐานของการศึกษา

ส่วนเข้ารหัสเอ็มคิวถือว่าเป็นส่วนสำคัญส่วนหนึ่งตามมาตรฐาน JPEG2000 โดยการทำงานของส่วนเข้ารหัสเอ็มคิวมีความซับซ้อนทำให้ใช้เวลาในการประมวลผลมาก เนื่องจากส่วนเข้ารหัสเอ็มคิวมีการทำงานแบบลำดับขั้นจึงมีขั้นตอนการทำงานเรียงตามลำดับกันไป ซึ่งข้อมูลที่ส่งผ่านแต่ละขั้นตอนมีความสัมพันธ์ขึ้นต่อกัน ทำให้สามารถประมวลผลทำได้เพียงครั้งละหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกาเท่านั้น ดังนั้นการออกแบบให้ส่วนเข้ารหัสเอ็มคิวทำงานแบบขนาน น่าจะสามารถปรับปรุงเวลาที่ใช้ในประมวลผลให้รวดเร็วขึ้น เนื่องจากสามารถประมวลผลได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา จึงเป็นแนวทางหนึ่งในการเพิ่มประสิทธิภาพของส่วนเข้ารหัสเอ็มคิวซึ่งผลที่ตามมาจะช่วยลดเวลาที่ใช้ในการประมวลผลลงได้ สำหรับการออกแบบให้ส่วนเข้ารหัสเอ็มคิวสามารถประมวลผลได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกาจำเป็นต้องอาศัยเทคนิคต่างๆ เข้าช่วยในการออกแบบ เช่น การคำนวณหรือคาดเดาผลลัพธ์ล่วงหน้า โดยผู้ออกแบบจำเป็นต้องทำการศึกษารูปแบบของผลลัพธ์ทั้งหมดที่สามารถเกิดขึ้นได้ จากนั้นทำการหาความสัมพันธ์ของผลลัพธ์รูปแบบต่างๆ เพื่อนำมาวิเคราะห์หาว่าส่วนใดสามารถคำนวณหรือคาดเดาผลลัพธ์ล่วงหน้าได้ ถ้าพบว่าส่วนใดสามารถคำนวณผลลัพธ์ล่วงหน้าได้ก็ให้ทำการคำนวณผลลัพธ์ล่วงหน้า แล้วเก็บผลลัพธ์ไว้ในรูปแบบที่เหมาะสมสำหรับการใช้งานต่อไป และถ้าพบว่าส่วนใดสามารถคาดเดาผลลัพธ์ล่วงหน้าได้ก็ให้ทำการออกแบบส่วนคาดเดาผลลัพธ์ล่วงหน้า ทำให้การทำงานในส่วนถัดไปสามารถเลือกผลลัพธ์จากการคาดเดาล่วงหน้าได้รวดเร็วยิ่งขึ้น

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย

เนื่องจากงานวิจัยเดิมการทำงานในของส่วนเข้ารหัสเอ็มคิวนั้นมีการทำงานแบบลำดับขั้น คือ มีขั้นตอนการทำงานเรียงตามลำดับต่อๆ กันไป ซึ่งข้อมูลที่ส่งผ่านในแต่ละขั้นตอนมีความสัมพันธ์ขึ้นต่อกัน ทำให้สามารถประมวลผลได้ครั้งละหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกาเท่านั้น

จากงานวิจัย [1] ได้ทำการปรับปรุงแก้ไขในขั้นตอนการทำรีนอร์มอลไลซ์ (Renormalize) โดยทำการคำนวณจำนวนรอบของการทำรีนอร์มอลไลซ์ เพื่อลดเวลาในการวนซ้ำของการทำรีนอร์มอลไลซ์ ทำให้สามารถลดเวลาในการคำนวณได้มาก

จากงานวิจัย [2] ได้ทำการปรับปรุงแก้ไขเพิ่มเติมจากงานวิจัย [1] โดยปรับรูปแบบการทำงานแบบลำดับขั้นเป็นแบบขนาน เพื่อให้สามารถประมวลผลได้สองสัญลักษณ์ต่อสัญญาณนาฬิกา ทำให้สามารถลดเวลาในการคำนวณได้มากกว่างานวิจัยเดิม แต่อย่างไรก็ดี ถึงแม้งานวิจัย [2] จะพยายามออกแบบการประมวลผลเป็นแบบขนาน แต่ยังคงมีส่วนที่เป็นคอขวด คือ ส่วนการคำนวณขอบเขตบน (A) และล่าง (C) ที่ยังคงมีการประมวลผลเป็นแบบลำดับขั้นอยู่ ทำให้การคำนวณในส่วนเข้ารหัสเอ็มคิวยังไม่สามารถเป็นแบบขนาน ได้เต็มที่ทั้งหมด ทำให้เวลาในการประมวลผลยังคงสูงอยู่

ดังนั้นในงานวิจัยที่นำเสนอนี้จะทำการปรับปรุงแก้ไขเพิ่มเติมจากงานวิจัย [2] โดยเพิ่มส่วนคำนวณหรือส่วนคาดเดาผลลัพธ์ล่วงหน้า เพื่อลดขั้นตอนที่เป็นลำดับขั้นในการคำนวณขอบเขตบนและล่าง ทำให้ช่วยลดเวลาที่ใช้ในการประมวลผลลงได้ โดยเริ่มจากการศึกษามาตรฐาน JPEG2000 เพื่อให้เข้าใจข้อดีข้อด้อยและขั้นตอนการทำงานทั้งหมดของ JPEG2000 จากนั้นทำการศึกษาการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวเพื่อให้เข้าใจขั้นตอนการทำงาน และสามารถวิเคราะห์ได้ว่าการทำงานส่วนใดของส่วนเข้ารหัสเอ็มคิวที่ใช้เวลาในการประมวลผลมาก จากนั้นทำการศึกษาแนวคิดในการออกแบบการทำงานแบบขนาน เพื่อนำไปประยุกต์ใช้ในการออกแบบส่วนเข้ารหัสเอ็มคิวจากการทำงานแบบลำดับขั้นเป็นแบบขนาน จากนั้นทำการศึกษาการทำงานเบื้องต้นของเอฟพีจีเอ(FPGA) เพื่อให้เข้าใจโครงสร้างพื้นฐานและการทำงานของเอฟพีจีเอจากนั้นทำการศึกษาทฤษฎีการออกแบบวงจรดิจิตอลบนเอฟพีจีเอเพื่อนำไปใช้ในการออกแบบวงจรดิจิตอลบนเอฟพีจีเอโดยส่วนหลักในการออกแบบการประมวลผลในส่วนเข้ารหัสเอ็มคิวนี้จะอยู่ในส่วนการคำนวณขอบเขตบนและล่าง รวมทั้งขั้นตอนการส่งผลลัพธ์การเข้ารหัสออกจากส่วนเข้ารหัสเอ็มคิวไปยังส่วนเข้ารหัสไทเออร์ทู (Tier-2 Encoder) ด้วย โดยการออกแบบจะนำผลจากการศึกษาความสัมพันธ์ของผลลัพธ์รูปแบบต่างๆ เพื่อนำมาวิเคราะห์และออกแบบส่วนคำนวณหรือคาดเดาผลลัพธ์ล่วงหน้า ทั้งในส่วนการคาดการณ์ผลลัพธ์ของขอบเขตบน ขอบเขตล่าง และส่วนการส่งผลลัพธ์ออกจากส่วนเข้ารหัสเอ็มคิวซึ่งคาดว่าด้วยโครงสร้างการคาดการณ์ผลลัพธ์ล่วงหน้าในแต่ละ

ส่วนที่ออกแบบในงานวิจัยนี้ จะช่วยให้เวลาที่ใช้ในการประมวลผลจะรวดเร็วขึ้นกว่างานวิจัย [1] และ [2]

1.5 ขอบเขตการวิจัย

งานวิจัยที่นำเสนอจะทำการศึกษาและดำเนินการวิจัย เพื่อออกแบบส่วนเข้ารหัสเอ็มคิวบนอุปกรณ์ประเภทเอฟพีจีเอให้สามารถลดเวลาที่ใช้ในการประมวลผลจากงานวิจัย [1] และ [2] ลงได้ โดยข้อมูลที่ใช้ในการทดลองจะได้จากการนำภาพชื่อ “peppers.jpg” ขนาด 512 × 512 พิกเซล นำไปผ่านกระบวนการตามมาตรฐาน JPEG2000 ด้วยโปรแกรม JJ2000 ซึ่งโปรแกรม JJ2000 นี้ถูกสร้างขึ้นเพื่อใช้สำหรับทดลองการทำงานตามมาตรฐาน JPEG2000 ผลลัพธ์จากโปรแกรม JJ2000 จะได้ไฟล์อินพุตและไฟล์เอาต์พุต โดยไฟล์อินพุตจะใช้เป็นอินพุตสำหรับการบีบอัด ส่วนไฟล์เอาต์พุตจะใช้สำหรับตรวจสอบความถูกต้องของการทำงาน สำหรับการเปรียบเทียบกับงานวิจัยก่อนหน้า จะทำการเปรียบเทียบจากจำนวนสัญลักษณ์ที่สามารถบีบอัดได้ต่อวินาที เวลาที่ใช้ในการประมวลผลและความเร็วสูงสุดที่สามารถทำงานได้ โดยในงานวิจัยนี้จะทำการออกแบบส่วนเข้ารหัสเอ็มคิวสำหรับทำการเข้ารหัสได้สองสัญลักษณ์ต่อสัญญาณนาฬิกาเท่านั้น อย่างไรก็ดีแนวทางการคาดการณ์ผลลัพธ์ล่วงหน้าที่ได้ศึกษาและออกแบบในงานวิจัยนี้ สามารถเป็นแนวทางในการพัฒนาการออกแบบส่วนเข้ารหัสเอ็มคิวสำหรับการเข้ารหัสมากกว่าสองสัญลักษณ์ต่อสัญญาณนาฬิกาต่อไปได้

1.6 ขั้นตอนของการศึกษา

1.6.1 กำหนดวัตถุประสงค์และขอบเขตของงานวิจัย

1.6.2 ศึกษาค้นคว้าทฤษฎีและงานวิจัยที่เกี่ยวข้องกับการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวสำหรับการนำไปใช้งานประเภทเรียลไทม์ แล้ววิเคราะห์ข้อดี ข้อด้อย และประเด็นที่น่าสนใจของงานวิจัยนั้นๆ แล้วนำมาปรับปรุงหรือประยุกต์ใช้กับงานวิจัยนี้ได้

1.6.3 ตั้งสมมุติฐานของการศึกษาและกำหนดวางแนวความคิดของงานวิจัย โดยมีการอ้างอิงทฤษฎีหรือหลักการที่เกี่ยวข้องเพื่อที่จะสามารถบรรลุตามวัตถุประสงค์ที่ได้กำหนดไว้

1.6.4 เตรียมข้อมูลสำหรับการบีบอัดด้วยส่วนเข้ารหัสเอ็มคิวเพื่อนำมาใช้สำหรับการทดลอง

1.6.5 ออกแบบและพัฒนาสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวตามแนวคิดหรืออัลกอริธึมที่งานวิจัยนี้ได้นำเสนอ

1.6.6 ทำการทดลองบีบอัดข้อมูลตามสถาปัตยกรรมที่ได้ออกแบบ โดยในขณะทำการทดลองนั้นจะเก็บข้อมูลและผลลัพธ์ของทุกๆ ขั้นตอนเอาไว้ เพื่อนำมาวิเคราะห์ปรับปรุงงานวิจัยต่อไป

1.6.7 นำผลลัพธ์จากการทดลองมาวิเคราะห์และประเมินงานวิจัย ทั้งในแง่ของคุณภาพและประสิทธิภาพ แล้วสรุปผลเพื่อนำเสนอผลงานวิจัย

1.7 เครื่องมือและอุปกรณ์ที่ใช้ในงานวิจัย

เครื่องมือและอุปกรณ์ที่ใช้ในการวิจัยในครั้งนี้ ได้แก่

1.7.1 เครื่องคอมพิวเตอร์แบบพกพาใช้หน่วยประมวลผลกลาง Intel Centrino Processor 1.5 GHz หน่วยความจำ (RAM) 768 MBytes จำนวน 1 เครื่อง

1.7.2 ระบบปฏิบัติการวินโดวส์ XP service pack 2

1.7.3 โปรแกรม Xilinx ISE เวอร์ชั่น 8.2i

1.7.4 โปรแกรม ModelSim เวอร์ชั่น 6.2b

1.7.5 โปรแกรม JJ2000 เวอร์ชั่น 1.4

1.8 โครงสร้างของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้แบ่งออกเป็น 6 บท แต่ละบทประกอบด้วยเนื้อหาดังต่อไปนี้

บทที่ 1 กล่าวถึงความเป็นมาและความสำคัญของปัญหา ความมุ่งหมายและวัตถุประสงค์ของการศึกษา สมมุติฐานของการศึกษา รวมทั้งทฤษฎีหรือแนวคิดที่ใช้ในการศึกษา ขอบเขตของการศึกษา และขั้นตอนของการศึกษา

บทที่ 2 กล่าวถึงความรู้พื้นฐานเกี่ยวกับ JPEG2000 และส่วนเข้ารหัสไทเออร์วันปัญหาการบีบอัดข้อมูลด้วยส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์ และงานวิจัยที่เกี่ยวกับการบีบอัดข้อมูลด้วยส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์

บทที่ 3 กล่าวถึงความรู้พื้นฐานที่เกี่ยวข้องกับงานวิจัย ได้แก่ ทฤษฎีการบีบอัดข้อมูล การบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิต การบีบอัดด้วยวิธีการเข้ารหัสเลขคณิตแบบไบนารี (Binary Arithmetic Coding) การบีบอัดข้อมูลด้วยวิธีส่วนเข้ารหัสเอ็มคิว

บทที่ 4 กล่าวถึงการออกแบบส่วนเข้ารหัสเอ็มคิวตามมาตรฐาน JPEG2000 สำหรับการใช้งานประเภทเรียลไทม์ การออกแบบส่วนตารางข้อมูล การออกแบบส่วนคำนวณค่าขอบเขตบน การออกแบบส่วนคำนวณค่าขอบเขตล่าง และการออกแบบส่วนจัดการเอาต์พุต

บทที่ 5 ผลการทดลองและการวิเคราะห์

บทที่ 6 สรุปผลการทดลอง ข้อเสนอแนะ และแนวทางในการพัฒนาออกแบบส่วนเข้ารหัสเอ็มคิวต่อไปในอนาคต

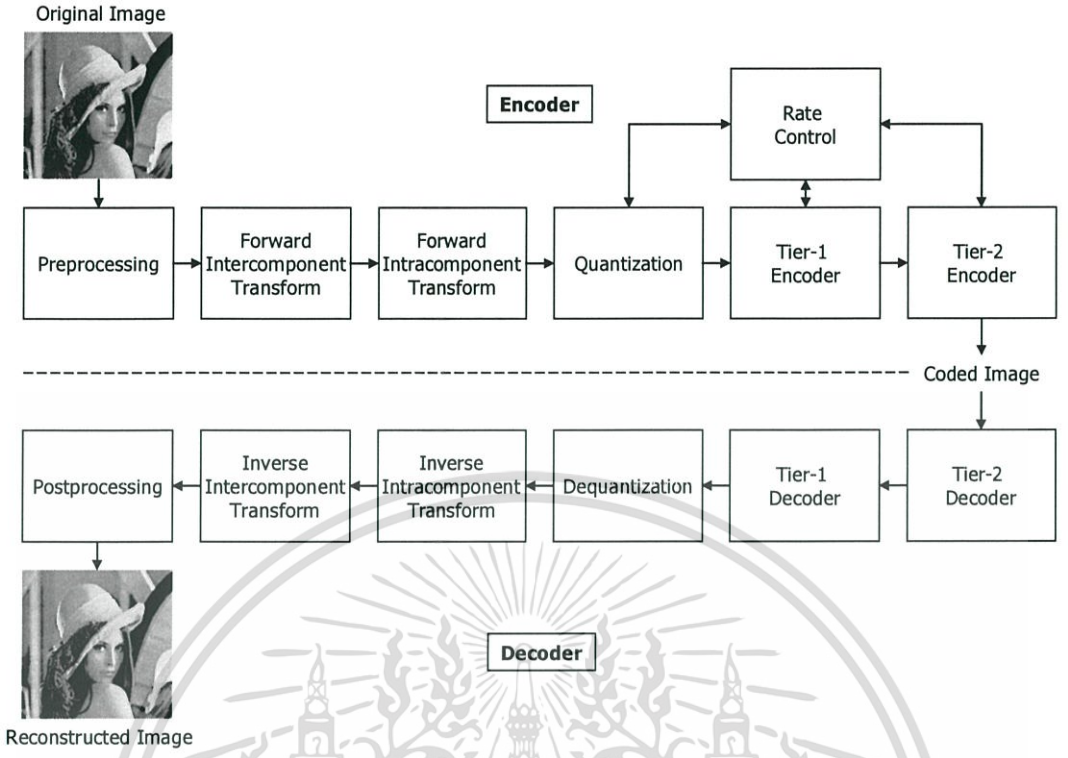
บทที่ 2
งานวิจัยที่เกี่ยวข้องกับส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์

จากความสำเร็จของมาตรฐาน JPEG ในปี ค.ศ. 1990 ทำให้แอพพลิเคชั่นต่างๆ นำมาตรฐาน JPEG ไปประยุกต์ใช้งานอย่างกว้างขวาง เช่น แอพพลิเคชั่นที่เกี่ยวข้องกับรูปภาพ มัลติมีเดีย และอินเตอร์เน็ต เป็นต้น ต่อมามีความต้องการเพิ่มความสามารถให้กับมาตรฐาน JPEG ซึ่งไม่สามารถตอบสนองความต้องการใหม่ๆ ได้ ทำให้เกิดการพัฒนามาตรฐาน JPEG ขึ้นใหม่เป็นมาตรฐาน JPEG2000

2.1 มาตรฐาน JPEG2000

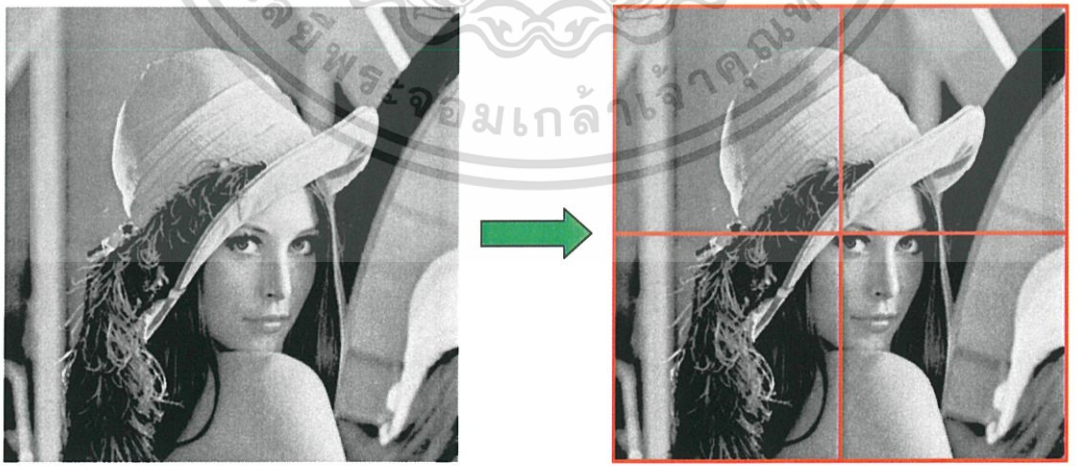
มาตรฐาน JPEG2000 ถูกนำเสนอในปี ค.ศ. 2001 โดยกลุ่ม ISO/IEC JTC1/SC29/WG1 เรียกว่า "กลุ่ม JPEG" มาตรฐานใหม่นี้สามารถใช้งานกับรูปภาพได้หลากหลายชนิด ได้แก่ ภาพสองระดับ ภาพระดับสีเทา ภาพองค์ประกอบหลายระดับ และภาพสี เป็นต้น ภาพหลากหลายคุณสมบัติ ได้แก่ ภาพธรรมชาติ ภาพทางวิทยาศาสตร์ ภาพทางการแพทย์ และภาพตัวอักษร เป็นต้น นอกจากนี้ยังสามารถประยุกต์ใช้งานได้หลากหลายด้านด้วยกัน ได้แก่ ไคลเอนต์/เซิร์ฟเวอร์ การรับส่งข้อมูลแบบเรียลไทม์ เป็นต้น จุดเด่นของมาตรฐาน JPEG2000 เช่น สามารถบีบอัดข้อมูลภาพด้วยอัตราส่วนการบิดเบือนที่อัตราการบีบอัดต่ำได้ โดยยังคงคุณภาพใกล้เคียงกับต้นฉบับเดิม สามารถบีบอัดข้อมูลภาพได้ทั้งแบบสูญเสีย และแบบไม่สูญเสีย ทนทานต่อการผิดพลาดจากสัญญาณรบกวนที่เกิดจากการสื่อสาร มีความปลอดภัยสูง เป็นต้น

โครงสร้างของตัวเข้ารหัสตามมาตรฐาน JPEG2000 ประกอบด้วยส่วนเข้ารหัส (Encoder) และส่วนถอดรหัส (Decoder) เนื่องจากส่วนถอดรหัสมีหลักการทำงานตรงข้ามกับส่วนเข้ารหัส ยกเว้นส่วนของการควบคุมอัตราการบีบอัด (Rate Control) ซึ่งไม่มีในส่วนถอดรหัส จึงขออธิบายเฉพาะส่วนเข้ารหัส ดังนี้



รูปที่ 2.1 แสดงโครงสร้างตัวเข้ารหัสตามมาตรฐาน JPEG2000

1) ส่วนเตรียมการก่อนการประมวลผล (Preprocessing) ทำหน้าที่แบ่งภาพออกเป็นส่วนเท่าๆ กัน (Tiling) แต่ละส่วนเรียกว่า "ไทล์ (Tile)" จากนั้นทำการปรับค่าสัมประสิทธิ์ข้อมูลให้เป็นแบบไม่คิดเครื่องหมาย (unsigned)



รูปที่ 2.2 แสดงตัวอย่างภาพภายหลังแบ่งออกเป็นส่วนเท่าๆ กัน

2) ส่วนฟอร์เวิร์ดอินเตอร์คอมโพเนนท์ทรานส์ฟอร์ม (Forward Intercomponent Transform) ทำหน้าที่แปลงองค์ประกอบจากแบบจำลองสีจากอาร์จีบี (RGB) เป็นแบบจำลองสีวายซีบีซีอาร์ (YCbCr) การแปลงองค์ประกอบสีสำหรับภาพที่ทำการบีบอัดแบบสูญเสียจะใช้เทคนิคการแปลงค่าสีแบบแปลงกลับไม่ได้ (Irreversible Color Transform : ICT) ส่วนการแปลงองค์ประกอบสีสำหรับภาพที่ทำการบีบอัดแบบไม่สูญเสียจะใช้เทคนิคการแปลงค่าสีแบบแปลงกลับได้ (Reversible Color Transform : RCT) ดังแสดงในสมการต่อไปนี้

Forward RCT:
$$Y_r = \left\lfloor \frac{R+2G+B}{4} \right\rfloor \tag{2.1}$$
$$U_r = B-G \tag{2.2}$$
$$V_r = R-G \tag{2.3}$$

Inverse RCT:
$$G = Y_r - \left\lfloor \frac{U_r+V_r}{4} \right\rfloor \tag{2.4}$$
$$R = V_r + G \tag{2.5}$$
$$B = U_r + G \tag{2.6}$$

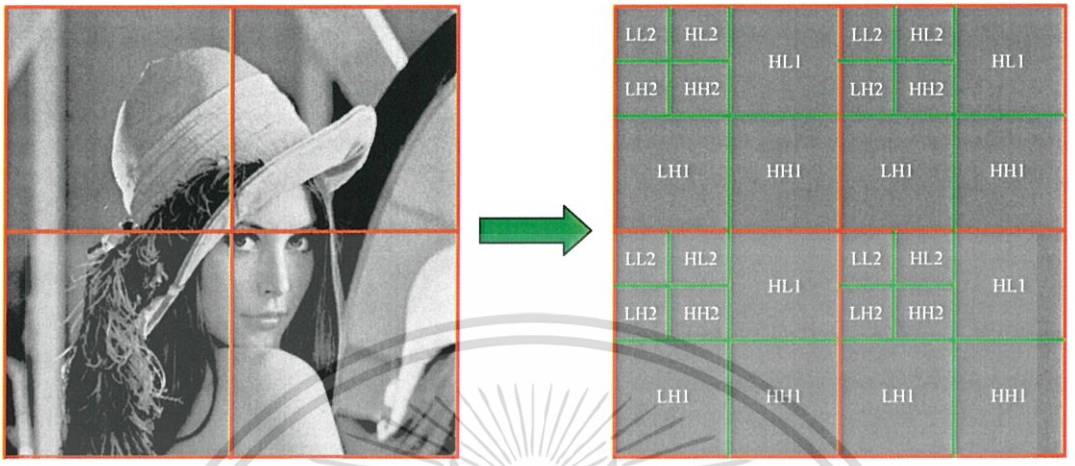
Forward ICT:
$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299000 & 0.587000 & 0.114000 \\ -0.168736 & -0.331264 & 0.500000 \\ 0.500000 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.7}$$

Inverse ICT:
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.000000 & 1.402000 \\ 1.0 & -0.344136 & -0.714136 \\ 1.0 & 1.772000 & 0.000000 \end{bmatrix} \times \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \tag{2.8}$$

โดยค่า R, G, และ B เป็นค่าความเข้มขององค์ประกอบสีแดง สีเขียว และสีน้ำเงิน ส่วนค่า Y, Cb และ Cr เป็นค่าของความเข้มขององค์ประกอบความส่องสว่าง สีน้ำเงินที่หักองค์ประกอบอื่นออก และสีแดงที่หักองค์ประกอบอื่นออก

3) ส่วนฟอร์เวิร์ดอินทราคอมโพเนนท์ทรานส์ฟอร์ม (Forward Intracomponent Transform) ทำหน้าที่แปลงข้อมูลเชิงตำแหน่งเป็นข้อมูลเชิงความถี่โดยใช้วิธีการแปลงเวฟเล็ตแบบไม่ต่อเนื่อง (Discrete Wavelet Transform) เพื่อกรองข้อมูลแต่ละไทล์ด้วยความถี่สูงและความถี่ต่ำทั้งในแนวแถวและแนวคอลัมน์ ผลลัพธ์จากการกรองด้วยความถี่สูงและความถี่ต่ำนี้เรียกว่า

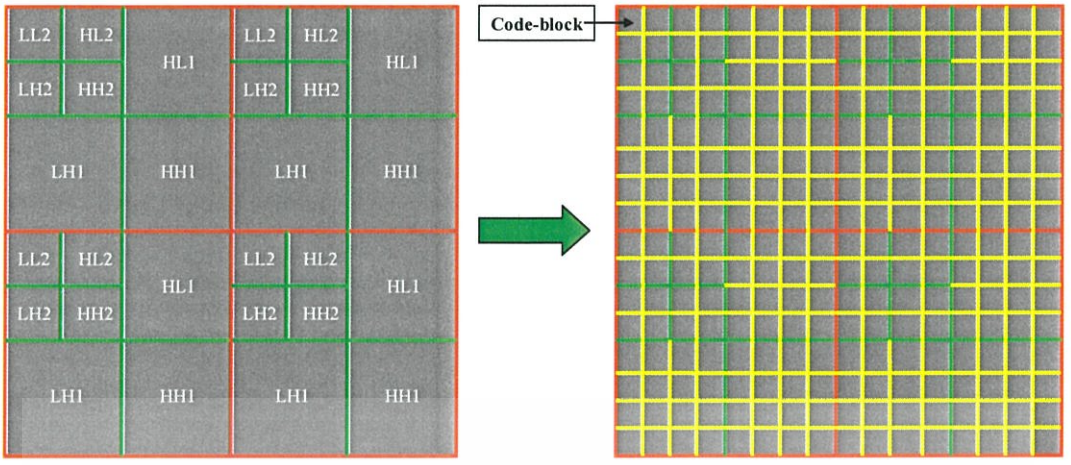
"ซับแบนด์ (Subband)" การแปลงเวฟเล็ตแบบไม่ต่อเนื่องหนึ่งครั้งจะได้ทั้งหมด 4 ซับแบนด์ คือ LL, LH, HL และ HH



รูปที่ 2.3 แสดงตัวอย่างภาพภายหลังการแปลงเวฟเล็ตสำหรับภาพตัดส่วนแต่ละไทล์

4) ส่วนของการจัดระดับข้อมูล (Quantization) ทำหน้าที่ลดทอนค่าของข้อมูล เนื่องจากผลลัพธ์หลังการแปลงเวฟเล็ตเป็นเลขทศนิยม (Floating point) ทำให้การบีบอัดข้อมูลทำได้ยาก ดังนั้นส่วนของการจัดระดับข้อมูล จึงทำหน้าที่แปลงเลขทศนิยมเป็นเลขจำนวนเต็มและลดค่าของข้อมูลให้อยู่ในช่วงที่แคบลง ช่วยให้การบีบอัดข้อมูลทำมีประสิทธิภาพสูงขึ้น แต่ความถูกต้องของข้อมูลก็ลดลงเช่นเดียวกัน ส่วนของการจัดระดับข้อมูล ใช้เฉพาะการบีบอัดข้อมูลภาพแบบสูญเสียเท่านั้น โดยอัตราการลดทอนข้อมูลขึ้นอยู่กับการควบคุมของส่วนควบคุมอัตราการบีบอัด (Rate Control)

5) ส่วนเข้ารหัสไทเออร์วัน ทำหน้าที่บีบอัดข้อมูล โดยนำผลลัพธ์หลังการทำควอนไตซ์แบ่งออกเป็นบล็อกย่อยๆ เรียกว่า "โค้ดบล็อก (Code Block)" จากนั้นนำค่าสัมประสิทธิ์เวฟเล็ตที่ถูกควอนไตซ์แล้วของแต่ละโค้ดบล็อกมาสร้างเป็นสัญลักษณ์ขึ้นมาใหม่ เพื่อลดค่าหรือลดจำนวนสัญลักษณ์ก่อนนำไปบีบอัด ทำให้การบีบอัดข้อมูลมีประสิทธิภาพมากยิ่งขึ้น โดยผลลัพธ์ของการบีบอัดจะอยู่ในรูปของไบต์ส่งออกไปยังส่วนเข้ารหัสไทเออร์ทู (Tier-2 Coding) ต่อไป



รูปที่ 2.4 แสดงตัวอย่างภาพภายหลังการแบ่งภาพออกเป็นโค้ดบล็อกย่อยๆ

6) ส่วนเข้ารหัสไทเออร์ทูทำหน้าที่จัดเรียงข้อมูลที่ถูกบีบอัดแล้วจากส่วนเข้ารหัสไทเออร์วันให้อยู่ในรูปของแพ็คเกจเพื่อนำไปใช้งานในรูปแบบต่างๆ ต่อไป

7) ส่วนควบคุมอัตราการบีบอัด (Rate Control) ทำหน้าที่ควบคุมอัตราการบีบอัดข้อมูลในแต่ละโค้ดบล็อกแต่ละซับแบนด์ตามลำดับ

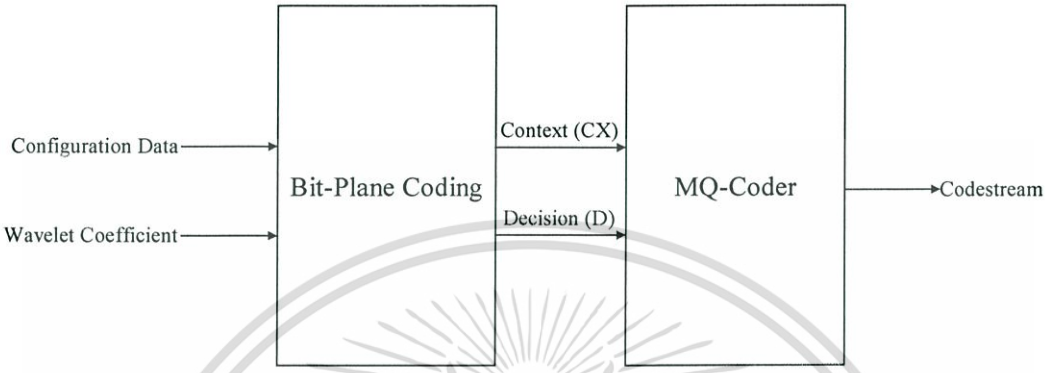
เนื่องจากมาตรฐาน JPEG2000 เป็นมาตรฐานใหม่ มีฟังก์ชั่นการทำงานที่หลากหลายและหลายฟังก์ชั่นมีหลักการทำงานที่ซับซ้อน ทำให้เวลาที่ใช้ในการประมวลผลมากขึ้น เมื่อเทียบกับมาตรฐาน JPEG เดิม ดังนั้นการนำมาตรฐาน JPEG2000 ไปใช้ในงานเรียลไทม์จริง ยังมีข้อจำกัดในเรื่องของเวลา ส่วนเข้ารหัสไทเออร์วันเป็นส่วนที่ใช้เวลาในการประมวลผลมากที่สุดของมาตรฐาน JPEG2000 ทำให้ส่วนเข้ารหัสไทเออร์วันถูกนำไปพัฒนาเป็นจำนวนมาก เพื่อที่จะลดเวลาในการประมวลผล แต่ยังรักษาประสิทธิภาพให้คงเดิม

2.2 ส่วนเข้ารหัสไทเออร์วัน

ส่วนเข้ารหัสไทเออร์วันทำหน้าที่บีบอัดข้อมูลที่ได้จากส่วนควอนไตซ์ แล้วนำผลลัพธ์จากการบีบอัดส่งไปยังส่วนเข้ารหัสไทเออร์ทู เพื่อนำไปจัดเรียงให้อยู่ในรูปของแพ็คเกจก่อนนำไปใช้งานในรูปแบบต่างๆ ต่อไป ส่วนเข้ารหัสไทเออร์วัน แบ่งการทำงานออกเป็น 2 ส่วน คือ

1) ส่วนเข้ารหัสแบบระนาบบิตทำหน้าที่สร้างสัญลักษณ์โดยสัญลักษณ์ที่สร้างขึ้นเหล่านี้ถูกนำไปใช้แทนค่าสัมประสิทธิ์ความถี่เวฟเล็ตที่ถูกควอนไตซ์แล้ว โดยสัญลักษณ์ที่สร้างขึ้นนี้จะมีรูปแบบเป็นคู่ลำดับ (Context : CX, Decision : D) โดยค่า CX จะพิจารณาจากรูปแบบของพิกเซลที่สนใจกับพิกเซลรอบข้าง ซึ่งค่า CX จะอยู่ในช่วง 0 - 18 ส่วนค่า D จะ มีค่าเป็น 0 หรือ 1 เท่านั้น

2) ส่วนเข้ารหัสเอ็มคิวทำหน้าที่บีบอัดข้อมูล (CX, D) ที่ส่งมาจากส่วนเข้ารหัสแบบระนาบบิต โดยใช้พื้นฐานของการเข้ารหัสเลขคณิตแบบไบนารี (Binary Arithematic Coding) โดยผลลัพธ์จากการบีบอัดข้อมูลนี้จะส่งออกไปในรูปของไบต์เรียกว่า "โค้ดสตรีม (Codestream)" ส่งออกไปยังส่วนเข้ารหัสไทเออร์ทู เพื่อนำไปจัดเรียงให้อยู่ในรูปของแพ็คเกจต่อไป



รูปที่ 2.5 แสดงโครงสร้างส่วนเข้ารหัสไทเออร์วันตามมาตรฐาน JPEG2000

ในปัจจุบันมีความต้องการนำมาตรฐาน JPEG2000 ไปใช้ในงานประเภทเรียลไทม์เป็นจำนวนมาก เช่น นำไปใช้กับกล้องดิจิตอล เครื่องสแกนเนอร์ เป็นต้น ช่วยให้ภาพมีขนาดเล็กลงแต่คุณภาพคงเดิม ทำให้ประหยัดพื้นที่ในการเก็บข้อมูล แต่การออกแบบฮาร์ดแวร์ตามมาตรฐาน JPEG2000 ทำได้ยาก เนื่องมาจากความซับซ้อนของการทำงานส่วนเข้ารหัสเอ็มคิวเป็นส่วนสำคัญส่วนหนึ่งที่มีการทำงานที่ซับซ้อน ทำให้มีงานวิจัยออกมาเป็นจำนวนมาก งานวิจัยส่วนใหญ่เป็นการแก้ไขปัญหาเรื่องการลดเวลาที่ใช้ในการประมวลผล โดยใช้วิธีปรับปรุงอัลกอริธึม เป็นต้น

2.3 ปัญหาการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์

ส่วนเข้ารหัสเอ็มคิวเป็นส่วนที่มีการทำงานซับซ้อนส่วนหนึ่งของมาตรฐาน JPEG2000 ทำให้ใช้เวลาในการบีบอัดข้อมูลมาก การทำงานของส่วนเข้ารหัสเอ็มคิวจะใช้การคำนวณเป็นหลัก โดยนำสัญลักษณ์ที่สร้างขึ้นจากส่วนเข้ารหัสแบบระนาบบิตเข้ามาประมวลผลครั้งละ 1 สัญลักษณ์ ผลของการคำนวณสัญลักษณ์ปัจจุบันจะส่งผลต่อการคำนวณของสัญลักษณ์ถัดไป การทำงานของส่วนเข้ารหัสเอ็มคิวจะคำนวณครั้งละ 1 โค้ดบล็อกหลังจากคำนวณเสร็จ 1 โค้ดบล็อก จะทำการรีเซตค่าที่ใช้ในการคำนวณทั้งหมด เพื่อเตรียมการคำนวณสำหรับโค้ดบล็อกถัดไป

เนื่องจากส่วนเข้ารหัสเอ็มคิวมีพื้นฐานการทำงานมาจากส่วนเข้ารหัสเลขคณิต (Arithmetic Coder) ซึ่งใช้การคำนวณสำหรับบีบอัดข้อมูล โดยส่วนเข้ารหัสเลขคณิตจะใช้เลขทศนิยมในการคำนวณส่วนเข้ารหัสเอ็มคิวจะใช้เลขจำนวนเต็มในการคำนวณ ทำให้การออกแบบฮาร์ดแวร์ทำ

ได้ง่ายยิ่งขึ้น แต่เนื่องจากการคำนวณสามารถทำได้ครั้งละหนึ่งสัญลักษณ์เท่านั้น ทำให้การคำนวณของสัญลักษณ์ถัดไปจำเป็นต้องรอผลลัพธ์จากการคำนวณของสัญลักษณ์ปัจจุบัน และจากงานวิจัย [3] ส่วนเข้ารหัสแบบระนาบบิตสามารถสร้างสัญลักษณ์ได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา ทำให้เกิดปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิต และอินพุตของส่วนเข้ารหัสเอ็มคิว นั่นคือ การทำงานส่วนเข้ารหัสเอ็มคิวไม่สามารถประมวลผลสัญลักษณ์ที่ส่งมาจากส่วนเข้ารหัสแบบระนาบบิตได้ทัน ส่งผลให้ผู้ออกแบบสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์นั้น จำเป็นต้องใช้เทคนิคต่างๆ เข้าช่วยในการออกแบบ เพื่อทำให้สามารถเร่งการประมวลผลของส่วนเข้ารหัสเอ็มคิว ให้ทันเวลากับการสร้างสัญลักษณ์ของส่วนเข้ารหัสระนาบบิต (Bit-Plane Coder)

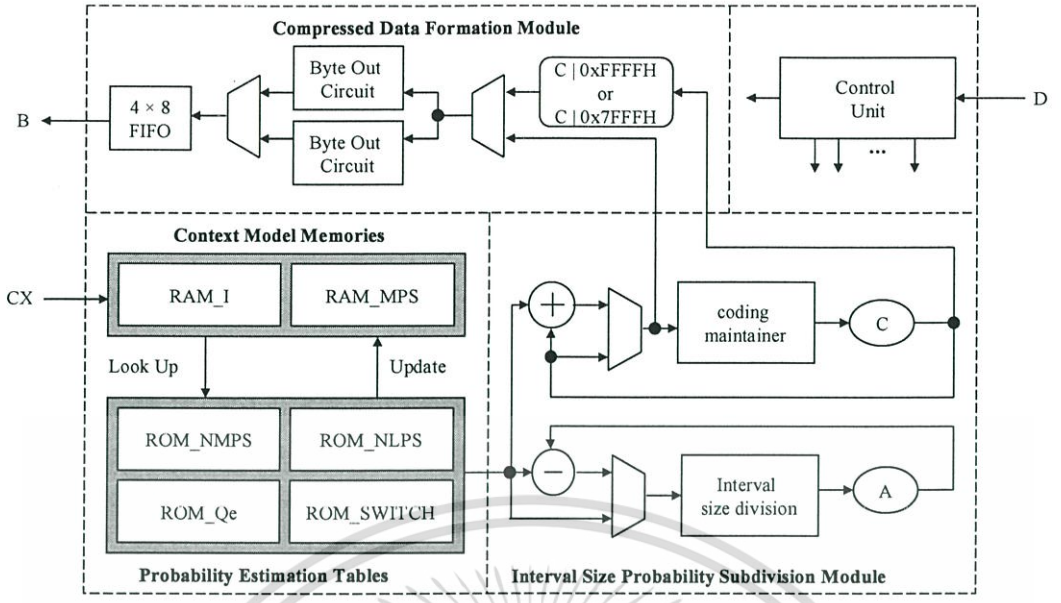
การออกแบบส่วนเข้ารหัสเอ็มคิวทำได้ยาก เนื่องมาจากการทำงานที่มีความซับซ้อน การแก้ไขปัญหาเรื่องการลดเวลาที่ใช้ในการประมวลผล ทำได้โดยการปรับปรุงอัลกอริธึมในส่วนของเส้นทางวิกฤต (Critical Path) เช่น การทำนายผลลัพธ์หรือการคำนวณล่วงหน้า การทำไปป์ไลน์ เป็นต้น จะช่วยลดเวลาในการประมวลผลลงได้

2.4 แนวทางแก้ไขการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์

งานวิจัยที่เกี่ยวข้องกับส่วนเข้ารหัสเอ็มคิวสำหรับการใช้งานประเภทเรียลไทม์ ส่วนใหญ่จะพัฒนาบนอุปกรณ์ประเภทเอสิก (ASIC) หรือเอฟพีจีเอ เนื่องจากผู้ออกแบบสามารถออกแบบวงจรได้ตามความต้องการ

2.4.1 การประมวลผลสัญลักษณ์เดี่ยวสำหรับส่วนเข้ารหัสเอ็มคิวของงานวิจัย [1]

งานวิจัยการประมวลผลสัญลักษณ์เดี่ยวสำหรับส่วนเข้ารหัสเอ็มคิวนี้เป็นงานวิจัยของ Chu Yu และ Hwai-Tsu Hu [1] งานวิจัยนี้พัฒนาบนอุปกรณ์ประเภทเอสิกสำหรับการประมวลผลสัญลักษณ์เดี่ยวนั่นคืองานวิจัยนี้สามารถทำการประมวลผลได้เพียงครั้งละหนึ่งสัญลักษณ์



รูปที่ 2.6 แสดงสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวของงานวิจัย [1]

จากรูปที่ 2.6 สถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวของงานวิจัย [1] สามารถแบ่งการทำงานออกเป็น 4 ส่วน คือ

1) ส่วนที่ 1 ประกอบไปด้วยส่วน Context Model Memories และส่วน Probability Estimation Tables (PET) เป็นส่วนของตารางข้อมูลทั้งหมด

2) ส่วนที่ 2 หรือส่วน Interval Size Probability Subdivision Module เป็นส่วนคำนวณค่าขอบเขตบน (Upper bound: A) และค่าขอบเขตล่าง (Lower bound: C)

3) ส่วนที่ 3 หรือส่วน Compressed Data Formation Module เป็นส่วนจัดการส่งผลลัพธ์จากการบีบอัดข้อมูลไปยังส่วนเข้ารหัสไทเออร์ทู (Tier-2 Encoder)

4) ส่วนที่ 4 หรือส่วนควบคุม (Control Unit) เป็นส่วนควบคุมการทำงานทั้งหมดของวงจร เริ่มตั้งแต่การอ่านค่าสัญลักษณ์เข้ามาประมวลผล จนกระทั่งส่งผลลัพธ์การเข้ารหัสออกไป

การทำงานของส่วนเข้ารหัสเอ็มคิวตามสถาปัตยกรรมของงานวิจัย [1] จะใช้ส่วนที่ 4 ทำหน้าที่ควบคุมการทำงานทั้งหมด แสดงขั้นตอนการทำงานของสถาปัตยกรรมของงานวิจัย [1] ในรูปที่ 2.7 การทำงานเริ่มจากอ่านค่าสัญลักษณ์จากส่วนเข้ารหัสแบบระนาบบิตเข้ามาประมวลผลครั้งละหนึ่งสัญลักษณ์ โดยหนึ่งสัญลักษณ์ประกอบด้วยค่าคอนเท็กซ์ และค่าดีซิชั่น เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน Read(CX, D) จากนั้นส่งค่า CX ไปยังส่วนที่ 1

2.4.1.1 ส่วนที่ 1

ส่วนที่ 1 เริ่มจากอ่านค่าจากตาราง I และตาราง MPS เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน $i = I(CX)$ และ $mps = MPS(CX)$ ค่า i เป็นผลลัพธ์จากตาราง I และค่า mps

เป็นผลลัพธ์จากตาราง MPS จากนั้นนำค่า i ไปอ่านค่าจากตาราง NLPS, ตาราง NMPS, ตาราง Qe และตาราง SWITCH เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน nlps = NLPS(i), nmps = NMPS(i), qe = Qe(i) และ switch = SWITCH(i) ค่า nlps เป็นผลลัพธ์จากตาราง NLPS, ค่า nmps เป็นผลลัพธ์จากตาราง NMPS, ค่า qe เป็นผลลัพธ์จากตาราง Qe และค่า switch เป็นผลลัพธ์จากตาราง SWITCH จากนั้นนำค่า mps, nlps, nmps, qe และ switch ส่งไปยังส่วนที่ 2

2.4.1.2 ส่วนที่ 2

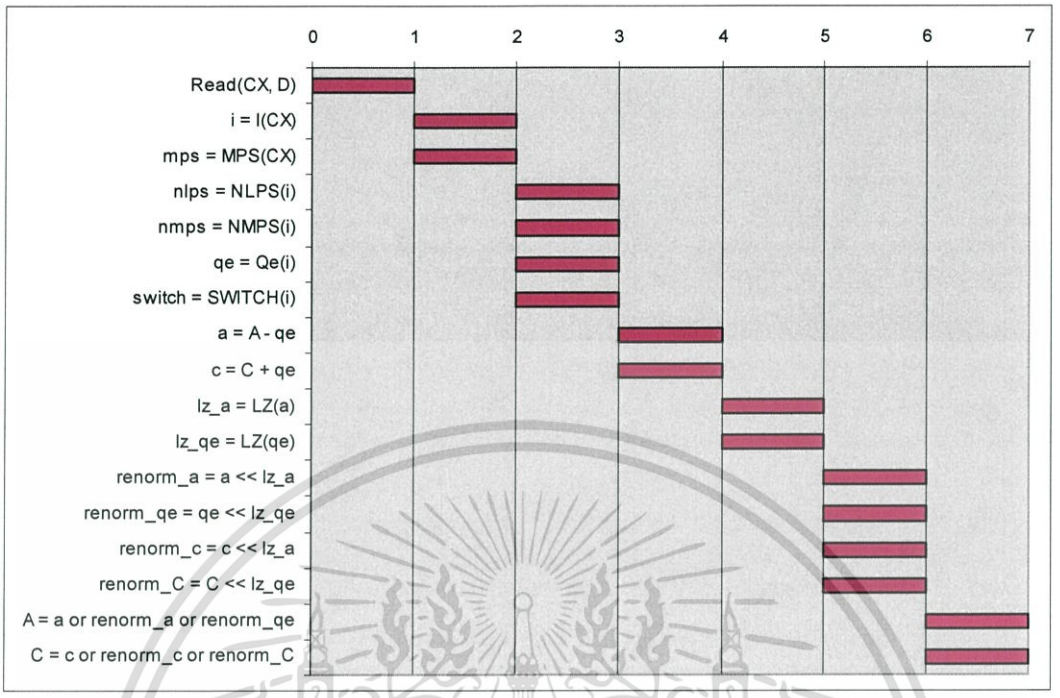
ส่วนที่ 2 เริ่มจากการคำนวณค่าขอบเขตบน (A) และค่าขอบเขตล่าง (C) การคำนวณค่า A เริ่มจากการนำค่าขอบเขตบนลบกับค่าความน่าจะเป็น เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน a = A - qe ค่า a เป็นผลลัพธ์จากการลบ ส่วนการคำนวณค่า C เริ่มจากการนำค่าขอบเขตล่างบวกกับค่าความน่าจะเป็น เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน c = C + qe ค่า c เป็นผลลัพธ์จากการบวก จากนั้นทำการหาจำนวนค่าศูนย์นำหน้า (Leading zero) ของค่า a และ qe เพื่อหาจำนวนการเลื่อนบิตไปทางซ้ายของค่า A และ C เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน lz_a = LZ(a) และ lz_qe = LZ(qe) ค่า lz_a เป็นจำนวนค่าศูนย์นำหน้า ของค่า a และค่า lz_qe เป็นจำนวนค่าศูนย์นำหน้าของค่า qe จากนั้นทำการคำนวณผลลัพธ์กรณีเกิดการรีนอร์มอลไลซ์เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน renorm_a = a << lz_a, renorm_qe = qe << lz_qe, renorm_c = c << lz_a และ renorm_C = C << lz_qe ค่า renorm_a เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า a, ค่า renorm_qe เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า qe, ค่า renorm_c เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า c และค่า renorm_C เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า C จากนั้นทำการอัพเดตค่า A และ C เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน A = a or renorm_a or renorm_qe และ C = c or renorm_c or renorm_C โดยขั้นตอน A = a or renorm_a or renorm_qe สามารถแสดงโค้ดเทียมการอัพเดตค่า A ดังรูปที่ 2.8 และขั้นตอน C = c or renorm_c or renorm_C สามารถแสดงโค้ดเทียมการอัพเดตค่า C ดังรูปที่ 2.9 การทำงานหลังการคำนวณค่าขอบเขตบนและขอบเขตล่างมี 2 กรณี คือ

2.4.1.2.1 กรณีที่ 1

ส่งค่า C ไปยังส่วนที่ 3 เพื่อจัดการผลลัพธ์จากการบีบอัดข้อมูลไปยังส่วนเข้ารหัสไทเออร์ทู (Tier-2 Encoder) ต่อไป

2.4.1.2.2 กรณีที่ 2

ทำการอ่านสัญลักษณ์ถัดไปเพื่อเริ่มการคำนวณใหม่ การทำงานจะเป็นเช่นนี้ไปจนจบโค้ดบล็อกจึงทำการรีเซตค่าต่างๆ สำหรับการบีบอัดข้อมูลในโค้ดบล็อกถัดไป



รูปที่ 2.7 แสดงแกนต์ชาร์ตของสถาปัตยกรรมงานวิจัย [1]

```
IF D = mps THEN
    IF a ≥ 0x8000 THEN
        A = a
    ELSE
        IF A < 2 × qe THEN
            A = renorm_a
        ELSE
            A = renorm_qe
        ENDIF
ELSE
    IF A < 2 × qe THEN
        A = renorm_a
    ELSE
        A = renorm_qe
    ENDIF
ENDIF
```

รูปที่ 2.8 แสดงโค้ดเทียมการปรับปรุงค่าขอบเขตบน

```
IF D = mps THEN
    IF a ≥ 0x8000 THEN
        C = c
    ELSE
        IF A < 2 × qe THEN
            C = renorm_c
        ELSE
            C = renorm_C
        ENDIF
ELSE
    IF A < 2 × qe THEN
        C = renorm_c
    ELSE
        C = renorm_C
    ENDIF
ENDIF
```

รูปที่ 2.9 แสดงโค้ดเทียมการปรับปรุงค่าขอบเขตล่าง

เนื่องมาจากการทำงานแบบลำดับขั้นของส่วนเข้ารหัสเอ็มคิวทำให้ต้องใช้เวลาในการเข้ารหัสมาก การออกแบบสถาปัตยกรรมของงานวิจัย [1] มีการพัฒนาส่วนรีนอร์มอลไลซ์โดยใช้เทคนิคการหาจำนวนค่าศูนย์นำหน้าเข้ามาช่วยคำนวณหาจำนวนการเลื่อนบิตไปทางซ้ายของค่า A และ C ทำให้ลดเวลาในการทำรีนอร์มอลไลซ์ได้มาก และยังมีการแยกส่วนจัดการผลลัพธ์ออกจากส่วนคำนวณค่าขอบเขตล่าง ทำให้สามารถประมวลผลสัญลักษณ์ถัดไปได้ทันที แต่ถึงอย่างไรก็ตามสถาปัตยกรรมของงานวิจัย [1] ยังคงทำงานแบบลำดับ คือ ทำงานได้ครั้งละ 1 สัญลักษณ์เท่านั้น ทำให้ปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิวยังคงมีอยู่

2.4.2 การประมวลผลสัญลักษณ์คู่สำหรับส่วนเข้ารหัสเอ็มคิวของงานวิจัย [2]

งานวิจัยการประมวลผลสัญลักษณ์คู่สำหรับส่วนเข้ารหัสเอ็มคิวนี้เป็นงานวิจัยของ Michael Dyer, David Taubman และ Saeid Nooshabadi [2] งานวิจัยนี้พัฒนาบนอุปกรณ์ประเภทเอฟพีจีเอ การประมวลผลสัญลักษณ์คู่ หมายถึง การประมวลผลครั้งละสองสัญลักษณ์พร้อมกัน

จากสถาปัตยกรรมของงานวิจัย [1] ถึงแม้จะมีการพัฒนาส่วนรีนอร์มอลไลซ์และมีการแยกส่วนจัดการผลลัพธ์ออกจากส่วนคำนวณค่าขอบเขตล่างแล้วก็ตาม แต่ขั้นตอนการทำงานยังคงเป็นแบบลำดับขั้นอยู่ ดังนั้นงานวิจัย [2] จึงออกแบบส่วนเข้ารหัสเอ็มคิวให้สามารถประมวลผลครั้งละสองสัญลักษณ์ต่อสัญญาณนาฬิกา เพื่อช่วยลดเวลาในการประมวลผลและลดปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิว

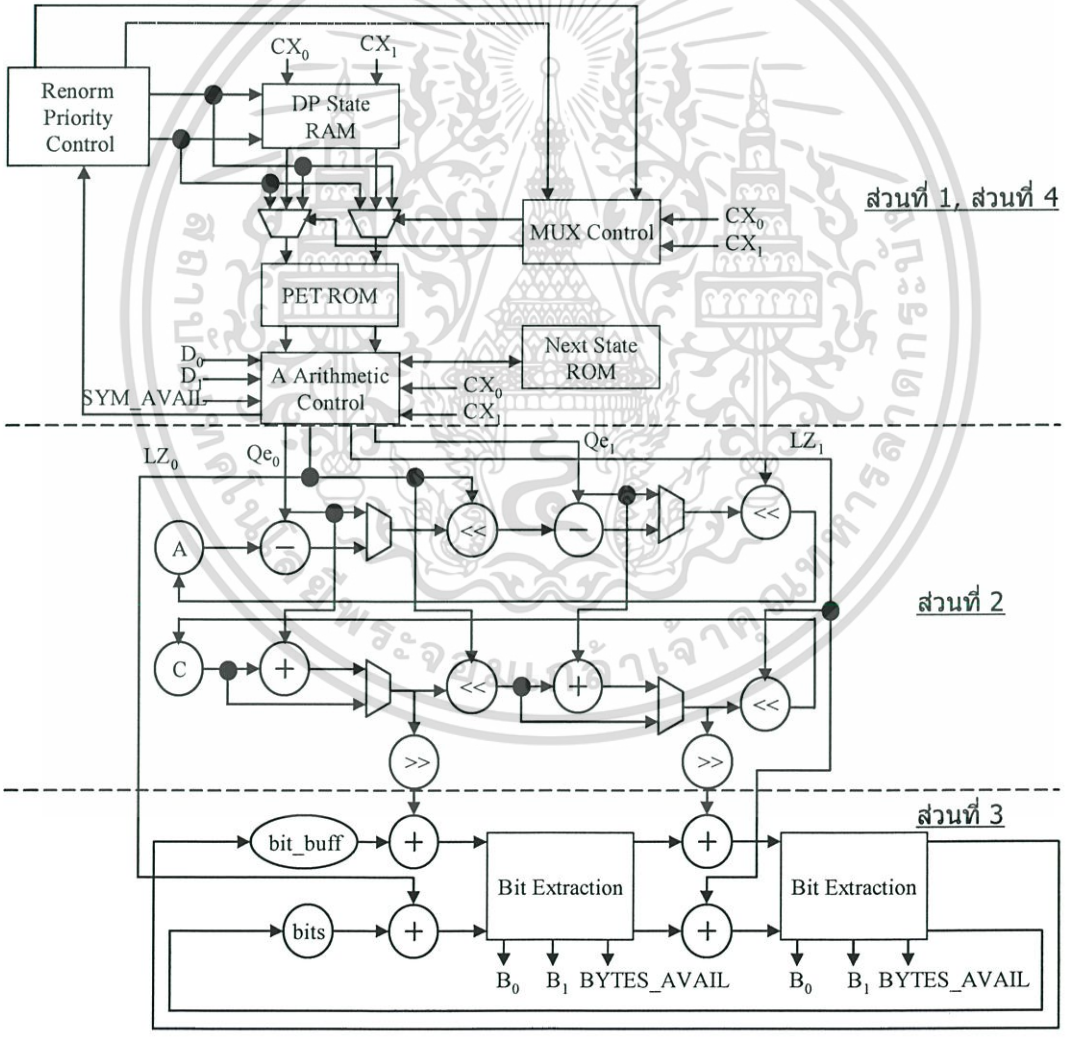
จากรูปที่ 2.10 แสดงสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวของงานวิจัย [2] เพื่อเปรียบเทียบให้เห็นความแตกต่างของการทำงานระหว่างงานวิจัย [1] และงานวิจัย [2] สามารถแบ่งการทำงานของงานวิจัย [2] ออกเป็น 4 ส่วน คือ

1) ส่วนที่ 1 เป็นส่วนของตารางข้อมูลทั้งหมด

2) ส่วนที่ 2 เป็นส่วนคำนวณค่าขอบเขตบน (Upper bound: A) และค่าขอบเขตล่าง (Lower bound: C)

3) ส่วนที่ 3 เป็นส่วนจัดการผลลัพธ์จากการบีบอัดข้อมูลไปยังส่วนเข้ารหัสไทเออร์ทู (Tier-2 Encoder)

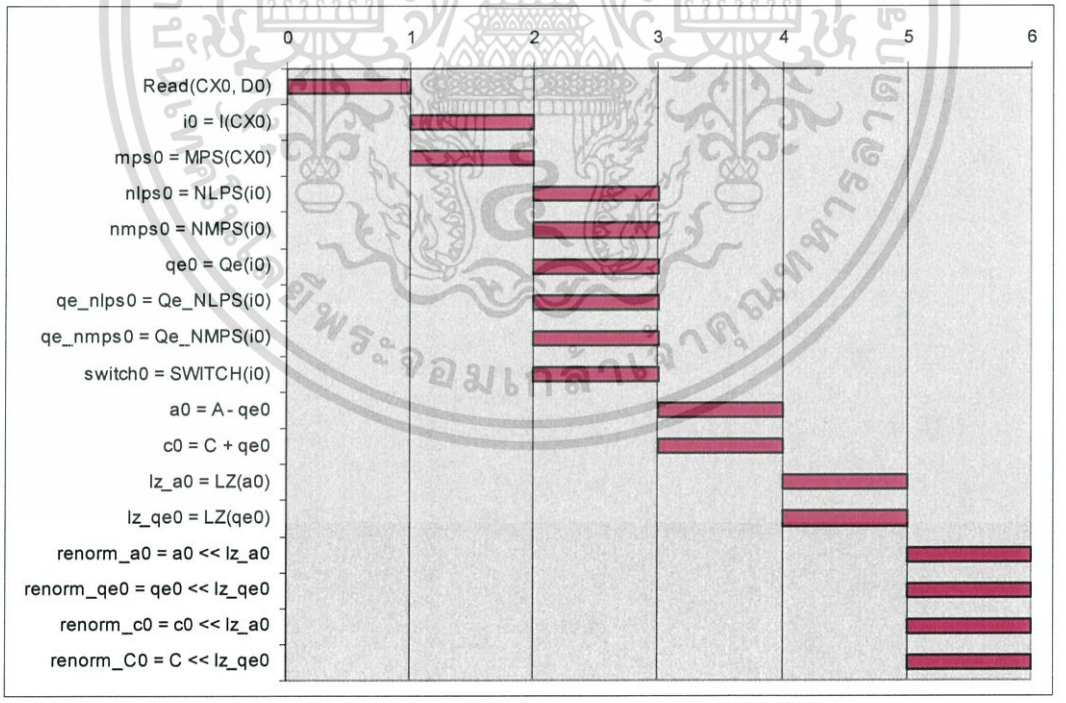
4) ส่วนที่ 4 เป็นส่วนควบคุมการทำงานทั้งหมดของวงจร เริ่มตั้งแต่การอ่านค่าสัญลักษณ์เข้ามาประมวลผล จนกระทั่งส่งผลลัพธ์การเข้ารหัสออกไป



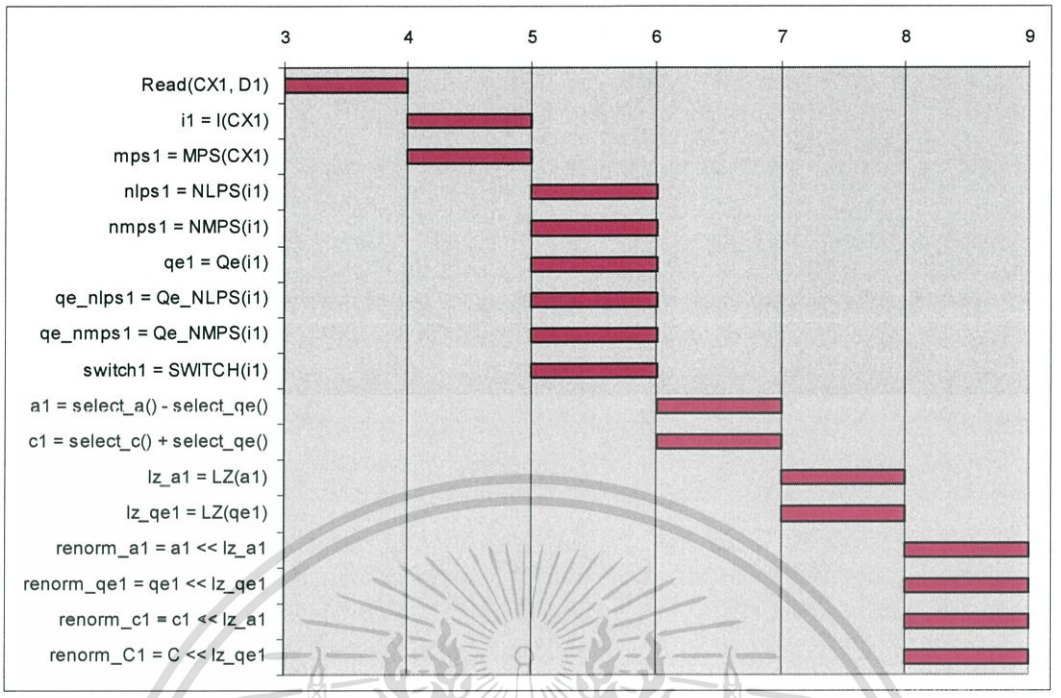
รูปที่ 2.10 แสดงสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวของงานวิจัย [?]

การทำงานของส่วนเข้ารหัสเอ็มคิวตามสถาปัตยกรรมของงานวิจัย [2] จะใช้ส่วนที่ 4 ทำหน้าที่ควบคุมการทำงานทั้งหมด แสดงขั้นตอนการทำงานของสถาปัตยกรรมของงานวิจัย [2] ในรูปที่ 2.11 และรูปที่ 2.12 โดยรูปที่ 2.11 แสดงขั้นตอนการทำงานของการเข้ารหัสสัญลักษณ์ที่หนึ่ง ส่วนรูปที่ 2.12 แสดงขั้นตอนการทำงานของการเข้ารหัสสัญลักษณ์ที่สอง การทำงานเริ่มจากอ่านค่าสัญลักษณ์จากส่วนเข้ารหัสแบบระนาบบิตเข้ามาประมวลผลครั้งละสองสัญลักษณ์ โดยสัญลักษณ์ที่หนึ่งประกอบด้วยค่า CX_0 และ D_0 ส่วนสัญลักษณ์ที่สองประกอบด้วยค่า CX_1 และ D_1 เปรียบเทียบกับรูปที่ 2.11 คือ ขั้นตอน Read(CX_0, D_0) และเปรียบเทียบกับรูปที่ 2.12 คือ ขั้นตอน Read(CX_1, D_1) จากรูปทั้งสองจะสังเกตได้ว่าการทำงานของสัญลักษณ์ที่สองจะช้ากว่าการทำงานของสัญลักษณ์ที่หนึ่ง เนื่องมาจากการคำนวณของสัญลักษณ์ที่หนึ่งจะส่งผลต่อการคำนวณของสัญลักษณ์ที่สอง ถึงแม้จะมีการทำงานเหมือนกัน แต่เป็นการทำงานคนละช่วงเวลากัน ดังนั้นจึงขออธิบายเฉพาะการเข้ารหัสของสัญลักษณ์ที่หนึ่งและส่วนเพิ่มเติมจากงานวิจัย [1] เริ่มจากการส่งค่า CX_0 ไปยังส่วนที่ 1 เพื่ออ่านค่าจากตาราง I และตาราง MPS เปรียบเทียบกับรูปที่ 2.11 คือ ขั้นตอน $i_0 = I(CX_0)$ และ $mps_0 = MPS(CX_0)$ ค่า i_0 เป็นผลลัพธ์จากตาราง I และค่า mps_0 เป็นผลลัพธ์จากตาราง MPS จากนั้นนำค่า i_0 ไปอ่านค่าจากตาราง NLPS, ตาราง NMPS, ตาราง Qe, ตาราง Qe_NLPS, ตาราง Qe_NMPS และตาราง SWITCH เปรียบเทียบกับรูปที่ 2.11 คือ ขั้นตอน $nlps_0 = NLPS(i_0)$, $nmps_0 = NMPS(i_0)$, $qe_0 = Qe(i_0)$, $qe\_nlps_0 = Qe\_NLPS(i_0)$, $qe\_nmps_0 = Qe\_NMPS(i_0)$ และ $switch_0 = SWITCH(i_0)$ ค่า $nlps_0$ เป็นผลลัพธ์จากตาราง NLPS, ค่า $nmps_0$ เป็นผลลัพธ์จากตาราง NMPS, ค่า qe_0 เป็นผลลัพธ์จากตาราง Qe, ค่า qe_nlps_0 เป็นผลลัพธ์จากตาราง Qe_NLPS, ค่า qe_nmps_0 เป็นผลลัพธ์จากตาราง Qe_NMPS และค่า $switch_0$ เป็นผลลัพธ์จากตาราง SWITCH จากนั้นนำค่า mps_0, $nlps_0$, $nmps_0$, qe_0, qe_nlps_0, qe_nmps_0 และ $switch_0$ ส่งไปยังส่วนที่ 2 เพื่อคำนวณค่าขอบเขตบน (A) และค่าขอบเขตล่าง (C) การคำนวณค่า A เริ่มจากการนำค่าขอบเขตบนลบกับค่าความน่าจะเป็น เปรียบเทียบกับรูปที่ 2.11 คือ ขั้นตอน $a_0 = A - qe_0$ ค่า a_0 เป็นผลลัพธ์จากการลบ ส่วนการคำนวณค่า C เริ่มจากการนำค่าขอบเขตล่างบวกกับค่าความน่าจะเป็น เปรียบเทียบกับรูปที่ 2.7 คือ ขั้นตอน $c_0 = C + qe_0$ ค่า c_0 เป็นผลลัพธ์จากการบวก จากนั้นทำการหาจำนวนค่าศูนย์นำหน้าของค่า a_0 และ qe_0 เพื่อหาจำนวนการเลื่อนบิตไปทางซ้ายของค่า A และ C เปรียบเทียบกับรูปที่ 2.11 คือ ขั้นตอน $lz\_a_0 = LZ(a_0)$ และ $lz\_qe_0 = LZ(qe_0)$ ค่า lz_a_0 เป็นจำนวนค่าศูนย์นำหน้า ของค่า a_0 และค่า lz_qe_0 เป็นจำนวนค่าศูนย์นำหน้า ของค่า qe_0 จากนั้นทำการคำนวณผลลัพธ์กรณีเกิดการรีนอร์มอลไลซ์เปรียบเทียบกับรูปที่ 2.11 คือ ขั้นตอน $renorm\_a_0 = a_0 << lz\_a_0$, $renorm\_qe_0 = qe_0 << lz\_qe_0$, $renorm\_c_0 = c_0 << lz\_a_0$ และ $renorm\_C_0 = C << lz\_qe_0$ ค่า $renorm\_a_0$ เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า a_0, ค่า $renorm\_qe_0$ เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า qe_0, ค่า $renorm\_c_0$ เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า c_0 และค่า $renorm\_C_0$ เป็นผลลัพธ์จากการทำรีนอร์มอลไลซ์ กับค่า C จากนั้นทำการเลือกผลลัพธ์จาก

การคำนวณค่า A ล่วงหน้า (a_0 หรือ renorm_a_0 หรือ renorm_qe_0) และทำการเลือกค่าความน่าจะเป็น (qe_nlps_0 หรือ qe_nmps_0 หรือ qe_1) นำผลลัพธ์จากการเลือกค่า A ลบกับผลลัพธ์จากการเลือกค่าความน่าจะเป็น เปรียบเทียบกับรูปที่ 2.12 คือ ขั้นตอน a_1 = select_a() - select_qe() ค่า a_1 เป็นผลลัพธ์จากการลบ และทำการเลือกผลลัพธ์จากการคำนวณค่า C ล่วงหน้า (c_0 หรือ renorm_c_0 หรือ renorm_C_0) และทำการเลือกค่าความน่าจะเป็น (qe_nlps_0 หรือ qe_nmps_0 หรือ qe_1) นำผลลัพธ์จากการเลือกค่า C บวกกับผลลัพธ์จากการเลือกค่าความน่าจะเป็น เปรียบเทียบกับรูปที่ 2.12 คือ ขั้นตอน c_1 = select_c() + select_qe() ค่า c_1 เป็นผลลัพธ์จากการบวก ผลจากการคาดเดาผลลัพธ์ล่วงหน้าจะช่วยลดเวลาในการประมวลผลลงการทำงานหลังจากนี้จะเหมือนกับการทำงานดังที่กล่าวไว้ข้างต้น คือ ในขณะที่กำลังคำนวณค่า A และค่า C ของสัญลักษณ์ปัจจุบัน ก็จะทำการอ่านค่าสัญลักษณ์ถัดไป เพื่อเตรียมค่าความน่าจะเป็นสำหรับการคำนวณของสัญลักษณ์ถัดไป สำหรับผลลัพธ์จากการคำนวณด้วยสัญลักษณ์ปัจจุบันมี 2 กรณี คือ กรณีที่ 1 ส่งผลลัพธ์จากการคำนวณไปยังส่วนที่ 3 เพื่อจัดการผลลัพธ์แล้วส่งไปยังส่วนเข้ารหัสไทเออร์ทู (Tier-2 Encoder) ต่อไป หรือกรณีที่ 2 ส่งผลลัพธ์จากการคำนวณไปยังส่วนคำนวณค่าขอบเขตบนและขอบเขตล่างจากสัญลักษณ์ถัดไป การทำงานจะเป็นเช่นนี้ไปจนจบโค้ดบล็อกจึงทำการรีเซตค่าต่างๆ สำหรับการบีบอัดข้อมูลในโค้ดบล็อกถัดไป



รูปที่ 2.11 แสดงแกนต์ชาร์ตของสถาปัตยกรรมงานวิจัย [2]



รูปที่ 2.12 แสดงแกนต์ชาร์ตของสถาปัตยกรรมงานวิจัย [2] (ต่อ)

เนื่องมาจากการทำงานแบบลำดับขั้นของงานวิจัย [1] ทำให้การเข้ารหัสยังคงใช้เวลามาก งานวิจัย [2] จึงมีการออกแบบสถาปัตยกรรมใหม่ เพื่อลดเวลาในการประมวลผล โดยใช้เทคนิคการประมวลผลสัญลักษณ์คู่และการคาดเดาผลลัพธ์ค่า A, ค่า C และค่าความน่าจะเป็นล่วงหน้าเข้ามาช่วย อีกทั้งยังมีการแยกส่วนจัดการผลลัพธ์ออกจากส่วนคำนวณค่าขอบเขตล่าง ทำให้สามารถประมวลผลสัญลักษณ์ถัดไปได้ทันที แต่ถึงอย่างไรก็ตามสถาปัตยกรรมของงานวิจัย [2] ยังสามารถปรับปรุงส่วนคาดเดาล่วงหน้าให้มีประสิทธิภาพมากยิ่งขึ้น เพื่อช่วยลดขั้นตอนที่เป็นลำดับขั้นลงทำให้ช่วยลดปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิวลงได้

บทที่ 3
ความรู้พื้นฐานในเรื่องที่เกี่ยวข้องกับงานวิจัย

ในปัจจุบันเทคโนโลยีสื่อสารเติบโตขึ้นอย่างรวดเร็ว การติดต่อสื่อสารกลายเป็นส่วนหนึ่งในชีวิตประจำวัน การติดต่อสื่อสารมีหลากหลายรูปแบบ เทคโนโลยีสื่อสารมีการพัฒนาจากระบบอนาล็อกเป็นระบบดิจิตอล เพื่อเพิ่มความสามารถในการรองรับความต้องการของผู้บริโภค การพัฒนาเทคโนโลยีสื่อสารเป็นระบบดิจิตอล ทำให้เกิดการเปลี่ยนแปลงครั้งสำคัญ การติดต่อสื่อสารสามารถส่งได้ทั้งภาพและเสียงจากประเทศหนึ่งไปยังอีกประเทศหนึ่ง การติดต่อสื่อสารในลักษณะนี้เกิดจากการนำเอาสัญญาณดิจิตอลไปใช้ในการประมวลผลข้อมูล การจัดการข้อมูล การจัดเก็บข้อมูล หรือการส่งข้อมูลในระยะทางไกล เป็นต้น

ในปัจจุบันเทคโนโลยีอินเตอร์เน็ตเริ่มเข้ามามีบทบาทในชีวิตประจำวันของเราเพิ่มขึ้น การติดต่อสื่อสารขยายวงกว้างมากยิ่งขึ้น เกิดเป็นเครือข่ายจำนวนมากในโลกของอินเตอร์เน็ต การติดต่อสื่อสารผ่านอินเตอร์เน็ตมีความหลากหลาย เช่น การสื่อสารผ่านจดหมายอิเล็กทรอนิกส์ การประชุมทางไกล หรือแม้กระทั่งการดูหนังฟังเพลง เป็นต้น การติดต่อสื่อสารเหล่านี้ทำให้เกิดการรับส่งข้อมูลจำนวนมาก การสื่อสารต้องตอบสนองความต้องการของผู้บริโภค ดังนั้นการลดปริมาณข้อมูลจะช่วยลดความหนาแน่นในการรับส่งข้อมูล ทำให้ตอบสนองความต้องการของผู้บริโภคได้ดีขึ้น

3.1 การบีบอัดข้อมูล

ในปัจจุบันการบีบอัดข้อมูลอยู่รอบๆ ตัวเรา เช่น เมื่อการดูหนังจากเครื่องเล่นดีวีดี ในแผ่นดีวีดีก็มีการบีบอัดข้อมูล การถ่ายรูปด้วยกล้องดิจิตอล ภาพถ่ายก็มีการบีบอัดข้อมูล เป็นต้น จะเห็นได้ว่าสิ่งต่างๆ รอบตัวเรามีการบีบอัดข้อมูล ข้อดีของการบีบอัดข้อมูล เช่น ช่วยลดพื้นที่ในการจัดเก็บ หากนำไปใช้กับการสื่อสารจะช่วยลดความหนาแน่นในการรับส่งข้อมูล เป็นต้น ส่วนข้อเสียของการบีบอัดข้อมูล เช่น ต้องใช้เวลาในการบีบอัดหรือคลายข้อมูล ทำให้ความถูกต้องของข้อมูลลดลง เป็นต้น

การบีบอัดข้อมูล (Compression) หรือเรียกว่า “การเข้ารหัส (Encoding)” และส่วนการคลายข้อมูล (Decompression) หรือรู้จักกันในชื่ออีกหนึ่งว่า “การถอดรหัส (Decoding)”

3.1.1 แนวคิดของทฤษฎีข้อมูล

แนวคิดของทฤษฎีข้อมูล (Information theory) เกิดขึ้นในปี ค.ศ. 1948 โดย Claude E.-Shannon กำหนดให้ข้อมูล (Data) ประกอบด้วยสารสนเทศ (Information) และส่วนซ้ำซ้อนใน

ข้อมูล (Redundancy) โดยสารสนเทศเป็นส่วนของข้อมูลที่จำเป็นต้องรักษาไว้ให้คงอยู่ในรูปแบบเดิม เพื่อให้การตีความถูกต้องตามความหมายของข้อมูล ส่วนซ้ำซ้อนในข้อมูล เป็นส่วนของข้อมูลที่สามารถตัดทิ้ง เมื่อไม่ต้องการและต้องสามารถแทรกเข้าไปในข้อมูลเมื่อต้องการได้ โดยทั่วไปส่วนซ้ำซ้อนในข้อมูล จะถูกแทรกกลับยังข้อมูลเพื่อสร้างกลับข้อมูลให้อยู่ในรูปแบบดั้งเดิมได้ เทคนิคที่นำไปใช้สำหรับการบีบอัดข้อมูล คือ การลดส่วนซ้ำซ้อนในข้อมูล ส่วนเทคนิคที่ใช้สำหรับการถอดรหัส คือ การถอดกลับเนื้อข้อมูลที่บีบอัด พร้อมการถอดกลับข้อมูลส่วนซ้ำซ้อนกลับคืนมา

การบีบอัดข้อมูลที่จริงแล้วก็ คือ การเทคนิคการลดความซ้ำซ้อนในข้อมูล โดยที่จะทำการแทรกกลับส่วนซ้ำซ้อนที่ลดลง เมื่อการคลายข้อมูลเพื่อสร้างข้อมูลให้กลับมาอยู่ในรูปแบบดั้งเดิม ในบางครั้งการบีบอัดข้อมูลอาจถูกเรียกว่า "การเข้ารหัส" เช่นเดียวกับการถอดการบีบอัดถูกเรียกว่า "การถอดรหัส"

การพัฒนาขั้นตอนในการบีบอัดข้อมูลสามารถแบ่งได้เป็น 2 ช่วง คือ

- การสร้างแบบจำลอง (Modeling) : เป็นขั้นตอนการดึงส่วนที่ซ้ำซ้อนในข้อมูลและนำมาอธิบายในแบบจำลอง (Model) ซึ่งทำให้เราบอกถึงความแตกต่างของแบบจำลองข้อมูลและข้อมูลจริงได้
- การเข้ารหัส (Coding) : เป็นขั้นตอนในการเข้ารหัสข้อมูลจากแบบจำลองและผลต่างของข้อมูลจริงและข้อมูลจากแบบจำลอง

จะเห็นว่าอัลกอริธึมในการบีบอัดข้อมูลจะยิ่งมีประสิทธิภาพ ถ้าแบบจำลองที่สร้างสามารถทำให้คุณลักษณะ (Characteristics) ใกล้เคียงกับขั้นตอนการสร้างข้อมูล โดยแบบจำลองอาจมาจากการสังเกตและติดตาม (Observation) ค่าทางสถิติของข้อมูลที่ถูกสร้างขึ้นมาในขั้นตอนหรือ จากแหล่งกำเนิด (Source)

3.1.2 แบบจำลองไม่ต่อเนื่องที่ไม่ต้องใช้หน่วยความจำและเอ็นโทรปี

นิยามทฤษฎีนี้ **ถ้าสัญลักษณ์ (Symbol) ต่างๆ ที่สร้างจากแหล่งกำเนิดข้อมูล (Information Source) เป็นสัญลักษณ์ ที่เป็นอิสระต่อกัน (Independent to each other) แล้วแหล่งกำเนิดนั้นจะเรียกว่า "Discrete Memoryless Source" ซึ่งสามารถอธิบายได้ด้วย สัญลักษณ์ที่สร้างขึ้น $A = \{a_1, a_2, a_3, \ldots, a_N\}$ และโอกาสในการเกิดของแต่ละสัญลักษณ์ $P = \{p(a_1), p(a_2), \ldots, p(a_N)\}$**

คำจำกัดความของแบบจำลองแหล่งกำเนิดที่ไม่ใช้หน่วยความจำแบบไม่ต่อเนื่อง (Discrete Memoryless Source Model) เป็นแนวคิดที่ดีสำหรับการเข้าใจถึงปริมาณของค่าเฉลี่ยของเนื้อหาข้อมูลสำหรับแต่ละสัญลักษณ์ ที่สร้างจากแหล่งกำเนิดที่ไม่ใช้หน่วยความจำแบบไม่ต่อเนื่อง โดยค่าปริมาณที่วัดได้นี้ เรียกว่า "เอ็นโทรปี (Entropy)" ของข้อมูลแนวคิดของเอ็นโทรปีถูกนำไปใช้ครั้ง

แรกในการวัดความผิดปกติในเทอร์โมไดนามิคส์ (Thermodynamics) แต่สำหรับการบีบอัดข้อมูลถูกนำไปใช้ในการวัดความสามารถสูงสุดในการบีบอัดต่อสัญลักษณ์

จากสมการที่ 3.1 ค่า $I(a_i)$ แทนผลรวมของปริมาณข้อมูล, a_i แทนสัญลักษณ์ตำแหน่งที่ i และ $p(a_i)$ แทนความน่าจะเป็นของสัญลักษณ์ตำแหน่งที่ i ส่วนลอการิธึมฐานสองเป็นการแทนข้อมูลด้วยเลขฐานสอง

จากสมการที่ 3.2 ค่า E แทนผลรวมค่าเฉลี่ยของข้อมูลต่อสัญลักษณ์จาก A

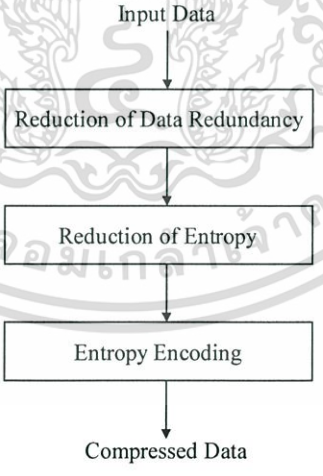
$$I(a_i) = \log_2 \frac{1}{p(a_i)} = -\log 2 p(a_i) \tag{3.1}$$

$$E = \sum_{i=1}^{N} p(a_i) I(a_i) = -\sum_{i=1}^{N} p(a_i) \log_2 p(a_i) \tag{3.2}$$

3.1.3 แบบจำลองการบีบอัดข้อมูล

รูปแบบของการบีบอัดข้อมูลสามารถแสดงได้ดังรูปที่ 3.1 มี 3 ขั้นตอน คือ

1) ลดส่วนซ้ำซ้อนในข้อมูล
2) ลดเอ็นโทรปี
3) บีบอัดข้อมูลโดยใช้พื้นฐานของเอ็นโทรปี



รูปที่ 3.1 แสดงแบบจำลองการบีบอัดข้อมูล

ส่วนซ้ำซ้อนในข้อมูลอาจปรากฏได้หลายรูปแบบ ตัวอย่างเช่น พิกเซลข้างเคียงในภาพมักมีความสัมพันธ์กันสูง (Highly Correlated) ความสัมพันธ์ (Correlation) ในที่นี้หมายถึง พิกเซล

ใกล้เคียงมักมีค่าสีและความเข้มแสงใกล้เคียงกัน ในบริเวณที่ไม่ใช่บริเวณขอบของวัตถุ (Non-edge Smooth Region) ในรูปภาพ สำหรับภาพเคลื่อนไหวที่เป็นวิดีโอนั้น ภาพที่อยู่ติดๆ กันมักจะเป็นภาพที่คล้ายคลึงกัน อาจไม่มีการเคลื่อนที่หรือเปลี่ยนแปลงแต่น้อยมาก เป็นการเปลี่ยนแปลงอย่างช้าๆ เช่นเดียวกับระเบียน (Record) ในฐานข้อมูล ที่อาจมีความสัมพันธ์กันในระหว่างเอนทิตี (Entity) ของแต่ละระเบียน ดังนั้นเพื่อให้การบีบอัดข้อมูลมีประสิทธิภาพควรลดความซ้ำซ้อนในข้อมูลเหล่านี้

การลดความซ้ำซ้อนในข้อมูลส่วนใหญ่จะใช้วิธี การแปลงข้อมูลต้นฉบับให้อยู่ในรูปแบบหรือตัวแทนอีกรูปแบบหนึ่ง ตัวอย่างเช่น การแปลงข้อมูลในทางตำแหน่ง (Spatial Domain) ไปอยู่ในรูปของข้อมูลในทางความถี่ (Frequency Domain) เทคนิคการแปลง (Transform) ข้อมูลแบบนี้ เช่น การแปลงโคซายน์แบบไม่ต่อเนื่อง (Discrete Cosine Transform) หรือการแปลงข้อมูลให้อยู่ในรูปของกลุ่มของความถี่ (Subband) เช่น การแปลงเวฟเล็ตแบบไม่ต่อเนื่อง (Discrete Wavelet Transform)

ขั้นตอนนี้เป็นขั้นตอนที่มีผลกระทบกับตัวแทนใหม่ที่จะนำมาใช้แทนข้อมูลต้นฉบับ นั่นคือช่วยให้มีจำนวนค่าสัมประสิทธิ์เท่าเดิมหรือลดน้อยลงได้ ในกรณีของการบีบอัดแบบไม่สูญเสีย ขั้นตอนนี้สามารถทำย้อนกลับได้อย่างสมบูรณ์ นั่นคือผลลัพธ์ที่ได้เหมือนกันทุกประการกับข้อมูลต้นฉบับนั่นเอง การแปลงนี้ช่วยลดเอ็นโทรปีในข้อมูลต้นฉบับโดยช่วยตัดความซ้ำซ้อนในข้อมูลที่ปรากฏในโครงสร้างลำดับของข้อมูลที่อ้างอิงโดยตำแหน่ง หลังจากได้ตัวแทนใหม่ของข้อมูลแล้ว สามารถนำมาทำการเข้ารหัสเพื่อบีบอัดข้อมูลได้ด้วย การเข้ารหัสโดยใช้หลักการของเอ็นโทรปี ได้แก่ การเข้ารหัสแบบฮัฟแมน (Huffman Coding) การเข้ารหัสเลขคณิต เป็นต้น

3.2 การบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิต

การบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตจะใช้การคำนวณเป็นหลัก โดยขอบเขตของการคำนวณจะอยู่ระหว่าง 0.0 ถึง 1.0 อัลกอริธึมของการเข้ารหัสเลขคณิตไม่มีความซับซ้อนมากนัก เพียงแต่ถ้ามีอินพุตเป็นจำนวนมาก จะทำให้การคำนวณต้องการความละเอียดของของทศนิยมที่ใช้ในการคำนวณมีจำนวนมากขึ้นเช่นเดียวกัน

3.2.1 อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิต

อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตแสดงตามโค้ดเทียมในรูปที่ 3.2 โดยตัวแปร L แทนค่าขอบเขตล่าง ตัวแปร H แทนค่าขอบเขตบน ตัวแปร N แทนจำนวนของอินพุต และฟังก์ชั่น $F(i)$ แทนผลรวมความน่าจะเป็นของสัญลักษณ์ที่ i โดยตัวแปร i แทนดัชนีของสัญลักษณ์

```
Algorithm: Arithmetic Coding
begin
    L = 0.0;
    H = 1.0;
    F(0) = 0;

    for (j = 1 to N) {
        i = index of Symbol(j);
        L = L + (H - L) * F(i - 1);
        H = L + (H - L) * F(i);
    }

    Output ((L+H)/2);
end
```

รูปที่ 3.2 แสดงอัลกอริธึมสำหรับการบีบอัดข้อมูลของการเข้ารหัสเลขคณิต

ตัวอย่างการบีบอัดข้อมูลของวิธีการเข้ารหัสเลขคณิตกำหนดให้ A = {*a*, *b*, *c*, *d*} โดยที่ค่าความน่าจะเป็นสำหรับแต่ละสัญลักษณ์มีดังนี้ p(*a*) = 0.3, p(*b*) = 0.2, p(*c*) = 0.4 และ p(*d*) = 0.1 ตามลำดับ เราสามารถแสดงแบบจำลองค่าความน่าจะเป็นได้ดังตารางที่ 3.1

ตารางที่ 3.1 แสดงตารางแบบจำลองค่าความน่าจะเป็น

ลำดับที่	สัญลักษณ์	ค่าความน่าจะเป็น	ผลรวมความน่าจะเป็น	ช่วง
1	a	0.3	0.3	[0.0, 0.3)
2	b	0.2	0.5	[0.3, 0.5)
3	c	0.4	0.9	[0.5, 0.9)
4	d	0.1	1.0	[0.9, 1.0)

สมมติให้บีบอัดข้อความ "*c a c b a d*" โดยกำหนดให้สัญลักษณ์ *d* เป็นสัญลักษณ์สุดท้ายเสมอ สามารถแสดงวิธีการบีบอัดได้ดังรูปที่ 3.3

1. การบีบอัดสัญลักษณ์ *c* สัญลักษณ์ที่หนึ่ง

ที่ตำแหน่ง R(start) เป็นการบีบอัดสัญลักษณ์ *c* โดยขอบเขตของสัญลักษณ์ *c* อยู่ในช่วง [0.5, 0.9) ค่าช่วงของสัญลักษณ์ *c* นี้จะถูกนำไปใช้เป็นขอบเขตบนและขอบเขตล่างสำหรับการบีบอัดของสัญลักษณ์ถัดไป ดังนั้นจำเป็นต้องมีการคำนวณขอบเขตของแต่ละสัญลักษณ์ใหม่ โดยขอบเขตใหม่ของสัญลักษณ์ *a* มีการคำนวณ ดังนี้

$$L_{new} = 0.5 + (0.9 - 0.5) \times 0.0 = 0.5 \quad (3.3)$$

$$H_{new} = 0.5 + (0.9 - 0.5) \times 0.3 = 0.62 \quad (3.4)$$

ดังนั้นค่าขอบเขตใหม่ของสัญลักษณ์ *a* คือ [0.5, 0.62), [0.62, 0.7) สำหรับสัญลักษณ์ *b*, [0.7, 0.86) สำหรับสัญลักษณ์ *c* และ [0.86, 0.9) สำหรับสัญลักษณ์ *d* ตามลำดับ

2. การบีบอัดสัญลักษณ์ *a* สัญลักษณ์ที่สอง

ที่ตำแหน่ง R(*c*) เป็นการบีบอัดสัญลักษณ์ *a* โดยขอบเขตของสัญลักษณ์ *a* อยู่ในช่วง [0.5, 0.62) สามารถแบ่งขอบเขตใหม่ได้เป็น [0.5, 0.536) สำหรับสัญลักษณ์ *a*, [0.536, 0.560) สำหรับสัญลักษณ์ *b*, [0.560, 0.608) สำหรับสัญลักษณ์ *c* และ [0.608, 0.62) สำหรับสัญลักษณ์ *d* ตามลำดับ

3. การบีบอัดสัญลักษณ์ *c* สัญลักษณ์ที่สาม

ที่ตำแหน่ง R(*ca*) เป็นการบีบอัดสัญลักษณ์ *c* โดยขอบเขตของสัญลักษณ์ *c* อยู่ในช่วง [0.560, 0.608) สามารถแบ่งขอบเขตใหม่ได้เป็น [0.560, 0.5744) สำหรับสัญลักษณ์ *a*, [0.5744, 0.5840) สำหรับสัญลักษณ์ *b*, [0.5840, 0.6032) สำหรับสัญลักษณ์ *c* และ [0.6032, 0.608) สำหรับสัญลักษณ์ *d* ตามลำดับ

4. การบีบอัดสัญลักษณ์ *b* สัญลักษณ์ที่สี่

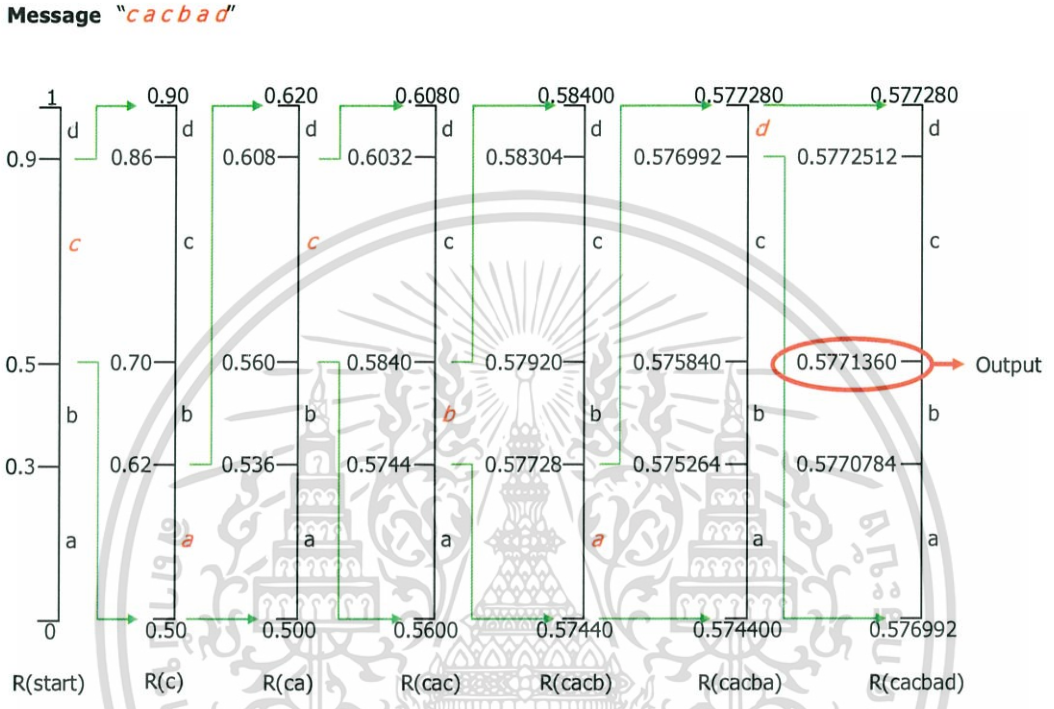
ที่ตำแหน่ง R(*cac*) เป็นการบีบอัดสัญลักษณ์ *b* โดยขอบเขตของสัญลักษณ์ *b* อยู่ในช่วง [0.5744, 0.5840) สามารถแบ่งขอบเขตใหม่ได้เป็น [0.5744, 0.57728) สำหรับสัญลักษณ์ *a*, [0.57728, 0.57920) สำหรับสัญลักษณ์ *b*, [0.57920, 0.58304) สำหรับสัญลักษณ์ *c* และ [0.58304, 0.584) สำหรับสัญลักษณ์ *d* ตามลำดับ

5. การบีบอัดสัญลักษณ์ *a* สัญลักษณ์ที่ห้า

ที่ตำแหน่ง R(*cacb*) เป็นการบีบอัดสัญลักษณ์ *a* โดยขอบเขตของสัญลักษณ์ *a* อยู่ในช่วง [0.5744, 0.57728) สามารถแบ่งขอบเขตใหม่ได้เป็น [0.5744, 0.575264) สำหรับสัญลักษณ์ *a*, [0.575264, 0.575840) สำหรับสัญลักษณ์ *b*, [0.575840, 0.576992) สำหรับสัญลักษณ์ *c* และ [0.576992, 0.57728) สำหรับสัญลักษณ์ *d* ตามลำดับ

6. การบีบอัดสัญลักษณ์ *d* สัญลักษณ์ที่หก

ที่ตำแหน่ง R(*cacba*) เป็นการบีบอัดสัญลักษณ์ *d* โดยขอบเขตของสัญลักษณ์ *d* อยู่ในช่วง [0.576992, 0.57728) ผลลัพธ์ของการบีบอัดข้อความ "*c a c b a d*" คือ จุดกึ่งกลางของขอบเขตสุดท้ายมีค่าเท่ากับ 0.577136 หรือประมาณเท่ากับ 0.577



รูปที่ 3.3 แสดงตัวอย่างการบีบอัดข้อมูลของการเข้ารหัสเลขคณิต

3.2.2 อัลกอริธึมสำหรับการถอดรหัสด้วยวิธีการเข้ารหัสเลขคณิต

อัลกอริธึมสำหรับการถอดรหัสมีรูปแบบเหมือนกับอัลกอริธึมสำหรับการบีบอัดข้อมูล การถอดรหัสเริ่มจากการกำหนดค่าขอบเขตเป็น [0.0, 1.0) และทำการแบ่งขอบเขตของแต่ละสัญลักษณ์เป็น [0.0, 0.3) สำหรับสัญลักษณ์ *a*, [0.3, 0.5) สำหรับสัญลักษณ์ *b*, [0.5, 0.9) สำหรับสัญลักษณ์ *c* และ [0.9, 1.0) สำหรับสัญลักษณ์ *d* เนื่องจากผลลัพธ์จากการบีบอัด คือ 0.577 ซึ่งค่า 0.577 อยู่ในช่วง [0.5, 0.9) สามารถถอดรหัสได้สัญลักษณ์ *c*

จากนั้นทำการแบ่งขอบเขตใหม่จากผลรวมค่าความน่าจะเป็นของสัญลักษณ์ *c* ได้เป็น [0.5, 0.62) สำหรับสัญลักษณ์ *a*, [0.62, 0.70) สำหรับสัญลักษณ์ *b*, [0.70, 0.86) สำหรับสัญลักษณ์ *c* และ [0.86, 0.9) สำหรับสัญลักษณ์ *d* ตามลำดับ ค่า 0.577 อยู่ในช่วง [0.5, 0.62) สามารถถอดรหัสได้สัญลักษณ์ *a*

จากนั้นทำการแบ่งขอบเขตใหม่จากผลรวมค่าความน่าจะเป็นของสัญลักษณ์ *a* ได้เป็น [0.5, 0.536) สำหรับสัญลักษณ์ *a*, [0.536, 0.560) สำหรับสัญลักษณ์ *b*, [0.560, 0.608) สำหรับสัญลักษณ์ *c* และ [0.608, 0.62) สำหรับสัญลักษณ์ *d* ตามลำดับ ค่า 0.577 อยู่ในช่วง [0.560, 0.608) สามารถถอดรหัสได้สัญลักษณ์ *c*

จากนั้นทำการแบ่งขอบเขตใหม่จากผลรวมค่าความน่าจะเป็นของสัญลักษณ์ *c* ได้เป็น [0.560, 0.5744) สำหรับสัญลักษณ์ *a*, [0.5744, 0.584) สำหรับสัญลักษณ์ *b*, [0.584, 0.6032) สำหรับสัญลักษณ์ *c* และ [0.6032, 0.608) สำหรับสัญลักษณ์ *d* ตามลำดับ ค่า 0.577 อยู่ในช่วง [0.5744, 0.584) สามารถถอดรหัสได้สัญลักษณ์ *b*

จากนั้นทำการแบ่งขอบเขตใหม่จากผลรวมค่าความน่าจะเป็นของสัญลักษณ์ *b* ได้เป็น [0.5744, 0.57728) สำหรับสัญลักษณ์ *a*, [0.57728, 0.5792) สำหรับสัญลักษณ์ *b*, [0.5792, 0.58304) สำหรับสัญลักษณ์ *c* และ [0.58304, 0.584) สำหรับสัญลักษณ์ *d* ตามลำดับ ค่า 0.577 อยู่ในช่วง [0.5744, 0.57728) สามารถถอดรหัสได้สัญลักษณ์ *a*

จากนั้นทำการแบ่งขอบเขตใหม่จากผลรวมค่าความน่าจะเป็นของสัญลักษณ์ *a* ได้เป็น [0.5744, 0.575264) สำหรับสัญลักษณ์ *a*, [0.575264, 0.575840) สำหรับสัญลักษณ์ *b*, [0.575840, 0.576992) สำหรับสัญลักษณ์ *c* และ [0.576992, 0.57728) สำหรับสัญลักษณ์ *d* ตามลำดับ ค่า 0.577 อยู่ในช่วง [0.576992, 0.57728) สามารถถอดรหัสได้สัญลักษณ์ *d* ดังนั้นผลลัพธ์ของการถอดรหัส คือ "*c a c b a d*"

3.3 การเข้ารหัสเลขคณิตแบบไบนารี

การบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตแบบไบนารีมีพื้นฐานมาจากการบีบอัดข้อมูลวิธีการเข้ารหัสเลขคณิตแต่พัฒนาให้ผลลัพธ์จากการบีบอัดจะอยู่ในรูปของเลขฐานสองแทนเลขทศนิยมแบบวิธีการเข้ารหัสเลขคณิตทำให้เหมาะสำหรับการประยุกต์ใช้ในการคำนวณบนฮาร์ดแวร์

3.3.1 อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตแบบไบนารี

อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตแบบไบนารีจะใช้กฎตามรูปที่ 3.4 ในการบีบอัด โดยตัวแปร LOW แทนค่าขอบเขตล่าง ตัวแปร HIGH แทนค่าขอบเขตบน และค่า SPCL_COUNT แทนค่าตัวนับพิเศษ

กรณีที่ 1: ถ้าค่า HIGH < 0.5 ให้ส่งผลลัพธ์ค่า 0 ออกไปและทำการปรับค่าขอบเขตใหม่เป็น [2 × LOW, 2 × HIGH)

กรณีที่ 2: ถ้าค่า LOW ≥ 0.5 ให้ส่งผลลัพธ์ค่า 1 ออกไปและทำการปรับค่าขอบเขตใหม่เป็น [2 × (LOW - 0.5), 2 × (HIGH - 0.5))

กรณีที่ 3: ถ้าค่า LOW ≥ 0.25 และค่า HIGH < 0.75 ให้เพิ่มค่า SPCL_COUNT ขึ้น 1 และทำการปรับค่าขอบเขตใหม่เป็น [LOW - 0.25, HIGH - 0.25) เมื่อใดก็ตามที่เกิดกรณีที่ 1 หรือกรณีที่ 2 ขึ้น ให้ตรวจสอบค่า SPCL_COUNT ว่ามีค่าเท่ากับ 1 หรือไม่

- ถ้ามีค่าเท่ากับกว่า 1 ให้ส่งผลลัพธ์ออกภายหลังจากส่งผลลัพธ์ของกรณีที่ 1 หรือกรณีที่ 2 ออกไป
  - ถ้าเป็นกรณีที่ 1 ให้ส่งผลลัพธ์ค่า 1 ออกไป
  - ถ้าเป็นกรณีที่ 2 ให้ส่งผลลัพธ์ค่า 0 ออกไป

รูปที่ 3.4 แสดงกฎการเข้ารหัสเลขคณิตแบบไบนารี

ตัวอย่างการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตแบบไบนารีจะใช้ตัวอย่างเดียวกับการบีบอัดข้อมูลด้วยวิธีการเข้ารหัสเลขคณิตสามารถแสดงวิธีการบีบอัดข้อมูลได้ดังรูปที่ 3.5 และรูปที่ 3.6 เริ่มตั้งแต่การกำหนดค่าเริ่มต้นตัวแปร LOW = 0.0, HIGH = 1.0 และ SPCL_COUNT = 0

ที่ตำแหน่ง R0 การบีบอัดเริ่มจากสัญลักษณ์ *c* อยู่ในช่วง [0.5, 0.9) ตรงกับกรณีที่ 2 ให้ส่งผลลัพธ์ค่า 1 ออกไปที่ตำแหน่ง R1 ทำการปรับค่าขอบเขตใหม่เป็น [0.0, 0.8)

ที่ตำแหน่ง R2 ต่อมาเป็นการบีบอัดสัญลักษณ์ *a* อยู่ในช่วง [0.0, 0.24) ตรงกับกรณีที่ 1 ให้ส่งผลลัพธ์ค่า 0 ออกไป

ที่ตำแหน่ง R3 ทำการปรับค่าขอบเขตใหม่เป็น [0.0, 0.48)

ที่ตำแหน่ง R4 ซึ่งตรงกับกรณีที่ 1 ให้ส่งผลลัพธ์ค่า 0 ออกไป และทำการปรับค่าขอบเขตใหม่เป็น [0.0, 0.96)

ที่ตำแหน่ง R5 ต่อมาเป็นการบีบอัดสัญลักษณ์ *c* อยู่ในช่วง [0.48, 0.864) ไม่ตรงกับกรณีใดๆ

ที่ตำแหน่ง R6 ต่อมาเป็นการบีบอัดสัญลักษณ์ *b* อยู่ในช่วง [0.672, 0.5952) ตรงกับกรณีที่ 2 ให้ส่งผลลัพธ์ค่า 1 ออกไป

ที่ตำแหน่ง R7 ซึ่งตรงกับกรณีที่ 1 ให้ส่งผลลัพธ์ค่า 0 ออกไป และทำการปรับค่าขอบเขตใหม่เป็น [0.1904, 0.344)

ที่ตำแหน่ง R8 ซึ่งตรงกับกรณีที่ 1 ให้ส่งผลลัพธ์ค่า 0 ออกไป และทำการปรับค่าขอบเขตใหม่เป็น [0.3808, 0.688)

ที่ตำแหน่ง R9 ซึ่งตรงกับกรณีที่ 3 ให้เพิ่มค่า SPCL_COUNT ขึ้น 1 และทำการปรับค่าขอบเขตใหม่เป็น [0.2616, 0.876)

ที่ตำแหน่ง R10 ต่อมาเป็นการบีบอัดสัญลักษณ์ *a* อยู่ในช่วง [0.2616, 0.44592) ตรงกับกรณีที่ 1 ให้ส่งผลลัพธ์ค่า 0 ออกไป

ที่ตำแหน่ง R11 ทำการปรับค่าขอบเขตใหม่เป็น [0.5232, 0.89184) แต่เนื่องจาก SPCL_COUNT มีค่าเท่ากับ 1 กรณีล่าสุด คือ กรณีที่ 1 ดังนั้นให้ส่งผลลัพธ์ค่า 1 ออกไป และกำหนดค่า SPCL_COUNT เป็น 0

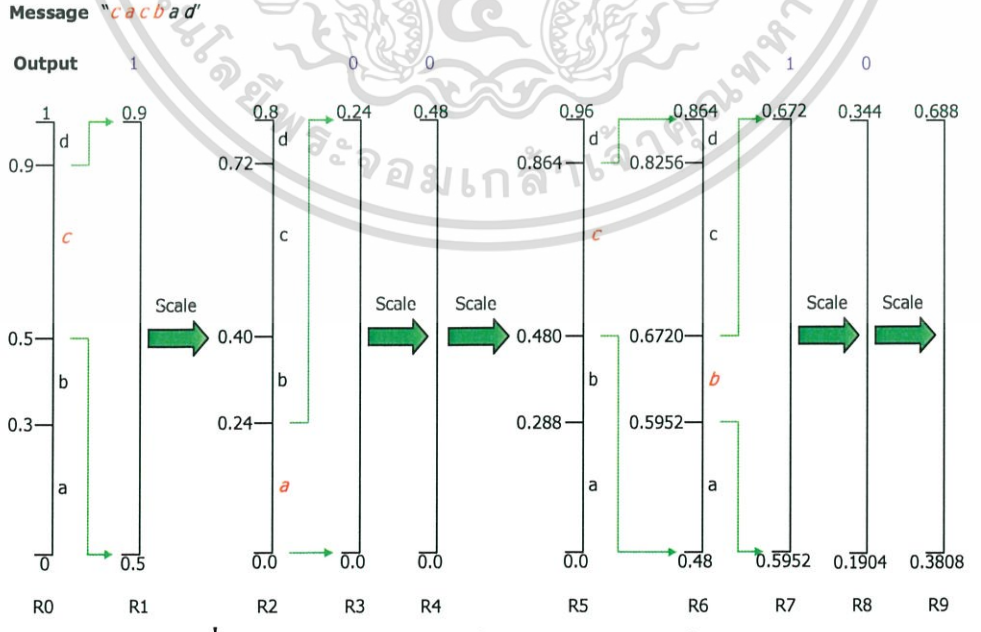
ที่ตำแหน่ง R12 ทำการปรับค่าขอบเขตใหม่เป็น [0.0464, 0.78368)

ที่ตำแหน่ง R13 สุดท้ายเป็นการบีบอัดสัญลักษณ์ *d* อยู่ในช่วง [0.709952, 0.78368) ตรงกับกรณีที่ 2 ให้ส่งผลลัพธ์ค่า 1 ออกไป

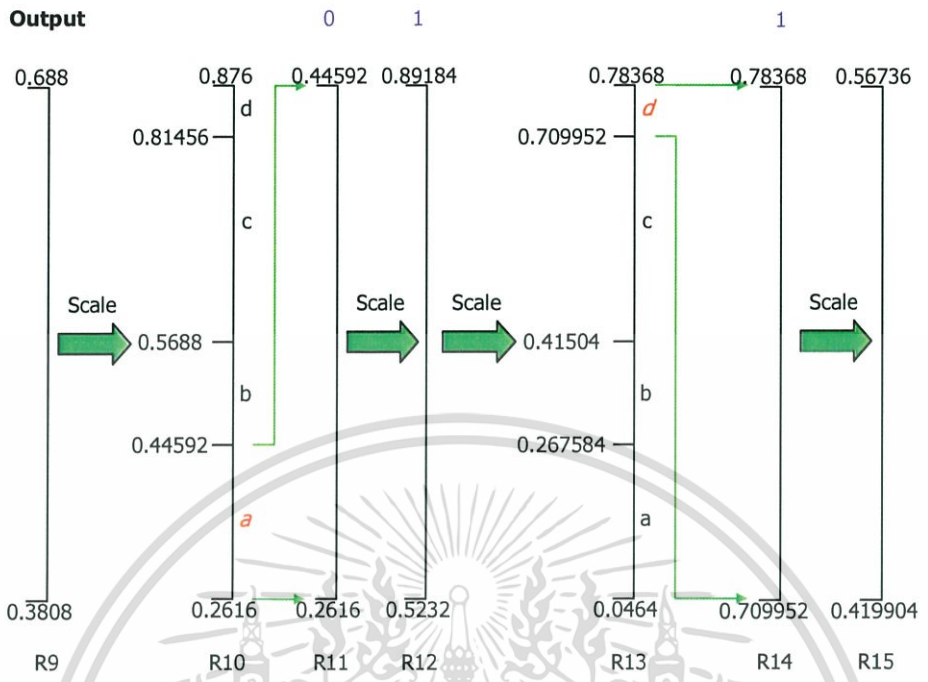
ที่ตำแหน่ง R14 ทำการปรับค่าขอบเขตใหม่เป็น [0.419904, 0.56736)

ที่ตำแหน่ง R15 เราสามารถหยุดการบีบอัดได้ ณ จุดนี้

หลังจากนั้นให้เลือกค่าขอบเขตสุดท้ายมาหนึ่งค่า ในที่นี้เลือกค่า 0.5 หรือ 0.1000... ดังนั้นเราจะทำการส่งค่า 1 นำแล้วตามด้วยค่า 0 โดยที่จำนวนของ 0 ขึ้นอยู่กับขนาดของ word เนื่องจากผลลัพธ์จากการบีบอัด คือ “1 0 0 1 0 0 1 1 1” มีขนาด 9 บิต สมมติให้ word มีขนาดเท่ากับ 16 บิต ดังนั้นเมื่อนำผลลัพธ์จากการบีบอัดมาต่อกับขอบเขตที่เลือกจะได้ผลลัพธ์เป็น “1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0”



รูปที่ 3.5 แสดงตัวอย่างการเข้ารหัสเลขคณิตแบบไบนารี (1)



รูปที่ 3.6 แสดงตัวอย่างการเข้ารหัสเลขคณิตแบบไบนารี (2)

3.3.2 อัลกอริธึมสำหรับการถอดรหัส

การถอดรหัสเริ่มจากการนำ 8 บิตแรกเก็บในตัวแปร C = 10010011 แทนเป็นเลขทศนิยมได้ 0.57421875 อยู่ในช่วง [0.5, 0.9) ถอดรหัสได้สัญลักษณ์ *c*

ที่ตำแหน่ง R1 ตรงกับกรณีที่ 2 ทำการปรับค่าขอบเขตใหม่เป็น [0.0, 0.8)

ที่ตำแหน่ง R2 ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนไปแทนบิต LSB จะได้ C = 00100111 แทนเป็นเลขทศนิยมได้ 0.15234375 อยู่ในช่วง [0.0, 0.24) ถอดรหัสได้สัญลักษณ์ *a*

ที่ตำแหน่ง R3 ตรงกับกรณีที่ 1 ทำการปรับค่าขอบเขตใหม่เป็น [0.0, 0.48)

ที่ตำแหน่ง R4 ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนทิ้ง ส่วนบิต LSB ถูกเลื่อนแทนด้วย 1 จะได้ C = 01001111 แทนเป็นเลขทศนิยมได้ 0.30859375 ตรงกับกรณีที่ 1 ทำการปรับค่าขอบเขตใหม่เป็น [0.0, 0.96)

ที่ตำแหน่ง R5 ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนทิ้ง ส่วนบิต LSB ถูกเลื่อนแทนด้วย 0 จะได้ C = 10011110 แทนเป็นเลขทศนิยมได้ 0.61718750 อยู่ในช่วง [0.48, 0.864) ถอดรหัสได้สัญลักษณ์ *c*

ที่ตำแหน่ง R6 ไม่ตรงกับกรณีใดๆ ไม่ต้องทำการปรับค่าขอบเขตใหม่ ค่าตัวแปร C แทนเป็นเลขทศนิยม 0.61718750 อยู่ในช่วง [0.5952, 0.672) ถอดรหัสได้สัญลักษณ์ *b*

ที่ตำแหน่ง R7 ตรงกับกรณีที่ 2 ทำการปรับค่าขอบเขตใหม่เป็น [0.1904, 0.344)

ที่ตำแหน่ง R8 ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนทิ้ง ส่วนบิต LSB ถูกเลื่อนแทนด้วย 0 จะได้ C = 00111100 ขอบเขต R8 ตรงกับกรณีที่ 1 ทำการปรับค่าขอบเขตใหม่เป็น [0.3808, 0.688)

ที่ตำแหน่ง R9 ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนทิ้ง ส่วนบิต LSB ถูกเลื่อนแทนด้วย 0 จะได้ C = 01111000 ขอบเขต R9 ตรงกับกรณีที่ 3 ให้เพิ่มค่า SPCL_COUNT ขึ้น 1 และทำการปรับค่าขอบเขตใหม่เป็น [0.2616, 0.876)

ที่ตำแหน่ง R10 ให้ลดค่า SPCL_COUNT ลง 1 และลบบิต Adjacent (ถูกขีดเส้นใต้) ในตัวแปร C = 01111000 และแทนบิต LSB ด้วย 0 จะได้ C = 01110000 แทนเป็นเลขทศนิยมได้ 0.4375 อยู่ในช่วง [0.2616, 0.44592) ถอดรหัสได้สัญลักษณ์ *a*

ที่ตำแหน่ง R11 ตรงกับกรณีที่ 1 ทำการปรับค่าขอบเขตใหม่เป็น [0.5232, 0.89184) ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนทิ้ง ส่วนบิต LSB ถูกเลื่อนแทนด้วย 0 จะได้ C = 11100000

ที่ตำแหน่ง R12 ตรงกับกรณีที่ 2 ทำการปรับค่าขอบเขตใหม่เป็น [0.0464, 0.78368) ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนทิ้ง ส่วนบิต LSB ถูกเลื่อนแทนด้วย 0 จะได้ C = 11000000

ที่ตำแหน่ง R13 แทนตัวแปร C เป็นเลขทศนิยมได้ 0.75 อยู่ในช่วง [0.709952, 0.78368) ถอดรหัสได้สัญลักษณ์ *d* หยุดทำการถอดรหัสเพราะสัญลักษณ์ *d* เป็นสัญลักษณ์สุดท้าย

จากนั้นทำการตรวจสอบความถูกต้องของ 8 บิตสุดท้ายในตัวแปร C ที่ตำแหน่ง R14 ตรงกับกรณีที่ 2 ทำการปรับค่าขอบเขตใหม่เป็น [0.419904, 0.56736) ทำการเลื่อนตัวแปร C ไปทางซ้าย โดยบิต MSB ถูกเลื่อนทิ้ง ส่วนบิต LSB ถูกเลื่อนแทนด้วย 0 จะได้ C = 10000000

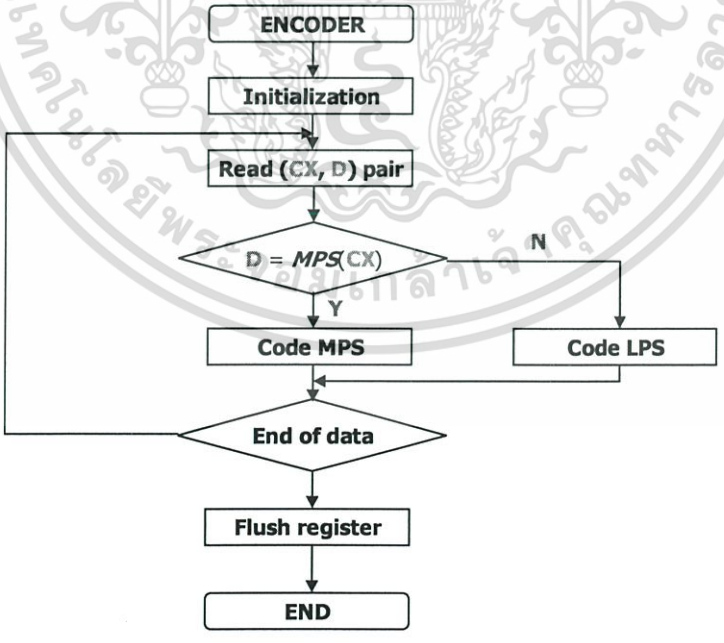
ที่ตำแหน่ง R15 แทนตัวแปร C เป็นเลขทศนิยมได้ 0.5 ซึ่งตรงกับค่าสุดท้ายของการบีบอัด ถือว่าเสร็จสิ้นการถอดรหัส

3.4 การบีบอัดข้อมูลด้วยวิธีส่วนเข้ารหัสเอ็มคิว

การบีบอัดข้อมูลด้วยวิธีส่วนเข้ารหัสเอ็มคิวมีพื้นฐานมาจากการบีบอัดข้อมูลวิธีการเข้ารหัสเลขคณิตแบบไบนารีมีการพัฒนาอัลกอริธึมและใช้เลขฐานสองในการคำนวณทั้งหมด ทำให้ออกแบบฮาร์ดแวร์ได้ง่ายขึ้น

3.4.1 อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีส่วนเข้ารหัสเอ็มคิว

อัลกอริธึมสำหรับการบีบอัดข้อมูลด้วยวิธีส่วนเข้ารหัสเอ็มคิวแสดงขั้นตอนการทำงานในรูปที่ 3.7 การทำงานเริ่มจากกำหนดค่าเริ่มต้นให้กับตัวแปร (Initialization) ซึ่งสามารถแสดงในรูปแบบของโค้ดเทียมในรูปที่ 3.8 จากนั้นทำการอ่านค่าสัญลักษณ์จากส่วนเข้ารหัสแบบระนาบบิตเข้ามาประมวลผลครั้งละหนึ่งสัญลักษณ์ โดยหนึ่งสัญลักษณ์ประกอบด้วยค่า CX และค่า D จากนั้นนำค่า CX ไปอ่านค่าจากตาราง I และตาราง MPS ผลลัพธ์จากตาราง I จะถูกนำไปอ่านค่าจากตาราง NLPS, NMPS, Qe และ SWITCH ผลลัพธ์จากตารางเหล่านี้จะถูกนำไปใช้ในการคำนวณค่าขอบเขตบน (A) และค่าขอบเขตล่าง (C) ส่วนผลลัพธ์จากตาราง MPS จะถูกนำไปเปรียบเทียบกับค่า D เพื่อเลือกการบีบอัดแบบ MPS หรือแบบ LPS แสดงโค้ดเทียม (Pseudo Code)การบีบอัดแบบ MPS ในรูปที่ 3.9 และแบบ LPS ในรูปที่ 3.10 ภายหลังการบีบอัดแบบ MPS หรือแบบ LPS ถ้าค่า A น้อยกว่า 0x8000 จะต้องทำการรีนอร์มอลไลซ์เพื่อปรับค่า A ให้มากกว่า 0x8000 โดยจำนวนรอบของการทำรีนอร์มอลไลซ์จะขึ้นอยู่กับค่า A และทุกครั้งที่ทำการรีนอร์มอลไลซ์จะลดค่า CT ลงหนึ่ง ซึ่งค่า CT ใช้เป็นตัวนับจำนวนรอบของการทำรีนอร์มอลไลซ์ เมื่อใดก็ตามที่ค่า CT ลดลงเท่ากับศูนย์ ให้ส่งผลลัพธ์จากการบีบอัดออกไปยังส่วนเข้ารหัสไทเออร์ทูแต่ถ้าค่า A มากกว่าเท่ากับ 0x8000 ให้ตรวจสอบว่ายังมีสัญลักษณ์เหลืออยู่อีกหรือไม่ หากยังมีสัญลักษณ์อยู่ให้กลับไปอ่านค่าสัญลักษณ์ใหม่เพื่อบีบอัดต่อไป แต่ถ้าไม่มีสัญลักษณ์ให้ทำการนำผลลัพธ์ที่อยู่ในค่า C ส่งไปเก็บในค่า B เพื่อจบการบีบอัดข้อมูลของโค้ดบล็อกนั้นๆ



รูปที่ 3.7 แสดงแผนผังการทำงานการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิว

```
A = 0x8000
C = 0x00000000

IF B = 0xFF THEN
    CT = 13
ELSE
    CT = 12
ENDIF
```

รูปที่ 3.8 แสดงโค้ดเทียมการกำหนดค่าเริ่มต้นของส่วนเข้ารหัสเอ็มคิว

```
CodeMPS()
BEGIN
    qe = Qe(I(CX))
    A = A - qe

    IF A < 0x8000 THEN
        IF A < qe THEN
            A = qe
        ELSE
            C = C + qe
            CALL Renormalization()
        ENDIF
    ELSE
        C = C + qe
    ENDIF
END
```

รูปที่ 3.9 แสดงโค้ดเทียมการบีบอัดข้อมูลแบบเอ็มพีเอสของส่วนเข้ารหัสเอ็มคิว

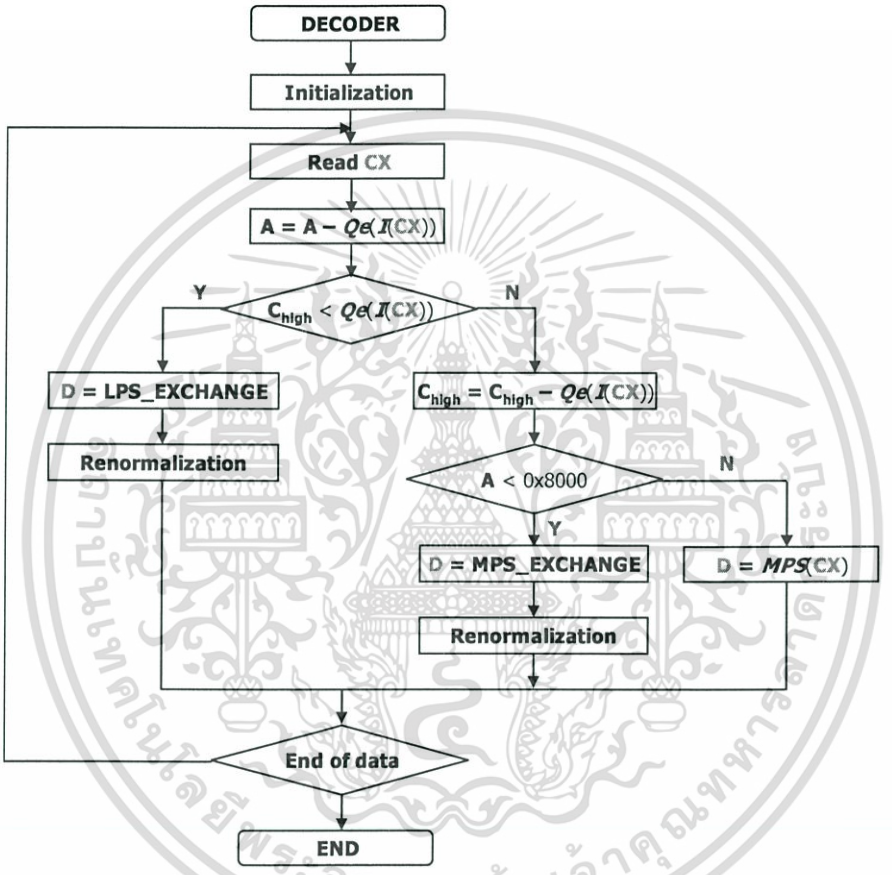
```
CodeLPS()
BEGIN
    qe = Qe(I(CX))
    A = A - qe

    IF A < qe THEN
        C = C + qe
    ELSE
        A = qe
    ENDIF

    CALL Renormalization()
END
```

รูปที่ 3.10 แสดงโค้ดเทียมการบีบอัดข้อมูลแบบแอลพีเอสของส่วนเข้ารหัสเอ็มคิว

3.4.2 อัลกอริธึมสำหรับการถอดรหัสด้วยวิธีส่วนเข้ารหัสเอ็มคิว

อัลกอริธึมสำหรับการถอดรหัสด้วยวิธีส่วนเข้ารหัสเอ็มคิวแสดงขั้นตอนการทำงานในรูปที่ 3.11 การทำงานเริ่มจากกำหนดค่าเริ่มต้นให้กับตัวแปร จากนั้นทำการอ่านค่า CX เพื่อนำไปอ่านค่า

จากตาราง I และตาราง MPS ผลลัพธ์จากตาราง I จะถูกนำไปอ่านค่าจากตาราง NLPS, NMPS, Qe และ SWITCH ผลลัพธ์จากตารางเหล่านี้จะถูกนำไปใช้ในการคำนวณค่าขอบเขตบน (A) และค่าขอบเขตล่าง (C) โดยผลลัพธ์จากตาราง Qe จะถูกนำไปเปรียบเทียบกับค่า C_{HIGH} เพื่อเลือกการถอดรหัสแบบ MPS_EXCHANGE หรือแบบ LPS_EXCHANGE เมื่อถอดรหัสเสร็จเรียบร้อยแล้วให้ตรวจสอบว่ายังมีค่า CX เหลืออยู่อีกหรือไม่ หากยังมีค่า CX อยู่ให้กลับไปอ่านค่า CX ใหม่เพื่อถอดรหัสต่อไป แต่ถ้าไม่มีค่า CX ให้จบการถอดรหัสของโค้ดบล็อกนั้นๆ



รูปที่ 3.11 แสดงแผนผังการทำงานการถอดรหัสของส่วนเข้ารหัสเอ็มคิว

บทที่ 4
การออกแบบส่วนเข้ารหัสเอ็มคิวตามมาตรฐาน JPEG2000 สำหรับการใช้งานประเภทเรียลไทม์

ในปัจจุบันมีความต้องการนำมาตรฐาน JPEG2000 ไปประยุกต์ใช้งานประเภทเรียลไทม์เป็นจำนวนมาก แต่การออกแบบฮาร์ดแวร์ตามมาตรฐาน JPEG2000 ทำได้ยาก เนื่องจากความซับซ้อนของการทำงานตามมาตรฐาน JPEG2000 และหนึ่งในส่วนสำคัญในการบีบอัดข้อมูลตามมาตรฐาน JPEG2000 ได้แก่ ส่วนของการเข้ารหัสส่วนเข้ารหัสเอ็มคิวซึ่งถูกออกแบบเพื่อให้สามารถทำการบีบอัดข้อมูลได้ประสิทธิภาพสูง ส่งผลให้การทำงานมีความซับซ้อนสูง และใช้เวลาประมวลผลมาก

การทำงานของส่วนเข้ารหัสเอ็มคิวเป็นแบบลำดับขั้นคือ สามารถประมวลผลได้ครั้งละหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา จากงานวิจัย [3] ส่วนเข้ารหัสแบบระนาบบิตสามารถสร้างสัญลักษณ์ได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา ทำให้ส่วนเข้ารหัสเอ็มคิวไม่สามารถทำงานได้ทันกับส่วนเข้ารหัสแบบระนาบบิตที่เป็นขั้นตอนการเข้ารหัสซึ่งอยู่ก่อนหน้า ส่งผลให้เกิดปัญหาคอขวดระหว่างเอาต์พุตของส่วนเข้ารหัสแบบระนาบบิตและอินพุตของส่วนเข้ารหัสเอ็มคิว ดังนั้นการออกแบบส่วนเข้ารหัสเอ็มคิว ให้สามารถประมวลผลได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา จึงเป็นแนวทางหนึ่งในการลดปัญหาคอขวดระหว่างส่วนเข้ารหัสแบบระนาบบิตและส่วนเข้ารหัสเอ็มคิว ซึ่งจะช่วยให้ลดระยะเวลาในการเข้ารหัสของส่วนเข้ารหัสเอ็มคิว และทำให้การเข้ารหัสของส่วนเข้ารหัสเอ็มคิวมีประสิทธิภาพดียิ่งขึ้นไป

การปรับการทำงานของส่วนเข้ารหัสเอ็มคิวจากแบบลำดับขั้นให้เป็นแบบขนาน เพื่อให้สามารถประมวลผลได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกานั้น จำเป็นต้องมีการออกแบบส่วนในการคาดเดาผลลัพธ์จากการประมวลผลของสัญลักษณ์ก่อนหน้า เพื่อนำผลลัพธ์จากการคาดเดาก่อนหน้ามาประมวลผลร่วมกับสัญลักษณ์ปัจจุบัน โดยส่วนการคาดเดาผลลัพธ์ที่ออกแบบเพิ่มเติมนี้ จะยังคงความถูกต้องของผลลัพธ์ของการเข้ารหัสเช่นเดียวกับการทำงานแบบลำดับขั้นแบบเดิม

จากสถาปัตยกรรมของงานวิจัย [2] ถึงแม้จะใช้เทคนิคการประมวลผลสัญลักษณ์คู่ การคาดเดาผลลัพธ์ของค่าขอบเขตบน (A) ค่าขอบเขตล่าง (C) ค่าความน่าจะเป็น ค่าดัชนีของตาราง I และตาราง MPS อีกทั้งยังมีการแยกส่วนนำส่งผลลัพธ์ออกจากส่วนคำนวณค่า C อย่างไรก็ดี สำหรับสถาปัตยกรรมของงานวิจัย [2] ยังสามารถปรับปรุงส่วนคาดเดาผลลัพธ์ค่า A, ค่า C และส่วนนำส่ง

ผลลัพธ์ให้มีประสิทธิภาพดียิ่งขึ้นได้ โดยส่วนคาดเดาผลลัพธ์ใหม่ที่เพิ่มเติมเข้าไป จะช่วยเพิ่มความสามารถในการคาดเดาผลลัพธ์และลดเวลาที่ใช้ในการประมวลผลลง

ดังนั้นงานวิจัยนี้จึงได้นำเสนอสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวใหม่ ที่สามารถประมวลผลได้ครั้งละสองสัญลักษณ์ต่อสัญญาณนาฬิกา โดยทำการปรับปรุงส่วนคาดเดาผลลัพธ์ของค่า A, ค่า C และส่วนนำส่งผลลัพธ์ให้มีประสิทธิภาพดียิ่งขึ้น ซึ่งสถาปัตยกรรมที่นำเสนอนี้แสดงได้ดังรูปที่ 4.1 สถาปัตยกรรมที่นำเสนอแบ่งการทำงานออกเป็น 5 ส่วน ดังนี้

1) ส่วนที่ 1 หรือส่วนตารางความน่าจะเป็น (Probability Estimate Table Unit) เป็นส่วนของตารางข้อมูลทั้งหมด ค่าที่อ่านได้จากตารางจะถูกนำไปใช้สำหรับการคำนวณค่า A, ค่า C การอ่านค่าและการอัพเดตตาราง I และตาราง MPS

2) ส่วนที่ 2 หรือส่วนการประมวลผลขอบเขตบน (Upper Bound Unit) เป็นส่วนคำนวณค่า A (Upper bound) สถาปัตยกรรมที่นำเสนอได้มีการเพิ่มส่วนคาดเดาผลลัพธ์จากการคำนวณค่า A เพื่อปรับการคำนวณจากแบบลำดับขั้นเป็นแบบขนาน ช่วยลดเวลาที่ใช้ในการประมวลผล

3) ส่วนที่ 3 หรือส่วนการประมวลผลขอบเขตล่าง (Lower Bound Unit) เป็นส่วนคำนวณค่า C (Lower bound) สถาปัตยกรรมที่นำเสนอได้มีการเพิ่มส่วนคาดเดาผลลัพธ์จากการคำนวณค่า C เพื่อปรับการคำนวณจากแบบลำดับขั้นเป็นแบบขนาน ช่วยลดเวลาที่ใช้ในการประมวลผล

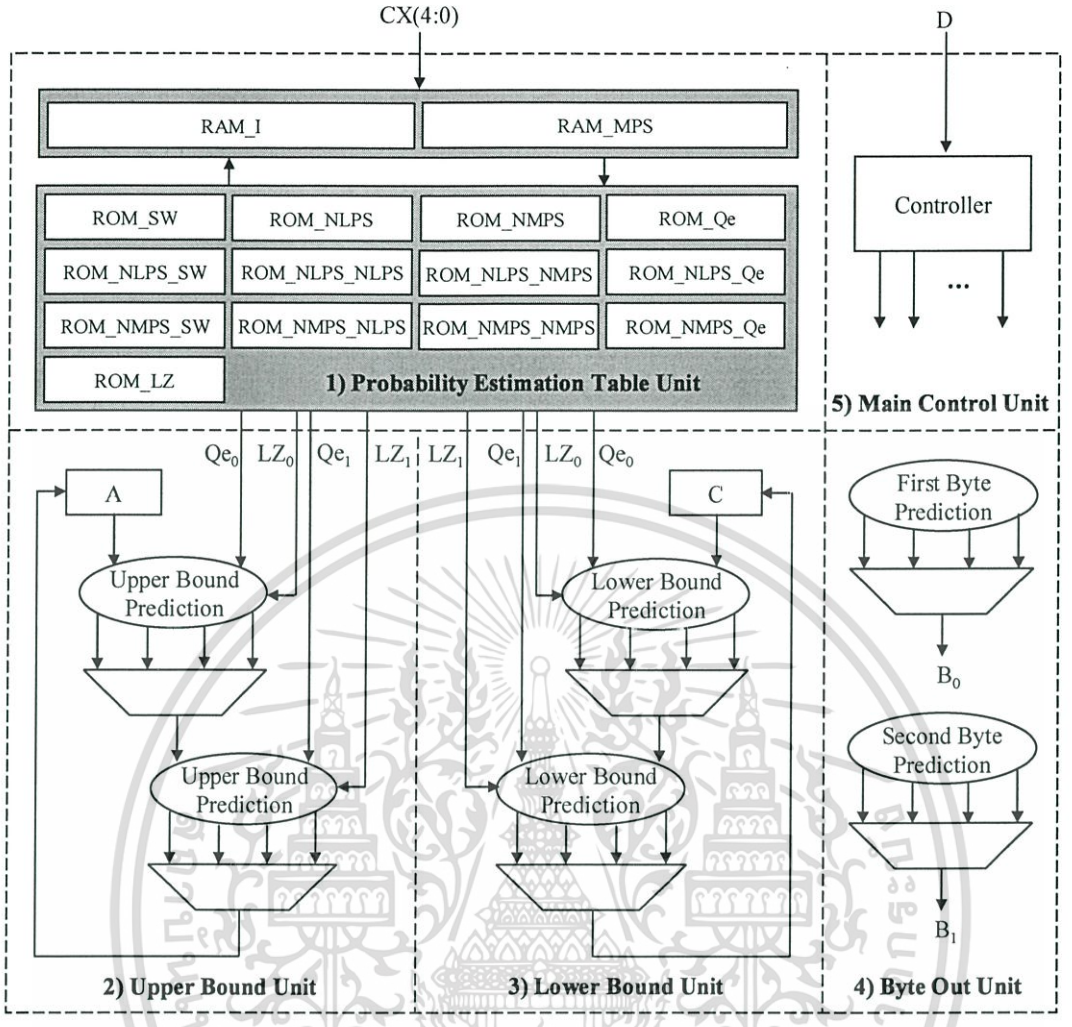
4) ส่วนที่ 4 หรือส่วนนำส่งผลลัพธ์ (Byte Out Unit) เป็นส่วนจัดการผลลัพธ์และส่งต่อไปยังส่วนเข้ารหัสไทเออร์ทูเพื่อนำผลลัพธ์จากการบีบอัดข้อมูลไปจัดเรียงให้อยู่ในรูปของแพ็คเกจก่อนนำไปใช้งานในรูปแบบต่างๆ ต่อไป

5) ส่วนที่ 5 หรือส่วนควบคุมหลัก (Main Control Unit) เป็นส่วนควบคุมการทำงานทั้งหมดของวงจร เริ่มตั้งแต่การอ่านค่าสัญลักษณ์เข้ามาประมวลผล จนกระทั่งส่งผลลัพธ์จากการบีบอัดไปยังส่วนถัดไป

เนื่องจากสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวที่นำเสนอนี้ มีการปรับปรุงโครงสร้างและการทำงานเพิ่มเติมจากงานวิจัย [2] ให้มีประสิทธิภาพดียิ่งขึ้น สามารถประมวลผลได้ครั้งละสองสัญลักษณ์ต่อสัญญาณนาฬิกา สามารถอธิบายการปรับปรุงโครงสร้างและการทำงานในแต่ละส่วนได้ดังต่อไปนี้

4.1 การออกแบบส่วนตารางความน่าจะเป็นของส่วนเข้ารหัสเอ็มคิว

จากสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวที่นำได้เสนอสามารถประมวลผลได้ครั้งละสองสัญลักษณ์ต่อสัญญาณนาฬิกานั้น ส่งผลต่อการออกแบบส่วนตารางข้อมูลของส่วนเข้ารหัสเอ็มคิว การออกแบบจำเป็นต้องคำนึงถึงเรื่องความเร็วในการอ่านค่าจากตารางและความน่าจะเป็นของค่าต่างๆ ที่จะถูกนำไปใช้ประมวลผลว่ามีโอกาสเกิดค่าใดขึ้นได้บ้าง ทำให้จำเป็นต้องเพิ่มตารางสำหรับการคาดเดาค่าที่จะถูกนำไปใช้ประมวลผล เพื่อเพิ่มความถูกต้องในการประมวลผล



รูปที่ 4.1 แสดงสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวที่นำเสนอ

ส่วนตารางความน่าจะเป็นประกอบด้วยตารางพื้นฐาน 6 ตาราง คือ ตาราง I, ตาราง MPS, ตาราง NLPS, ตาราง NMPS, ตาราง Qe และตาราง SWITCH แต่เนื่องจากความต้องการดังที่กล่าวไว้ข้างต้น ทำให้มีการออกแบบตารางเพิ่มขึ้นอีก 9 ตาราง คือ ตาราง LZ_Qe, ตาราง NLPS_ NLPS, ตาราง NMPS_ NLPS, ตาราง NLPS_ NMPS, ตาราง NMPS_ NMPS, ตาราง NLPS_Qe, ตาราง NMPS_Qe, ตาราง NLPS_SWITCH และตาราง NMPS_SWITCH สามารถอธิบายแต่ละตารางได้ดังต่อไปนี้

ตารางพื้นฐาน 6 ตารางประกอบด้วย

1) ตาราง I เป็นตารางเก็บค่าดัชนี ใช้สำหรับอ่านค่าจากตาราง NLPS, ตาราง NMPS, ตาราง Qe, ตาราง SWITCH, ตาราง LZ_Qe, ตาราง NLPS_ NLPS, ตาราง NMPS_ NLPS, ตาราง NLPS_ NMPS, ตาราง NMPS_ NMPS, ตาราง NLPS_Qe, ตาราง NMPS_Qe, ตาราง NLPS_SWITCH และตาราง NMPS_SWITCH

2) ตาราง MPS เป็นตารางเก็บค่าการบีบอัดข้อมูลแบบ Code MPS

3) ตาราง NLPS เป็นตารางเก็บค่าดัชนีถัดไปสำหรับการบีบอัดข้อมูลแบบ Code LPS

4) ตาราง NMPS เป็นตารางเก็บค่าดัชนีถัดไปสำหรับการบีบอัดข้อมูลแบบ Code MPS

5) ตาราง Qe เป็นตารางเก็บค่าความน่าจะเป็น ใช้สำหรับการคำนวณค่า A และค่า C

6) ตาราง SWITCH เป็นตารางเก็บค่าการเปลี่ยนรูปแบบการบีบอัดข้อมูลจากแบบ Code LPS เป็นแบบ Code MPS หรือจากแบบ Code MPS เป็นแบบ Code LPS

ตารางที่ออกแบบเพิ่มเติม 9 ตารางประกอบด้วย

1) ตาราง LZ_Qe เป็นตารางเก็บจำนวนค่าศูนย์นำหน้าของค่าความน่าจะเป็นในตาราง Qe ถูกนำไปใช้ในการคำนวณผลลัพธ์ล่วงหน้าสำหรับค่า A

2) ตาราง NLPS_NLPS เป็นตารางเก็บค่าดัชนีถัดไป ถูกนำไปใช้สำหรับการคาดเดาค่าดัชนีล่วงหน้าในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ปัจจุบันเป็นแบบ Code LPS และการบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code LPS

3) ตาราง NMPS_ NLPS เป็นตารางเก็บค่าดัชนีถัดไป ถูกนำไปใช้สำหรับการคาดเดาค่าดัชนีล่วงหน้าในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ปัจจุบันเป็นแบบ Code MPS และการบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code LPS

4) ตาราง NLPS_ NMPS เป็นตารางเก็บค่าดัชนีถัดไป ถูกนำไปใช้สำหรับการคาดเดาค่าดัชนีล่วงหน้าในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ปัจจุบันเป็นแบบ Code LPS และการบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code MPS

5) ตาราง NMPS_ NMPS เป็นตารางเก็บค่าดัชนีถัดไป ถูกนำไปใช้สำหรับการคาดเดาค่าดัชนีล่วงหน้าในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ปัจจุบันเป็นแบบ Code MPS และการบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code MPS

6) ตาราง NLPS_Qe เป็นตารางเก็บค่าความน่าจะเป็น ถูกนำไปใช้สำหรับการคำนวณล่วงหน้าของค่า A และค่า C ในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code LPS

7) ตาราง NMPS_Qe เป็นตารางเก็บค่าความน่าจะเป็น ถูกนำไปใช้สำหรับการคำนวณล่วงหน้าของค่า A และค่า C ในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code MPS

8) ตาราง NLPS_SWITCH เป็นตารางเก็บค่าการเปลี่ยนรูปแบบการบีบอัดข้อมูลจากแบบ Code LPS เป็นแบบ Code MPS หรือจากแบบ Code MPS เป็นแบบ Code LPS ถูกนำไปใช้สำหรับการคาดเดารูปแบบการบีบอัดข้อมูลล่วงหน้า ในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code LPS

9) ตาราง NMPS_SWITCH เป็นตารางเก็บค่าการเปลี่ยนรูปแบบการบีบอัดข้อมูลจากแบบ Code LPS เป็นแบบ Code MPS หรือจากแบบ Code MPS เป็นแบบ Code LPS ถูกนำไปใช้สำหรับ

การคาดเดารูปแบบการบีบอัดข้อมูลล่วงหน้า ในกรณีที่การบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้าเป็นแบบ Code MPS

4.2 การออกแบบส่วนคำนวณค่าขอบเขตบน

จากความต้องการให้สถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวที่นำเสนอนี้ ให้สามารถประมวลผลได้ครั้งละสองสัญลักษณ์ต่อสัญญาณนาฬิกานั้น ส่งผลต่อการออกแบบส่วนคำนวณค่า A การออกแบบจำเป็นต้องคำนึงถึงเรื่องความเร็วในการคำนวณเป็นหลัก ทำให้ต้องเพิ่มส่วนคาดเดาผลลัพธ์จากการคำนวณ เพื่อลดเวลาที่ใช้ในการประมวลผล

จากการสังเกตขั้นตอนการคำนวณค่า A พบว่าการคำนวณค่า A มี 3 รูปแบบ ดังแสดงในรูปที่ 4.2 สามารถอธิบายได้ ดังนี้

1) P1: A = A - qe

2) P2: A = qe แล้วถูกรีนอร์มอลไลซ์

3) P3: A = A - qe แล้วถูกรีนอร์มอลไลซ์

จากการสังเกตรูปแบบ P1 พบว่าสามารถประมวลผลเสร็จทันภายในหนึ่งสัญญาณนาฬิกา แต่รูปแบบ P2 และ P3 ไม่สามารถประมวลผลเสร็จภายในหนึ่งสัญญาณนาฬิกา ทำให้การออกแบบสถาปัตยกรรมจำเป็นต้องใช้เทคนิคต่างๆ เข้ามาช่วยในการออกแบบ เพื่อลดเวลาในการประมวลผล

<u>การลดเวลาในการประมวลผลของรูปแบบ P2</u>

เทคนิคที่นำมาใช้สำหรับการลดเวลาในการประมวลผลของรูปแบบ P2 คือ การคำนวณล่วงหน้า โดยที่มาของการคำนวณล่วงหน้าเกิดจากการสังเกตค่าที่นำมาใช้ในการคำนวณ โดยจำนวนค่าศูนย์นำหน้า (lz_qe) ของค่าความน่าจะเป็น (qe) ในตาราง Qe ถูกนำมาใช้เพื่อลดเวลาในการคำนวณ โดยค่าต่ำสุดของค่า lz_qe คือ Min(LZ(Qe)) = 1 และค่าสูงสุดของค่า lz_qe คือ Max(LZ(Qe)) = 15 เนื่องจากค่า qe เป็นค่าคงที่ ดังนั้นเพื่อลดเวลาที่ใช้ในการคำนวณค่า lz_qe ทุกครั้งที่ทำการบีบอัดข้อมูล งานวิจัยที่นำเสนอนี้จึงได้ทำการคำนวณค่า lz_qe ไว้ล่วงหน้า แล้วเก็บผลลัพธ์ลงในตาราง LZ_Qe จะช่วยลดเวลาในการคำนวณของรูปแบบ P2 ลงได้

<u>การลดเวลาในการประมวลผลของรูปแบบ P3</u>

การลดเวลาในการประมวลผลของรูปแบบ P3 นี้ทำได้ยากกว่ารูปแบบ P2 เนื่องจากการประมวลผลของรูปแบบ P2 ใช้ค่า lz_qe จากตาราง LZ_Qe ในการคำนวณล่วงหน้า ซึ่งการประมวลผลของรูปแบบ P3 นี้ไม่สามารถใช้เทคนิคเช่นเดียวกับรูปแบบ P2

ได้ เนื่องจากค่า A - qe ไม่ได้เป็นค่าคงที่ แต่จากการสังเกตค่าที่ใช้ในการคำนวณของรูปแบบ P3 พบว่าจำนวนค่าศูนย์นำหน้าสูงสุดของค่า A – qe เท่ากับ 2 สามารถคำนวณได้จาก

ค่าต่ำสุดของ A: Min(A) = 0x8000

ค่าสูงสุดของ qe: Max(qe) = 0x5601

Min(A) – Max(qe) = 0x29FF = 0010 1001 1111 1111

จะสังเกตเห็นได้ว่าจำนวนศูนย์นำหน้าของผลลัพธ์เท่ากับสอง โดยนับจากบิต MSB ไปยังบิต LSB ทำให้การออกแบบสถาปัตยกรรมของรูปแบบ P3 แบ่งได้ 3 รูปแบบ คือ

1) $P3_1$: A = A - qe

2) $P3_2$: A = (A - qe) << 1 (ปรับค่า A - qe เพิ่มขึ้น 2 เท่า)

3) $P3_3$: A = (A - qe) << 2 (ปรับค่า A - qe เพิ่มขึ้น 4 เท่า)

จากการสังเกตพบว่าผลลัพธ์ของรูปแบบ P1 และ $P3_1$ เหมือนกัน ทำให้สามารถใช้ฮาร์ดแวร์ร่วมกันได้ ดังแสดงสถาปัตยกรรมส่วนคำนวณค่าขอบเขตบน (A) ในรูปที่ 4.3

จากสถาปัตยกรรมที่ได้นำเสนอในรูปที่ 4.3 เทคนิคการคาดเดาและคำนวณผลลัพธ์ล่วงหน้าได้ถูกนำมาใช้ในการออกแบบ การเลือกผลลัพธ์จากการคาดเดาล่วงหน้า ($P1/P3_1$, P2, $P3_2$ และ $P3_3$) จะใช้มัลติเพล็กเซอร์ (Multiplexer) ในการเลือกผลลัพธ์ โดยสัญญาณที่นำมาใช้ในการเลือกผลลัพธ์นำมาจากเงื่อนไขตามแผนผังการทำงานในรูปที่ 4.2 แสดงโค้ดเทียมการเลือกผลลัพธ์ได้ในรูปที่ 4.4 โดยเงื่อนไขในการเลือกผลลัพธ์ของค่า A จะขึ้นอยู่กับ 4 ค่าหลัก คือ

1) จำนวนค่าศูนย์นำหน้าของผลต่างระหว่างค่า A ลบกับค่าจากตาราง Qe

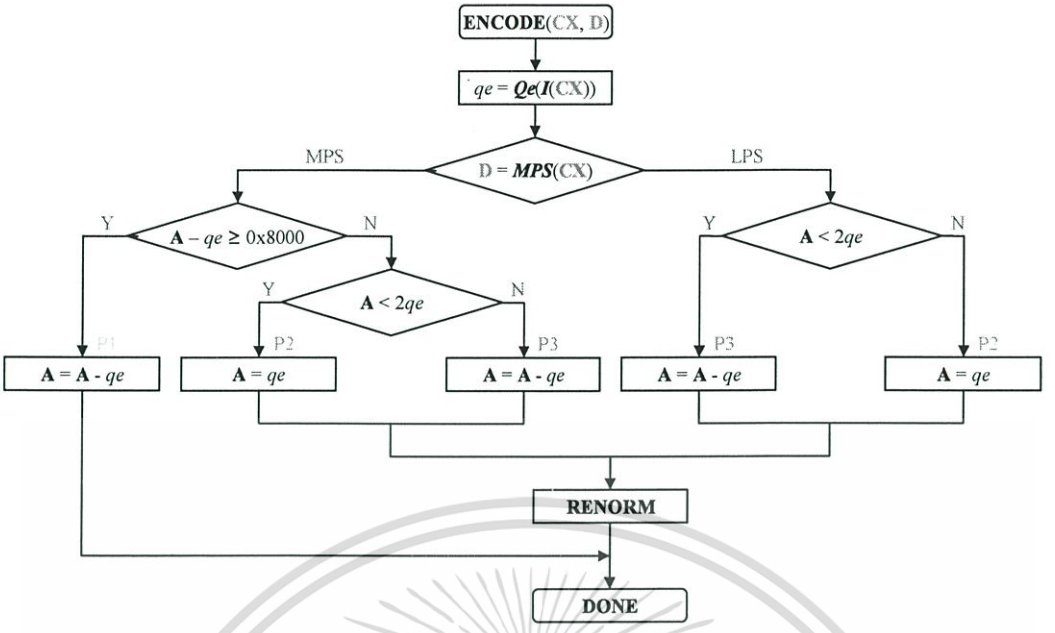
2) บิต MSB ของผลต่างระหว่างค่า A ลบกับค่าจากตาราง Qe

3) ผลการเปรียบเทียบน้อยกว่าของค่า A กับสองเท่าของค่าจากตาราง Qe

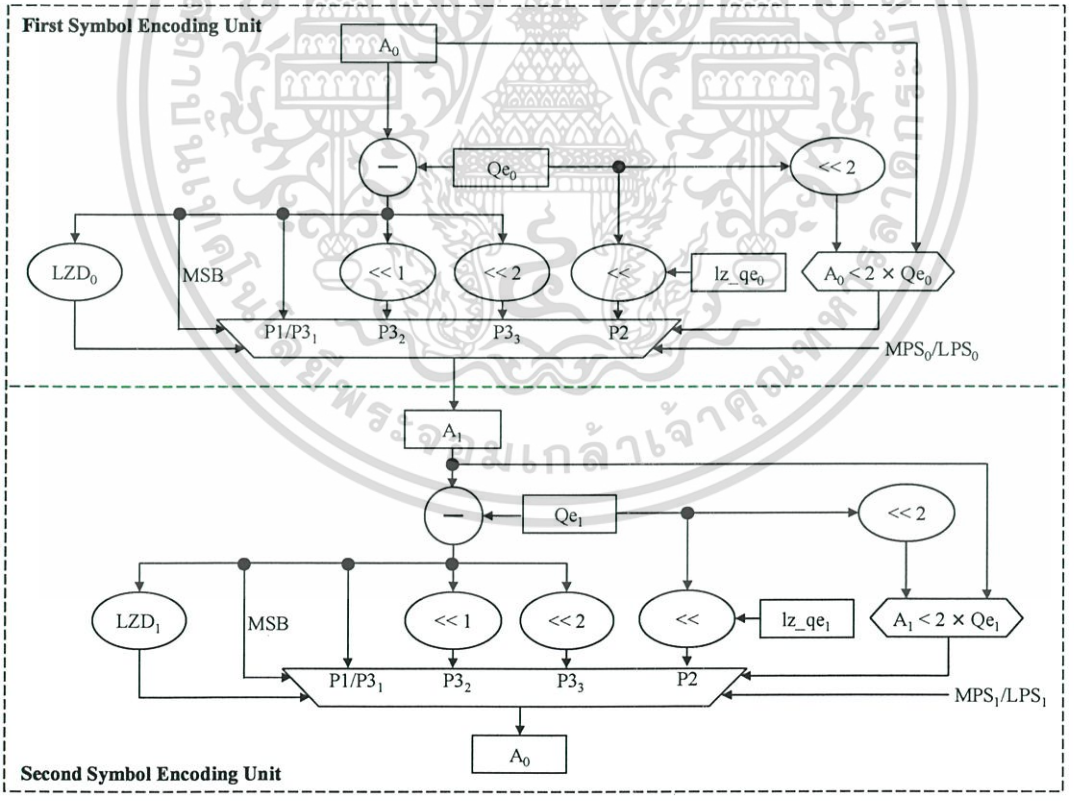
4) รูปแบบการบีบอัดข้อมูล MPS หรือ LPS

4.3 การออกแบบส่วนคำนวณค่าขอบเขตล่าง

การออกแบบส่วนคำนวณค่า C จะใช้หลักการเหมือนกับการออกแบบส่วนคำนวณค่า A แต่จะมีการออกแบบส่วนคาดเดาเพิ่มเติม เนื่องจากค่า C จะมีการเปลี่ยนแปลงทุกครั้งที่ส่วนนำส่งผลลัพธ์ (Byte out) ทำงาน ดังนั้นการออกแบบจำเป็นจะต้องคำนึงถึงการคาดเดาค่า C หลังส่วนนำส่งผลลัพธ์ทำงานด้วย



รูปที่ 4.2 แสดงแผนผังการทำงานของส่วนคำนวณค่าขอบเขตบน



รูปที่ 4.3 แสดงสถาปัตยกรรมของส่วนคำนวณค่าขอบเขตบน

```
IF D0 = mps0 THEN                          // Code MPS
    IF dif_a0[15] THEN
        a1 = dif_a0                        // P1
    ELSE
        IF a0 < 2 × qe0 THEN
            a1 = qe0 << lz_qe0             // P2
        ELSE
            CASE lz_qe0 OF
                0: a1 = dif_a0             // P31
                1: a1 = dif_a0 << 1        // P32
                2: a1 = dif_a0 << 2        // P33
            ENDCASE
        ENDIF
    ENDIF
ELSE                                       // Code LPS
    IF a0 < 2 × qe0 THEN
        CASE lz_qe0 OF
            0: a1 = dif_a0                 // P31
            1: a1 = dif_a0 << 1            // P32
            2: a1 = dif_a0 << 2            // P33
        ENDCASE
    ELSE
        a1 = qe0 << lz_qe0                 // P2
    ENDIF
ENDIF
```

รูปที่ 4.4 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่าขอบเขตบน

จากโค้ดเทียมการทำงานของส่วนนำส่งผลลัพธ์ในรูปที่ 4.5 พบว่าส่วนนำส่งผลลัพธ์ทำงานภายในขั้นตอนการทำรีนอร์มอลไลซ์ โดยจำนวนรอบของการทำรีนอร์มอลไลซ์ จะขึ้นอยู่กับค่า A ส่วนค่า CT ทำหน้าที่นับจำนวนบิตของผลลัพธ์ โดยที่ทุกรอบของการทำรีนอร์มอลไลซ์ ค่า A และค่า C จะเพิ่มขึ้นสองเท่า ส่วนค่า CT จะลดลงหนึ่ง ถ้าค่า CT ลดลงเท่ากับศูนย์ ผลลัพธ์จากการบีบอัดข้อมูลจะถูกส่งไปยังส่วนเข้ารหัสไทเออร์ทูเพื่อนำไปใช้งานต่อไป

การทำงานของส่วนนำส่งผลลัพธ์ (Byte out) ตามมาตรฐาน JPEG2000 แสดงโค้ดเทียมได้ดังรูปที่ 4.6 ค่า B ทำหน้าที่เก็บผลลัพธ์ของการบีบอัดข้อมูล สามารถแบ่งการทำงานออกได้เป็น 2 รูปแบบ คือ

1) กรณีเพิ่มบิตผลลัพธ์ (Bit Stuffing) -> คิดบิตทด (Carry bit)

2) กรณีไม่เพิ่มบิตผลลัพธ์ (No bit stuffing) -> ไม่คิดบิตทด (Carry bit)

การทำงานของกรณีเพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์แสดงโค้ดเทียมได้ดังรูปที่ 4.7 และ 4.8 ตามลำดับ ผลลัพธ์ (B) ของการบีบอัดข้อมูลได้มาจากข้อมูลไบต์บน (High Byte) ของค่า C อย่างไรก็ดีหลังการส่งผลลัพธ์ออกไป จำเป็นต้องกำหนดค่า C และค่า CT ใหม่ โดยค่าที่ถูก

กำหนดใหม่นี้จะขึ้นกับเงื่อนไขกรณีเพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์ซึ่งสามารถจำแนกข้อแตกต่างระหว่างการทำงานของกรณีเพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์ได้ดังต่อไปนี้

1) การเลือกตำแหน่งผลลัพธ์จากค่า C

2) การกำหนดค่า C ใหม่

3) การกำหนดค่า CT ใหม่

```
REPEAT
     A = A << 1
     C = C << 1
     CT = CT - 1
     IF CT = 0 THEN
          CALL byte_out()
     ENDIF
UNTIL A ≥ 0x8000
```

รูปที่ 4.5 แสดงโค้ดเทียมของการทำรีนอมอลไลซ์

```
IF B = 0xFF THEN
    CALL bit_stuffing()
ELSE
    IF C < 0x08000000 THEN
        CALL no_bit_stuffing()
    ELSE
        B = B + 1
        IF B = 0xFF THEN
            C = C & 0x07FFFFFF
            CALL bit_stuffing()
        ELSE
            CALL no_bit_stuffing()
        ENDIF
    ENDIF
ENDIF
```

รูปที่ 4.6 แสดงโค้ดเทียมของส่วนนำส่งผลลัพธ์

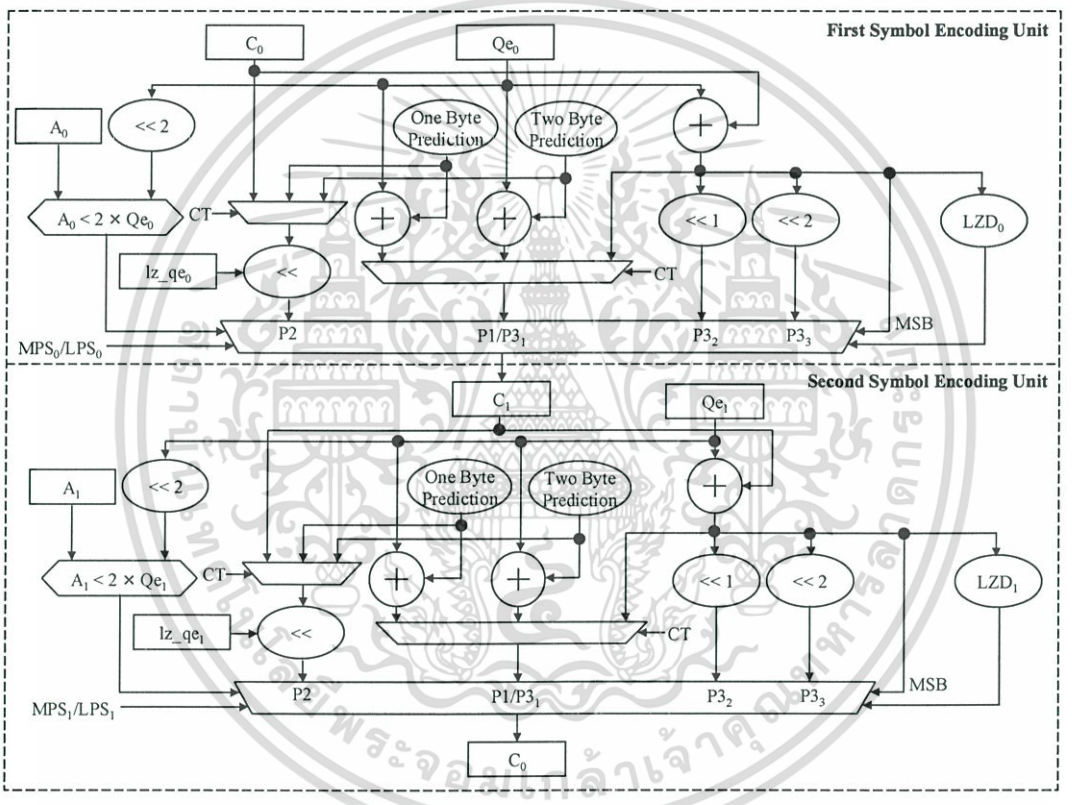
```
B = C >> 20         // shift "cbbb bbbb" bits of C to B
C = C & 0x000FFFFF // clear "cbbb bbbb" bits of C
CT = 7              // initial CT
```

รูปที่ 4.7 แสดงโค้ดเทียมของฟังก์ชัน bit_stuffing

```
B = C >> 19         // shift "bbb bbbb b" bits of C to B
C = C & 0x0007FFFF // clear "bbb bbbb b" bits of C
CT = 8              // initial CT
```

รูปที่ 4.8 แสดงโค้ดเทียมของฟังก์ชัน no_bit_stuffing

จากการทำงานของส่วนนำส่งผลลัพธ์ในรูปที่ 4.6 จะสังเกตได้ว่าส่งผลให้ค่า C มีการเปลี่ยนแปลง ทำให้การออกแบบส่วนคำนวณค่า C มีความซับซ้อนเพิ่มขึ้นจากการออกแบบส่วนคำนวณค่า A เพื่อให้ส่วนคาดเดาค่า C ทำงานได้ทันกับส่วนคำนวณค่า A ดังแสดงในภาพที่ 4.9



รูปที่ 4.9 แสดงสถาปัตยกรรมของส่วนคำนวณค่าขอบเขตล่าง

ดังนั้นการออกแบบส่วนคำนวณค่า C จึงสามารถแบ่งการพิจารณาออกเป็น 2 รูปแบบ คือ

1) การคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ไม่ทำงาน

2) การคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ทำงาน

4.3.1 การคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ไม่ทำงาน

จากการสังเกตขั้นตอนการคำนวณค่า C กรณีส่วนนำส่งผลลัพธ์ไม่ทำงานพบว่ามีรูปแบบที่ซ้ำกัน คล้ายคลึงกับรูปแบบในการคำนวณค่า A ดังแสดงในรูปที่ 4.9 แบ่งได้ 3 รูปแบบ คือ

1) P1: C = C + qe

2) P2: C = C แล้วถูก renormalize

3) P3: C = C + qe แล้วถูก renormalize

จากการสังเกตการคำนวณค่า C พบว่ารูปแบบ P1 สามารถประมวลผลเสร็จภายในหนึ่งสัญญาณนาฬิกา แต่รูปแบบ P2 และ P3 ไม่สามารถประมวลผลเสร็จภายในหนึ่งสัญญาณนาฬิกา ทำให้การออกแบบสถาปัตยกรรมจำเป็นต้องใช้เทคนิคต่างๆ เข้ามาช่วยในการออกแบบ เพื่อลดเวลาในการประมวลผล ดังนี้

<u>การลดเวลาในการประมวลผลของรูปแบบ P2</u>

เทคนิคที่นำมาใช้ลดเวลาในการประมวลผลของรูปแบบ P2 จะใช้หลักการเดียวกับการคำนวณค่า A คือ การคำนวณล่วงหน้า โดยนำค่าจากตาราง LZ_Qe มาใช้ในการคำนวณ

<u>การลดเวลาในการประมวลผลของรูปแบบ P3</u>

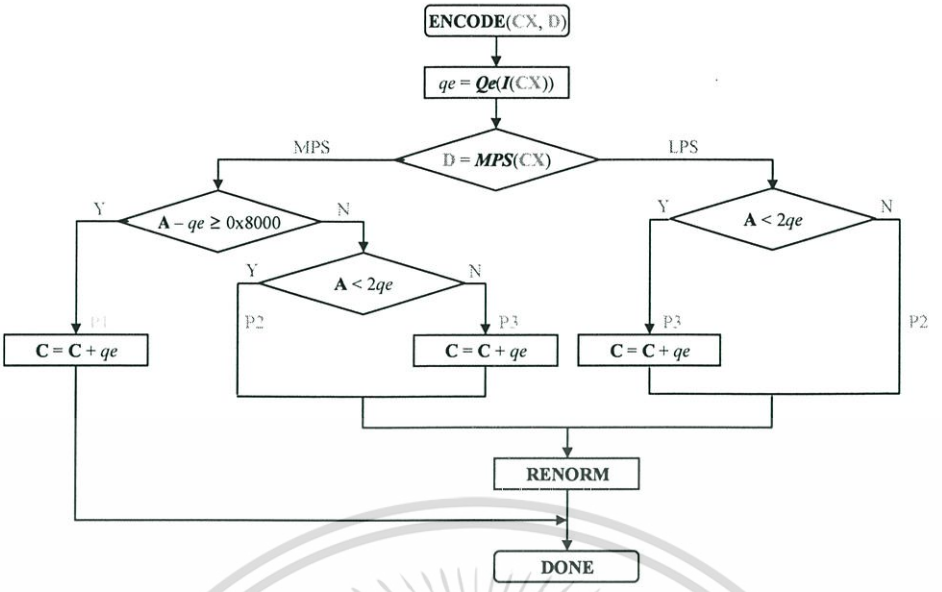
เทคนิคที่นำมาใช้ลดเวลาในการประมวลผลของรูปแบบ P3 จะใช้หลักการเดียวกับการคำนวณค่า A ดังนั้นสามารถแบ่งรูปแบบ P3 ได้ 3 รูปแบบ คือ

1) $P3_1$: C = C + qe

2) $P3_2$: C = (C + qe) << 1 (ปรับค่า C + qe เพิ่มขึ้น 2 เท่า)

3) $P3_3$: C = (C + qe) << 2 (ปรับค่า C + qe เพิ่มขึ้น 4 เท่า)

จากการสังเกตพบว่าผลลัพธ์ของรูปแบบ P1 และ $P3_1$ เหมือนกัน ทำให้สามารถใช้ฮาร์ดแวร์ร่วมกันได้ ดังแสดงสถาปัตยกรรมส่วนคำนวณค่า C ในรูปที่ 4.9



รูปที่ 4.10 แสดงแผนผังการทำงานของส่วนคำนวณค่าขอบเขตล่าง

```
IF D0 = mps0 THEN                          // Code MPS
    IF dif_a0[15] THEN
        c1 = sum_c0                        // P1
    ELSE
        IF a0 < 2 × qe0 THEN
            c1 = c0 << lz_qe0              // P2
        ELSE
            CASE lz_qe0 OF
                0: c1 = sum_c0             // P31
                1: c1 = sum_c0 << 1        // P32
                2: c1 = sum_c0 << 2        // P33
            ENDCASE
        ENDIF
    ENDIF
ELSE                                       // Code LPS
    IF a0 < 2 × qe0 THEN
        CASE lz_qe0 OF
            0: c1 = sum_c0                 // P31
            1: c1 = sum_c0 << 1            // P32
            2: c1 = sum_c0 << 2            // P33
        ENDCASE
    ELSE
        c1 = c0 << lz_qe0                  // P2
    ENDIF
ENDIF
```

รูปที่ 4.11 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่าขอบเขตล่าง

จากสถาปัตยกรรมที่ได้นำเสนอในรูปที่ 4.9 เทคนิคการคาดเดาและคำนวณผลลัพธ์ล่วงหน้าได้ถูกนำมาใช้ในการออกแบบ การเลือกผลลัพธ์จากการคาดเดาล่วงหน้า ($P1/P3_1$, P2, $P3_2$ และ $P3_3$) ซึ่งใช้มัลติเพล็กเซอร์ในการเลือกผลลัพธ์ โดยสัญญาณที่นำมาใช้ในการเลือกผลลัพธ์นำมาจากเงื่อนไขตามแผนผังการทำงานในรูปที่ 4.10 แสดงโค้ดเทียมการเลือกผลลัพธ์ได้ในรูปที่ 4.11 โดยเงื่อนไขในการเลือกผลลัพธ์ของค่า A จะขึ้นอยู่กับ 4 ค่าหลัก คือ

1) จำนวนค่าศูนย์นำหน้าของผลต่างระหว่างค่า A ลบกับค่าจากตาราง Qe

2) บิต MSB ของผลต่างระหว่างค่า A ลบกับค่าจากตาราง Qe

3) ผลการเปรียบเทียบน้อยกว่าของค่า A กับสองเท่าของค่าจากตาราง Qe

4) รูปแบบการบีบอัดข้อมูล MPS หรือ LPS

จากเงื่อนไขในการเลือกผลลัพธ์ของค่า C ดังที่กล่าวไว้ข้างต้น จะสังเกตได้ว่าเหมือนกับเงื่อนไขการเลือกผลลัพธ์ของค่า A ดังนั้นเงื่อนไขในการเลือกผลลัพธ์ของค่า A และค่า C สามารถใช้งานร่วมกันได้

4.3.2 การคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ทำงาน

จากการทดลองส่วนนำส่งผลลัพธ์ในโปรแกรม JJ2000 พบว่าจำนวนผลลัพธ์สูงสุดต่อสัญลักษณ์มีค่าเท่ากับ 2 ไบต์ นั่นหมายความว่าส่วนนำส่งผลลัพธ์ทำงาน 2 ครั้งต่อการบีบอัดหนึ่งครั้ง ส่งผลต่อการออกแบบส่วนคำนวณค่า C ให้สามารถทำงานได้ทันกับส่วนคำนวณค่า A ดังนั้นการออกแบบจึงแบ่งการพิจารณาออกเป็น 2 รูปแบบ คือ

1) การคาดเดาค่า C กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์

2) การคาดเดาค่า C กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์

<u>การคาดเดาค่า C กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์</u>

การคาดเดาค่า C กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์ แบ่งออกเป็น 2 กรณี คือ

1) กรณีเพิ่มบิตผลลัพธ์

2) กรณีไม่เพิ่มบิตผลลัพธ์

การพิจารณาค่า C ของทั้งสองกรณีจะใช้หลักการเดียวกัน แต่จะแตกต่างกันเฉพาะค่าที่นำมาใช้ในการมาร์คบิต (Mask Bits) เพื่อกำหนดค่า C ใหม่ การออกแบบเริ่มจากการสังเกตพฤติกรรมของค่า C ดังแสดงตัวอย่างในตารางที่ 4.1 และ 4.2

ตารางที่ 4.1 ตัวอย่างที่ 1 การคำนวณค่า C กรณีจำนวนลัพธ์จากการบีบอัดเท่ากับ 1 ไบต์

Statement	A	B	C	CT	Qe
<-- CX = 1, D = 1 -->	0x8DE0	0xC3	0x0007FCB4	8	0x0085
A = A - Qe	0x8D5B				
IF D = MPS(CX) **THEN**					
...					
ELSE					
IF A < Qe **THEN**					
C = C + Qe					
ELSE					
A = Qe	0x0085				
ENDIF					
A << 8 // P2(A)	0x8500				
C << 8 // P2(C)			0x07FCB400		
CT = CT - 8				0	
CALL BYTE OUT()					
<-- BYTE OUT() BEGIN -->					
IF B = 0xFF					
...					
ELSE IF C < 0x8000000					
B = C >> 19		0xFF			
C = C & 0x0007FFFF			0x0004B400		
CT = 8				8	
ENDIF					
<-- BYTE OUT() END -->					
ENDIF					

ตารางที่ 4.2 ตัวอย่างที่ 2 การคำนวณค่า C กรณีจำนวนลัพธ์จากการบีบอัดเท่ากับ 1 ไบต์

Statement	A	B	C	CT	Qe
<-- CX = 1, D = 1 -->	0xAC03	0x4B	0x01075421	1	0x3401
A = A - Qe	0x7802				
IF D = MPS(CX) **THEN**					
...					
ELSE					
IF A < Qe **THEN**					
C = C + Qe					
ELSE					
A = Qe	0x3401				
ENDIF					
A << 2 // P2(A)	0xD004				
C << 1 // P2(C)			0x020EA842		
CT = CT - 1				0	
CALL BYTE OUT()					
<-- BYTE OUT() BEGIN -->					
IF B = 0xFF					
...					
ELSE IF C < 0x8000000					
B = C >> 19		0x41			
C = C & 0x0007FFFF			0x0006A842		
CT = 8				8	
ENDIF					
<-- BYTE OUT() END -->					
C << 1 // P2(C)			0x000D5084		
CT = CT - 1				7	
ENDIF					

จากตัวอย่างในตารางที่ 4.1 จะสังเกตเห็นว่าการคำนวณค่า C อยู่ในกรณี P2 ซึ่งค่า C ก่อนทำการรีนอร์มอลไลซ์มีค่าเท่ากับ 0x07FCB400 แต่เนื่องจากจำนวนค่าศูนย์นำหน้ามีค่าเท่ากับ 8 ด้วยเงื่อนไขนี้ทำให้ค่า A ถูกรีนอร์มอลไลซ์จำนวน 8 รอบ โดยการเลื่อนบิตไปทางซ้ายจำนวน 8 ครั้ง ส่วนการคำนวณค่า C จะขึ้นอยู่กับค่า CT ก่อนการประมวลผล ในที่นี้มีค่าเท่ากับ 8 เท่ากับจำนวนค่าศูนย์นำหน้าซึ่งเป็นจำนวนรอบที่ต้องทำการรีนอร์มอลไลซ์ทำให้ต้องทำการรีนอร์มอลไลซ์ด้วยการเลื่อนบิตจำนวน 8 ครั้ง ทำให้ค่า CT ลดลงเป็นศูนย์ จากนั้นได้ทำการนำส่งผลลัพธ์จากการบีบอัด ค่า C หลังนำส่งผลลัพธ์มีค่าเท่ากับ 0x0004B4000

จากตัวอย่างในตารางที่ 4.2 จะสังเกตเห็นว่าการคำนวณค่า C อยู่ในกรณี P2 ซึ่งค่า C ก่อนทำการรีนอร์มอลไลซ์มีค่าเท่ากับ 0x01075421 แต่เนื่องจากจำนวนค่าศูนย์นำหน้ามีค่าเท่ากับ “2” ด้วยเงื่อนไขนี้ทำให้ค่า A ถูกรีนอร์มอลไลซ์จำนวน 2 รอบ โดยการเลื่อนบิตไปทางซ้ายจำนวน 2 ครั้ง ส่วนการคำนวณค่า C จะขึ้นอยู่กับค่า CT ก่อนการประมวลผลมีค่าเท่ากับ “1” แต่น้อยกว่า “2” ซึ่งเป็นจำนวนรอบที่ต้องทำการรีนอร์มอลไลซ์ทำให้ต้องทำการรีนอร์มอลไลซ์ด้วยการเลื่อนบิตจำนวน 1 ครั้งก่อน ทำให้ค่า CT ลดลงเป็นศูนย์ จากนั้นทำการนำส่งผลลัพธ์จากการบีบอัด และกลับมาทำรีนอร์มอลไลซ์ที่ค้างไว้ ทำให้ต้องเลื่อนค่า C ที่ค้างอีก 1 ครั้งจะได้ผลลัพธ์ 0x000D5084

จากตัวอย่างในตารางที่ 4.1 และ 4.2 จะเห็นได้ว่าการคำนวณค่า C อยู่ในกรณี P2 และกรณีไม่เพิ่มบิตผลลัพธ์เหมือนกัน แต่ขั้นตอนการคำนวณค่า C แตกต่างกัน ซึ่งความแตกต่างนี้ขึ้นอยู่กับค่า CT ถ้าค่า CT ก่อนการประมวลผลมีค่ามากกว่าเท่ากับจำนวนค่าศูนย์นำหน้าการทำงานจะเหมือนกับตัวอย่างในตารางที่ 4.1 แต่ถ้าค่า CT ก่อนการประมวลผลมีค่าน้อยกว่าจำนวนค่าศูนย์นำหน้าการทำงานจะเหมือนกับตัวอย่างในตารางที่ 4.2

จากการทดลองเพื่อหาความสัมพันธ์ของการคำนวณค่า C กรณีส่วนนำส่งผลลัพธ์ทำงานและจำนวนผลลัพธ์เท่ากับ 1 ไบต์ สามารถแบ่งกลุ่มความสัมพันธ์ได้ ดังนี้

1) P1, $P3_1$ และ $P3_2$ -> เหมือนกับตัวอย่างในตารางที่ 4.1

2) P2 และ $P3_3$ -> เหมือนกับตัวอย่างในตารางที่ 4.2

จากความต้องการให้ส่วนคำนวณค่า C ทำงานได้ทันกับส่วนคำนวณค่า A ดังนั้นการคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ทำงานและจำนวนผลลัพธ์เท่ากับ 1 ไบต์ ซึ่งมีความซับซ้อนกว่าการคำนวณค่า A เป็นอย่างมาก แต่จำเป็นจะต้องทำการประมวลผลให้เสร็จภายใน 1 สัญญาณนาฬิกา เพื่อให้สัญญาณนาฬิกาถัดไปสามารถคำนวณค่า A และค่า C ใหม่ได้พร้อมกัน ทั้งนี้จากความสัมพันธ์ของการคำนวณค่า C ดังที่กล่าวไว้ข้างต้นสามารถนำมาเป็นแนวทางในการออกแบบฮาร์ดแวร์ได้ ดังนี้

1) การออกแบบฮาร์ดแวร์สำหรับกรณี P1, $P3_1$ และ $P3_2$

สามารถใช้เงื่อนไขในการคาดเดาค่า C เหมือนกับกรณีที่ส่วนนำส่งผลลัพธ์ไม่ทำงาน เพียงแต่ค่า C หลังการทำงานตามขั้นตอนปกติเสร็จสิ้น จะต้องทำการมาร์คบิต เพื่อล้างค่าในตำแหน่งของผลลัพธ์ด้วย 0 การมาร์คบิตจะทำงานเสร็จภายใน 1 สัญญาณนาฬิกา เพื่อช่วยลดความซับซ้อนและเวลาในการประมวลผล แสดงโค้ดเทียมการคาดเดาค่า C ในรูปที่ 4.12

จากโค้ดเทียมการคาดเดาค่า C กรณีไม่เพิ่มบิตผลลัพธ์ในรูปที่ 4.12 จะสังเกตเห็นว่ากรณี P1, $P3_1$ และ $P3_2$ นั้น ค่าที่ใช้ในการมาร์คบิต จะมีค่าเท่ากับ 0x7FFFF แต่หากเป็นการทำงานกรณีกรณีเพิ่มบิตผลลัพธ์นั้น ค่าที่นำมาใช้ในการมาร์คบิตจะมีค่าเท่ากับ 0xFFFFF

2) การออกแบบฮาร์ดแวร์สำหรับกรณี P2 และ $P3_3$

การเลื่อนบิตหลังส่วนนำส่งผลลัพธ์ทำงาน ทำให้การคาดเดาค่า C ทำได้ยากขึ้น เนื่องจากจำนวนการเลื่อนบิตก่อนและหลังส่วนนำส่งผลลัพธ์ทำงานไม่คงที่ ดังนั้นการออกแบบจำเป็นจะต้องหาความสัมพันธ์ของการเลื่อนบิตที่ส่งผลต่อการเปลี่ยนแปลงค่า C ซึ่งสามารถแสดงพารามิเตอร์สำคัญที่สัมพันธ์กับการเลื่อนบิตได้ ดังนี้

- กรณี P2 -> initial CT - CT
- กรณี $P3_3$ -> LZ(A - Qe) – (initial CT – CT)

จากความสัมพันธ์ของการเลื่อนบิตดังที่กล่าวไว้ข้างต้น แสดงโค้ดเทียมได้ในรูปที่ 4.12 ค่า initial CT เกิดจากความต้องการให้สามารถนำส่งผลลัพธ์จำนวน 2 ไบต์เสร็จภายใน 1 สัญญาณนาฬิกา เพื่อให้สามารถประมวลผลสัญลักษณ์ถัดไปได้ทันที ซึ่งการนำส่งผลลัพธ์จะมีความเกี่ยวข้องกับค่า CT เนื่องจากค่า CT จะเป็นตัวกำหนดการนำส่งผลลัพธ์ โดยที่ค่า CT เดิมนั้นมีขนาด 5 บิต และมีค่าเริ่มต้นเท่ากับ “13” ซึ่งค่า CT จะทำงานในขั้นตอนรีนอร์มอลไลซ์และค่า CT จะลดลงหนึ่งต่อรอบการทำงาน โดยจำนวนรอบของการทำรีนอร์มอลไลซ์ จะขึ้นอยู่กับจำนวนค่าศูนย์นำหน้าของผลต่างของค่า A ลบด้วยค่าจากตาราง Qe แต่เนื่องจากจำนวนค่าศูนย์นำหน้าสูงสุดมีค่าเท่ากับ “15” ทำให้ค่า CT เดิมมีโอกาสติดลบหากจำนวนค่าศูนย์นำหน้ามากกว่าค่า CT ปัจจุบัน ดังนั้นค่า CT เดิมจึงไม่สามารถตอบสนองความต้องการใหม่ได้ ดังนั้นจำเป็นจะขยายขนาดของค่า CT เป็น 6 บิต และเปลี่ยนการคำนวณค่า CT จากลดค่าลงหนึ่งเป็นบวกค่าขึ้นหนึ่ง ซึ่งจะช่วย

แก้ปัญหาค่า CT ติดลบลงได้ จากการเปลี่ยนแปลงรูปแบบการคำนวณค่า CT ใหม่นั้น จะส่งผลต่อเงื่อนไขของการนำส่งผลลัพธ์ เนื่องจากการนำส่งผลลัพธ์เดิมจะเกิดขึ้นก็ต่อเมื่อค่า CT มีค่าเท่ากับ “0” แต่รูปแบบการคำนวณค่า CT ใหม่นี้จะกำหนดให้ค่า CT เริ่มต้นเท่ากับ “0” และเงื่อนไขของการนำส่งผลลัพธ์ใหม่นี้ ค่า CT จะมีค่าเท่ากับ “13” หรือเรียกว่าค่า “initial CT” ซึ่งมีค่าเท่ากับค่าเริ่มต้นของค่า CT เดิม

จากโค้ดเทียมการคาดเดาค่า C กรณีไม่เพิ่มบิตผลลัพธ์ในรูปที่ 4.12 จะสังเกตเห็นว่าการคาดเดาค่า C กรณี P2 นั้น ค่าที่นำมาใช้ในการมาร์คบิตจะไม่คงที่ โดยขึ้นอยู่กับเงื่อนไข initial CT - CT เช่น ถ้าผลต่างมีค่าเท่ากับ “0” จะถูกมาร์คบิตด้วยค่า 0x7FFFF แต่ถ้าเป็นกรณีกรณีเพิ่มบิตผลลัพธ์และอยู่ในเงื่อนไขเดียวกัน ค่ามาร์คบิตจะถูกแทนด้วยค่า 0xFFFFF เป็นต้น

จากโค้ดเทียมการคาดเดาค่า C กรณีไม่เพิ่มบิตผลลัพธ์ในรูปที่ 4.12 จะสังเกตเห็นว่าการคาดเดาค่า C กรณี $P3_3$ นั้น จะเกิดขึ้นเฉพาะในเงื่อนไขการบีบอัดข้อมูลแบบ LPS เท่านั้น โดยค่าที่นำมาใช้ในการมาร์คบิตจะไม่คงที่ และขึ้นอยู่กับเงื่อนไข LZ(A - Qe) - (initial CT - CT) เช่น ถ้าผลต่างมีค่าเท่ากับ “0” จะถูกมาร์คบิตด้วยค่า 0x7FFFF แต่ถ้าผลต่างไม่เท่ากับ “0” จะถูกมาร์คบิตด้วยค่า 0xFFFFF แต่ถ้าเป็นกรณีกรณีเพิ่มบิตผลลัพธ์ถ้าผลต่างมีค่าเท่ากับ “0” จะถูกมาร์คบิตด้วยค่า 0xFFFFF แต่ถ้าผลต่างไม่เท่ากับ “0” จะถูกมาร์คบิตด้วยค่า 0x1FFFFF

เทคนิคที่นำมาใช้ในการคาดเดาค่า C กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์นี้จะใช้การเลื่อนบิตค่า C (ขั้นตอนการทำรีนอมอลไลซ์) ภายในครั้งเดียว และทำการคาดเดาค่าที่ใช้ในการมาร์คบิต (ขั้นตอนนำส่งผลลัพธ์) ส่งผลให้การคำนวณค่า C ซึ่งมีความซับซ้อนมากกว่าการคำนวณค่า A สามารถทำงานได้ทันกันอย่างมีประสิทธิภาพ

```
IF D0 = mps0 THEN                                              // Code MPS
  IF MSB of a0 - qe0 = 1 THEN
    one_byte_c1 = sum_c0                                       // P1
  ELSE
    IF a0 < 2 × qe0 THEN
      CASE LZ(dif_a0) - (initial_ct - ct) OF
        0: one_byte_c1 = renorm_c0 & 0x0000007FFFF             // P2
        1: one_byte_c1 = renorm_c0 & 0x000000FFFFF             // P2
        2: one_byte_c1 = renorm_c0 & 0x000001FFFFF             // P2
        3: one_byte_c1 = renorm_c0 & 0x000003FFFFF             // P2
        4: one_byte_c1 = renorm_c0 & 0x000007FFFFF             // P2
        5: one_byte_c1 = renorm_c0 & 0x00000FFFFFF             // P2
        6: one_byte_c1 = renorm_c0 & 0x00001FFFFFF             // P2
        7: one_byte_c1 = renorm_c0 & 0x00003FFFFFF             // P2
      ENDCASE
    ELSE
      one_byte_c1 = double_sum_c0 & 0x000000FFFFF              // P32
    ENDIF
  ENDIF
ELSE                                                           // Code LPS
  IF a0 < 2 × qe0 THEN
    IF LZ(a0 - qe0) - (initial_ct   ct) = 0 THEN
      CASE lz_qe0 OF
        0: one_byte_c1 = sum_c0 & 0x0000007FFFF                // P31
        1: one_byte_c1 = double_sum_c0 & 0x0000007FFFF         // P32
        2: one_byte_c1 = four_sum_c0 & 0x0000007FFFF           // P33
      ENDCASE
    ELSE
      one_byte_c1 = four_sum_c0 & 0x000000FFFFF                // P33
    ENDIF
  ELSE
    CASE LZ(a0 - qe0) - (initial_ct - ct) OF
      0: one_byte_c1 = renorm_c0 & 0x0000007FFFF               // P2
      1: one_byte_c1 = renorm_c0 & 0x000000FFFFF               // P2
      2: one_byte_c1 = renorm_c0 & 0x000001FFFFF               // P2
      3: one_byte_c1 = renorm_c0 & 0x000003FFFFF               // P2
      4: one_byte_c1 = renorm_c0 & 0x000007FFFFF               // P2
      5: one_byte_c1 = renorm_c0 & 0x00000FFFFFF               // P2
      6: one_byte_c1 = renorm_c0 & 0x00001FFFFFF               // P2
      7: one_byte_c1 = renorm_c0 & 0x00003FFFFFF               // P2
    ENDCASE
  ENDIF
ENDIF
```

รูปที่ 4.12 แสดงโค้ดเทียมกรณีไม่เพิ่มบิตผลลัพธ์ของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์

การคาดเดาค่า C กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์

จากการทดลองพบว่าส่วนนำส่งผลลัพธ์มีโอกาสทำงานสูงสุด 2 ครั้งต่อการบีบอัดข้อมูลหนึ่งสัญลักษณ์ ทำให้จำนวนของผลลัพธ์สูงสุดเท่ากับ 2 ไบต์ กรณีนี้จะส่งผลต่อการเปลี่ยนแปลงค่า C ทำให้ต้องมีการออกแบบส่วนการคาดเดาค่า C ล่วงหน้าในกรณีนี้เพิ่มเติมอีกด้วย ดังแสดงตัวอย่างในตารางที่ 4.3 และ 4.4

ตารางที่ 4.3 ตัวอย่างที่ 1 กรณีไม่เพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์

```
                CT = 2, LZ(Qe) = 10
                Ccurrent = 0x000AC3300
= 0000 0000 0000 1010 1100 0011 0011 0000 0000
  Crenorm = Ccurrent << LZ(Qe) = 0x2B0CC0000
= 0010 1011 0000 1100 1100 0000 0000 0000 0000
                Cmask    = 0x00007FFFF
= 0000 0000 0000 0000 0111 1111 1111 1111 1111
    Cnew    = Crenorm & Cmask = 0x000040000
= 0000 0000 0000 0000 0100 0000 0000 0000 0000
          B0      = 0x56 = 0101 0110
          B1      = 0x19 = 0001 1001
```

ตารางที่ 4.4 ตัวอย่างที่ 2 กรณีไม่เพิ่มบิตผลลัพธ์และกรณีเพิ่มบิตผลลัพธ์

```
                CT = 2, LZ(Qe) = 10
                Ccurrent = 0x001FE82F3
= 0000 0000 0001 1111 1110 1000 0010 1111 0011
  Crenorm = Ccurrent << LZ(Qe) = 0x7FA0BCC00
= 0111 1111 1010 0000 1011 1100 1100 0000 0000
                Cmask    = 0x0000FFFFF
= 0000 0000 0000 0000 1111 1111 1111 1111 1111
    Cnew    = Crenorm & Cmask = 0x0000BCC00
= 0000 0000 0000 0000 1011 1100 1100 0000 0000
          B0      = 0xFF = 1111 1111
          B1      = 0x20 = 0010 0000
```

จากตัวอย่างในตารางที่ 4.3 และ 4.4 แสดงการคำนวณค่า C กรณีที่ส่วนนำส่งผลลัพธ์ทำงานถึง 2 ครั้งต่อการบีบอัดข้อมูลหนึ่งสัญลักษณ์ โดยการทำงานจะเกิดขึ้นเฉพาะกรณี P2 เท่านั้น เนื่องจากจำนวนค่าศูนย์นำหน้าสูงสุดในกรณี P2 มีค่าเท่ากับ "15" ทำให้โอกาสในการนำส่งผลลัพธ์สูงสุดเท่ากับ 2 ไบต์ จากตัวอย่างกำหนดให้ค่า $C_{current}$ เก็บค่า C ก่อนการบีบอัด, ค่า C_{renorm} เก็บค่า C หลังถูกรีนอมอลไลซ์, ค่า C_{mask} เก็บค่าที่ใช้ในการ mask bits, ค่า C_{new} เก็บค่า C หลังส่งผลลัพธ์ออก 2 ไบต์, ค่า B_0 เก็บผลลัพธ์ไบต์ที่หนึ่ง และส่วนค่า B_1 เก็บผลลัพธ์ไบต์ที่สอง

จากตารางที่ 4.3 และตารางที่ 4.4 จะสังเกตเห็นการระบายสีต่างๆ ลงในตัวอย่าง โดยความหมายของสีต่างๆ มีดังนี้

1) สีเหลือง -> ผลลัพธ์ของ B_0
2) สีเขียว -> ผลลัพธ์ของ B_1
3) สีฟ้า -> ตำแหน่งของค่ามาร์คบิตที่ใช้ล้างค่า C
4) สีแดง -> บิต MSB ของผลลัพธ์ที่มีค่าเป็น 0 เสมอ

จากตัวอย่างในตารางที่ 4.3 จะสังเกตเห็นว่าการคำนวณค่า C อยู่ในกรณีไม่เพิ่มบิตผลลัพธ์สำหรับเงื่อนไขการนำส่งผลลัพธ์ทั้ง 2 ไบต์ จากตัวอย่างค่า C ก่อนรีนอร์มอลไลซ์มีค่าเท่ากับ 0x000AC3300 และหลังการทำรีนอร์มอลไลซ์ มีค่าเท่ากับ 0x2B0CC0000 ส่วน

ค่าที่นำมาใช้ในการมาร์คบิตเท่ากับ 0x00007FFFF และผลลัพธ์หลังการทำมาร์คบิตมีค่าเท่ากับ 0x000040000

จากตัวอย่างในตารางที่ 4.4 จะสังเกตเห็นว่าการคำนวณค่า C อยู่ในกรณีไม่เพิ่มบิตผลลัพธ์และกรณีเพิ่มบิตผลลัพธ์ตามลำดับ สำหรับเงื่อนไขการนำส่งผลลัพธ์ทั้ง 2 ไบต์ จากค่า C ก่อนรีนอร์มอลไลซ์มีค่าเท่ากับ 0x001FE82F3 และหลังการทำรีนอร์มอลไลซ์มีค่าเท่ากับ 0x7FA0BCC00 ส่วนค่าที่นำมาใช้ในการมาร์คบิตเท่ากับ 0x0000FFFFF และผลลัพธ์หลังการทำมาร์คบิตมีค่าเท่ากับ 0x0000BCC00

จากตัวอย่างในตารางที่ 4.4 จะสังเกตได้ว่าผลลัพธ์จากการบีบอัดข้อมูลไบต์ที่หนึ่งส่งผลต่อผลลัพธ์ไบต์ที่สอง ดังนั้นส่วนคำนวณค่า C จำเป็นจะต้องมีการออกแบบส่วนคาดเดาผลลัพธ์ของไบต์ที่หนึ่งล่วงหน้า เพื่อหาว่าผลลัพธ์ของไบต์ที่หนึ่งส่งผลกระทบต่อการคาดเดาผลลัพธ์ไบต์ที่สองอย่างไร

จากการทดลองพบว่ากรณีที่สามารถเกิดขึ้นได้สำหรับการคำนวณค่า C กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์ มีดังนี้

1) กรณีเพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์
2) กรณีไม่เพิ่มบิตผลลัพธ์และกรณีเพิ่มบิตผลลัพธ์
3) กรณีไม่เพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์

จากกรณีที่สามารถเกิดขึ้นได้ดังที่กล่าวไว้ข้างต้นนั้น กรณีที่ 1 และกรณีที่ 3 จะใช้หลักการเดียวกันในการออกแบบ แต่จะแตกต่างกันเฉพาะค่าที่นำมาใช้ในการมาร์คบิต

จากตัวอย่างกรณีที่ 3 ในตารางที่ 4.3 สามารถนำมาใช้ในการออกแบบฮาร์ดแวร์ได้ดังนี้ การเลื่อนบิตค่า C ในขั้นตอนการรีนอร์มอลไลซ์จะทำภายในครั้งเดียว จากนั้นจึงทำการคาดเดาค่าที่ใช้ในการมาร์คบิตในขั้นตอนการนำส่งผลลัพธ์ เพื่อแทนค่าในตำแหน่งของผลลัพธ์ด้วย 0 แต่เนื่องจากจำนวนการเลื่อนบิตในขั้นตอนการรีนอร์มอลไลซ์ไม่คงที่ ทำให้การออกแบบจะต้องทำการคาดเดาค่าที่นำมาใช้ในการมาร์คบิต ซึ่งจะสัมพันธ์กับผลต่างของจำนวนค่าศูนย์นำหน้าและผลต่างของค่า CT และ initial CT สามารถเขียนได้ดังสมการ LZ(Qe) – (initial CT – CT) และแสดงโค้ดเทียมการเลือกค่าที่นำมาใช้ในการมาร์คบิตได้ดังรูปที่ 4.14 ตัวอย่างเช่น ถ้าผลต่างของสมการที่กล่าวไว้ข้างต้นมีค่าเท่ากับ “8” ค่าที่ใช้ในการมาร์คบิตจะมีค่าเท่ากับ 0x7FFFF เป็นต้น แต่ถ้าเป็นกรณีที่ 1 ค่าที่นำมาใช้ในการมาร์คบิต จะแสดงโค้ดเทียมได้ดังรูปที่ 4.15 ตัวอย่างเช่น ถ้าผลต่างมีค่าเท่ากับ “8” ค่าที่ใช้ในการมาร์คบิตจะมีค่าเท่ากับ 0xFFFFF โดยหลักการที่ใช้ในการมาร์คบิตจะเป็นเช่นเดียวกับกรณีที่ 3 เป็นต้น

อย่างไรก็ดี เงื่อนไขการออกแบบกรณีที่ 2 นั้นทำได้ยากกว่าการออกแบบกรณีที่ 1 และ 3 เนื่องมาจากการทำงานของกรณีที่ 2 มีขั้นตอนเพิ่มเติมในการตรวจสอบค่าของผลลัพธ์จากการบีบอัดข้อมูลไบต์ที่หนึ่งก่อน จึงจะสามารถคำนวนค่าการคาดเดาค่า C ได้ ดังนั้นเพื่อให้สามารถรู้ค่าของผลลัพธ์จากการบีบอัดข้อมูลไบต์ที่หนึ่งล่วงหน้า จำเป็นต้องระบุตำแหน่งผลลัพธ์ไบต์ที่หนึ่งจากค่า C_{renorm} ให้ได้ก่อน ซึ่งตำแหน่งผลลัพธ์ไบต์ที่หนึ่งนี้จะขึ้นอยู่กับผลต่างของค่า initial CT กับค่า CT ก่อนการคำนวณค่า C ซึ่งค่าผลต่างจะเป็นตัวกำหนดจำนวนรอบในการทำรีนอร์มอลไลซ์ ก่อนการนำส่งผลลัพธ์ครั้งที่ 1 ดังแสดงได้ในโค้ดเทียมรูปที่ 4.13 ตัวอย่างเช่น ถ้าผลต่างมีค่าเท่ากับ "1" ผลลัพธ์ของไบต์ที่หนึ่งจะอยู่ในตำแหน่งที่ 18 ถึง 25 เป็นต้น

```
CASE initial_ct - ct OF
    1: IF c0[25:18] = 0xFF THEN
           CALL mask_bit_bit_stuffing()
       ELSE
           CALL mask_bit_no_bit_stuffing()
       ENDIF
    2: IF c0[24:17] = 0xFF THEN
           CALL mask_bit_bit_stuffing()
       ELSE
           CALL mask_bit_no_bit_stuffing()
       ENDIF
    3: IF c0[23:16] = 0xFF THEN
           CALL mask_bit_bit_stuffing()
       ELSE
           CALL mask_bit_no_bit_stuffing()
       ENDIF
    4: IF c0[22:15] = 0xFF THEN
           CALL mask_bit_bit_stuffing()
       ELSE
           CALL mask_bit_no_bit_stuffing()
       ENDIF
    5: IF c0[21:14] = 0xFF THEN
           CALL mask_bit_bit_stuffing()
       ELSE
           CALL mask_bit_no_bit_stuffing()
       ENDIF
    6: IF c0[20:13] = 0xFF THEN
           CALL mask_bit_bit_stuffing()
       ELSE
           CALL mask_bit_no_bit_stuffing()
       ENDIF
    7: IF c0[19:12] = 0xFF THEN
           CALL mask_bit_bit_stuffing()
       ELSE
           CALL mask_bit_no_bit_stuffing()
       ENDIF
ENDCASE
```

รูปที่ 4.13 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณีจำนวนผลลัพธ์เท่ากับ "2" ไบต์

```
CASE lz_qe0 - (initial_ct - ct) OF
    8 : two_byte_c1 = renorm_c0 & 0x0000007FFFF
    9 : two_byte_c1 = renorm_c0 & 0x000000FFFFF
    10: two_byte_c1 = renorm_c0 & 0x000001FFFFF
    11: two_byte_c1 = renorm_c0 & 0x000003FFFFF
    12: two_byte_c1 = renorm_c0 & 0x000007FFFFF
    13: two_byte_c1 = renorm_c0 & 0x00000FFFFFF
    14: two_byte_c1 = renorm_c0 & 0x00001FFFFFF
    OTHER: two_byte_c1 = renorm_c0 & 0x0000007FFFF
ENDCASE
```

รูปที่ 4.14 แสดงโค้ดเทียมการเลือกค่า C กรณีไม่เพิ่มบิตผลลัพธ์ของรูปที่ 4.11

```
CASE lz_qe0 - (initial_ct - ct) OF
    8 : two_byte_c1 = renorm_c0 & 0x000000FFFFF
    9 : two_byte_c1 = renorm_c0 & 0x000001FFFFF
    10: two_byte_c1 = renorm_c0 & 0x000003FFFFF
    11: two_byte_c1 = renorm_c0 & 0x000007FFFFF
    12: two_byte_c1 = renorm_c0 & 0x00000FFFFFF
    13: two_byte_c1 = renorm_c0 & 0x00001FFFFFF
    14: two_byte_c1 = renorm_c0 & 0x00003FFFFFF
    OTHER: two_byte_c1 = renorm_c0 & 0x000000FFFFF
ENDCASE
```

รูปที่ 4.15 แสดงโค้ดเทียมการเลือกค่า C กรณีกรณีเพิ่มบิตผลลัพธ์ของรูปที่ 4.13

จากเงื่อนไขต่างๆ ของการคาดเดาค่า C กรณีส่วนนำส่งผลลัพธ์ทำงานนั้น ส่งผลต่อการเปลี่ยนแปลงการออกแบบแตกต่างไปจากมาตรฐานในหลายๆ ส่วน สามารถสรุปการเปลี่ยนแปลงได้ ดังนี้

1) ค่า C เปลี่ยนจากขนาด 28 บิตเป็น 41 บิต เพื่อให้สามารถตรวจสอบและนำส่งผลลัพธ์ได้สูงสุดถึง 2 ไบต์ พร้อมกัน ส่งผลให้ค่า C_{renorm}, $C_{one\ byte}$ และ $C_{two\ byte}$ ที่ใช้ในการคาดเดามีขนาด 41 บิตด้วย

2) ค่า CT เปลี่ยนจากขนาด 4 บิตเป็น 5 บิต และการทำงานเปลี่ยนจากนับลงเป็นนับขึ้น เนื่องจากการเลือกค่า C กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์ หากใช้วิธีการนับลงค่า CT จะติดลบ ดังนั้นการออกแบบจึงเปลี่ยนวิธีการนับค่า CT จากวิธีการนับลงเป็นนับขึ้น

จากสถาปัตยกรรมการคำนวณค่า C ในรูปที่ 4.9 การทำงานของมัลติเพล็กเซอร์ในส่วนของกรณี P2 จะต้องทำการเลือกค่า C ที่ถูกต้อง เนื่องจากค่า C หลังส่วนนำส่งผลลัพธ์ทำงานจะถูกมาร์คบิต เพื่อล้างค่าในตำแหน่งผลลัพธ์ด้วย 0 ดังนั้นในการบีบอัดสัญลักษณ์ปัจจุบัน จำเป็นจะต้องทำการคาดเดาการบีบอัดสัญลักษณ์ก่อนหน้าว่า มีการนำส่งผลลัพธ์

หรือไม่ ถ้ามีการนำส่งผลลัพธ์แล้วจำนวนของการนำส่งผลลัพธ์มีจำนวนเท่ากับ 1 หรือ 2 ไบต์ แสดงโค้ดเทียมการคาดเดาค่า C ของกรณี P1 และ $P3_1$ ในรูปที่ 4.16 และแสดงโค้ดเทียม ส่วนกรณี P2 ในรูปที่ 4.17 ถ้าค่า CT มีค่ามากกว่าหรือเท่ากับ "13" หมายความว่าการบีบอัดข้อมูลของสัญลักษณ์ก่อนหน้ามีจำนวนเท่ากับ 1 ไบต์ ส่วนกรณีอื่นๆ ให้ถือว่าไม่มีผลลัพธ์จากการบีบอัดข้อมูล

```
IF ct ≥ 21 THEN
    sum_c1 = two_byte_c1 + qe1
ELSEIF ct ≥ 13 THEN
    sum_c1 = one_byte_c1 + qe1
ELSE
    sum_c1 = c1 + qe1
ENDIF
```

รูปที่ 4.16 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณี P1 และ $P3_1$

```
IF ct ≥ 21 THEN
    renorm_c1 = two_byte_c1 << lz_qe1
ELSEIF ct ≥ 13 THEN
    renorm_c1 = one_byte_c1 << lz_qe1
ELSE
    renorm_c1 = c1 << lz_qe1
ENDIF
```

รูปที่ 4.17 แสดงโค้ดเทียมของมัลติเพล็กเซอร์ที่ใช้ในการเลือกค่า C กรณี P2

4.4 การออกแบบส่วนนำส่งผลลัพธ์

จากความสัมพันธ์ระหว่างส่วนคำนวณค่า C และส่วนนำส่งผลลัพธ์ดังที่กล่าวไว้ในหัวข้อที่ 4.3 นั้น จะเห็นได้ว่าหากส่วนนำส่งผลลัพธ์ทำงาน ค่า C จะถูกมาร์คบิตเพื่อล้างค่าในตำแหน่งผลลัพธ์ด้วย 0 ทำให้ค่า C เปลี่ยนแปลงไป ดังนั้นการออกแบบส่วนคำนวณค่า C จะต้องคำนึงว่า ถ้าส่วนนำส่งผลลัพธ์ทำงาน รูปแบบการทำงานจะเป็นอย่างไร ส่งผลอย่างไรต่อค่า C บ้าง แต่ในส่วนของการออกแบบส่วนนำส่งผลลัพธ์นั้น จะมีความแตกต่างจากการออกแบบส่วนคำนวณค่า C พอสมควร เนื่องจากส่วนนำส่งผลลัพธ์จะสนใจเฉพาะการเลือกผลลัพธ์จากค่า C ที่ผ่านการรีนอร์มอลไลซ์แล้วเท่านั้น แต่จากข้อจำกัดของการออกแบบให้ส่วนนำส่งผลลัพธ์ทำงานภายใน 1 สัญญาณนาฬิกา เพื่อให้สามารถทำงานได้ทันกับส่วนคำนวณค่า A ดังนั้นจำเป็นจะต้องปรับการทำงานของส่วนนำส่งผลลัพธ์ให้ทำงานได้ภายใน 1 สัญญาณนาฬิกา ด้วยการปรับลดเงื่อนไข

การนำส่งผลลัพธ์ แสดงโค้ดเทียมก่อนปรับการทำงานได้ในรูปที่ 4.6 และแสดงโค้ดเทียมหลังปรับการทำงานใหม่ในรูปที่ 4.18

```
IF B = 0xFF THEN
    CALL bit_stuffing_1()
ELSE IF B = 0xFE AND Carry bit = 1 THEN
    CALL bit_stuffing_2()
ELSE
    CALL no_bit_stuffing()
ENDIF
```

รูปที่ 4.18 แสดงโค้ดเทียมใหม่ของส่วนนำส่งผลลัพธ์

จากโค้ดเทียมการทำงานใหม่ของส่วนนำส่งผลลัพธ์ในรูปที่ 4.18 นั้น จะมีการปรับเงื่อนไขในการเลือกผลลัพธ์ เพื่อลดเวลาที่ใช้ในการประมวลผลให้เสร็จภายใน 1 สัญญาณนาฬิกา โดยสามารถแบ่งได้ 3 กรณี คือ

1) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF

2) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFE และบิตทดมีค่าเท่ากับ “1”

3) กรณีอื่นๆ

1) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF

การเลือกผลลัพธ์ของกรณีนี้จะมีขนาด 8 บิต (ตำแหน่งบิตที่ 21 ถึง 28 ของค่า C) โดยตำแหน่งบิต MSB จะเป็นตำแหน่งของบิตทดเสมอ และต่อไปจะเรียกการทำงานของกรณีนี้ว่า “bit_stuffing_1”

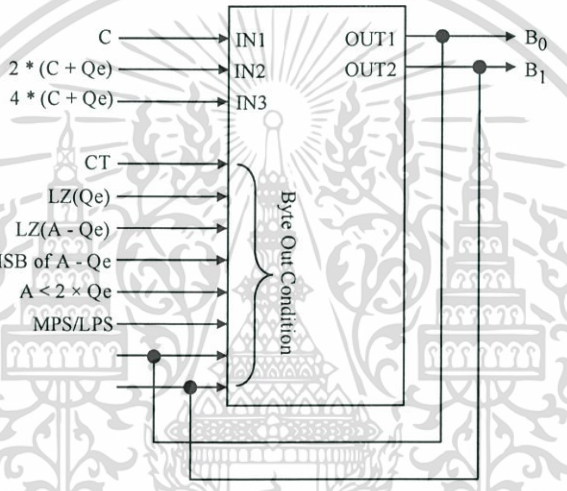
2) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFE และบิตทดมีค่าเท่ากับ “1”

การเลือกผลลัพธ์ของกรณีนี้จะมีขนาด 7 บิต (ตำแหน่งบิตที่ 21 ถึง 27 ของค่า C) โดยตำแหน่งบิต MSB จะมีค่าเท่ากับ “0” เสมอ และต่อไปจะเรียกการทำงานของกรณีนี้ว่า “bit_stuffing_2” ส่วนสาเหตุของการแบ่งการทำงานของรูปแบบกรณีเพิ่มบิตผลลัพธ์ออกเป็น 2 กรณี เนื่องจากการเลือกขนาดของผลลัพธ์ที่แตกต่างกัน ส่วนสาเหตุของความแตกต่างนี้เกิดจากเงื่อนไข C < 0x8000000 ในโค้ดเทียมการทำงานตามมาตรฐานของส่วนนำส่งผลลัพธ์รูปที่ 4.6 จะสังเกตได้ว่าถ้าเงื่อนไขนี้เป็นเท็จ แสดงว่าบิตทดจะมีค่าเท่ากับ “1” (บิตทดอยู่ในตำแหน่งที่ 28 ของค่า C) ดังนั้นตามขั้นตอนการทำงานปกติจะต้องทำการเพิ่มค่า B ขึ้นหนึ่ง แล้วตรวจสอบค่า B ว่ามีค่าเท่ากับ 0xFF หรือไม่ จากสองขั้นตอนดังที่กล่าวมานี้เราสามารถยุบการทำงานให้ทำภายในขั้นตอนเดียวกันได้ ดังนี้ ตรวจสอบค่า B ว่ามีค่าเท่ากับ 0xFE และบิตทดมีค่าเท่ากับ “1” หรือไม่ ซึ่งจะให้ผลลัพธ์ในการทำงานเท่ากัน

3) กรณีอื่นๆ

การเลือกผลลัพธ์ของกรณีนี้จะมีขนาด 8 บิต (ตำแหน่งบิตที่ 20 ถึง 27 ของค่า C) และต่อไปจะเรียกการทำงานของกรณีนี้ว่า “no_bit_stuffing”

จากโค้ดเทียมการทำงานใหม่ของส่วนนำผลลัพธ์ในรูปที่ 4.18 จะพบว่าการเลือกผลลัพธ์ปัจจุบัน จำเป็นจะต้องทำการพิจารณาค่าผลลัพธ์ก่อนหน้าของทั้งค่า B_0 และค่า B_1 ด้วย ดังนั้นในการออกแบบส่วนนำส่งผลลัพธ์จะต้องส่งค่าผลลัพธ์ก่อนหน้ากลับเข้ามาพิจารณาในการเลือกผลลัพธ์ปัจจุบันด้วย แสดงโครงสร้างฮาร์ดแวร์ของส่วนการนำส่งผลลัพธ์ในรูปที่ 4.19 ซึ่งประกอบด้วย ส่วนอินพุต ส่วนผลลัพธ์ และส่วนเงื่อนไขในการเลือกผลลัพธ์



รูปที่ 4.19 แสดงโครงสร้างการทำงานของส่วนนำส่งผลลัพธ์

ส่วนอินพุต

เนื่องจากส่วนนำส่งผลลัพธ์จะทำงานเฉพาะกรณี P2, $P3_2$ และ $P3_3$ ดังนั้นสัญญาณอินพุตของส่วนนำส่งผลลัพธ์จะประกอบไปด้วย 1). ค่า C -> กรณี P2 2). ค่าสองเท่าของผลรวมระหว่างค่า C กับค่าจากตาราง Qe -> กรณี $P3_2$ 3). ค่าสี่เท่าของผลรวมระหว่างค่า C กับค่าจากตาราง Qe -> กรณี $P3_3$

จะเห็นว่าผลลัพธ์ของการบีบอัดนี้ขึ้นอยู่กับสัญญาณอินพุตดังที่กล่าวไว้ข้างต้น ดังนั้นสัญญาณอินพุตที่ส่งเข้ามายังส่วนนำส่งผลลัพธ์ จะถูกนำมาเลือกตำแหน่งของผลลัพธ์ แล้วเก็บผลลัพธ์ของการบีบอัดข้อมูลลงในค่า B_0 และค่า B_1 ตามเงื่อนไขการนำส่งผลลัพธ์ที่ได้ออกแบบไว้ในงานวิจัยนี้

<u>ส่วนผลลัพธ์</u>

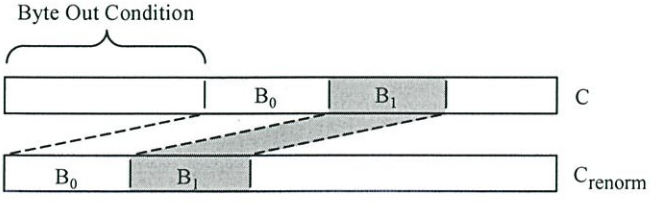
เป็นส่วนผลลัพธ์ของการบีบอัดข้อมูล โดยกำหนดให้กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์เก็บค่าผลลัพธ์ใน B_0 ส่วนในกรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์ เก็บค่าผลลัพธ์ไบต์ที่หนึ่งใน B_0 และเก็บค่าผลลัพธ์ไบต์ที่สองใน B_1

<u>ส่วนเงื่อนไขการเลือกผลลัพธ์</u>

ส่วนเงื่อนไขในการเลือกผลลัพธ์ในที่นี้จะเรียกว่า "Byte Out Condition" ซึ่งเป็นเงื่อนไขในการประมวลผลสัญญาณอินพุต เพื่อให้ได้มาซึ่งผลลัพธ์จากการบีบอัดข้อมูล โดยเงื่อนไขต่างๆ ที่ถูกออกแบบในงานวิจัยนี้ มีจุดประสงค์เพื่อกระชับการทำงานของการประมวลผลส่วนนำส่งผลลัพธ์ เนื่องจากการนำส่งผลลัพธ์มีความซับซ้อนในการประมวลผลสูง โดยใช้เวลาในการประมวลผลสูงสุดถึง 8 สัญญาณนาฬิกา (กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์) แต่เพื่อให้ส่วนนำส่งผลลัพธ์ทำงานได้ทันกับส่วนคำนวณค่า A และค่า C ดังนั้นจำเป็นที่ต้องออกแบบให้ส่วนนำส่งผลลัพธ์ทำงานเสร็จสิ้นภายใน 1 สัญญาณนาฬิกา

ดังนั้นงานวิจัยนี้จึงได้นำเสนอโครงสร้างส่วนเลือกผลลัพธ์ที่มีประสิทธิภาพ เพื่อให้ส่วนนำส่งผลลัพธ์สามารถทำงานได้อย่างรวดเร็วตามที่กำหนด โดยสัญญาณที่ถูกนำมาใช้เพื่อสร้างเงื่อนไขในส่วนของการคาดเดาผลลัพธ์ภายในส่วนนำส่งผลลัพธ์นี้ประกอบไปด้วย ทั้งหมด 8 ค่า ได้แก่ 1). ค่า CT 2). จำนวนค่าศูนย์นำหน้าของค่าจากตาราง Qe 3). จำนวนค่าศูนย์นำหน้าของผลต่างระหว่างค่า A ลบค่าจากตาราง Qe 4). บิต MSB ของผลต่างระหว่างค่า A ลบค่าจากตาราง Qe 5). ผลการเปรียบเทียบค่า A น้อยกว่าสองเท่าของค่าจากตาราง Qe 6). รูปแบบการบีบอัด MPS หรือ LPS 7). ค่า B_0 และ 8). ค่า B_1

จากรูปที่ 4.20 แสดงความสัมพันธ์ของการเลื่อนตำแหน่งผลลัพธ์ในค่า C ก่อนและหลังการทำรีนอร์มอลไลซ์ จะสังเกตเห็นว่าผลลัพธ์อยู่ในค่า C อยู่ก่อนแล้ว แต่หลังการทำรีนอร์มอลไลซ์ ทำให้ตำแหน่งของผลลัพธ์เปลี่ยนแปลงไป ดังนั้นหากเราทราบความสัมพันธ์ก่อนและหลังการทำรีนอร์มอลไลซ์ได้ จะทำให้สามารถคาดเดาตำแหน่งของผลลัพธ์ได้



รูปที่ 4.20 แสดงการเลื่อนบิตของผลลัพธ์ในค่า C

เนื่องจากการทำงานของส่วนนำส่งผลลัพธ์จะต้องทำการตรวจสอบค่าผลลัพธ์ล่าสุดว่ามีค่าเท่ากับ 0xFE หรือ 0xFF หรือไม่ ดังนั้นเราจำเป็นจะต้องตรวจสอบผลลัพธ์ล่าสุดเก็บไว้ในค่า B_0 หรือ B_1 ดังนั้นวิธีการเพื่อให้ทราบว่าผลลัพธ์ล่าสุดเก็บไว้ที่ค่าใด ในที่นี้เราได้สร้างแฟล็ก (flag) ชื่อ "twobyte" เพื่อช่วยในการตรวจสอบปัญหาดังกล่าว แสดงโค้ดเทียมในรูปที่ 4.21 โดยถ้าการนำส่งผลลัพธ์ล่าสุดมีจำนวนผลลัพธ์เท่ากับ 1 ไบต์ ค่า twobyte จะเท่ากับ "0" แต่ถ้าจำนวนผลลัพธ์เท่ากับ 2 ไบต์ ค่า twobyte จะเท่ากับ "1" ทำให้เราสามารถทราบได้ว่าตำแหน่งของผลลัพธ์ล่าสุดอยู่ที่ค่า B_0 หรือ B_1

```
IF twobyte = 1 THEN
    IF B1 = 0xFF THEN
        CALL bit_stuffing_1()
    ELSE IF B1 = 0xFE AND carry = 1 THEN
        CALL bit_stuffing_2()
    ELSE
        CALL no_bit_stuffing()
    ENDIF
ELSE
    IF B0 = 0xFF THEN
        CALL bit_stuffing_1()
    ELSE IF B0 = 0xFE AND carry = 1 THEN
        CALL bit_stuffing_2()
    ELSE
        CALL no_bit_stuffing()
    ENDIF
ENDIF
```

รูปที่ 4.21 แสดงโค้ดเทียมการตรวจสอบผลลัพธ์ล่าสุด โดยใช้แฟล็กชื่อ "twobyte"

จากผลการทดลองดังที่กล่าวไว้ในหัวข้อที่ 4.3 นั้นพบว่าจำนวนของผลลัพธ์สูงสุดต่อการบีบอัดข้อมูลหนึ่งสัญลักษณ์เท่ากับ 2 ไบต์ ดังนั้นในการออกแบบส่วนนำส่งผลลัพธ์จะแยกการพิจารณาออกเป็น 2 กรณี คือ

1) กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์

2) กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์

4.4.1 กรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์

การออกแบบกรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์ จะแยกการพิจารณาออกเป็น 2 กรณี คือ

1) กรณีเพิ่มบิตผลลัพธ์

2) กรณีไม่เพิ่มบิตผลลัพธ์

การทำงานของกรณีเพิ่มบิตผลลัพธ์และกรณีไม่เพิ่มบิตผลลัพธ์นั้น จะทำหน้าที่ในการเลือกผลลัพธ์จากค่า C แต่จะมีความแตกต่างกันเฉพาะตำแหน่งในการเลือกผลลัพธ์ โดยหลังการเลือกผลลัพธ์จะทำการล้างบิตในตำแหน่งของผลลัพธ์ด้วย เพื่อเตรียมประมวลผลสัญลักษณ์ถัดไป

1) กรณี Bit stuffng

การตรวจสอบว่าเป็นกรณีเพิ่มบิตผลลัพธ์หรือไม่ จะพิจารณาจากเงื่อนไข

1. กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF

ในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 8 บิต และต่อไปจะเรียกการทำงานของกรณีนี้ว่า “bit_stuffing_1” สำหรับเงื่อนไขในการเลือกผลลัพธ์สามารถแสดงโค้ดเทียมได้ดังรูปที่ 4.22 เมื่อพิจารณาเงื่อนไขในการเลือกผลลัพธ์ของกรณี bit_stuffing_1 นี้ จะสังเกตได้ว่าเหมือนกับเงื่อนไขการคาดเดาค่า A และค่า C ในการพิจารณากรณี P1, P2, $P3_1$, $P3_2$ และ $P3_3$ ดังนั้นสามารถนำเงื่อนไขการคาดเดาค่า A, ค่า C และผลลัพธ์มาใช้งานร่วมกันได้ แต่จะมีเงื่อนไขเพิ่มเติมในการเลือกผลลัพธ์ สามารถอธิบายได้ ดังนี้

กรณี P1 และ $P3_1$ ในรูปที่ 4.22 จะไม่มีการเลือกผลลัพธ์ เนื่องจากไม่มีการทำรีนอร์มอลไลซ์ ดังนั้นค่า CT ซึ่งใช้ในการนับจำนวนบิตของผลลัพธ์จะไม่มีโอกาสลดลงเป็น 0 ส่งผลให้ส่วนนำส่งผลลัพธ์ไม่ทำงาน ดังนั้นจะกำหนดให้ค่าผลลัพธ์คงเดิม

ส่วนกรณี $P3_2$ และ $P3_3$ ในรูปที่ 4.22 เงื่อนไขกรณีนี้ส่งผลให้หลังจากการรีนอร์มอลไลซ์ค่า C แล้ว ส่งผลให้ผลลัพธ์อยู่ในตำแหน่งบิตที่ 20 ถึง 27 เสมอ

และกรณี P2 ในรูปที่ 4.22 เป็นกรณีที่มีความซับซ้อนที่สุด เมื่อเทียบกับกรณีอื่นๆที่ได้กล่าวมาแล้ว เนื่องจากตำแหน่งของผลลัพธ์ในค่า C ไม่คงที่ จะขึ้นอยู่กับเงื่อนไขผลต่าง LZ(Qe) - (initial CT - CT) ทำให้จำเป็นต้องหาความสัมพันธ์ของการเลื่อนตำแหน่งผลลัพธ์ในค่า C ดังแสดงในโค้ดเทียมเช่นถ้าผลต่างมีค่าเท่ากับ “1” จะทำให้ตำแหน่งของผลลัพธ์อยู่ที่ตำแหน่งบิตที่ 20 ถึง 27 ของค่า C เป็นต้น

```
IF D0 = mps0 THEN                                          // Code MPS
  IF dif_a0[15] THEN
    b0 = b0                                                // P1
  ELSE
    IF a0 < 2 × qe0 THEN
      CASE lz_qe0 - (initial_ct - ct) OF
        1: b0 = renorm_c0[27:20]                           // P2
        2: b0 = renorm_c0[28:21]                           // P2
        3: b0 = renorm_c0[29:22]                           // P2
        4: b0 = renorm_c0[30:23]                           // P2
        5: b0 = renorm_c0[31:24]                           // P2
        6: b0 = renorm_c0[32:25]                           // P2
        7: b0 = renorm_c0[33:26]                           // P2
        OTHER: b0 = b0
      ENDCASE
    ELSE
      CASE lz_qe0 OF
        1: b0 = double_sum_c0[27:20]                       // P3_2
        2: b0 = four_sum_c0[27:20]                         // P3_3
        OTHER: b0 = b0
      ENDCASE
    ENDIF
  ENDIF
ELSE                                                       // Code LPS
  IF a0 < 2 × qe0 THEN
    CASE initial_ct - ct OF
      1: b0 = double_sum_c0[27:20]                         // P3_2
      2: b0 = four_sum_c0[27:20]                           // P3_3
      OTHER: b0 = b0
    ENDCASE
  ELSE
    CASE lz_qe0 - (initial_ct - ct) OF
      1: b0 = renorm_c0[27:20]                             // P2
      2: b0 = renorm_c0[28:21]                             // P2
      3: b0 = renorm_c0[29:22]                             // P2
      4: b0 = renorm_c0[30:23]                             // P2
      5: b0 = renorm_c0[31:24]                             // P2
      6: b0 = renorm_c0[32:25]                             // P2
      7: b0 = renorm_c0[33:26]                             // P2
      OTHER: b0 = b0
    ENDCASE
  ENDIF
ENDIF
```

รูปที่ 4.22 แสดงโค้ดเทียมการเลือกผลลัพธ์ของฟังก์ชั่น bit_stuffing_1

2. กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFE และค่าบิตทดเท่ากับ "1"

ในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 7 บิต และกำหนดให้บิตที่ 8 ซึ่งเป็นตำแหน่งบิต MSB ของผลลัพธ์มีค่าเป็น "0" เสมอ และต่อไปจะเรียกการทำงานของกรณีนี้ว่า "bit_stuffing_2" สำหรับเงื่อนไขในการเลือกผลลัพธ์จะใช้หลักการเดียวกันกับกรณีที่ 1 ที่อธิบายในหัวข้อก่อนหน้านี้ แต่จะมีความแตกต่างกันเฉพาะขนาดในการเลือกผลลัพธ์

กล่าวคือ กรณีที่ 1 จะเลือกผลลัพธ์ขนาด 8 บิต ส่วนกรณีที่ 2 จะเลือกผลลัพธ์ขนาด 7 บิต ส่วนบิตที่ 8 ซึ่งเป็นตำแหน่งบิต MSB ของผลลัพธ์จะมีค่าเป็น "0" เสมอ ตัวอย่างเช่น การเลือกผลลัพธ์กรณี $P3_2$ จะทำการเลือกผลลัพธ์ในตำแหน่งบิตที่ 20 ถึง 26 ส่วนบิตที่ 8 ของผลลัพธ์จะกำหนดให้มีค่าเป็น "0" เสมอ เป็นต้น

หลังจากทราบจำนวนบิตที่ต้องการเลือกเป็นผลลัพธ์ของการบีบอัดแล้ว ปัญหาต่อมา คือ ปัญหาการระบุตำแหน่งของบิตทดในค่า C เนื่องจากตำแหน่งบิตทดเดิมนั้นอยู่ในตำแหน่งบิต MSB หรือตำแหน่งที่ 28 เสมอ แต่เนื่องจากค่า C ที่ออกแบบใหม่ในงานวิจัยนี้มีขนาดเพิ่มขึ้นเป็น 41 บิต เพื่อให้สามารถรองรับการนำส่งผลลัพธ์พร้อมกันได้สูงสุดถึง 2 ไบต์ โดยค่า CT ไม่มีโอกาสติดลบ จากการออกแบบขนาดของค่า C แบบใหม่นี้ ทำให้บิตทดไม่จำเป็นที่จะต้องอยู่ที่ตำแหน่งบิต MSB เสมอ ดังนั้นในงานวิจัยนี้จึงจำเป็นต้องออกแบบส่วนคาดเดาตำแหน่งของบิตทดได้ดังโค้ดเทียมในรูปที่ 4.23

```
CASE initial_ct - ct OF
  1 : carry = c1[26]
  2 : carry = c1[25]
  3 : carry = c1[24]
  4 : carry = c1[23]
  5 : carry = c1[22]
  6 : carry = c1[21]
  7 : carry = c1[20]
  8 : carry = c1[19]
  9 : carry = c1[18]
  10: carry = c1[17]
  11: carry = c1[16]
  12: carry = c1[15]
  13: carry = c1[14]
  14: carry = c1[13]
  15: carry = c1[12]
  OTHER: carry = 0
ENDCASE
```

รูปที่ 4.23 แสดงโค้ดเทียมการเลือกค่าบิตทด

2) กรณีไม่เพิ่มบิตผลลัพธ์

ในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 8 บิต และต่อไปจะเรียกการทำงานของกรณีนี้ว่า "no_bit_stuffing" สำหรับเงื่อนไขในการเลือกผลลัพธ์จะใช้หลักการเดียวกรณี bit_stuffing_1 แต่จะมีความแตกต่างกันเฉพาะตำแหน่งในการเลือกผลลัพธ์ ตัวอย่างเช่น การเลือกผลลัพธ์กรณี $P3_2$ จะทำการเลือกผลลัพธ์ในตำแหน่งบิตที่ 19 ถึง 26 เป็นต้น

4.4.2 กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์

การออกแบบกรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์ จะแยกการพิจารณาออกเป็น 2 กรณี คือ

1) การเลือกผลลัพธ์ไบต์ที่หนึ่ง

2) การเลือกผลลัพธ์ไบต์ที่สอง

1) การเลือกผลลัพธ์ไบต์ที่หนึ่ง

การเลือกผลลัพธ์ไบต์ที่หนึ่งแบ่งออกได้ 2 กรณี คือ

1. กรณีเพิ่มบิตผลลัพธ์

2. กรณีไม่เพิ่มบิตผลลัพธ์

1. กรณีกรณีเพิ่มบิตผลลัพธ์

การเลือกผลลัพธ์กรณีเพิ่มบิตผลลัพธ์แบ่งออกได้ 2 กรณี คือ

1) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF

2) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFE และค่าบิตทดเท่ากับ "1"

(1) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF

เนื่องจากส่วนนำส่งผลลัพธ์กรณีจำนวนผลลัพธ์เท่ากับ 2 ไบต์นี้จะทำงานเฉพาะกรณี P2 เท่านั้น ดังนั้นหลักในการเลือกผลลัพธ์ทั้งสองไบต์จะพิจารณาเฉพาะกรณี P2 เท่านั้น โดยในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 8 บิต โดยเงื่อนไขในการเลือกผลลัพธ์ไบต์ที่หนึ่ง คือ ผลต่าง LZ(Qe) - (initial CT - CT) เช่นเดียวกับกรณี P2 ของฟังก์ชั่น bit_stuffing_1 ซึ่งเป็นกรณีจำนวนผลลัพธ์เท่ากับ 1 ไบต์ แสดงโค้ดเทียมในรูปที่ 4.24 แต่จะมีความแตกต่างกันเฉพาะตำแหน่งในการเลือกผลลัพธ์ ตัวอย่างเช่น ถ้าผลต่างของความสัมพันธ์ดังที่กล่าวไว้ข้างต้นมีค่าเท่ากับ 8 ผลลัพธ์ในค่า C จะอยู่ที่ตำแหน่งบิตที่ 28 ถึง 35 เป็นต้น

```
CASE LZ(Qe) - (initial_ct - ct) OF
    8 : b0 = renorm_c0[35:28]
    9 : b0 = renorm_c0[36:29]
    10: b0 = renorm_c0[37:30]
    11: b0 = renorm_c0[38:31]
    12: b0 = renorm_c0[39:32]
    13: b0 = renorm_c0[40:33]
    14: b0 = renorm_c0[41:34]
    OTHER: b0 = renorm_c0[35:28]
ENDCASE
```

รูปที่ 4.24 แสดงโค้ดเทียมการเลือกผลลัพธ์ไบต์ที่หนึ่งกรณีกรณีเพิ่มบิตผลลัพธ์และค่า B = 0xFF

(2) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFE และค่าบิตทดเท่ากับ "1"

ในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 7 บิต และกำหนดให้บิตที่ 8 ซึ่งเป็นตำแหน่งบิต MSB ของผลลัพธ์มีค่าเป็น "0" เสมอ โดยเงื่อนไขในการเลือกผลลัพธ์ไบต์ที่หนึ่ง คือ ผลต่าง LZ(Qe) - (initial CT - CT) เช่นเดียวกับกรณีที่ 1 แต่จะมีความแตกต่างกันเฉพาะขนาดในการเลือกผลลัพธ์ กล่าวคือ กรณีที่ 1 จะเลือกผลลัพธ์ขนาด 8 บิต ส่วนกรณีที่ 2 จะเลือกผลลัพธ์ขนาด 7 บิต ส่วนบิตที่ 8 ซึ่งเป็นตำแหน่งบิต MSB ของผลลัพธ์จะมีค่าเป็น "0" เสมอ ตัวอย่างเช่น ถ้าผลต่างมีค่าเท่ากับ 8 ผลลัพธ์ในค่า C จะอยู่ที่ตำแหน่งบิตที่ 28 ถึง 34 ส่วนบิตที่ 8 ของผลลัพธ์จะกำหนดให้มีค่าเป็น "0" เสมอ เป็นต้น

2. กรณีไม่เพิ่มบิตผลลัพธ์ (No bit stuffing)

ในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 8 บิต โดยเงื่อนไขในการเลือกผลลัพธ์ไบต์ที่หนึ่ง คือ ผลต่าง LZ(Qe) - (initial CT - CT) เช่นเดียวกับกรณี Bit stuffing แต่จะมีความแตกต่างกันเฉพาะตำแหน่งในการเลือกผลลัพธ์ ตัวอย่างเช่น ถ้าผลต่างมีค่าเท่ากับ 8 ผลลัพธ์ในค่า C จะอยู่ที่ตำแหน่งบิตที่ 27 ถึง 34 เป็นต้น

2) การเลือกผลลัพธ์ไบต์ที่สอง

การเลือกผลลัพธ์ไบต์ที่สองแบ่งออกได้ 2 กรณี คือ

1. กรณีเพิ่มบิตผลลัพธ์
2. กรณีไม่เพิ่มบิตผลลัพธ์

1. กรณีกรณีเพิ่มบิตผลลัพธ์

การเลือกผลลัพธ์กรณีเพิ่มบิตผลลัพธ์แบ่งออกได้ 2 กรณี คือ

1) กรณี B = 0xFF

2) กรณี B = 0xFE และบิตทดเท่ากับ 1

(1) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF

ในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 8 บิต โดยเงื่อนไขในการเลือกผลลัพธ์ไบต์ที่สอง คือ ผลต่าง LZ(Qe) - (initial CT - CT) เช่นเดียวกับการเลือกผลลัพธ์ไบต์ที่หนึ่งกรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF แสดงโค้ดเทียมในรูปที่ 4.25 แต่จะมีความแตกต่างกันเฉพาะตำแหน่งในการเลือกผลลัพธ์ ตัวอย่างเช่น ถ้าผลต่างของความสัมพันธ์ดังที่กล่าวไว้ข้างต้นมีค่าเท่ากับ 8 ผลลัพธ์ในค่า C จะอยู่ที่ตำแหน่งบิตที่ 20 ถึง 27 เป็นต้น

```
CASE LZ(Qe) - (initial_ct - ct) OF
   8 : b1 = renorm_c0[27:20]
   9 : b1 = renorm_c0[28:21]
  10: b1 = renorm_c0[29:22]
  11: b1 = renorm_c0[30:23]
  12: b1 = renorm_c0[31:24]
  13: b1 = renorm_c0[32:25]
  14: b1 = renorm_c0[33:26]
  OTHER: b1 = renorm_c0[27:20]
ENDCASE
```

รูปที่ 4.25 แสดงโค้ดเทียมการเลือกผลลัพธ์ไบต์ที่สองกรณีกรณีเพิ่มบิตผลลัพธ์และค่า B = 0xFF

(2) กรณีค่าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFE และค่าบิตทดเท่ากับ “1”

ในกรณีนี้จะทำการเลือกบิตผลลัพธ์จำนวน 7 บิต และกำหนดให้บิตที่ 8 ซึ่งเป็นตำแหน่งบิต MSB ของผลลัพธ์มีค่าเป็น “0” เสมอ โดยเงื่อนไขในการเลือกผลลัพธ์ไบต์ที่หนึ่ง คือ ผลต่าง LZ(Qe) - (initial CT - CT) เช่นเดียวกับกรณีที่ 1 แต่จะมีความแตกต่างกันเฉพาะขนาดในการเลือกผลลัพธ์ กล่าวคือ กรณีที่ 1 จะเลือกผลลัพธ์ขนาด 8 บิต ส่วนกรณีที่ 2 จะเลือกผลลัพธ์ขนาด 7 บิต ส่วนบิตที่ 8 ซึ่งเป็นตำแหน่งบิต MSB ของผลลัพธ์จะมีค่าเป็น “0” เสมอ ตัวอย่างเช่น ถ้าผลต่างมีค่าเท่ากับ 8

ผลลัพธ์ในค่า C จะอยู่ที่ตำแหน่งบิตที่ 20 ถึง 26 ส่วนบิตที่ 8 ของผลลัพธ์จะกำหนดให้มีค่าเป็น “0” เสมอ เป็นต้น

2. กรณีไม่เพิ่มบิตผลลัพธ์ (No bit stuffing)

การออกแบบกรณีไม่เพิ่มบิตผลลัพธ์นั้นทำได้ยากกว่าการออกแบบกรณีกรณีเพิ่มบิตผลลัพธ์เนื่องมาจากการทำงานของกรณีไม่เพิ่มบิตผลลัพธ์จะต้องทำการตรวจสอบผลลัพธ์ไบต์ที่หนึ่งว่าเท่ากับ 0xFF หรือไม่ ดังนั้นปัญหาในการออกแบบกรณีนี้ คือ เราจะทราบได้อย่างไรว่าผลลัพธ์ไบต์ที่หนึ่งอยู่ในตำแหน่งใดของค่า C โดยผลลัพธ์ไบต์ที่หนึ่งจะอยู่ที่ตำแหน่งใดของค่า C จะขึ้นอยู่กับผลต่าง LZ(Qe) - (initial CT - CT) ดังแสดงได้ในโค้ดเทียมรูปที่ 4.26 ตัวอย่างเช่น ถ้าผลต่างของความสัมพันธ์ดังที่กล่าวไว้ข้างต้นมีค่าเท่ากับ 8 และค่า C ในตำแหน่งบิตที่ 27 ถึง 34 มีค่าเท่ากับ 0xFF ผลลัพธ์ไบต์ที่สองจะอยู่ในตำแหน่งที่ 20 ถึง 26 ส่วนบิตที่ 8 ซึ่งเป็นตำแหน่งบิต MSB ของผลลัพธ์จะมีค่าเป็น “0” เสมอ เป็นต้น

4.5 ตัวอย่างการคำนวณของสถาปัตยกรรมที่นำเสนอ

จากรายละเอียดการออกแบบสถาปัตยกรรมที่นำเสนอดังที่กล่าวไว้ข้างต้นนั้น ทำให้ทราบถึงปัญหาในการออกแบบส่วนเข้ารหัสเอ็มคิวให้สามารถประมวลผลเสร็จภายในสองสัญญาณนาฬิกาต่อสัญลักษณ์ รวมถึงวิธีการแก้ไขปัญหาต่างๆ ที่เกิดขึ้นจากการเร่งความเร็วในการประมวลผล ทั้งนี้เพื่อเพิ่มความเข้าใจในขั้นตอนการทำงานของสถาปัตยกรรมที่นำเสนอนั้น จึงได้นำเสนอตัวอย่างการบีบอัดข้อมูลดังต่อไปนี้

4.5.1 กรณีค่าดัชนีเปลี่ยนแปลง

จากการทำงานของส่วนเข้ารหัสเอ็มคิวตามมาตรฐาน JPEG2000 นั้น จะมีการนำสัญลักษณ์ปัจจุบันมาทำการเปิดตารางค่าดัชนี ซึ่งค่าดัชนีนี้จะถูกนำมาใช้ในการเปิดตาราง Qe และผลลัพธ์จากตาราง Qe จะถูกนำไปใช้ในการประมวลผลการบีบอัดสัญญาณต่อไป แต่จากการเร่งความเร็วในการประมวลผลของสถาปัตยกรรมที่นำเสนอนั้น ทำให้จำเป็นจะต้องประมวลผลสัญลักษณ์ปัจจุบันไปพร้อมๆ กับการอ่านค่าจากตารางทั้งหมด เพื่อให้สามารถประมวลผลสัญลักษณ์ถัดไปได้ทันที ทั้งที่การประมวลผลสัญลักษณ์ปัจจุบันยังทำไม่เสร็จสิ้น ดังนั้นถ้าการประมวลผลสัญลักษณ์ปัจจุบันส่งผลให้ค่าดัชนีเกิดการเปลี่ยนแปลง จะทำให้ผลลัพธ์จากการเปิดตาราง Qe สำหรับสัญลักษณ์ถัดไปเกิดความ

ผิดพลาดขึ้นด้วย เนื่องจากค่าดัชนีที่ใช้ในการเปิดตารางสำหรับสัญลักษณ์ถัดไปยังคงเป็นค่าก่อนการเปลี่ยนแปลง

```
CASE LZ(Qe) - (initial_ct - CT) OF
    8 :IF renorm_c0[34:27] = 0xFF THEN
           b1 = '0' & renorm_c0[26:20]
       ELSE
           b1 = renorm_c0[27:20]
       ENDIF
    9 :IF renorm_c0[35:28] = 0xFF THEN
           b1 = '0' & renorm_c0[27:21]
       ELSE
           b1 = renorm_c0[28:21]
       ENDIF
   10:IF renorm_c0[36:29] = 0xFF THEN
           b1 = '0' & renorm_c0[28:22]
       ELSE
           b1 = renorm_c0[29:22]
       ENDIF
   11:IF renorm_c0[37:30] = 0xFF THEN
           b1 = '0' & renorm_c0[29:23]
       ELSE
           b1 = renorm_c0[30:23]
       ENDIF
   12:IF renorm_c0[38:31] = 0xFF THEN
           b1 = '0' & renorm_c0[30:24]
       ELSE
           b1 = renorm_c0[31:24]
       ENDIF
   13:IF renorm_c0[39:32] = 0xFF THEN
           b1 = '0' & renorm_c0[31:25]
       ELSE
           b1 = renorm_c0[32:25]
       ENDIF
   14:IF renorm_c0[40:33] = 0xFF THEN
           b1 = '0' & renorm_c0[32:26]
       ELSE
           b1 = renorm_c0[33:26]
       ENDIF
ENDCASE
```

รูปที่ 4.26 แสดงโค้ดเทียมการเลือกผลลัพธ์ไบต์ที่สองกรณีไม่เพิ่มบิตผลลัพธ์

จากปัญหาที่กล่าวไว้ข้างต้นนั้น สถาปัตยกรรมที่นำเสนอได้แก้ไขปัญหาดังกล่าวด้วยการเพิ่มตาราง NLPS_Qe และตาราง NMPS_Qe ซึ่งทั้งสองตารางนี้จะทำหน้าที่เก็บค่าล่วงหน้าจากตาราง Qe ในกรณีที่ค่าดัชนีมีการเปลี่ยนแปลง โดยจะทำการอ่านค่าจากตาราง Qe, ตาราง NLPS_Qe และตาราง NMPS_Qe ไปพร้อมๆ กัน และจะทำการเลือกผลลัพธ์จากตารางใดตารางหนึ่งไปใช้งานสำหรับการประมวลผลของสัญลักษณ์ถัดไป ด้วยเทคนิคนี้ทำให้เราสามารถอ่านค่า Qe ที่จะนำไปใช้คำนวณได้ถูกต้องและแม่นยำ

นอกจากปัญหากรณีค่าดัชนีเปลี่ยนแปลงแล้ว การเร่งความเร็วในการประมวลผลของสถาปัตยกรรมที่นำเสนอนั้น ยังส่งผลกระทบต่อการคำนวณค่า A, B และ C โดยสถาปัตยกรรมที่นำเสนอได้แก้ไขปัญหาเหล่านี้ด้วยการคาดเดาผลลัพธ์ค่า A, B และ C ล่วงหน้า ดังแสดงตัวอย่างในตารางที่ 4.5 – 4.8

ตารางที่ 4.5 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Qe ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง

Symbol Number	CX	D	I	I_{NLPS}	I_{NMPS}	I_{Update}	Qe	Qe_{NLPS}	Qe_{NMPS}	Qe_{Final}
4374	1	0	29	27	30	30	0x1101	0x1401	0x0AC1	0x1101
4375	1	0	29	27	30	30	0x1101	0x1401	0x0AC1	0x0AC1
4376	1	0	30	28	31	30	0x0AC1	0x1201	0x09C1	0x0AC1

ตารางที่ 4.6 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง

Symbol Number	A of $P1/P3_1$	A of P2	A of $P3_2$	A of $P3_3$	A_{Final}
4374	0x7F54	0x8808	0xFEA8	0xFD50	0xFEA8
4375	0xF3E7	0xAC10	0xE7CE	0xCF9C	0xF3E7
4376	0xE926	0xAC10	0xD24C	0xA498	0xE926

ตารางที่ 4.7 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง

Symbol Number	C of $P1/P3_1$	C of P2	C of $P3_2$	C of $P3_3$	$C_{1\ Byte}$	$C_{2\ Bytes}$	C_{Final}
4374	0x01F15B4	0x0F82598	0x03E2B68	0x07C56D0	0x0062B68	0x0062B68	0x03E2B68
4375	0x03E3629	0x3E2B680	0x07C6C52	0x0F8D8A4	0x0063629	0x0063629	0x03E3629
4376	0x03E40EA	0x3E36290	0x07C81D4	0x0F903A8	0x00640EA	0x00640EA	0x03E40EA

ตารางที่ 4.8 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีค่าดัชนีเปลี่ยนแปลง

Symbol Number	$B_{1st\ byte}$	$B_{2nd\ byte}$	Byte Out	Two Bytes
4374	0xF7	0xDC	No	No
4375	0xF7	0xDC	No	No
4376	0xF7	0xDC	No	No

จากตัวอย่างในตารางที่ 4.5 – 4.8 การทำงานของสถาปัตยกรรมที่นำเสนอจะเริ่มจากการนำค่า CX (*CX = 1*) ของสัญลักษณ์ที่หนึ่งมาเปิดตาราง I และผลลัพธ์จากตาราง I (*I(1) = 29*) จะถูกนำมาเปิดตาราง Qe (*Qe(I(1)) = 0x1101*), ตาราง NLPS_Qe (*NLPS_Qe(I(1)) = 0x1401*) และตาราง NMPS_Qe (*NMPS_Qe(I(1)) = 0x0AC1*) โดยค่าจากตาราง NLPS_Qe และ NMPS_Qe เป็นค่าที่ได้จากการคาดเดาโอกาสของค่า Qe ในกรณีที่อาจมีการปรับปรุงค่าในตาราง I ซึ่งส่งผลมาจากการประมวลผลสัญลักษณ์ก่อนหน้า แต่ยังเปลี่ยนค่าตารางไม่ทันในขณะที่อ่านค่า I เพื่อใช้สำหรับสัญลักษณ์ปัจจุบัน

เนื่องจากสถาปัตยกรรมที่นำเสนอถูกออกแบบเพื่อเร่งให้สามารถประมวลผลแต่ละสัญลักษณ์ให้เสร็จภายใน 2 สัญญาณนาฬิกา

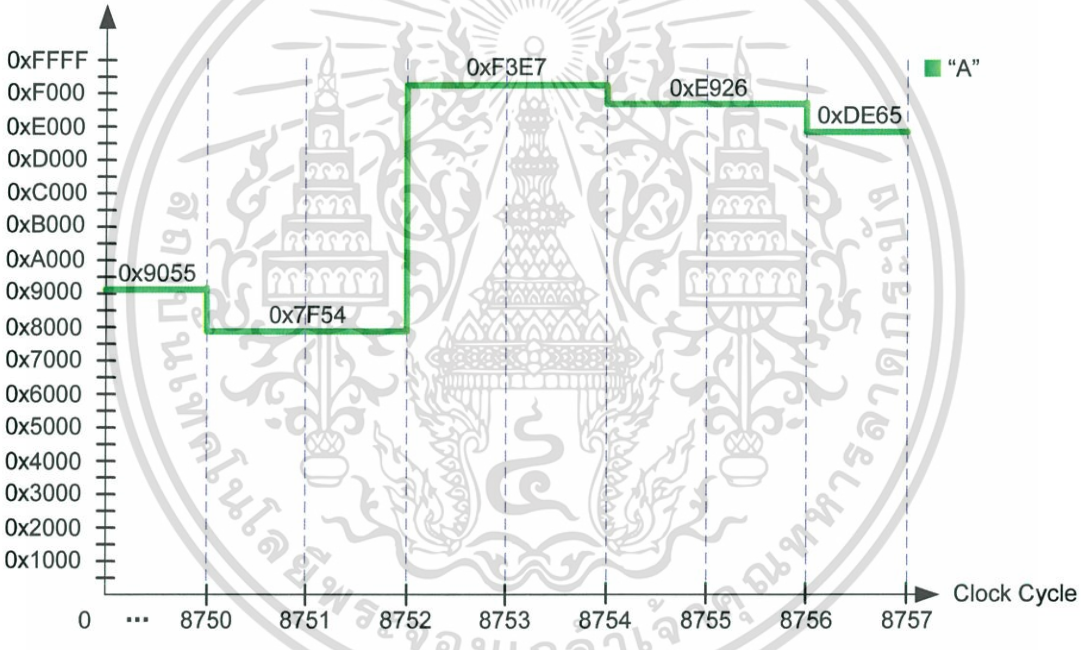
จากค่า Qe และค่าที่ได้จากการคาดเดา Qe ในกรณี NLPS และ NMPS นั้น จะทำการเลือกค่า Qe ที่ถูกต้องตามเงื่อนไขที่ได้กำหนด (*Qe(I(1)) = 0x1101*) มาใช้ในการคำนวณค่า A และ C โดยในขณะที่กำลังคำนวณค่า A และ C อยู่ สถาปัตยกรรมที่นำเสนอจะต้องนำค่า CX (*CX = 1*) ของสัญลักษณ์ที่สองมาเปิดตาราง I และนำผลลัพธ์จากตาราง I (*I(1) = 29*) มาเปิดตาราง Qe, NLPS_QE และ NMPS_QE เพื่อให้สามารถประมวลผลสัญลักษณ์ที่สองได้ทันทีที่การประมวลผลสัญลักษณ์ที่หนึ่งเสร็จสิ้น แต่เนื่องจากค่าดัชนีในตาราง I จะมีการเปลี่ยนแปลงจาก 29 เป็น 30 ภายหลังการประมวลผลสัญลักษณ์ที่หนึ่งเสร็จสิ้น ซึ่งในขณะที่นำค่า CX (*CX = 1*) ของสัญลักษณ์ที่สองมาเปิดตาราง I ค่าดัชนีในตาราง I ยังคงมีค่าเท่ากับ 29 (*I(1) = 29*) อยู่ เนื่องจากการประมวลผลสัญลักษณ์ที่หนึ่งยังไม่เสร็จสิ้น ดังนั้นค่าดัชนีที่อ่านได้จะผิดพลาด และค่าดัชนีที่ผิดพลาดนั้นจะถูกนำไปเปิดตาราง Qe และผลลัพธ์จากตาราง Qe จะนำไปประมวลผลค่า A และ C ต่อไป ถ้าเป็นเช่นนี้จะทำให้การประมวลผลเกิดความผิดพลาดเช่นนี้ไปเรื่อยๆ โดยวิธีที่นำมาแก้ปัญหานี้ คือ การเพิ่มตาราง NLPS_Qe และ NMPS_Qe ซึ่งเป็นการคาดเดาค่าของตาราง Qe ล่วงหน้าในกรณีที่ค่าดัชนีมีการเปลี่ยนแปลง ถ้าเราสามารถพิสูจน์ได้ว่าค่าดัชนีมีการเปลี่ยนแปลงอย่างไร เราก็จะสามารถเลือกค่า Qe ที่ถูกต้องตามการเปลี่ยนแปลงของค่าดัชนีนั้นได้ โดยในตัวอย่างนี้ในขณะที่กำลังประมวลผลสัญลักษณ์ที่สองอยู่นั้น สามารถพิสูจน์ได้ว่าค่าดัชนีหลังการประมวลผลสัญลักษณ์ที่หนึ่งมีการเปลี่ยนแปลงแบบ NMPS ดังนั้นค่า Qe ที่ถูกต้องนั้นจะเลือกจากตาราง NMPS_Qe (*NMPS_Qe(I(1)) = 0x0AC1*) และนำผลลัพธ์ที่ได้ไปใช้คำนวณค่า A และ C ต่อไป โดยสถาปัตยกรรมที่นำเสนอนี้จะทำการคาดเดาค่าต่างๆ ของสัญลักษณ์ที่หนึ่ง ดังนี้

- คาดเดาค่า A ล่วงหน้าทั้งหมด 4 กรณี คือ กรณี P1/P3$_1$ (*0x7F54*), กรณี P2 (*0x8808*), กรณี P3$_2$ (*0xFEA8*) และ กรณี P3$_3$ (*0xFD50*)
- คาดเดาค่า C ล่วงหน้าทั้งหมด 6 กรณี คือ กรณี P1/P3$_1$ (*0x01F15B4*), กรณี P2 (*0x0F82598*), กรณี P3$_2$ (*0x03E2B68*), กรณี P3$_3$ (*0x07C56D0*), กรณี $C_{1\,Byte}$ (*0x0062B68*) และกรณี $C_{2\,Byte}$ (*0x0062B68*)

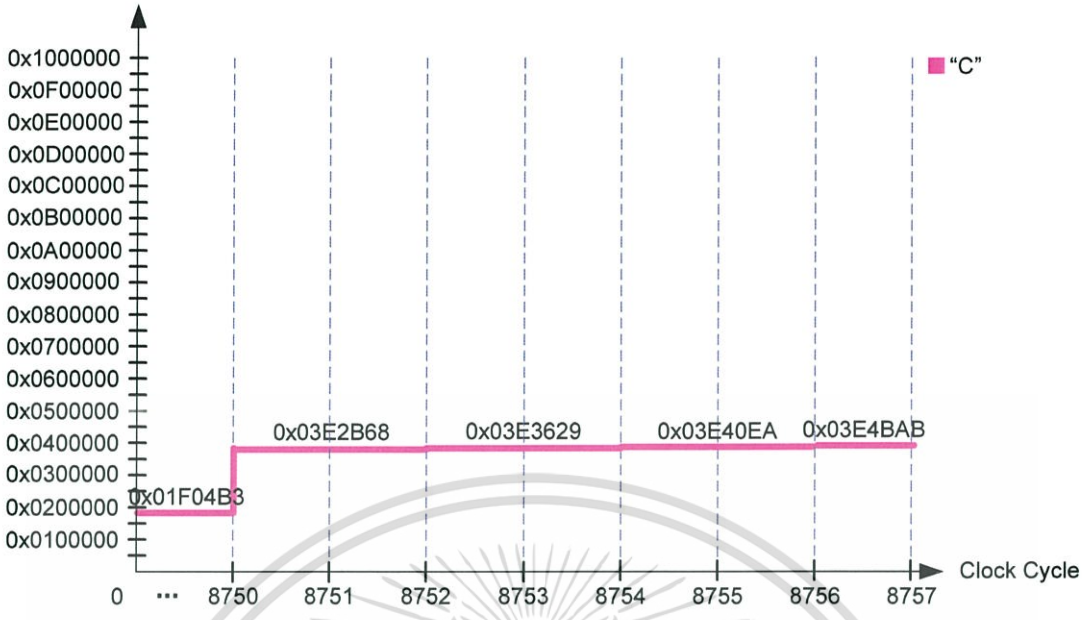
การคาดเดาค่าต่างๆ เหล่านี้จะทำงานไปพร้อมๆ กัน ส่วนขั้นตอนการเลือกค่าที่ถูกต้องนั้น (A_{Final} = *0xFEA8*, C_{Final} = *0x03E2B68*) จะกระทำภายหลังด้วยเงื่อนไขที่แตกต่างกันออกไป จากตัวอย่างในตารางที่ 4.6 และ 4.7 จะพบว่าเงื่อนไขในการเลือกค่า A และ C ที่ถูกต้องสำหรับสัญลักษณ์ที่หนึ่งเป็นไปตามกรณี P3$_2$ (A_{Final} = *0xFEA8*, C_{Final} =

0x03E2B68) สำหรับสัญลักษณ์ที่สองเป็นไปตามกรณี P1/P3$_1$ (A_{Final} = *0xF3E7*, C_{Final} = *0x03E3629*) และสำหรับสัญลักษณ์ที่สามเป็นไปตามกรณี P1/P3$_1$ (A_{Final} = *0xE926*, C_{Final} = *0x03E40EA*)

จากตัวอย่างในตารางที่ 4.5 – 4.8 สามารถแสดงการเปลี่ยนแปลงของค่า A และ C ได้ดังรูปที่ 4.27 และ 4.28 ซึ่งในตัวอย่างนี้ตรงกับเงื่อนไขที่ไม่มีการนำส่งผลลัพธ์ของการบีบอัด อย่างไรก็ดีการทำงานของสถาปัตยกรรมที่นำเสนอนั้น จำเป็นจะต้องทำการคาดเดาผลลัพธ์ล่วงหน้า เพื่อเตรียมพร้อมเผื่อไว้สำหรับเงื่อนไขที่อาจเกิดกรณีมีการนำส่งผลลัพธ์การบีบอัด ดังนั้นในตัวอย่างนี้จะทำการคาดเดาผลลัพธ์ล่วงหน้า แต่เนื่องจากเงื่อนไขไม่ตรงกับการนำส่งผลลัพธ์ จึงไม่มีการนำส่งผลลัพธ์ออกไป ทำให้ค่าผลลัพธ์การบีบอัดครั้งก่อนยังคงค้างอยู่ในค่า B ซึ่งเป็นค่าของรีจิสเตอร์เก็บผลลัพธ์อยู่



รูปที่ 4.27 แสดงการเปลี่ยนแปลงของค่า A จากตัวอย่างกรณีค่าดัชนีเปลี่ยนแปลง



รูปที่ 4.28 แสดงการเปลี่ยนแปลงของค่า C จากตัวอย่างกรณีค่าดัชนีเปลี่ยนแปลง

4.5.2 กรณีนำส่งผลลัพธ์ 1 ไบต์

จากการทำงานของส่วนเข้ารหัสเอ็มคิวตามมาตรฐาน JPEG2000 นั้น การนำส่งผลลัพธ์จะเกิดขึ้นภายหลังการคำนวณค่า A และค่า C โดยการนำส่งผลลัพธ์จะทำงานก็ต่อเมื่อค่า CT ลดลงเท่ากับศูนย์ ซึ่งค่า CT จะลดลงเท่ากับจำนวนรอบของการทำรีนอร์มอลไลซ์ การนำส่งผลลัพธ์มีความสัมพันธ์กับค่า B และ C โดยค่า B จะใช้เก็บผลลัพธ์จากการบีบอัดข้อมูล และผลลัพธ์จากการบีบอัดข้อมูลมาจากค่า C ภายหลังการนำส่งผลลัพธ์ค่า CT จะถูกกำหนดค่าใหม่ และตำแหน่งของผลลัพธ์ในค่า C จะถูกกำหนดให้มีค่าเท่ากับศูนย์

จากการเร่งความความเร็วในการประมวลผลของสถาปัตยกรรมที่นำเสนอให้สามารถประมวลผลแต่ละสัญลักษณ์เสร็จภายในสองสัญญาณนาฬิกา ทำให้ต้องปรับการทำงานของส่วนนำส่งผลลัพธ์ จากที่จะสามารถนำส่งผลลัพธ์ได้ ก็ต่อเมื่อทำการคำนวณค่า A และ C เสร็จสิ้นแล้ว เปลี่ยนมาเป็นทำการคำนวณและนำส่งผลลัพธ์ขนานไปกับการคำนวณค่า A และ C ซึ่งทำให้การปรับการทำงานของส่วนนำส่งผลลัพธ์นี้มีความซับซ้อนและยุ่งยากมาก เนื่องจากค่าของผลลัพธ์จากการบีบอัดนี้ขึ้นอยู่กับค่า A และ C ซึ่งยังคำนวณไม่แล้วเสร็จ แต่เพื่อให้สามารถประมวลผลสัญลักษณ์ถัดไปได้ทันที ทำให้จำเป็นจะต้องทำการคาดเดาผลลัพธ์ล่วงหน้าแล้ว ดังนั้นเพื่อให้สามารถประมวลผลเสร็จภายในสองสัญญาณนาฬิกา จำเป็นจะต้องทำการคาดเดาค่า A, B และ C ล่วงหน้าไป

พร้อมๆ กัน โดยเฉพาะค่า C จะต้องทำการคาดเดาทั้งในกรณีที่มีผลลัพธ์และไม่มีผลลัพธ์ เนื่องจากค่า C จะมีการเปลี่ยนแปลงทุกครั้งที่มีการนำส่งผลลัพธ์เกิดขึ้น

ในการประมวลผลสำหรับผลลัพธ์จากการบีบอัดของสัญลักษณ์ปัจจุบัน นอกจากจะพิจารณาเงื่อนไขของสัญลักษณ์ปัจจุบันแล้ว ยังจะต้องพิจารณาผลลัพธ์ที่ถูกนำส่งไปก่อนหน้าอีกด้วย ดังนั้นในตัวอย่างต่อไปนี้จะนำเสนอการนำส่งผลลัพธ์ใน 2 กรณี คือ กรณีที่ผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน และกรณีที่ผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

4.4.2.1 กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

วิธีพิจารณาว่าผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบันหรือไม่ สามารถพิจารณาได้จากค่าของผลลัพธ์ก่อนหน้า โดยถ้าผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF (*B = 0xFF*) หรือ มีค่าเท่ากับ 0xFE และบิตทดเท่ากับ 1 (*B = 0xFE, Carry Bit = 1*) จะส่งผลกระทบต่อผลลัพธ์ปัจจุบัน หากมีค่านอกเหนือจากเงื่อนไขเหล่านี้ผลลัพธ์ก่อนหน้าจะไม่ส่งผลกระทบต่อผลลัพธ์ไบต์ปัจจุบัน

จากการปรับการทำงานของส่วนนำส่งผลลัพธ์ให้ทำงานพร้อมกับการคำนวณค่า A และ C ดังที่กล่าวไว้ข้างต้น สามารถแสดงตัวอย่างการทำงานของส่วนนำส่งผลลัพธ์ได้ดังตารางที่ 4.9 – 4.12

ตารางที่ 4.9 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Qe ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งกระทบผลต่อผลลัพธ์ปัจจุบัน

Symbol Number	CX	D	I	I_{NLPS}	I_{NMPS}	I_{Final}	Qe	Qe_{NLPS}	Qe_{NMPS}	Qe_{Final}
4282	18	1	2	9	3	9	0x1801	0x2201	0x1601	0x1801
4283	1	1	39	37	40	37	0x0085	0x0141	0x0049	0x0085

ตารางที่ 4.10 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

Symbol Number	A of $P1/P3_1$	A of P2	A of $P3_2$	A of $P3_3$	A_{Final}
4282	0x7F83	0xC008	0xFF06	0xFE0C	0xC008
4283	0xBF83	0x8500	0x7F06	0xFE0C	0x8500

ตารางที่ 4.11 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

Symbol Number	C of $P1/P3_1$	C of P2	C of $P3_2$	C of $P3_3$	$C_{1\ Byte}$	$C_{2\ Bytes}$	C_{Final}
4282	0x0EA16FD	0x74FF7E0	0x1D42DFA	0x3A85BF4	0x007F7E0	0x007F7E0	0x007F7E0
4283	0x007F865	0x7F7E000	0x00FF0CA	0x001FE194	0x007E000	0x007E000	0x007E000

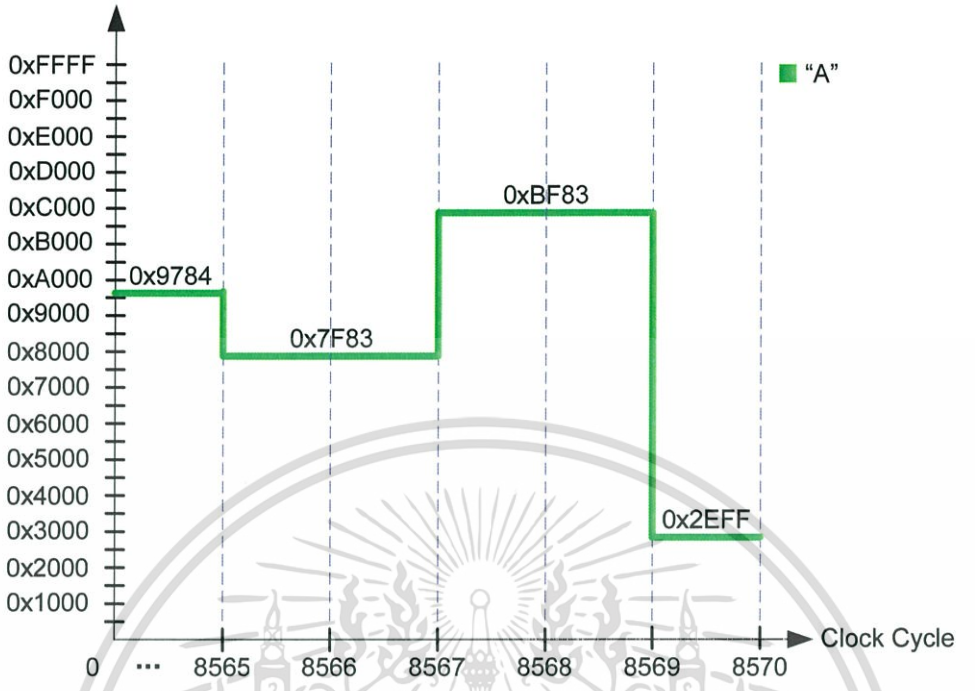
ตารางที่ 4.12 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าไม่ส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

Symbol Number	$B_{1st\ byte}$	$B_{2nd\ byte}$	Byte Out	Two Bytes
4282	0xE9	0xDC	Yes	No
4283	0xFE	0xDC	Yes	No

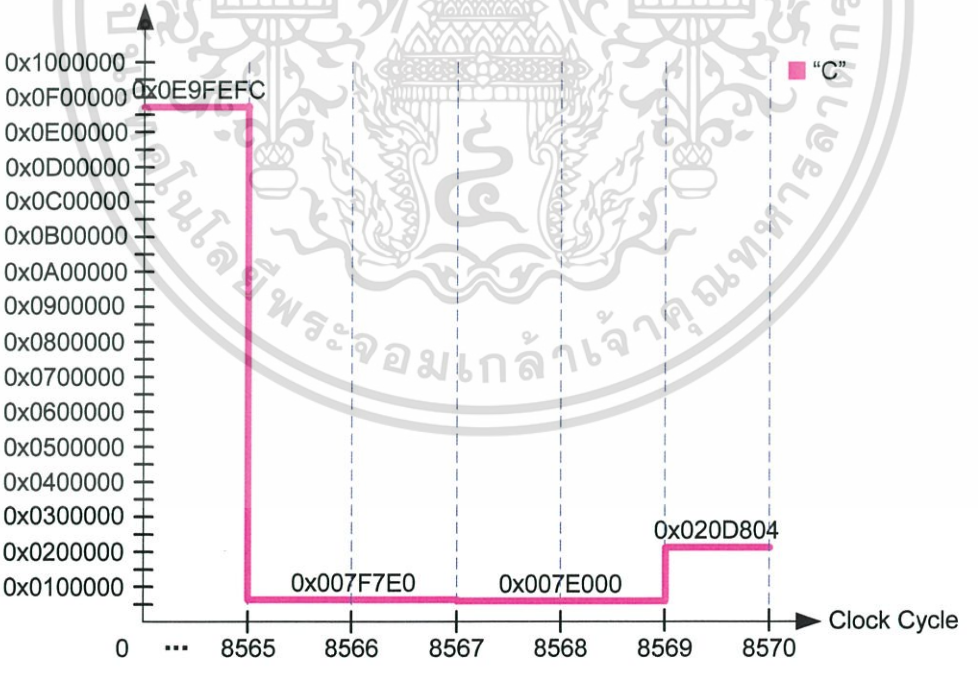
จากตัวอย่างในตารางที่ 4.9 – 4.12 การทำงานของสถาปัตยกรรมที่นำเสนอจะเริ่มจากการนำค่า CX (*CX = 18*) ของสัญลักษณ์ที่หนึ่งมาเปิดตาราง I และผลลัพธ์จากตาราง I (*I(18) = 29*) จะถูกนำมาเปิดตาราง Qe (*Qe(I(18)) = 0x1801*), ตาราง NLPS_Qe (*NLPS_Qe(I(18)) = 0x2201*) และตาราง NMPS_Qe (*NMPS_Qe(I(18)) = 0x1601*) จากนั้นจะทำการเลือกผลลัพธ์ที่ถูกต้อง (*Qe(I(18)) = 0x1801*) มาใช้ในการคำนวณค่า A, B และ C โดยสถาปัตยกรรมที่นำเสนอนี้จะทำการคาดเดาค่าต่างๆ ดังนี้

- คาดเดาค่า A ล่วงหน้าทั้งหมด 4 กรณี คือ กรณี P1/$P3_1$ (*0x7F83*), กรณี P2 (*0xC008*), กรณี $P3_2$ (*0xFF06*) และ กรณี $P3_3$ (*0xFE0C*)
- คาดเดาค่า B ล่วงหน้าทั้งหมด 2 กรณี คือ $B_{1st\ Byte}$ (*0xE9*) และ $B_{2nd\ Byte}$ (*0xFE*)
- คาดเดาค่า C ล่วงหน้าทั้งหมด 6 กรณี คือ กรณี P1/$P3_1$ (*0x0EA16FD*), กรณี P2 (*0x74FF7E0*), กรณี $P3_2$ (*0x1D42DFA*), กรณี $P3_3$ (*0x3A85BF4*), กรณี $C_{1\ Byte}$ (*0x007F7E0*) และกรณี $C_{2\ Byte}$ (*0x007F7E0*)

การคาดเดาค่าต่างๆ เหล่านี้จะทำงานไปพร้อมๆ กัน ส่วนขั้นตอนการเลือกค่าที่ถูกต้องนั้น (A_{Final} = *0xC008*, $B_{1st\ Byte}$ = *0xE9*, C_{Final} = *0x007F7E0*) จะกระทำภายหลังด้วยเงื่อนไขที่แตกต่างกันออกไป เนื่องจากการประมวลผลสัญลักษณ์ที่หนึ่งจะได้ผลลัพธ์จำนวน 1 ไบต์ ดังนั้นค่า C ที่ถูกต้องจะเลือกจากกรณี $C_{1\ Byte}$ และเลือกผลลัพธ์จาก $B_{1st\ Byte}$ โดยผลลัพธ์จะอยู่ในช่วงบิตที่ 17 ถึง 24 ($B_{1st\ Byte}$ = *0xE9*) ของ C (*C = 0x0E9FEFC*) และผลลัพธ์ปัจจุบันจะไม่ส่งผลกระทบต่อผลลัพธ์ถัดไป เนื่องจากผลลัพธ์ก่อนหน้ามีค่าไม่เท่ากับ 0xFF ($B \neq 0xFF$) และ ไม่เท่ากับ 0xFE และ Carry Bit ที่เหลือไม่เท่ากับ 1 ($B \neq 0xFE$, *Carry Bit* $\neq 1$) ส่วนผลลัพธ์จาก $B_{2nd\ Byte}$ จะทำการคาดเดาล่วงหน้าเช่นเดียวกันแต่เนื่องจากเงื่อนไขตรงกับการนำส่งผลลัพธ์จำนวน 1 ไบต์ จึงไม่มีการนำส่งผลลัพธ์ไบต์ที่สองออกไป ทำให้ค่าผลลัพธ์การบีบอัดครั้งก่อนยังคงค้างอยู่ในค่า $B_{2nd\ Byte}$ ส่วนการประมวลผลสัญลักษณ์ที่สองจะได้ผลลัพธ์จำนวน 1 ไบต์เช่นเดียวกัน ดังนั้นค่า C ที่ถูกต้องจะเลือกจากกรณี $C_{1\ Byte}$ และเลือกผลลัพธ์จาก $B_{1st\ Byte}$ เช่นเดียวกัน สามารถแสดงการเปลี่ยนแปลงของค่า A และ C ได้ดังรูปที่ 4.29 และ 4.30



รูปที่ 4.29 แสดงการเปลี่ยนแปลงของค่า A จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าไม่ส่งกระทบผลต่อผลลัพธ์ปัจจุบัน



รูปที่ 4.30 แสดงการเปลี่ยนแปลงของค่า C จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าไม่ส่งกระทบผลต่อผลลัพธ์ปัจจุบัน

4.5.2.2 กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

จากการทำงานของส่วนเข้ารหัสเอ็มคิวตามมาตรฐาน JPEG2000 นั้น ผลลัพธ์ที่ถูกนำส่งก่อนหน้าอาจจะส่งผลกระทบต่อผลลัพธ์ปัจจุบัน ดังนั้นการทำงานของส่วนนำส่งผลลัพธ์จะต้องทำการตรวจสอบผลลัพธ์ก่อนหน้าทุกครั้ง เพื่อหาว่าผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบันหรือไม่ โดยผลกระทบที่ได้รับจากผลลัพธ์ก่อนหน้า คือ ตำแหน่งของผลลัพธ์ในค่า C และการกำหนดค่าเริ่มต้นของ CT ที่เปลี่ยนแปลงไป สามารถแสดงตัวอย่างการทำงานของส่วนนำส่งผลลัพธ์ได้ดังตารางที่ 4.13 – 4.16

ตารางที่ 4.13 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Qe ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งกระทบผลต่อผลลัพธ์ปัจจุบัน

Symbol Number	CX	D	I	I_{NLPS}	I_{NMPS}	I_{Final}	Qe	Qe_{NLPS}	Qe_{NMPS}	Qe_{Final}
1052	1	1	40	38	41	38	0x0049	0x0111	0x0025	0x0049
1053	0	1	46	46	46	46	0x5601	0x5601	0x5601	0x5601

ตารางที่ 4.14 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

Symbol Number	A of $P1/P3_1$	A of $P2$	A of $P3_2$	A of $P3_3$	A_{Final}
1052	0xF3C4	0x9200	0xE788	0xCF10	0x9200
1053	0x3BFF	0XAC02	0x77FE	0xEFFC	0xEFFC

ตารางที่ 4.15 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

Symbol Number	C of $P1/P3_1$	C of $P2$	C of $P3_2$	C of $P3_3$	$C_{1\ Byte}$	$C_{2\ Bytes}$	C_{Final}
1052	0x0FF2C3C	0xE57E600	0x1FE5878	0x3FCB0F0	0x057E600	0x007E600	0x057E600
1053	0x0583C01	0x0AFCC00	0x0B07802	0x160F004	0x000F004	0x000F004	0x000F004

ตารางที่ 4.16 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีผลลัพธ์ก่อนหน้าส่งผลกระทบต่อผลลัพธ์ปัจจุบัน

Symbol Number	$B_{1st\ byte}$	$B_{2nd\ byte}$	Byte Out	Two Bytes
1052	0xFF	0x00	Yes	No
1053	0x26	0x00	Yes	No

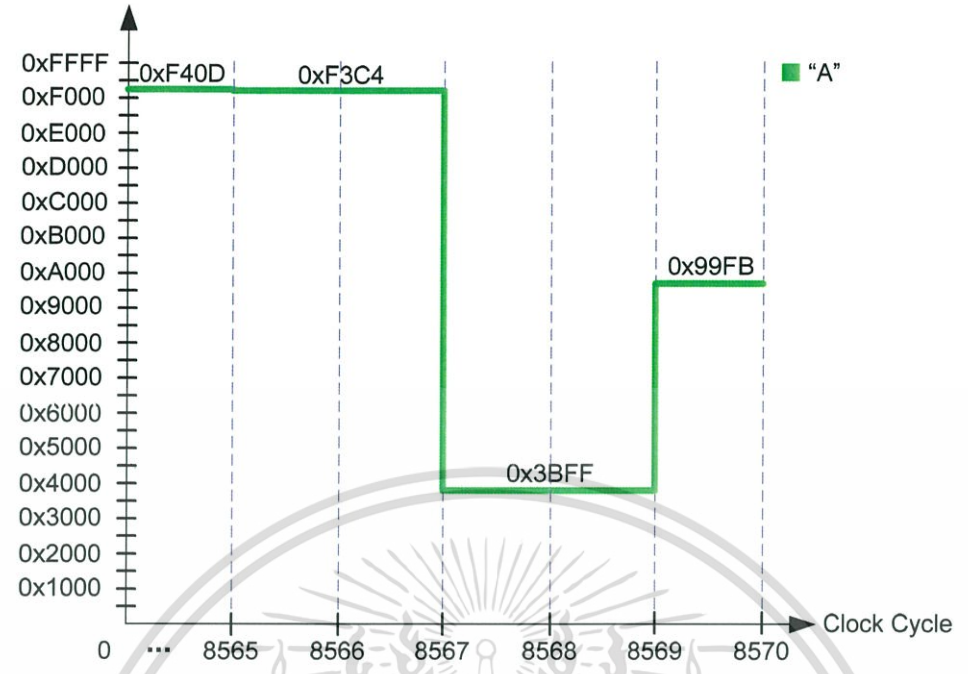
จากตัวอย่างในตารางที่ 4.13 – 4.16 การทำงานของสถาปัตยกรรมที่นำเสนอจะเริ่มจากการนำค่า CX (*CX = 1*) ของสัญลักษณ์ที่หนึ่งมาเปิดตาราง I และผลลัพธ์จากตาราง I (*I(1) = 40*) จะถูกนำมาเปิดตาราง Qe (*Qe(I(1)) = 0x0049*), ตาราง NLPS_Qe (*NLPS_Qe(I(1)) = 0x0111*) และตาราง NMPS_Qe (*NMPS_Qe(I(1)) = 0x0025*) จากนั้นจะ

ทำการเลือกผลลัพธ์ที่ถูกต้อง (*Qe(I(1)) = 0x0049*) มาใช้ในการคำนวณค่า A, B และ C โดยสถาปัตยกรรมที่นำเสนอนี้จะทำการคาดเดาค่าต่างๆ ดังนี้

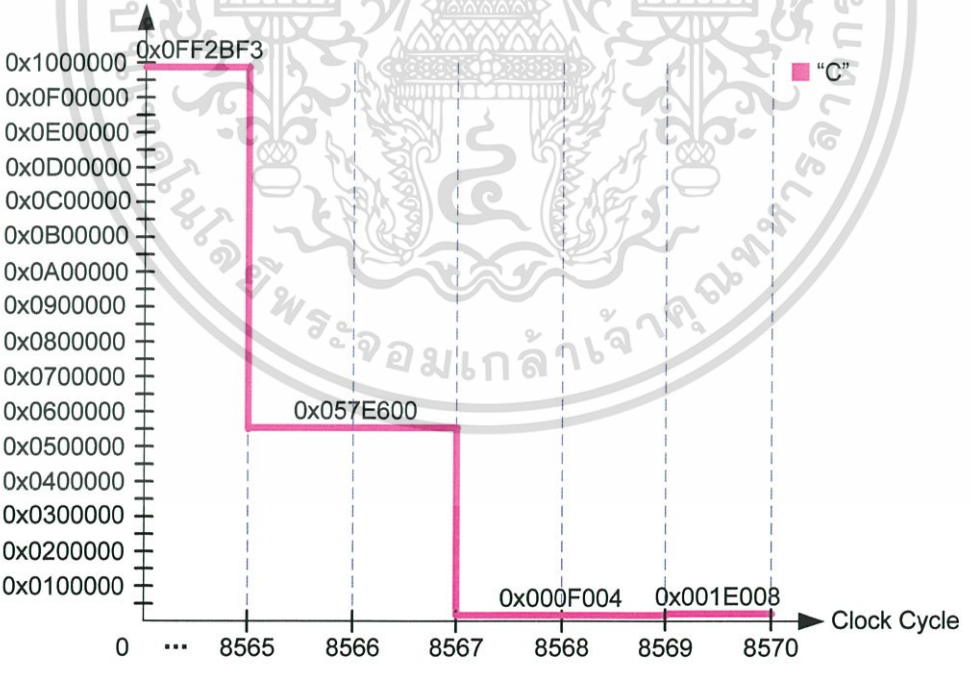
- คาดเดาค่า A ล่วงหน้าทั้งหมด 4 กรณี คือ กรณี $P1/P3_1$ (*0xF3C4*), กรณี P2 (*0x9200*), กรณี $P3_2$ (*0xE788*) และ กรณี $P3_3$ (*0xCF10*)
- คาดเดาค่า B ล่วงหน้าทั้งหมด 2 กรณี คือ $B_{1st\ Byte}$ (*0xFF*) และ $B_{2nd\ Byte}$ (*0x00*)
- คาดเดาค่า C ล่วงหน้าทั้งหมด 6 กรณี คือ กรณี $P1/P3_1$ (*0x0FF2C3C*), กรณี P2 (*0xE57E600*), กรณี $P3_2$ (*0x1FE5878*), กรณี $P3_3$ (*0x3FCB0F0*), กรณี $C_{1\ Byte}$ (*0x057E600*) และกรณี $C_{2\ Byte}$ (*0x007E600*)

การคาดเดาค่าต่างๆ เหล่านี้จะทำงานไปพร้อมกัน ส่วนขั้นตอนการเลือกค่าที่ถูกต้องนั้น ($A_{Final} = 0x9200$, $B_{1st\ Byte} = 0xFF$, $C_{Final} = 0x057E600$) จะกระทำภายหลังด้วยเงื่อนไขที่แตกต่างกันออกไป เนื่องจากการประมวลผลสัญลักษณ์ที่หนึ่งจะได้ผลลัพธ์จำนวน 1 ไบต์ ดังนั้นค่า C ที่ถูกต้องจะเลือกจากกรณี $C_{1\ Byte}$ และเลือกผลลัพธ์จาก $B_{1st\ Byte}$ โดยผลลัพธ์จะอยู่ในช่วงบิตที่ 17 ถึง 24 ($B_{1st\ Byte} = 0xFF$) ของ C ($C = 0x0FF2BF3$) และผลลัพธ์ปัจจุบันจะส่งผลกระทบต่อผลลัพธ์ของสัญลักษณ์ถัดไป เนื่องจากผลลัพธ์ปัจจุบันมีค่าเท่ากับ 0xFF ($B = 0xFF$) โดยตำแหน่งของผลลัพธ์ถัดไปจะเปลี่ยนแปลงจากช่วงบิตที่ 17 ถึง 24 เป็นช่วงบิตที่ 18 ถึง 25 ของค่า C และช่วงของผลลัพธ์ใหม่นี้มีขนาดเท่ากับ 7 บิต ดังนั้นเพื่อให้ผลลัพธ์มีขนาดครบ 1 ไบต์ จำเป็นจะต้องทำการเติมค่า "0" ลงในบิตที่ 8 และกำหนดค่าเริ่มต้นใหม่ให้กับค่า CT จาก 8 เป็น 7 ซึ่งผลกระทบที่เกิดขึ้นนี้จะเกิดขึ้นก็ต่อเมื่อผลลัพธ์ก่อนหน้ามีค่าเท่ากับ 0xFF ($B = 0xFF$) และ เท่ากับ 0xFE และ Carry Bit เท่ากับ 1 ($B = 0xFE$, $Carry\ Bit = 1$) เท่านั้น

ส่วนผลลัพธ์จาก $B_{2nd\ Byte}$ จะทำการคาดเดาล่วงหน้าเช่นเดียวกัน แต่เนื่องจากเงื่อนไขตรงกับการนำส่งผลลัพธ์จำนวน 1 ไบต์ จึงไม่มีการนำส่งผลลัพธ์ไบต์ที่สองออกไป ทำให้ค่าผลลัพธ์การบีบอัดครั้งก่อนยังคงค้างอยู่ในค่า $B_{2nd\ Byte}$ ส่วนการประมวลผลสัญลักษณ์ที่สองจะได้ผลลัพธ์จำนวน 1 ไบต์เช่นเดียวกัน ดังนั้นค่า C ที่ถูกต้องจะเลือกจากกรณี $C_{1\ Byte}$ ($C = 0x000F004$) และเลือกผลลัพธ์จาก $B_{1st\ Byte}$ ($B = 0x26$) เช่นเดียวกัน โดยตำแหน่งของผลลัพธ์และค่า CT จะเปลี่ยนแปลงดังที่กล่าวไว้ข้างต้น สามารถแสดงการเปลี่ยนแปลงของค่า A และ C ได้ดังรูปที่ 4.31 และ 4.32



รูปที่ 4.31 แสดงการเปลี่ยนแปลงของค่า A จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าส่งกระทบผลต่อผลลัพธ์ปัจจุบัน



รูปที่ 4.32 แสดงการเปลี่ยนแปลงของค่า C จากตัวอย่างกรณีผลลัพธ์ก่อนหน้าส่งกระทบผลต่อผลลัพธ์ปัจจุบัน

4.5.3 กรณีการนำส่งผลลัพธ์ 2 ไบต์

จากการเร่งความความเร็วในการประมวลผลของสถาปัตยกรรมที่นำเสนอ ส่งผลให้การคาดเดาค่า B และ C จะต้องทำการคาดเดาทั้งในกรณีที่ไม่มีผลลัพธ์ มีผลลัพธ์จำนวน 1 ไบต์ และมีผลลัพธ์จำนวน 2 ไบต์ และในกรณีที่จำนวนผลลัพธ์เท่ากับ 2 ไบต์นั้น จะต้องทำการคาดเดาว่าผลลัพธ์ไบต์ที่หนึ่งควรจะอยู่ในตำแหน่งใด ส่งผลกระทบอย่างไรต่อผลลัพธ์ไบต์ที่สองหรือไม่ และถ้าส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง ตำแหน่งของผลลัพธ์ไบต์ที่สองจะเปลี่ยนแปลงอย่างไร นอกจากการคาดเดาผลลัพธ์ทั้งสองไบต์แล้ว สถาปัตยกรรมที่นำเสนอนั้นจะต้องทำการคาดเดาค่า C ด้วยว่า หากมีการนำส่งผลลัพธ์จำนวน 2 ไบต์ จะส่งผลให้ค่า C เปลี่ยนแปลงอย่างไร

จากการปรับการทำงานของส่วนนำส่งผลลัพธ์ให้ทำงานพร้อมกับการคำนวณค่า A และ C อีกทั้งส่วนนำส่งผลลัพธ์ยังสามารถคาดเดาผลกระทบจากผลลัพธ์ไบต์ที่หนึ่งซึ่งมีต่อผลลัพธ์ไบต์ที่สอง ดังที่กล่าวไว้ข้างต้น สามารถแสดงตัวอย่างการทำงานของส่วนนำส่งผลลัพธ์ได้ดังตารางที่ 4.13 – 4.16

ตารางที่ 4.17 ตัวอย่างการคาดเดาผลลัพธ์จากตาราง Qe ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง

Symbol Number	CX	D	I	I_{NLPS}	I_{NMPS}	I_{Final}	Qe	Qe_{NLPS}	Qe_{NMPS}	Qe_{Final}
1600	1	1	41	39	42	39	0x0025	0x0085	0x0015	0x0025

ตารางที่ 4.18 ตัวอย่างการคาดเดาค่า A ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง

Symbol Number	A of $P1/P3_1$	A of P2	A of $P3_2$	A of $P3_3$	A_{Final}
1600	0xC0F8	0x9400	0x81F0	0x03E0	0x9400

ตารางที่ 4.19 ตัวอย่างการคาดเดาค่า C ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง

Symbol Number	C of $P1/P3_1$	C of P2	C of $P3_2$	C of $P3_3$	$C_{1\ Byte}$	$C_{2\ Bytes}$	C_{Final}
1600	0x1FE8318	0xA0BCC00	0x3FD0630	0x7FA0C60	0x003CC00	0x00BCC00	0x00BCC00

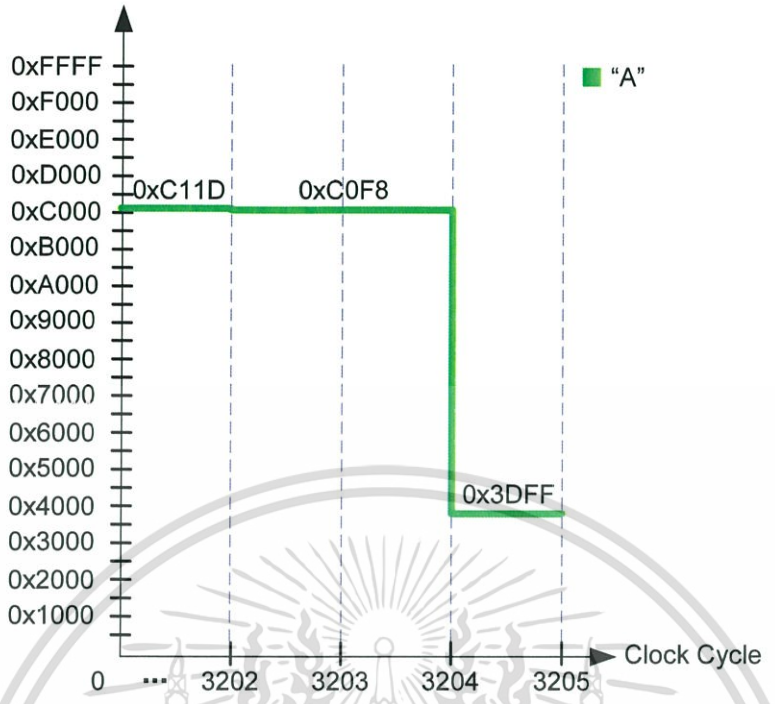
ตารางที่ 4.20 ตัวอย่างการคาดเดาค่า B ล่วงหน้า กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง

Symbol Number	$B_{1st\ byte}$	$B_{2nd\ byte}$	Byte Out	Two Bytes
1600	0xFF	0x20	Yes	Yes

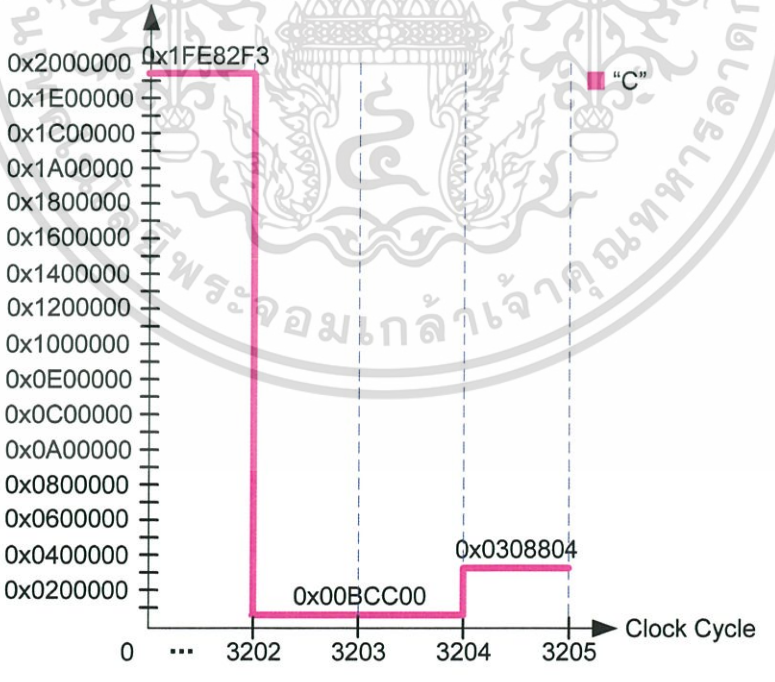
จากตัวอย่างในตารางที่ 4.13 – 4.16 การทำงานของสถาปัตยกรรมที่นำเสนอจะเริ่มจากการนำค่า CX (*CX = 1*) ของสัญลักษณ์ที่หนึ่งมาเปิดตาราง I และผลลัพธ์จากตาราง I (*I(1) = 41*) จะถูกนำมาเปิดตาราง Qe (*Qe(I(1)) = 0x0025*), ตาราง NLPS_Qe (*NLPS_Qe(I(1)) = 0x0085*) และตาราง NMPS_Qe (*NMPS_Qe(I(1)) = 0x0015*) จากนั้นจะทำการเลือกผลลัพธ์ที่ถูกต้อง (*Qe(I(1)) = 0x0025*) มาใช้ในการคำนวณค่า A, B และ C โดยสถาปัตยกรรมที่นำเสนอนี้จะทำการคาดเดาค่าต่างๆ ดังนี้

- คาดเดาค่า A ล่วงหน้าทั้งหมด 4 กรณี คือ กรณี P1/$P3_1$ (*0xC0F8*), กรณี P2 (*0x9400*), กรณี $P3_2$ (*0x81F0*) และ กรณี $P3_3$ (*0x03E0*)
- คาดเดาค่า B ล่วงหน้าทั้งหมด 2 กรณี คือ $B_{1st\ Byte}$ (*0xFF*) และ $B_{2nd\ Byte}$ (*0x20*)
- คาดเดาค่า C ล่วงหน้าทั้งหมด 6 กรณี คือ กรณี P1/$P3_1$ (*0x1FE8318*), กรณี P2 (*0xA0BCC00*), กรณี $P3_2$ (*0x3FD0630*), กรณี $P3_3$ (*0x7FA0C60*), กรณี $C_{1\ Byte}$ (*0x003CC00*) และกรณี $C_{2\ Byte}$ (*0x00BCC00*)

การคาดเดาค่าต่างๆ เหล่านี้จะทำงานไปพร้อมๆ กัน ส่วนขั้นตอนการเลือกค่าที่ถูกต้องนั้น ($A_{Final} = 0x9400$, $B_{1st\ Byte} = 0xFF$, $B_{2nd\ Byte} = 0x20$, $C_{Final} = 0x00BCC00$) จะกระทำภายหลังด้วยเงื่อนไขที่แตกต่างกันออกไป เนื่องจากการประมวลผลจะได้ผลลัพธ์จำนวน 2 ไบต์ ดังนั้นค่า C ที่ถูกต้องจะเลือกจากกรณี $C_{2\ Byte}$ ส่วนผลลัพธ์ไบต์ที่หนึ่งจะเลือกจาก $B_{1st\ Byte}$ โดยผลลัพธ์ไบต์ที่หนึ่งจะอยู่ในช่วงบิตที่ 18 ถึง 25 ($B_{1st\ Byte} = 0xFF$) ของ C (*C = 0x01FE82F3*) และผลลัพธ์ไบต์ที่หนึ่งจะส่งผลกระทบต่อผลลัพธ์ที่สอง เนื่องจากผลลัพธ์ไบต์ที่หนึ่งมีค่าเท่ากับ 0xFF ดังนั้นตำแหน่งของผลลัพธ์ไบต์ที่สองจะเปลี่ยนแปลงไป โดยตำแหน่งของผลลัพธ์ไบต์ที่สองที่เปลี่ยนแปลงไปจะอยู่ในช่วงบิตที่ 9 ถึง 16 ซึ่งมีขนาดเท่ากับ 7 บิต ดังนั้นเพื่อให้ผลลัพธ์มีขนาดครบ 1 ไบต์ จำเป็นจะต้องทำการเติมค่า “0” ลงในบิตที่ 8 โดยวิธีการเติมค่า “0” ลงในผลลัพธ์นี้มีชื่อเรียกว่า “Bit Stuffing” สามารถแสดงการเปลี่ยนแปลงของค่า A และ C ได้ดังรูปที่ 4.31 และ 4.32



รูปที่ 4.33 แสดงการเปลี่ยนแปลงของค่า A กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง



รูปที่ 4.34 แสดงการเปลี่ยนแปลงของค่า C กรณีผลลัพธ์ไบต์ที่หนึ่งส่งผลกระทบต่อผลลัพธ์ไบต์ที่สอง

บทที่ 5
ผลการทดลองและการวิเคราะห์

งานวิจัยที่นำเสนอเป็นสถาปัตยกรรมการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวแบบประมวลผลสัญลักษณ์คู่เหมาะสำหรับการนำไปประยุกต์ใช้งานประเภทเรียลไทม์ เนื่องจากได้ถูกออกแบบเพื่อปรับปรุงส่วนประมวลผลให้สามารถประมวลผลได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา ความสำเร็จนี้เกิดจากการออกแบบส่วนคาดเดาค่าพารามิเตอร์ต่างๆ ได้อย่างมีประสิทธิภาพ ทำให้ช่วยกระชับขั้นตอนการทำงาน และลดเวลาในการประมวลผลลง ทำให้ใช้เวลาในการบีบอัดข้อมูลน้อย การออกแบบสถาปัตยกรรมการบีบอัดข้อมูลที่ได้นำเสนอในงานวิจัยนี้ได้ใช้เทคนิคต่างๆ เข้าช่วยดังที่ได้กล่าวไว้ในบทที่ 4

ในบทนี้จะนำเสนอผลการทดลองและการวิเคราะห์ผลลัพธ์ที่ได้จากการทดลองการทำงานสำหรับการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวด้วยโครงสร้างสถาปัตยกรรมที่นำเสนอในงานวิจัยนี้ การทดลองเริ่มจากการเตรียมข้อมูลสำหรับการบีบอัด ในการทดลองนี้ได้เลือกใช้ภาพ "lena.jpg" ขนาด 512 × 512 พิกเซล เป็นภาพต้นฉบับสำหรับการทดลอง ซึ่งจะนำมาผ่านกระบวนการตามมาตรฐาน JPEG2000 ด้วยโปรแกรม JJ2000 เพื่อสร้างไฟล์อินพุตและไฟล์เอาต์พุตสำหรับส่วนเข้ารหัสเอ็มคิวโดยไฟล์อินพุตเป็นไฟล์ซึ่งประกอบด้วยสัญลักษณ์ (CX, D) ที่จะถูกนำไปใช้สำหรับการทดลองบีบอัด ส่วนไฟล์เอาต์พุตเป็นผลลัพธ์หลังจากการบีบอัดข้อมูล ซึ่งจะใช้สำหรับตรวจสอบความถูกต้องของโครงสร้างสถาปัตยกรรมการบีบอัดข้อมูลที่นำเสนอในงานวิจัยนี้ ซึ่งโปรแกรม JJ2000 นี้ถูกสร้างขึ้นเพื่อใช้ทดลองการทำงานตามมาตรฐาน JPEG2000

สถาปัตยกรรมการบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวที่นำเสนอในงานวิจัยนี้ แบ่งออกได้เป็น 3 ส่วน คือ

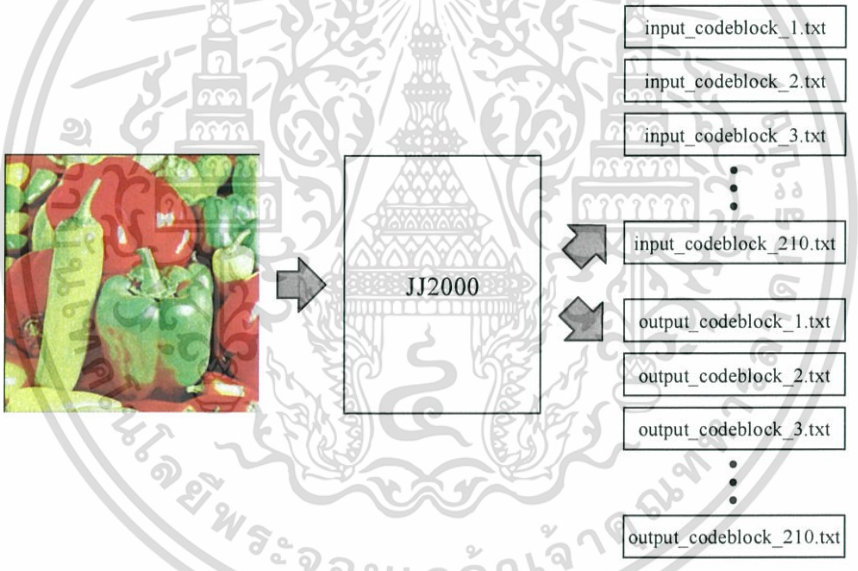
1) ส่วนติดต่ออินพุต
2) ส่วนเข้ารหัสเอ็มคิว
3) ส่วนติดต่อเอาต์พุต

โดยอินพุตสำหรับส่วนเข้ารหัสเอ็มคิวจะอยู่ในส่วนติดต่ออินพุตและเอาต์พุตสำหรับส่วนเข้ารหัสเอ็มคิวจะอยู่ในส่วนติดต่อเอาต์พุตเมื่อทำการทดลองเสร็จสิ้น ผลการทดลองจะถูกนำมาเปรียบเทียบความถูกต้องกับไฟล์เอาต์พุตจากโปรแกรม JJ2000 จากนั้นทำการวิเคราะห์ขนาดของทรัพยากร ความเร็วของสัญญาณนาฬิกา และเวลาที่ใช้ในการบีบอัดข้อมูลเปรียบเทียบกับงานวิจัย [2]

5.1 ขั้นตอนการเตรียมข้อมูลสำหรับการบีบอัดด้วยส่วนเข้ารหัสเอ็มคิว

การทดลองเริ่มจากการเตรียมข้อมูลสำหรับการบีบอัด ในการทดลองนี้ได้เลือกใช้ภาพ "peppers.ppm" ขนาด 512 × 512 พิกเซล เป็นภาพตั้งต้นสำหรับการทดลอง นำมาผ่านกระบวนการตามมาตรฐาน JPEG2000 ด้วยโปรแกรม JJ2000 ซึ่งโปรแกรม JJ2000 นี้ถูกสร้างขึ้นเพื่อใช้ทดลองการทำงานตามมาตรฐาน JPEG2000 ส่วนผลลัพธ์หลังผ่านกระบวนการตามมาตรฐาน JPEG2000 ด้วยโปรแกรม JJ2000 จะได้ไฟล์อินพุตและไฟล์เอาต์พุตสำหรับการทดลอง โดยไฟล์อินพุตจะใช้สำหรับการทดลองบีบอัด ส่วนไฟล์เอาต์พุตจะใช้สำหรับตรวจสอบความถูกต้อง

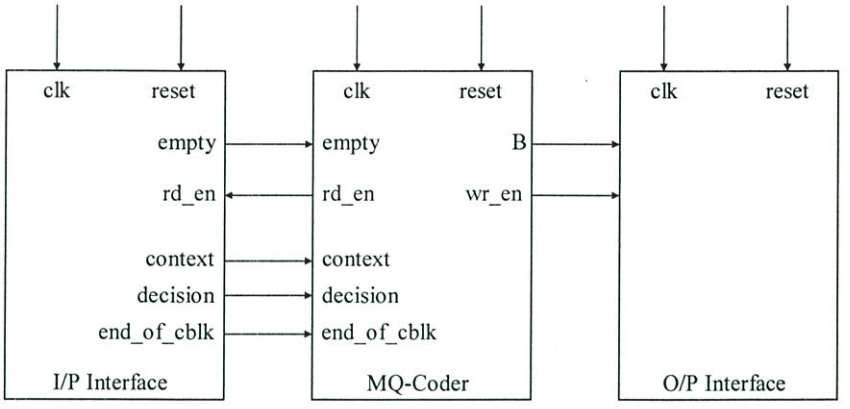
ในรูปที่ 5.1 แสดงตัวอย่างการเตรียมข้อมูลสำหรับการบีบอัด ผลลัพธ์จากการเตรียมข้อมูลประกอบด้วยไฟล์อินพุตจำนวน 210 ไฟล์ และไฟล์เอาต์พุตจำนวน 210 ไฟล์ โดยจำนวน 210 ไฟล์ของทั้งไฟล์อินพุตและไฟล์เอาต์พุตจะเท่ากับจำนวนของโค้ดบล็อกที่ทำการบีบอัด ในตัวอย่างนี้กำหนดให้ขนาดของโค้ดบล็อกเท่ากับ 64 × 64 พิกเซล



รูปที่ 5.1 แสดงตัวอย่างการสร้างไฟล์อินพุตและไฟล์เอาต์พุตด้วยโปรแกรม JJ2000

5.2 ขั้นตอนการทดลองการบีบอัดด้วยส่วนเข้ารหัสเอ็มคิว

ผลลัพธ์จากการเตรียมข้อมูลจะประกอบด้วยไฟล์อินพุตสำหรับการบีบอัด และไฟล์เอาต์พุตสำหรับการตรวจสอบความถูกต้อง จากนั้นทำการออกแบบสถาปัตยกรรมสำหรับการทดลองส่วนเข้ารหัสเอ็มคิวดังแสดงในรูปที่ 5.2 โดยสถาปัตยกรรมสำหรับการทดลองนี้แบ่งออกเป็น 3 ส่วน คือส่วนติดต่ออินพุต ส่วนเข้ารหัสเอ็มคิวและส่วนติดต่อเอาต์พุต



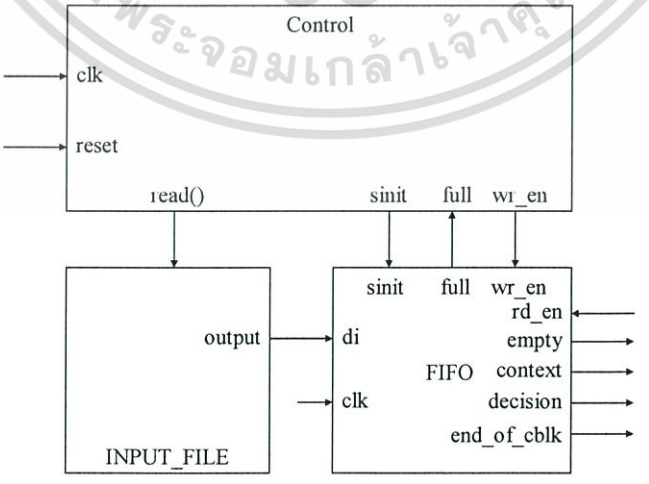
รูปที่ 5.2 แสดงสถาปัตยกรรมสำหรับการทดลองส่วนเข้ารหัสเอ็มคิว

5.2.1 ส่วนติดต่ออินพุต

การติดต่อระหว่างส่วนติดต่ออินพุตและส่วนเข้ารหัสเอ็มคิวจะใช้ FIFO (First In First Out) เป็นส่วนติดต่อระหว่างการทำงาน สามารถออกแบบสถาปัตยกรรมสำหรับส่วนติดต่ออินพุตได้ดังรูปที่ 5.3 โดยสถาปัตยกรรมสำหรับส่วนติดต่ออินพุตแบ่งออกเป็น 3 ส่วน คือ

1) ส่วน Input File
2) ส่วน Control
3) ส่วน FIFO

โดยส่วน Input File คือ ไฟล์อินพุตที่ได้จากการเตรียมข้อมูล ส่วนควบคุมการทำงานทำหน้าที่อ่านข้อมูลจากส่วนไฟล์อินพุต แล้วทำการเขียนผลลัพธ์จากการอ่านไปยังส่วน FIFO



รูปที่ 5.3 แสดงสถาปัตยกรรมส่วนติดต่ออินพุต

การทำงานของสถาปัตยกรรมสำหรับการทดลองนี้จะเริ่มจากส่วนติดต่ออินพุต อ่านข้อมูลจากไฟล์อินพุต แล้วเขียนข้อมูลไปยัง FIFO ซึ่งเป็นส่วนติดต่อกันระหว่างส่วนติดต่ออินพุตและส่วนเข้ารหัสเอ็มคิวจากนั้นเมื่อพิจารณาส่วนของส่วนเข้ารหัสเอ็มคิวซึ่งจะทำการตรวจสอบสัญญาณ empty ของ FIFO ถ้าสัญญาณ empty มีค่าเป็น "0" แสดงว่ามีข้อมูล ให้ทำการอ่านค่าจาก FIFO เพื่อนำค่าสัญลักษณ์ไปบีบอัดข้อมูล แต่ถ้าสัญญาณ empty มีค่าเป็น "1" แสดงว่าไม่มีข้อมูล ให้รอจนกว่าจะมีข้อมูลเข้ามา

5.2.2 ส่วนเข้ารหัสเอ็มคิว

ส่วนเข้ารหัสเอ็มคิวเป็นส่วนหลักในโครงสร้างการบีบอัดข้อมูลที่นำเสนอ โดยเมื่อรับข้อมูลจากส่วนติดต่ออินพุตมาแล้ว จะเริ่มทำการบีบอัดข้อมูลตามขั้นตอนที่ได้นำเสนอในบทที่ 4 ในขณะที่บีบอัดข้อมูล ถ้ามีผลลัพธ์จากการบีบอัดเกิดขึ้นจากส่วนนำส่งผลลัพธ์ ส่วนเข้ารหัสเอ็มคิวจะทำการเขียนผลลัพธ์ไปยังส่วนติดต่อเอาต์พุตเพื่อพักข้อมูลก่อนนำไปใช้งานต่อไป

5.2.3 ส่วนติดต่อเอาต์พุต

ส่วนติดต่อเอาต์พุตนี้จะนำผลลัพธ์จากการเข้ารหัสด้วยส่วนเข้ารหัสเอ็มคิวที่ส่งมานำไปเปรียบเทียบกับข้อมูลในไฟล์เอาต์พุต ถ้าผลจากการเปรียบเทียบถูกต้อง แสดงว่าส่วนเข้ารหัสเอ็มคิวทำงานถูกต้อง แต่ถ้าผลจากการเปรียบเทียบไม่ถูกต้อง ผู้ออกแบบส่วนเข้ารหัสเอ็มคิวต้องปรับปรุงการทำงานจนกว่าจะถูกต้อง

การทำงานจะดำเนินการเช่นนี้ไปเรื่อยๆ จนกระทั่งสัญญาณ end_of_cblk มีค่าเป็น "1" แสดงว่าไม่มีข้อมูลสำหรับการบีบอัดในโค้ดบล็อกนี้อีกแล้ว การทำงานหลังสัญญาณ end_of_cblk มีค่าเป็น "1" คือ ส่วนติดต่ออินพุตจะต้องอ่านไฟล์อินพุตไฟล์ถัดไปส่วนเข้ารหัสเอ็มคิวจะต้องทำการรีเซตค่าที่ใช้ในการประมวลผลทั้งหมด และส่วนติดต่อเอาต์พุตจะต้องอ่านไฟล์เอาต์พุตถัดไปเพื่อเตรียมสำหรับการบีบอัดข้อมูลในโค้ดบล็อกถัดไป

5.3 ผลการทดลอง

หัวข้อนี้จะทำการทดลองเพื่อตรวจสอบการทำงานของส่วนเข้ารหัสเอ็มคิวพร้อมทั้งแสดงผลการสังเคราะห์วงจร เพื่อแสดงการทดสอบการทำงานของวงจรในส่วนต่างๆ รวมทั้งขนาดของทรัพยากรที่ใช้ในส่วนเข้ารหัสเอ็มคิวโดยจะแบ่งการทดลองออกเป็นการทดลองส่วนติดต่ออินพุต ส่วนเข้ารหัสเอ็มคิวคอร์ (MQ-Core) และส่วนติดต่อเอาต์พุต

5.3.1 ผลการทดลองส่วนติดต่ออินพุต

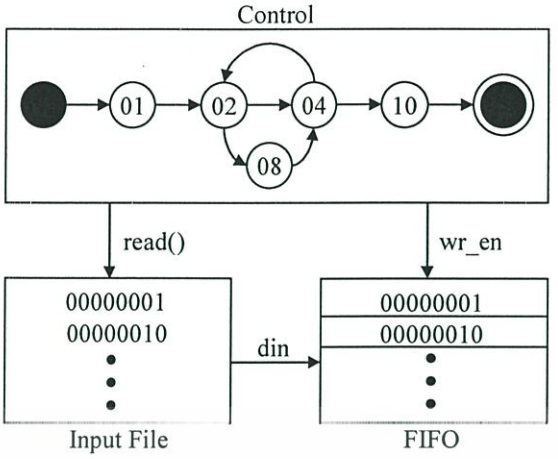
หัวข้อนี้จะแสดงตัวอย่างผลการทดลองทดสอบการทำงานของส่วนติดต่ออินพุต โดยใช้การจำลองการทำงาน ดังแสดงวงจรเบื้องต้นในรูปที่ 5.4 และผลการทดลองในรูปที่ 5.5

จากวงจรเบื้องต้นของส่วนติดต่ออินพุตในรูปที่ 5.4 ส่วนควบคุมการทำงานจะใช้ไฟไนต์สเตจแมชชีน (Finite State Machine) ควบคุมการทำงานของส่วนติดต่ออินพุต สามารถแบ่งการทำงานเป็น 5 สถานะ (State) โดยที่

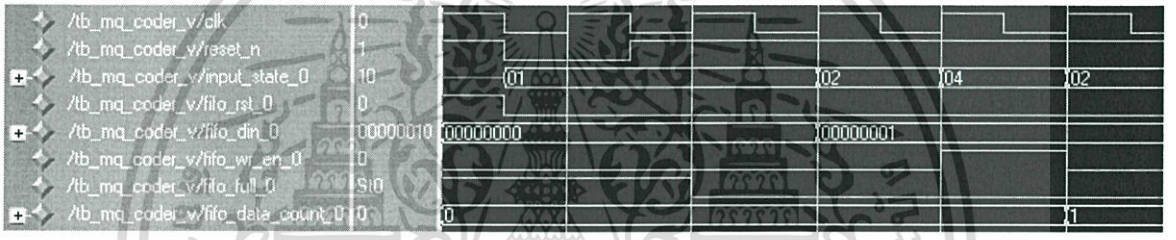
- สถานะ “01” ทำหน้าที่เปิดไฟล์อินพุต
- สถานะ “02” ทำหน้าที่อ่านไฟล์อินพุต
- สถานะ “04” ทำหน้าที่เขียนผลลัพธ์จากการอ่านไฟล์อินพุตไปยัง FIFO
- สถานะ “08” ทำหน้าที่รอ FIFO ว่าง
- สถานะ “10” ทำหน้าที่บอกสถานะสิ้นสุดไฟล์ (End of file)

จากผลการทดลองในรูปที่ 5.5 มาจากการทำงานของวงจรเบื้องต้นในรูปที่ 5.4 ได้โดยสัญญาณ input_state_0 จะใช้สำหรับเก็บค่าสถานะปัจจุบันของส่วนควบคุมการทำงาน เริ่มจากการเปิดไฟล์อินพุตในสถานะ “01” หลังจากเปิดไฟล์สำเร็จค่าของสถานะจะเปลี่ยนเป็นสถานะ “02” เพื่ออ่านค่าจากไฟล์อินพุต แล้วเก็บผลลัพธ์จากการอ่านไว้ในสัญญาณ fifo_din_0 จะสังเกตได้ว่าค่าของสัญญาณ fifo_din_0 = 00000001 ที่สถานะ “02” จากนั้นทำการตรวจสอบสถานะของสัญญาณ fifo_full_0 ถ้ามีค่าเป็น “1” ให้เปลี่ยนค่าของสถานะ เป็นสถานะ “08” เพื่อรอ FIFO ว่าง แต่ถ้าเป็น “0” ให้เปลี่ยนค่าของสถานะ เป็นสถานะ “04” เพื่อเขียนผลลัพธ์จากการอ่านไปยัง FIFO จะสังเกตได้ว่าค่าของสัญญาณ fifo_full_0 มีค่าเป็น “0” ดังนั้นค่าของสถานะ จะเปลี่ยนเป็นสถานะ “04” การเขียนข้อมูลจะกระทำโดยกำหนดให้สัญญาณ fifo_wr_en_0 = “1” หลังจากเขียนผลลัพธ์เสร็จสิ้นจะสังเกตเห็นว่าสัญญาณ fifo_data_count_0 = “1” แสดงว่ามีข้อมูลอยู่ใน FIFO จำนวน 1 ตัว จากนั้นทำการตรวจสอบว่าอ่านข้อมูลจนหมดไฟล์แล้วหรือยัง ถ้ายังอ่านไม่หมดให้เปลี่ยนค่าของสถานะ เป็นสถานะ “02” เพื่ออ่านข้อมูลถัดไป แต่ถ้าอ่านข้อมูลจนหมดไฟล์แล้วให้เปลี่ยนค่าของสถานะ เป็นสถานะ “10” เพื่อบอกสถานะสิ้นสุดไฟล์ การทำงานจะวนเช่นนี้จนกว่าจะอ่านข้อมูลจนหมดไฟล์

การติดต่อระหว่างส่วนติดต่ออินพุตและส่วนเข้ารหัสเอ็มคิวนั้นส่วนเข้ารหัสเอ็มคิวจะใช้สัญญาณ fifo_rd_en_0 ในการอ่านค่าจากส่วนติดต่ออินพุตโดยกำหนดให้สัญญาณ fifo_rd_en_0 = “1” เพื่ออ่านข้อมูลจากส่วนติดต่ออินพุตโดยผลลัพธ์จากการอ่านจะเก็บไว้ในสัญญาณ fifo_dout_0



รูปที่ 5.4 แสดงตัวอย่างการทำงานของส่วนติดต่ออินพุต



รูปที่ 5.5 แสดงผลลัพธ์การจำลองการทำงานของส่วนติดต่ออินพุต

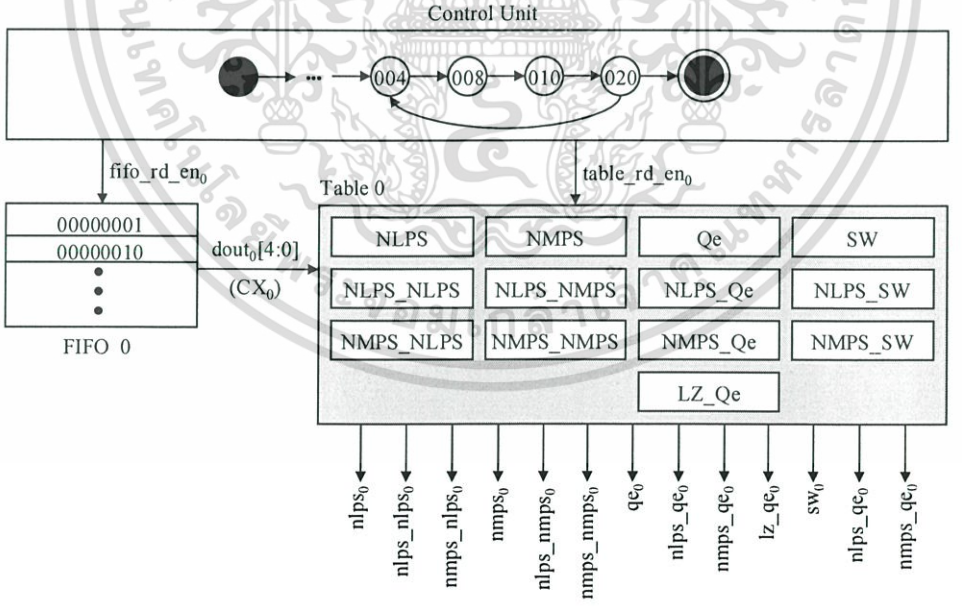
5.3.2 ผลการทดลองส่วนเข้ารหัสเอ็มคิว

หัวข้อนี้จะแสดงตัวอย่างผลการทดลองในส่วนเข้ารหัสเอ็มคิวโดยใช้การจำลองการทำงาน แสดงผลการทดลองได้ในรูปที่ 5.7, 5.8 และ 5.9 โดยรูปที่ 5.7 เป็นผลการทดลองการอ่านค่าสัญลักษณ์จากส่วนติดต่ออินพุตรูปที่ 5.8 เป็นผลการทดลองส่วนคำนวณขอบเขตบน (A) ของส่วนเข้ารหัสเอ็มคิวและรูปที่ 5.9 เป็นผลการทดลองส่วนคำนวณขอบเขตล่าง (C) ของส่วนเข้ารหัสเอ็มคิวจากผลการทดลองในรูปที่ 5.7 สามารถเขียนวงจรเบื้องต้นเพื่ออธิบายการทำงานของการอ่านค่าสัญลักษณ์จากส่วนติดต่ออินพุตได้ในรูปที่ 5.6

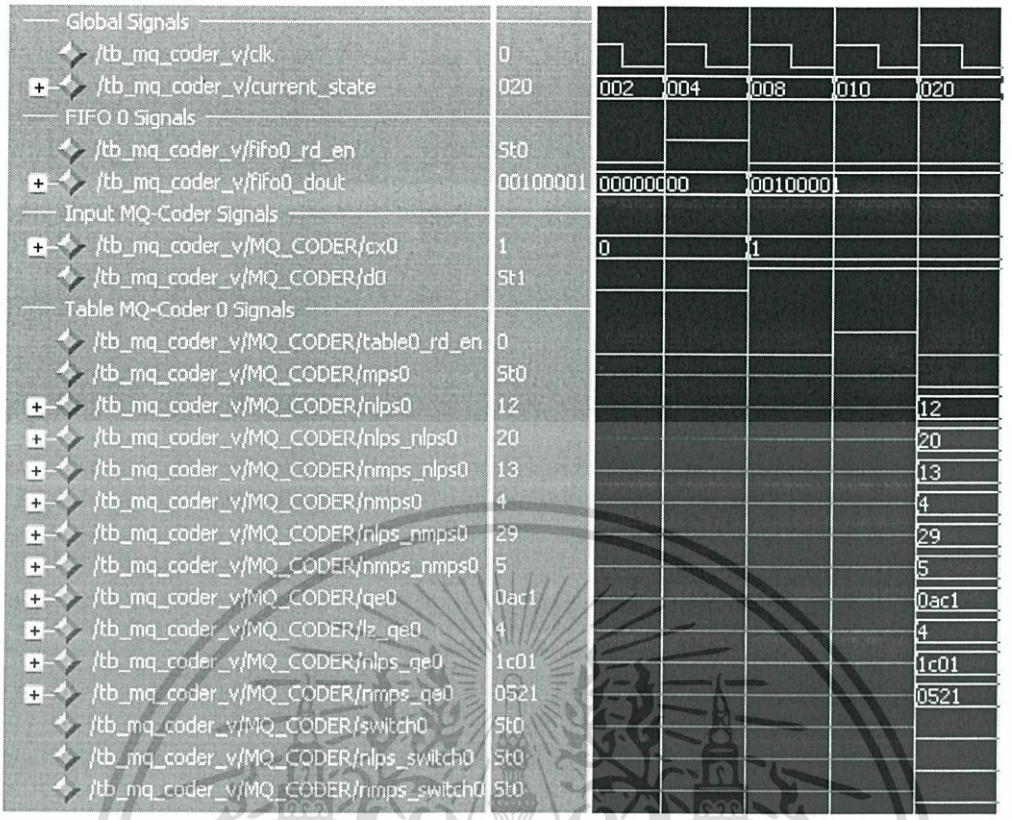
จากรูปที่ 5.6 เป็นวงจรเบื้องต้นสำหรับการอ่านค่าสัญลักษณ์จากส่วนติดต่ออินพุต ส่วนควบคุมการทำงานของส่วนเข้ารหัสเอ็มคิวจะใช้ไฟไนต์สเตจแมชชีนควบคุมการอ่านค่าสัญลักษณ์ โดยในส่วนของการอ่านค่าสัญลักษณ์จะมีการทำงานอยู่ 4 สถานะ คือ

- สถานะ “004” ทำหน้าที่อ่านข้อมูลจาก FIFO ของส่วนติดต่ออินพุต
- สถานะ “008” ทำหน้าที่รอผลลัพธ์จาก FIFO
- สถานะ “010” ทำหน้าที่อ่านข้อมูลจากตารางของส่วนเข้ารหัสเอ็มคิว
- สถานะ “020” ทำหน้าที่รอผลลัพธ์จากตาราง

จากผลการทดลองในรูปที่ 5.7 สามารถนำมาเปรียบเทียบกับวงจรเบื้องต้นในรูปที่ 5.6 ได้ โดยสัญญาณ current_state จะใช้สำหรับเก็บค่าของสถานะปัจจุบันของส่วนควบคุม การอ่านค่าสัญลักษณ์เริ่มจากกำหนดค่าให้สัญญาณ fifo0_rd_en = “1” ในสถานะ “004” เพื่ออ่านค่าสัญลักษณ์จาก FIFO โดยผลลัพธ์จากการอ่านจะได้จากสัญญาณ fifo0_dout = 00100001 ในสถานะ “008” แต่การอ่านค่าจากตารางจะใช้สัญญาณจำนวน 5 บิต (cx0) ในที่นี้กำหนดให้ค่า cx0 เท่ากับ 5 บิตล่างของสัญญาณ fifo0_dout (fifo0_dout[4:0]) และค่า d0 จะอยู่ในตำแหน่งบิตที่ 6 (fifo0_dout[5]) จากนั้นกำหนดค่าให้สัญญาณ table0_rd_en = “1” ในสถานะ “010” เพื่ออ่านค่าจากตาราง โดยผลลัพธ์จากการอ่านค่าจะได้จากสัญญาณ mps0 = “0”, nlps0 = “12”, nlps_nlps0 = “20”, nmps_nlps0 = “13”, nmps0 = “4”, nlps_nmps0 = “29”, nmps_nmps0 = “5”, qe0 = 0x0AC1, lz_qe0 = “4”, qe_nlps0 = 0x1C01, qe_nmls0 = 0x0521, switch0 = “0”, nlps_switch0 = “0” และ nmps_switch0 = “1” ในสถานะ “020”



รูปที่ 5.6 แสดงวงจรเบื้องต้นสำหรับการอ่านค่าสัญลักษณ์จากส่วนติดต่ออินพุต



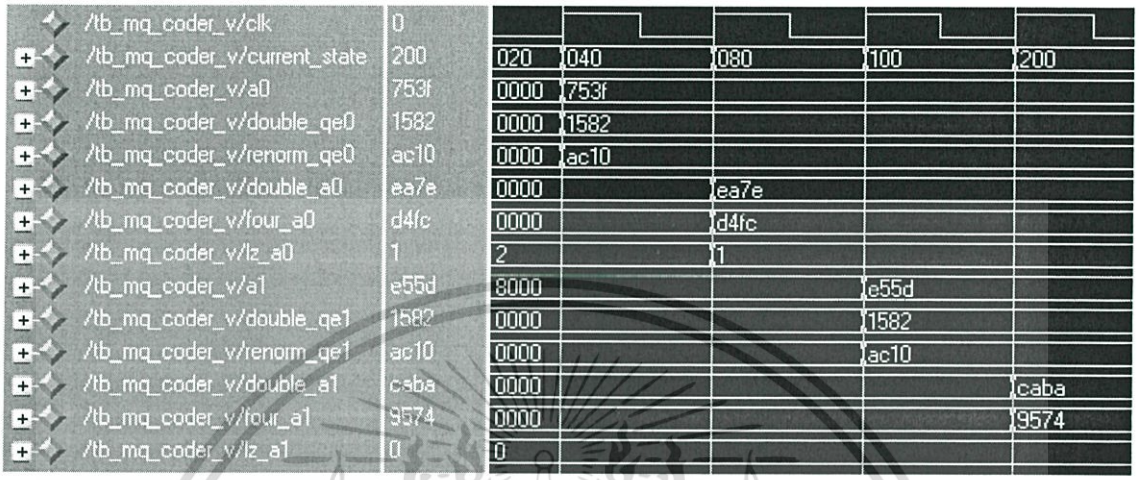
รูปที่ 5.7 แสดงผลการทดลองการอ่านค่าสัญลักษณ์จากส่วนติดต่ออินพุต

เนื่องจากส่วนควบคุมการทำงานของส่วนเข้ารหัสเอ็มคิวจะใช้ไฟไนต์สเตจแมชชีนควบคุมการคำนวณขอบเขตบน โดยในส่วนของการคำนวณขอบเขตบนจะมีการทำงานอยู่ 4 สถานะ คือ

- สถานะ “040” ทำหน้าที่คำนวณค่า a_0, $double\_qe_0$ และ $renorm\_qe_0$
- สถานะ “080” ทำหน้าที่คำนวณค่า $double\_a_0$, $four\_a_0$ และ lz_a_0
- สถานะ “100” ทำหน้าที่คำนวณค่า a_1, $double\_qe_1$ และ $renorm\_qe_1$
- สถานะ “200” ทำหน้าที่คำนวณค่า $double\_a_1$, $four\_a_1$ และ lz_a_1

จากผลการทดลองในรูปที่ 5.8 สัญญาณ current_state จะใช้สำหรับเก็บค่าค่าของสถานะปัจจุบันของส่วนควบคุมการทำงานของส่วนคำนวณขอบเขตบนเริ่มจากการคำนวณค่า a_0, $double\_qe_0$ และ $renorm\_qe_0$ ในสถานะ “040” ผลลัพธ์จากการคำนวณค่า a_0 = 0x753F, $double\_qe_0$ = 0x1582 และ $renorm\_qe_0$ = 0xAC10 จากนั้นคำนวณค่า $double\_a_0$, $four\_a_0$ และ lz_a_0 ในสถานะ “080” ผลลัพธ์จากการคำนวณค่า $double\_a_0$ = 0xEA7E, $four\_a_0$ = 0xD4FC และ lz_a_0 = “1” จากนั้นคำนวณค่า a_1, $double\_qe_1$ และ $renorm\_qe_1$ ในสถานะ “100” ผลลัพธ์จากการคำนวณค่า a_1 = 0xE55D, $double\_qe_1$ =

0x1582 และ renorm_qe$_1$ = 0xAC10 และคำนวณค่า double_a$_1$, four_a$_1$ และ lz_a$_1$ ใน สถานะ “200” ผลลัพธ์จากการคำนวณค่า double_a$_1$ = 0xCABA, four_a$_1$ = 0x9574 และ lz_a$_1$ = “0”



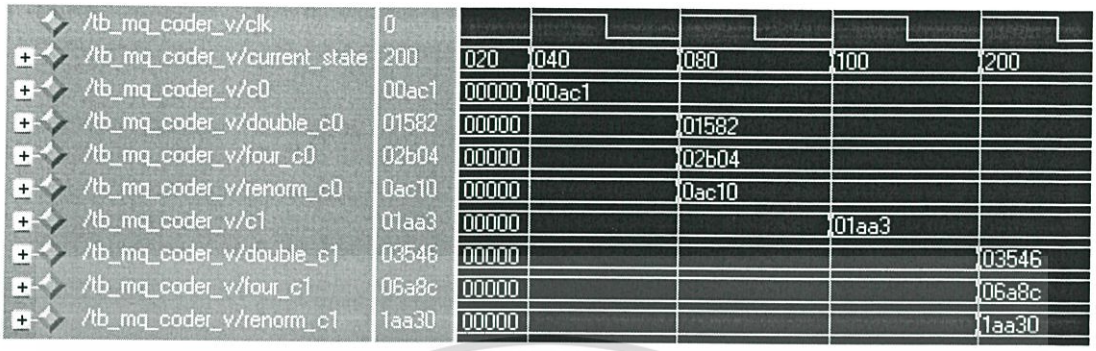
รูปที่ 5.8 แสดงผลลัพธ์การจำลองการทำงานส่วนคำนวณขอบเขตบนของส่วนเข้ารหัสเอ็มคิว

เนื่องจากส่วนควบคุมการทำงานของส่วนเข้ารหัสเอ็มคิวจะใช้ไฟไนต์สเตจแมชชีนควบคุมการคำนวณขอบเขตล่าง โดยในส่วนของการคำนวณขอบเขตล่างจะมีการทำงานอยู่ 4 สถานะ คือ

- สถานะ “040” ทำหน้าที่คำนวณค่า c_0
- สถานะ “080” ทำหน้าที่คำนวณค่า double_c_0, four_c_0 และ renorm_c_0
- สถานะ “100” ทำหน้าที่คำนวณค่า c_1
- สถานะ “200” ทำหน้าที่คำนวณค่า double_c_1, four_c_1 และ renorm_c_1

จากผลการทดลองในรูปที่ 5.9 สัญญาณ current_state จะใช้สำหรับเก็บค่าของสถานะปัจจุบันของส่วนควบคุมการทำงานของส่วนคำนวณขอบเขตบนเริ่มจากการคำนวณค่า c_0 ในสถานะ “040” ผลลัพธ์จากการคำนวณค่า c_0 = 0x00AC1 จากนั้นคำนวณค่า double_c_0, four_c_0 และ renorm_c_0 ในสถานะ “080” ผลลัพธ์จากการคำนวณค่า double_c_0 = 0x01582, four_c_0 = 0x02B04 และ renorm_c_0 = 0x0AC10 จากนั้นคำนวณค่า c_1 ในสถานะ “100” ผลลัพธ์จากการคำนวณค่า c_1 = 0x01AA3 และคำนวณค่า double_c_1, four_c_1

และ renorm_c_1 ในสถานะ “200” ผลลัพธ์จากการคำนวณค่า double_c_1 = 0x03546, four_c_1 = 0x06A8C และ renorm_c_1 = 0x01AA30



รูปที่ 5.9 แสดงผลลัพธ์การจำลองการทำงานส่วนคำนวณขอบเขตล่างของส่วนเข้ารหัสเอ็มคิว

งานวิจัยนี้ ได้นำเสนอสถาปัตยกรรมการประมวลผลสัญลักษณ์คู่สำหรับส่วนเข้ารหัสเอ็มคิวโดยการออกแบบโครงสร้างจะพยายามปรับการประมวลผลแบบลำดับขั้นเป็นแบบขนาน ที่มีโครงสร้างกระชับ สามารถดำเนินการบีบอัดข้อมูล 1 สัญลักษณ์ได้ภายใน 2 สัญญาณนาฬิกา ซึ่งเป็นผลมาจากความสามารถของส่วนคาดเดาผลลัพธ์ที่ได้ถูกออกแบบ โครงสร้างสถาปัตยกรรมการประมวลผลสัญลักษณ์คู่ในงานวิจัยนี้ ถูกทดสอบการทำงานในอุปกรณ์ประเภทเอฟพีจีเอของบริษัท Xilinx ตระกูล Spartan3 เบอร์ xc3s200-5tq144 โดยเครื่องมือที่ใช้ในการสังเคราะห์ คือ โปรแกรม Project Navigator เวอร์ชัน 8.2.03i ตารางที่ 5.1 แสดงผลการสังเคราะห์ของส่วนคำนวณขอบเขตบน ส่วนคำนวณขอบเขตล่าง (C) และส่วนนำส่งผลลัพธ์ (Byte Out) ได้ในตารางที่ 5.1 โดยความถี่สูงสุด (Maximum Frequency) ที่วงจรสามารถทำงานได้ คือ 56.20 MHz

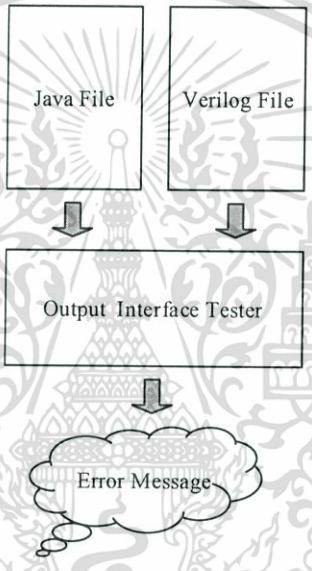
ตารางที่ 5.1 แสดงตารางทรัพยากรที่ใช้ในการออกแบบส่วนเข้ารหัสเอ็มคิวส่วนคำนวณ A และ C

	จำนวนทั้งหมด	จำนวนที่ใช้ไป	เปอร์เซ็นต์
Slices	1920	1914	99%
Flip Flops	3840	707	18%
4 input LUT	3840	3678	95%
BRAMs	12	6	50%
GCLK	8	1	12%

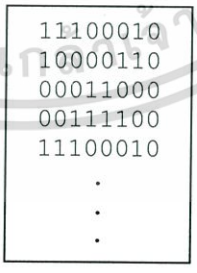
5.3.3 ผลการทดลองส่วนส่วนติดต่อเอาต์พุต

หัวข้อนี้จะแสดงตัวอย่างผลการทดลองในส่วนติดต่อเอาต์พุตโดยใช้การเปรียบเทียบของไฟล์ตั้งต้น (Java File) และไฟล์ทดสอบ (Verilog File) เพื่อตรวจสอบความถูกต้องของผลลัพธ์ที่เกิดจากการบีบอัดข้อมูล แสดงขั้นตอนการทำงานในรูปที่ 5.10 ตัวอย่างไฟล์ตั้งต้นและไฟล์ทดสอบในรูปที่ 5.11 และผลการทดลองในรูปที่ 5.12

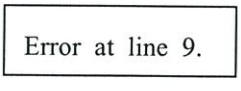
ขั้นตอนการทำงานของส่วนติดต่อเอาต์พุตเริ่มจากนำไฟล์ตั้งต้นจากขั้นตอนการเตรียมข้อมูลสำหรับการบีบอัด และไฟล์ทดสอบจากการทดลองบีบอัดของส่วนเข้ารหัสเอ็มคิวนำมาเปรียบเทียบกันด้วยส่วนทดสอบผลลัพธ์ (Output Interface Tester) ถ้าผลการเปรียบเทียบไม่ตรงกันจะเกิดข้อความแสดงความผิดพลาดขึ้น



รูปที่ 5.10 แสดงขั้นตอนการทดลองส่วนส่วนติดต่อเอาต์พุต



รูปที่ 5.11 แสดงตัวอย่างไฟล์ตั้งต้นและไฟล์ทดสอบ



รูปที่ 5.12 แสดงตัวอย่างข้อผิดพลาดจากการทดสอบ

จากการเตรียมข้อมูลสำหรับการบีบอัดในรูปที่ 5.1 ผลลัพธ์จากการเตรียมข้อมูลประกอบด้วยไฟล์เอาต์พุตหรือไฟล์ตั้งต้นจำนวน 210 ไฟล์ ส่วนติดต่อเอาต์พุตจะนำไฟล์ตั้งต้นทั้งหมด 210 ไฟล์ทำการเปรียบเทียบกับไฟล์ทดสอบทั้ง 210 ไฟล์เช่นกัน ถ้าผลจากการเปรียบเทียบถูกต้องทั้งหมด แสดงว่าส่วนเข้ารหัสเอ็มคิวทำงานถูกต้อง แต่ถ้าผลจากการเปรียบเทียบไม่ถูกต้องทั้งหมด ผู้ออกแบบส่วนเข้ารหัสเอ็มคิวต้องปรับปรุงการทำงานจนกว่าจะถูกต้องทั้งหมด

5.4 ผลการทดลองวัดประสิทธิภาพในการประมวลผลที่ใช้สำหรับส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว

ในส่วนของการทดลองจะทำการวัดความเร็วในการทำงานของการบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิวโดยจะทำการวิเคราะห์เปรียบเทียบเวลาที่ใช้ในการประมวลผล ความเร็วสูงสุดที่วงจรสามารถทำงานได้ (Maximum Clock Rate) ค่าปริมาณงานและขนาดทรัพยากรที่ใช้สำหรับส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว โดยจะทำการเปรียบเทียบกับผลการทดลองของงานวิจัย [2]

5.4.1 ผลการทดลองความเร็วในการทำงานของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว

จากการศึกษางานวิจัย [2] พบว่าถึงแม้ว่าจะมีความพยายามในการปรับการทำงานของส่วนบีบอัดข้อมูลจากลำดับขั้นเป็นการทำงานแบบขนาน แต่อย่างไรก็ดีการทำงานบางส่วนยังคงมีส่วนที่มีการทำงานแบบลำดับขั้นอยู่ โดยส่วนการทำงานแบบขนานจะทำการออกแบบและพัฒนาเพิ่มเติมจากมาตรฐานเดิม ได้แก่ ส่วนคาดเดาจำนวนครั้งของการทำรีนอร์มอลไลซ์ โดยวิธีการหาจำนวนค่าศูนย์นำหน้า และส่วนนำส่งผลลัพธ์ซึ่งสามารถนำส่งผลลัพธ์ในเวลาเดียวกันได้สูงถึง 2 ไบต์ต่อการบีบอัดข้อมูลหนึ่งสัญลักษณ์ ซึ่งนับว่าเป็นจุดเด่นของงานวิจัยนี้ สำหรับส่วนนำส่งผลลัพธ์ในงานวิจัย [2] ยังคงมีข้อจำกัดในเรื่องความล่าช้าในช่วงรอยต่อในการคำนวณค่า C และ การนำส่งผลลัพธ์ ซึ่งเกิดจากโครงสร้างการนำส่งผลลัพธ์ในงานวิจัย [2] นี้ยังคงมีบางส่วนเป็นแบบลำดับขั้นอยู่ โดยหากนำมาเปรียบเทียบเวลา (จำนวนสัญญาณนาฬิกา) ที่ใช้ในการบีบอัดข้อมูล จะแสดงให้เห็นถึงความเร็วของการทำงาน ดังตารางที่ 5.2 โดยงานวิจัย [2] จะใช้เวลาในการบีบอัดข้อมูล 1 สัญลักษณ์ต่อ 3 สัญญาณนาฬิกา แต่เฉพาะสัญลักษณ์แรกเท่านั้นที่จะใช้เวลา 5 สัญญาณนาฬิกา เนื่องจากในช่วงเริ่มต้นจะเสียเวลาในการเปิดตารางข้อมูล 2 สัญญาณนาฬิกา นอกจากนี้ข้อจำกัดของงานวิจัย [2] ยังอยู่ที่ส่วนการนำส่งผลลัพธ์ นั่นคือเมื่อใดก็ตามที่มีการนำส่งผลลัพธ์จะต้องใช้เวลาในการประมวลผลเพิ่มขึ้นอีก 4 สัญญาณนาฬิกา รวมทั้งสิ้น

7 สัญญาณนาฬิกา ต่อ 1 สัญลักษณ์ เมื่อเปรียบเทียบกับงานวิจัยที่นำเสนอนี้ จะพบว่าใช้เวลาในการบีบอัดข้อมูลเพียง 2 สัญญาณนาฬิกา แต่เฉพาะสัญลักษณ์แรกเท่านั้นที่จะใช้เวลา 4 สัญญาณนาฬิกา เนื่องจากใช้เวลาในการเปิดตารางข้อมูล 2 สัญญาณนาฬิกา จุดเด่นของงานวิจัยที่นำเสนออยู่ที่โครงสร้างการออกแบบส่วนคาดเดาค่าเงื่อนไขและตำแหน่งของผลลัพธ์ที่กระชับ ทำให้สามารถประมวลผลได้พร้อมกับส่วนคำนวณค่า A และค่า C ส่งผลให้การนำส่งผลลัพธ์ไม่จำเป็นต้องใช้เวลาในการประมวลผลเพิ่มเติมจากการประมวลผลค่า A และ C ปกติ

ตารางที่ 5.2 แสดงจำนวนสัญญาณนาฬิกาที่ใช้ในการบีบอัดข้อมูล 1 สัญลักษณ์

ส่วนเข้ารหัสเอ็มคิว	จำนวนสัญญาณนาฬิกาต่อสัญลักษณ์
งานวิจัย [2]	3 (*)
งานวิจัยที่นำเสนอ	2 (**)

* การบีบอัดข้อมูลของสัญลักษณ์ที่หนึ่งจะใช้เวลา 5 สัญญาณนาฬิกา แต่หลังจากนั้นจะใช้เวลา 3 สัญญาณนาฬิกา จนกระทั่งจบการทำงาน ยกเว้นภายหลังการบีบอัดข้อมูลมีการนำส่งผลลัพธ์ให้บวกเพิ่มขึ้น 4 สัญญาณนาฬิกา รวมเป็น 7 สัญญาณนาฬิกา

** การบีบอัดข้อมูลของสัญลักษณ์ที่หนึ่งจะใช้เวลา 4 สัญญาณนาฬิกา หลังจากนั้นจะใช้เวลา 2 clock cycle จนกระทั่งจบการทำงาน

5.4.1.1 สำหรับงานวิจัย [2]

การทำงานของงานวิจัย [2] แบ่งเป็น 5 ขั้นตอน แสดงในรูปที่ 2.11 และ 2.12 เนื่องจากสถาปัตยกรรมของงานวิจัย [2] ออกแบบให้สามารถประมวลผลได้สองสัญลักษณ์พร้อมกัน ดังนั้นการทำงานของการเข้ารหัสสัญลักษณ์ที่หนึ่งจะเหมือนกับการเข้ารหัสสัญลักษณ์ที่สอง แต่เป็นการทำงานคนละช่วงเวลากัน ดังนั้นจะขออธิบายเฉพาะส่วนเพิ่มเติมจากงานวิจัย [2] ดังนี้

1) หลังการอ่านค่าจากส่วนเข้ารหัสแบบระนาบบิตจะได้ค่า CX_0 และค่า D_0 นำค่า CX_0 ไปอ่านค่าจากตาราง I ($i_0 = I(CX_0)$)และตาราง MPS ($mps_0 = MPS(CX_0)$)
2) นำผลลัพธ์จากตาราง I ไปอ่านค่าจากตาราง NLPS ($nlps_0 = NLPS(i_0)$), NMPS ($nmps_0 = NMPS(i_0)$), Qe ($qe_0 = Qe(i_0)$),

NLPS_Qe ($nlps\_qe_0$ = NLPS_Qe(i_0)), NMPS_Qe ($nmps\_qe_0$ = NMPS_Qe(i_0)) และ SWITCH ($switch_0$ = SWITCH(i_0))

3) คำนวณค่าขอบเขตบน (A) และค่าขอบเขตล่าง (C) การคำนวณค่า A เริ่มจากการนำค่าขอบเขตบนลบกับค่าความน่าจะเป็น ($a_0 = A - qe_0$) ส่วนการคำนวณค่า C เริ่มจากการนำค่าขอบเขตล่างบวกกับค่าความน่าจะเป็น ($c_0 = C + qe_0$)

4) หาจำนวนค่าศูนย์นำหน้าของผลลัพธ์จากการลบ (lz_a_0 = LZ(a_0)) จำนวนค่าศูนย์นำหน้าของค่าความน่าจะเป็น (lz_qe_0 = LZ(qe_0))

5) หาค่าของ a หลังทำการรีนอร์มอลไลซ์($renorm\_a_0 = a_0 << lz\_a_0$) ค่าของ qe หลังการทำรีนอร์มอลไลซ์ ($renorm\_qe_0 = qe_0 << lz\_qe_0$) ค่าของ c_0 หลังการทำรีนอร์มอลไลซ์ ($renorm\_c_0 = c_0 << lz\_a_0$) และค่าของ C หลังการทำรีนอร์มอลไลซ์ ($renorm\_C_0 = C << lz\_qe_0$)

6) ทำการเลือกผลลัพธ์จากการคำนวณค่า A ล่วงหน้า (a_0 หรือ $renorm\_a_0$ หรือ $renorm\_qe_0$) และทำการเลือกค่าความน่าจะเป็น (qe_nlps_0 หรือ qe_nmps_0 หรือ qe_1) นำผลลัพธ์จากการเลือกค่า A ลบกับผลลัพธ์จากการเลือกค่าความน่าจะเป็น (a_1 = select_a() - select_qe())

5.4.1.2 สำหรับงานวิจัยที่นำเสนอ

การทำงานของงานวิจัยที่นำเสนอแบ่งเป็น 3 ขั้นตอน แสดงในรูปที่ 4.2 และ 4.3 เนื่องจากสถาปัตยกรรมของงานวิจัยที่นำเสนอนี้ ถูกออกแบบให้สามารถประมวลผลได้สองสัญลักษณ์พร้อมกัน ดังนั้นการทำงานของการเข้ารหัสสัญลักษณ์ที่หนึ่งจะเหมือนกับการเข้ารหัสสัญลักษณ์ที่สอง แต่เป็นการทำงานคนละช่วงเวลากัน ดังนั้นจึงจะอธิบายเฉพาะส่วนเพิ่มเติมจากงานวิจัย [2] ดังนี้

1) หลังการอ่านค่าจากส่วนเข้ารหัสแบบระนาบบิตจะได้ค่า CX และค่า D นำค่า CX ไปอ่านค่าจากตาราง MPS (mps_0 = MPS(CX)), NLPS ($nlps_0$ = NLPS(I(CX))), NLPS_NLPS ($nlps\_nlps_0$ = NLPS_NLPS(I(CX))), NMPS_NLPS ($nmps\_nlps_0$ = NMPS_NLPS(I(CX))), NMPS ($nmps_0$ = NMPS(I(CX))), NLPS_NMPS ($nlps\_nmps_0$ = NLPS_NMPS(I(CX))), NMPS_NMPS ($nmps\_nmps_0$ = NMPS_NMPS(I(CX))), Qe (qe_0 = Qe(I(CX))), LZ_Qe (lz_qe_0 = LZ_Qe(I(CX))), NLPS_Qe ($nlps\_qe_0$

= NLPS_Qe((I(CX))), NMPS_Qe (nmps_qe$_0$ = NMPS_Qe(I(CX))), SWITCH (switch$_0$ = SWITCH(I(CX))), NLPS_SWITCH (nlps_switch$_0$ = NLPS_SWITCH(I(CX))) และ NMPS_SWITCH (nmps_switch$_0$ = NMPS_SWITCH(I(CX)))

2) คำนวณค่าขอบเขตบน (A) และค่าขอบเขตล่าง (C) การคำนวณค่า A เริ่มจากการนำค่าขอบเขตบนลบกับค่าความน่าจะเป็น (a_0 = A - qe_0) ส่วนการคำนวณค่า C เริ่มจากการนำค่าขอบเขตล่างบวกกับค่าความน่าจะเป็น (c_0 = C + qe_0)

3) หาค่าสองเท่าของ A (double_a_0 = a_0 << 1), ค่าสี่เท่าของ A (four_a_0 = a_0 << 2), ค่าของ a หลังทำการรีนอร์มอลไลซ์ (renorm_a_0 = a_0 << lz_a_0), ค่าสองเท่าของ C (double_c_0 = c_0 << 1), ค่าสี่เท่าของ C (four_c_0 = c_0 << 2) และค่าของ c_0 หลังการทำรีนอร์มอลไลซ์ (renorm_c_0 = c_0 << lz_qe_0)

4) ทำการเลือกผลลัพธ์จากการคำนวณค่า A ล่วงหน้า (a_0 หรือ renorm_a_0 หรือ renorm_qe_0) และทำการเลือกค่าความน่าจะเป็น (qe_nlps$_0$ หรือ qe_nmps$_0$ หรือ qe_1) นำผลลัพธ์จากการเลือกค่า A ลบกับผลลัพธ์จากการเลือกค่าความน่าจะเป็น (a_1 = select_a() - select_qe())

สถาปัตยกรรมการบีบอัดข้อมูลสัญลักษณ์คู่ที่นำเสนอในงานวิจัยมีการออกแบบให้ลดขั้นตอนการทำงานจากงานวิจัย [2] ลง ได้แก่

1) การอ่านค่าจากตารางทั้งหมดถูกออกแบบให้สามารถทำงานได้ภายในหนึ่งสัญญาณนาฬิกา

2) การเปลี่ยนการทำงานจากลำดับขั้นเป็นขนาน

3) การลดขั้นตอนการคำนวณค่าขอบเขตบน (A) ค่าขอบเขตล่าง (C) และการเลือกผลลัพธ์ โดยเพิ่มความสามารถในการคาดเดาค่า A ค่า C และผลลัพธ์ล่วงหน้าได้

4) การยุบขั้นตอนการเลือกค่า A ไปรวมกับขั้นตอนการลบกันระหว่างค่า A และค่าความน่าจะเป็น

5.4.2 ผลการทดลองความเร็วสูงสุดของสัญญาณนาฬิกาที่ส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิวสามารถทำงานได้

ตารางที่ 5.3 แสดงความเร็วสูงสุดของสัญญาณนาฬิกาที่ส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิวสามารถทำงานได้

ส่วนเข้ารหัสเอ็มคิว	ความถี่สูงสุด (MHz)
งานวิจัย [2] (Altera)	26.29
งานวิจัยที่นำเสนอ (Altera)	47.86
งานวิจัยที่นำเสนอ (Xilinx)	56.20

จากผลการทดลองของงานวิจัย [2] พบว่า โครงสร้างสถาปัตยกรรมที่ออกแบบถูกนำไปทดสอบกับอุปกรณ์เอฟพีจีเอของบริษัท Altera ในขณะที่งานวิจัยที่นำเสนอใช้อุปกรณ์ประเภทเอฟพีจีเอของบริษัท Xilinx จึงทำให้การเปรียบเทียบขนาดของทรัพยากร ความเร็วของสัญญาณนาฬิกา และเวลาที่ใช้ในการประมวลผลโดยตรงนั้นทำได้ยาก เนื่องจากได้ทำการทดสอบกับอุปกรณ์ที่มีโครงสร้างทางฮาร์ดแวร์ที่ค่อนข้างแตกต่างกัน ดังนั้นจึงได้ทำการนำโครงสร้างสถาปัตยกรรมสำหรับการบีบอัดข้อมูลที่นำเสนอในงานวิจัยไปทดสอบกับอุปกรณ์เอฟพีจีเอของบริษัท Altera เช่นเดียวกับงานวิจัย [2] เพื่อให้การเปรียบเทียบเป็นไปอย่างเสมอภาคกัน จากผลการทดลองวัดความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำงานได้ ดังแสดงในตารางที่ 5.3 จากผลการทดลองจะสังเกตได้ว่าค่าความเร็วสูงสุดของงานวิจัยที่นำเสนอมีความเร็ว (47.86 MHz) สูงกว่างานวิจัย [2] (26.29 MHz)

5.4.3 ผลการทดลองวัดค่าปริมาณงานของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว

จากผลการทดลองความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำงานได้ในตารางที่ 5.3 สามารถนำมาคำนวณหาค่าปริมาณงานสูงสุดได้ผลลัพธ์ดังตารางที่ 5.4 โดยวิธีการคำนวณหาค่าปริมาณงานสามารถหาได้จากค่าความเร็วสูงสุดหารด้วยจำนวนของสัญญาณนาฬิกาในการประมวลผลหนึ่งสัญลักษณ์ แต่เมื่อพิจารณาจำนวนรอบของสัญญาณนาฬิกาที่ใช้ในงานวิจัย [2] ไม่คงที่ เนื่องจากความต้องการเวลาในการประมวลผลเพิ่มเติมในกรณีมีการนำส่งผลลัพธ์หลังการบีบอัดข้อมูล ทำให้จำนวนสัญญาณนาฬิกาเพิ่มขึ้น ซึ่งส่งผลกระทบต่อการคำนวณค่าปริมาณงานของงานวิจัย [2] เนื่องจากงานวิจัย [2] จะใช้เวลาในการนำส่งผลลัพธ์ 4 สัญญาณนาฬิกา ทำให้ผลรวมจำนวนของสัญญาณนาฬิกามีค่าเท่ากับ “7” ซึ่งส่งผลให้ค่าปริมาณงานลดลง แต่จากปัญหาดังกล่าวจะไม่ส่งผล

กระทบต่อการคำนวณค่าปริมาณงานของงานวิจัยที่นำเสนอ เนื่องจากงานวิจัยที่นำเสนอได้ถูกออกแบบให้มีส่วนคาดเดาผลลัพธ์ ซึ่งสามารถประมวลผลได้พร้อมกับการคำนวณค่า A และค่า C ดังนั้นค่าปริมาณงานของงานวิจัยที่นำเสนอจะมีค่าตามตารางที่ 5.4 ส่วนค่าปริมาณงานของงานวิจัย [2] จะลดลงขึ้นอยู่กับจำนวนครั้งของการนำส่งผลลัพธ์

ตารางที่ 5.4 แสดงปริมาณงานของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว

ส่วนเข้ารหัสเอ็มคิว	ปริมาณงาน (MSymbols/Sec)
งานวิจัย [2] (Altera)	8.76
งานวิจัยที่นำเสนอ (Altera)	23.93
งานวิจัยที่นำเสนอ (Xilinx)	28.10

5.4.4 ผลการทดลองทรัพยากรของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว

จากปัญหาการใช้อุปกรณ์ในการทดลองแตกต่างกันในหัวข้อที่ 5.4.2 เพื่อให้สามารถเปรียบเทียบผลการทดลองของงานวิจัย [2] กับงานวิจัยที่นำเสนอได้ ดังนั้นจึงได้ทำการทดลองเพิ่มเติมบนอุปกรณ์ชนิดเดียวกันกับงานวิจัย [2] สามารถแสดงการใช้ทรัพยากรของงานวิจัย [2] เทียบกับงานวิจัยที่นำเสนอได้ในตารางที่ 5.5 โดยจะสังเกตได้ว่าจำนวนตรรกะพื้นฐาน (Logic Elements) ของงานวิจัยที่นำเสนอมีค่าเท่ากับ 26.93% ซึ่งมีค่ามากกว่างานวิจัย [2] ที่ใช้ไปจำนวน 7.44% สาเหตุที่ทำให้งานวิจัยที่นำเสนอใช้จำนวนทรัพยากรมากกว่างานวิจัย [2] เนื่องมาจากการเพิ่มส่วนคาดเดาค่า A ค่า C และผลลัพธ์ ซึ่งมีความซับซ้อนสูงมาก โดยโครงสร้างที่ออกแบบในงานวิจัยนี้ได้ถูกออกแบบให้กระชับและสามารถทำได้ทันช่วงเวลาคำนวณค่า A และ C เพื่อให้สัญญาณนาฬิกาที่ใช้ในการประมวลผลคงที่ ไม่ว่าจะมีการนำส่งผลลัพธ์หรือไม่ ส่งผลให้ไม่เกิดการล่าช้าในการประมวลผลในกรณีที่มีการนำส่งผลลัพธ์ ส่วนในกรณีของจำนวนบิตหน่วยความจำ (Memory Bits) ของงานวิจัยที่นำเสนอใช้ 3.16% แต่งานวิจัย [2] บอกขนาดของหน่วยความจำเป็นบล็อกหน่วยความจำ (Memory Block) หรือเรียกว่า “Block Ram” ซึ่งใช้จำนวนหน่วยความจำ 100 บล็อก แต่จากการศึกษาพบว่าการคำนวณหาค่าบิตหน่วยความจำสามารถหาได้จากการนำค่าบล็อกหน่วยความจำคูณด้วยความกว้างของข้อมูล (มีหน่วยเป็นบิต) ตัวอย่างเช่น กำหนดให้จำนวนบล็อกหน่วยความจำเท่ากับ 100 และความกว้างของข้อมูลเท่ากับ 8 บิต ดังนั้นจำนวนบิตหน่วยความจำทั้งหมดจะมีค่าเท่ากับ 800 บิต เป็นต้น แต่เนื่องจากงานวิจัย [2] ไม่ได้แสดงขนาดความกว้างของข้อมูล ทำให้ไม่สามารถคำนวณค่าบิตหน่วยความจำได้เพื่อเปรียบเทียบได้

ตารางที่ 5.5 แสดงทรัพยากรของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว

	จำนวนทั้งหมด	งานวิจัย [2] (Altera)		งานวิจัยที่นำเสนอ (Altera)	
		จำนวนที่ใช้งาน	เปอร์เซ็นต์	จำนวนที่ใช้งาน	เปอร์เซ็นต์
Total Logic Elements	24,320	1811	7.44%	6551	26.93%
Total Memory Bits	327,680	100 (BRAMs)	**NA**	10,368	3.16%

5.4.5 การวิเคราะห์ประสิทธิภาพของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิว

จากผลการทดลองของงานวิจัยที่นำเสนอนี้พบว่าความเร็วที่ใช้ในการบีบอัดข้อมูลสูงขึ้นเมื่อเทียบกับงานวิจัย [2] เนื่องจากการลดขั้นตอนที่ใช้ในการประมวลผลค่า A และค่า C ลง โดยเทคนิคที่นำมาใช้เพื่อลดขั้นตอนในการประมวลผลลง คือ การคาดเดาค่า A และค่า C ล่วงหน้า ส่งผลให้เวลาที่ใช้ในการประมวลผลลดลง 1 สัญญาณนาฬิกาต่อการบีบอัด 1 สัญลักษณ์ เมื่อเทียบกับงานวิจัย [2] นอกจากนี้งานวิจัยที่นำเสนอยังได้ออกแบบส่วนนำส่งผลลัพธ์ด้วยเทคนิคการคาดเดาตำแหน่งของผลลัพธ์ล่วงหน้า ทำให้สามารถทำงานได้พร้อมกับการประมวลผลค่า A และ C ส่งผลให้สามารถลดเวลาที่ใช้ในการประมวลผลลง 4 สัญญาณนาฬิกาต่อการบีบอัด 1 สัญลักษณ์

นอกจากเวลาที่ใช้ในการบีบอัดข้อมูลแล้ว ความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำงานได้ และค่าปริมาณงานที่ได้จากการบีบอัด ก็เป็นอีกดัชนีในการวัดประสิทธิภาพของการเข้ารหัสส่วนเข้ารหัสเอ็มคิวจากการทดลองพบว่าความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำได้สำหรับงานวิจัยที่นำเสนอสามารถทำได้ 47.86 MHz เมื่อเทียบกับความเร็วสูงสุดของวงจรในงานวิจัย [2] ทำได้ 26.29 MHz ส่วนค่าปริมาณงานของส่วนบีบอัดข้อมูลส่วนเข้ารหัสเอ็มคิวสำหรับงานวิจัยที่นำเสนอมีค่าเท่ากับ 23.93 MSymbols/Sec ซึ่งเทียบกับค่าปริมาณงานของวงจรในงานวิจัยที่ [1] ซึ่งมีค่าเท่ากับ 8.76 MSymbols/Sec

จากผลการทดลองแสดงให้เห็นว่าสถาปัตยกรรมส่วนเข้ารหัสเอ็มคิวที่นำเสนอเมื่อเทียบกับงานวิจัย [2] จะมีความเร็วในการบีบอัดข้อมูลสูงกว่า ค่าความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำได้สูงกว่า และค่าปริมาณงานยังสูงกว่า อย่างไรก็ดีประสิทธิภาพในการทำงานที่สูงขึ้นของงานวิจัยที่ได้นำเสนอแลกมาด้วยขนาดของทรัพยากรที่ใช้ในสถาปัตยกรรมการบีบอัดข้อมูลที่สูงขึ้นด้วยเช่นกัน

บทที่ 6
สรุปผลการทดลอง

วิทยานิพนธ์นี้ได้นำเสนอสถาปัตยกรรมในส่วนของการบีบอัดข้อมูลการเข้ารหัสเอ็มคิวแบบประมวลผลสัญลักษณ์คู่สำหรับการใช้งานประเภทเรียลไทม์ เนื่องจากส่วนเข้ารหัสเอ็มคิวทั่วไปมีการทำงานแบบลำดับขั้น โดยมีขั้นตอนการทำงานเรียงตามลำดับขั้น ซึ่งข้อมูลที่ส่งผ่านในแต่ละขั้นตอนนั้นมีความสัมพันธ์ขึ้นต่อกัน ทำให้สามารถประมวลผลทำได้เพียงครั้งละหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกาเท่านั้น ดังนั้นจึงออกแบบให้ส่วนเข้ารหัสเอ็มคิวทำงานแบบขนานกล่าวคือ ทำให้สามารถประมวลผลได้มากกว่าหนึ่งสัญลักษณ์ต่อสัญญาณนาฬิกา ซึ่งจะช่วยเพิ่มประสิทธิภาพในการประมวลผลลงได้ ทั้งในแง่ของความเร็วในการเข้ารหัสความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำงานได้ และค่าปริมาณงาน

6.1 สรุปและวิเคราะห์ผลการดำเนินงานวิจัย

จากผลการทดลองส่วนเข้ารหัสเอ็มคิวพบว่าความเร็วในการบีบอัดข้อมูลของงานวิจัยที่นำเสนอสูงกว่างานวิจัย [2] ด้วยประสิทธิภาพของส่วนคาดเดา ค่า A และ ค่า C รวมทั้งส่วนการคาดเดาตำแหน่งผลลัพธ์ล่วงหน้า ซึ่งทำให้การทำงานของโครงสร้างสถาปัตยกรรมที่นำเสนอสามารถนำส่งผลลัพธ์ ได้พร้อมกับการคำนวณค่า A และค่า C ส่งผลให้สามารถประมวลผลสัญลักษณ์ถัดไปได้ทันที โดยไม่จำเป็นต้องเสียเวลารอการนำส่งผลลัพธ์ให้เสร็จสิ้นเสียก่อนเหมือนในงานวิจัย [2] และด้วยโครงสร้างส่วนเข้ารหัสเอ็มคิวที่นำเสนอในงานวิจัย ทำให้สามารถบีบอัดข้อมูลในรูปแบบขนานได้อย่างมีประสิทธิภาพ

การออกแบบส่วนคาดเดาค่า A ค่า C และผลลัพธ์ล่วงหน้านั้นจะใช้มัลติเพล็กเซอร์เป็นตัวเลือกผลลัพธ์ที่ถูกต้องจากโอกาสทั้งหมดที่สามารถเกิดขึ้นได้ ซึ่งช่วยให้การประมวลผลของส่วนเข้ารหัสเอ็มคิวรวดเร็วและมีประสิทธิภาพดีกว่างานวิจัย [2] และผลจากการเพิ่มส่วนคาดเดาค่า A ส่งผลให้สามารถประมวลผลค่า A เสร็จภายใน 2 สัญญาณนาฬิกา โดยถ้าต้องการให้สามารถประมวลผลสัญลักษณ์ถัดไปได้ทันที จำเป็นจะต้องทำการออกแบบส่วนคำนวณค่า C และส่วนนำส่งผลลัพธ์ให้สามารถทำงานได้ทันกับส่วนคำนวณค่า A จากเหตุผลดังที่กล่าวมานี้ทำให้การออกแบบส่วนคำนวณค่า C มีความซับซ้อนมากยิ่งขึ้น เนื่องจากค่า C จะมีการเปลี่ยนแปลงทุกครั้งที่ส่วนนำส่งผลลัพธ์ทำงาน ดังนั้นการออกแบบส่วนคาดเดาค่า C จำเป็นจะต้องคำนึงถึงความเป็นไปได้ของค่า C ทั้งในกรณีส่วนนำผลลัพธ์ของการประมวลผลสัญลักษณ์ก่อนหน้าทำงานและไม่ทำงาน ซึ่งส่งผลให้ค่า C ของทั้งสองกรณีมีความแตกต่างกัน ส่วนความซับซ้อนในการออกแบบส่วนนำส่งผลลัพธ์นั้น จำเป็นจะต้องคำนึงถึงความเป็นไปได้ของการเกิดผลลัพธ์

รูปแบบของการเลือกผลลัพธ์ ตำแหน่งของผลลัพธ์ จำนวนของผลลัพธ์ และผลลัพธ์ก่อนหน้าซึ่งจะส่งผลต่อผลลัพธ์ปัจจุบัน เป็นต้น จะเห็นได้ว่าเงื่อนไขในการเปลี่ยนแปลงค่าของ C และ ผลลัพธ์ที่ต้องถูกนำส่งเป็นผลลัพธ์ของการบีบอัดข้อมูลนี้ มีความซับซ้อนมาก ทำให้ต้องออกแบบส่วนคาดเดาให้ครอบคลุมทุกกรณีที่เป็นไปได้ และที่สำคัญต้องสามารถประมวลผลส่วนที่ซับซ้อนเหล่านี้ให้เสร็จทันกับการคำนวณค่า A ซึ่งใช้เวลาเพียง 2 สัญญาณนาฬิกาเท่านั้น

การเปรียบเทียบงานวิจัยที่นำเสนอกับงานวิจัย [2] นอกจากเวลาที่ใช้ในการบีบอัดข้อมูลแล้ว ความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำงานได้ และค่าปริมาณงานที่ได้จากการบีบอัดข้อมูล ก็เป็นอีกดัชนีหนึ่งในการวัดประสิทธิภาพของส่วนเข้ารหัสเอ็มคิวจากการทดลองพบว่าความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำได้สำหรับงานวิจัยที่นำเสนอมีค่าเท่ากับ 47.86 MHz (Altera) เมื่อเทียบกับความเร็วสูงสุดของวงจรในงานวิจัย [2] มีค่าเท่ากับ 26.29 MHz ส่วนค่าปริมาณงานของงานวิจัยที่นำเสนอมีค่าเท่ากับ 23.93 MSymbols/Sec (Altera) ซึ่งเทียบกับค่าปริมาณงานของวงจรงานวิจัย [2] มีค่าเท่ากับ 8.76 MSymbols/Sec

จากผลการทดลองแสดงให้เห็นว่าส่วนเข้ารหัสเอ็มคิวที่นำเสนอในงานวิจัยนี้มีประสิทธิภาพในการบีบอัดข้อมูลสูงกว่างานวิจัย [2] ทั้งในเรื่องของความเร็วในการบีบอัดข้อมูล ค่าความเร็วสูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำงานได้ และค่าปริมาณงาน

อย่างไรก็ดีถึงแม้ว่างานวิจัยที่นำเสนอนี้จะมุ่งเน้นให้การบีบอัดข้อมูลของส่วนเข้ารหัสเอ็มคิวที่มีประสิทธิภาพในการบีบอัดในแง่ของความเร็ว ความเร็วสูงสุดของสัญญาณนาฬิกา และค่าปริมาณงานเพิ่มสูงขึ้น แต่ต้องแลกมาด้วยความต้องการทรัพยากรที่เพิ่มสูงมากขึ้นด้วยเช่นกัน ดังนั้นผลการทดลองในส่วนของทรัพยากรที่ใช้สำหรับโครงสร้างสถาปัตยกรรมที่นำเสนอในงานวิจัยนี้ จะพบว่าสูงกว่าที่ถูกออกแบบในงานวิจัย [2]

6.2 แนวทางในการพัฒนาต่อ

เนื่องจากเวลาที่ใช้ในการบีบอัดข้อมูลของงานวิจัยที่นำเสนอมีค่าเท่ากับ 2 สัญญาณนาฬิกาต่อการประมวลผลหนึ่งสัญลักษณ์ ทำให้การลดเวลาในการประมวลผลให้ต่ำกว่านี้ทำได้ยาก เพราะการบีบอัดข้อมูลมีการทำงานที่ซับซ้อนมาก ดังนั้นแนวทางในการพัฒนาต่อแบ่งได้ 3 แนวทาง คือ

1) การลดขนาดของฮาร์ดแวร์ เพื่อประหยัดจำนวนทรัพยากรและประหยัดพลังงาน
2) การเพิ่มความถี่ในการทำงาน เพื่อเพิ่มความเร็วในการบีบอัดข้อมูล
3) การลดเวลาในการบีบอัดข้อมูล โดยผู้พัฒนาต่อจำเป็นจะต้องศึกษาการทำงานทั้งส่วนเข้ารหัสแบบระนาบบิตและส่วนเข้ารหัสเอ็มคิวเพื่อหารูปแบบหรือความสัมพันธ์ของทั้งสองส่วน โดยออกแบบให้ทั้งสองส่วนทำงานสอดคล้องกันและสามารถประมวลผลหลายสัญลักษณ์พร้อมกันได้ เป็นต้น

เอกสารอ้างอิง

[1] Chu Yu and Hwai-Tsu Hu "**Design of an Area-Efficient ASIC Architecture for Context-Based Binary Arithmetic Coding**", *International Computer Symposium 2006 (ICS 2006)*, 2006, pp. 129-132.

[2] M. Dyer, D. Taubman, and S. Nooshabadi "**Improved throughput Arithmetic Coder for JPEG2000**", *International Conference on Image Processing 2004 (ICIP'04)*, vol. 4, 2004, pp. 2817-2820.

[3] Mathiang and O.Chitsobhuk "**Efficient pass-pipelined VLSI architecture for context modeling of JPEG2000**", *Asia-Pacific Conference on Communication*, Oct. 2007, pp. 63-66.

[4] JPEG2000 Part 1 020719 (Final Publication Draft), ISO/IEC JTC1/SC29/WG1 N1636R, Jul. 2002.

[5] Tinku Acharya and Ping-Sing Tsai, "**JPEG2000 Standard for Image Compression Concepts, Algorithms and VLSI Architectures",** John Wiley & Sons, 111 River Street, Hoboken, NJ 07030, 2005.

[6] Grzegorz Pastuszak, "**A novel architecture of Arithmetic Coder in JPEG2000 based on parallel symbol encoding**", *The International Conference on Parallel Computing in Electrical Engineering, 2004 (PARELEC'04)*, Sept. 2004, pp. 303-308.

[7] Wei He, Haiping Sun, Jinging Cai and Minglun Gao, "**Logarithmic Complexity Implementation for Leading Zeros Detector**", *The Conference on ASIC 2003*, vol. 2, Oct. 2003, pp. 761-764.

ภาคผนวก

ภาคผนวก ก

แสดงตารางที่ออกแบบเพิ่มเติมจากตารางมาตรฐาน

ตารางที่ ก.1 แสดงค่าของตาราง NLPS_Qe, ตาราง NMPS_Qe, ตาราง LZ_Qe, ตาราง NLPS_SWITCH และตาราง NMPS_SWITCH

Index	NLPS_Qe	NMPS_Qe	LZ_Qe	NLPS_SWITCH	NMPS_ SWITCH
0	0x3401	0x3401	1	0	0
1	0x5601	0x1801	2	1	0
2	0x3801	0x0AC1	3	0	0
3	0x1C01	0x0521	4	0	0
4	0x1101	0x0221	5	0	0
5	0x0521	0x0111	6	0	0
6	0x5601	0x5401	1	1	0
7	0x5601	0x4801	1	1	0
8	0x5601	0x3801	1	1	0
9	0x5601	0x3001	2	1	0
10	0x4801	0x2401	2	0	0
11	0x3801	0x1C01	2	0	0
12	0x3001	0x1601	3	0	0
13	0x2801	0x1101	3	0	0
14	0x5601	0x5401	1	1	0
15	0x5601	0x5101	1	1	0
16	0x5401	0x4801	1	0	0
17	0x5101	0x3801	1	0	0
18	0x4801	0x3401	2	0	0
19	0x3801	0x3001	2	0	0
20	0x3401	0x2801	2	0	0
21	0x3401	0x2401	2	0	0
22	0x3001	0x2201	2	0	0
23	0x2801	0x1C01	2	0	0
24	0x2401	0x1801	3	0	0
25	0x2201	0x1601	3	0	0
26	0x1C01	0x1401	3	0	0
27	0x1801	0x1201	3	0	0

ตารางที่ ก.1 (ต่อ)

Index	NLPS_Qe	NMPS_Qe	LZ_Qe	NLPS_SWITCH	NMPS_SWITCH
28	0x1601	0x1101	3	0	0
29	0x1401	0x0AC1	3	0	0
30	0x1201	0x09C1	4	0	0
31	0x1101	0x08A1	4	0	0
32	0x0AC1	0x0521	4	0	0
33	0x09C1	0x0441	5	0	0
34	0x08A1	0x02A1	5	0	0
35	0x0521	0x0221	6	0	0
36	0x0441	0x0141	6	0	0
37	0x02A1	0x0111	7	0	0
38	0x0221	0x0085	7	0	0
39	0x0141	0x0049	8	0	0
40	0x0111	0x0025	9	0	0
41	0x0085	0x0015	10	0	0
42	0x0049	0x0009	11	0	0
43	0x0025	0x0005	12	0	0
44	0x0015	0x0001	13	0	0
45	0x0009	0x0001	15	0	0
46	0x5601	0x5601	1	0	0

ตารางที่ ก.2 แสดงค่าของตาราง NLPS_NLPS, ตาราง NMPS_NLPS, ตาราง NLPS_NMPS และตาราง NMPS_NMPS

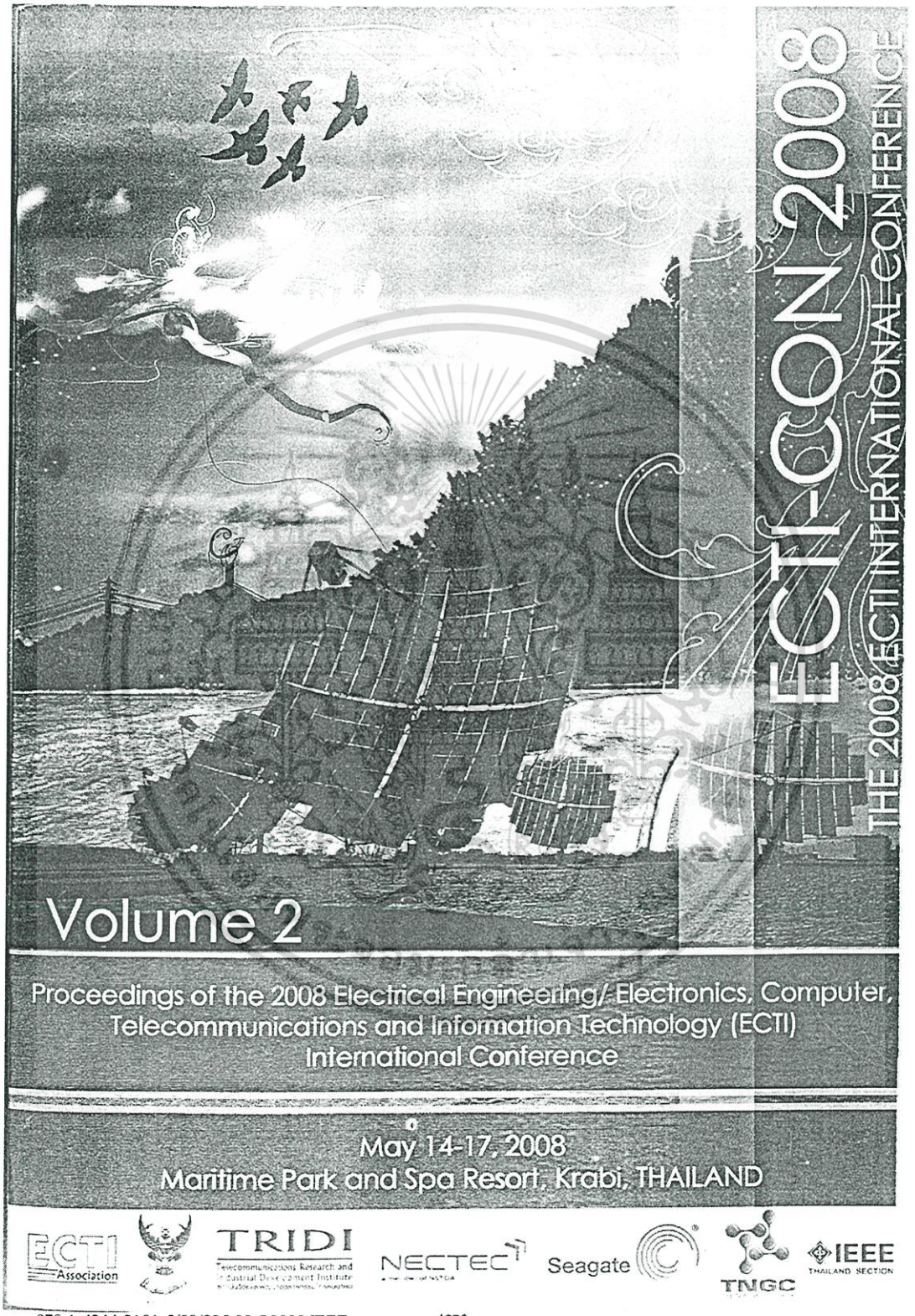
Index	NLPS_NLPS	NMPS_NLPS	NLPS_NMPS	NMPS_NMPS
0	6	2	6	2
1	6	7	9	3
2	14	10	12	4
3	20	13	29	5
4	27	30	33	38
5	31	34	36	39
6	6	7	14	8
7	14	15	14	9
8	14	15	14	10
9	14	15	17	11
10	16	18	18	12
11	17	19	20	13
12	19	21	21	29
13	19	22	27	30
14	14	15	14	16
15	14	15	15	17
16	14	16	16	18
17	15	17	17	19
18	16	18	18	20
19	17	19	19	21
20	18	20	19	22
21	18	20	20	23
22	19	21	21	24
23	19	22	22	25
24	20	23	23	26
25	21	24	24	27
26	22	25	25	28
27	23	26	26	29

ตารางที่ ก.2 (ต่อ)

Index	NLPS_NLPS	NMPS_NLPS	NLPS_NMPS	NMPS_NMPS
28	24	27	27	30
29	25	28	28	31
30	26	29	29	32
31	27	30	30	33
32	28	31	31	34
33	29	32	32	35
34	30	33	33	36
35	31	34	34	37
36	32	35	35	38
37	33	36	36	39
38	34	37	37	40
39	35	38	38	41
40	36	39	39	42
41	37	40	40	43
42	38	41	41	44
43	39	42	42	45
44	40	43	43	45
45	41	44	43	45
46	46	46	46	46

ภาคผนวก ข

งานวิจัยที่ได้รับการตีพิมพ์



978-1-4244-2101-5/08/$25.00 ©2008 IEEE

byte emission technique of the Q-Coder. MQ-Coder uses context based probability estimation of 19 contexts. Each context has an associated probability state that identifies value of the most probable symbol (MPS) and index (I) pointing to a probability estimation table (PET). After a symbol has been coded, the corresponding state is updated according to the probability mapping rules. PET determines the probability estimation (Qe) for the least probable symbol (LPS), the next index value (NLPS, NMPS) and the exchange indicator (SWITCH) of the MPS. MQ-Coder module contains a set of registers: A, B, C, CT and L. MQ-Coder is implemented using fixed precision integer arithmetic and using an integer representation of fractional values in which 0x8000 is equivalent to decimal 0.75. The interval A is kept in the range $0.75 \leq A < 1.5$. Whenever, the value of register A falls below 0.75, its value will be doubled together with the value of the register C. Then, the down counter, CT, is decremented. If the CT value reaches 0, the previous value of the register B is released to the output codestream. The value of the byte counter (L) is incremented. Finally, the high order bits of the register C are moved out to temporal buffer B.

3. The proposed MQ-Coder Architecture

Usually, the regular operation of the MQ-Coder is sequential, which can process only one symbol at a time. However, the bit-plane coding can generate more than one symbol per clock cycle. Consequently, the coding speed will be limited and bottlenecked at the interface between the output of the bit-plane coder and the input of the MQ-Coder. If the MQ-Coder is designed to support more than one input symbol, it can help to accelerate the coding process. Therefore, the proposed MQ-Coder architecture, as shown in figure 1, is a dual symbol processor, which can process two symbols for each clock cycle to minimize the bottleneck problem. This can help to increase the performance of the EBCOT. The proposed MQ-Coder can be separated into 4 main units. The first unit is the main control unit. The second unit is the PET unit, which stores all the probabilities and the index prediction values. The third unit is the upper bound unit used to calculate the value of the register A. The last unit is the lower bound unit used to calculate the value of the register C. Figure 2 shows the operation of MQ-Coder. In order to process two symbols per clock cycle, there is a need to predict the upper bound value (A) and the lower bound value (C). The regular process to calculate the upper bound value and the lower bound value is presented in the dashed box. This can be grouped into 3 patterns, for the upper bound value is (Figure 3): 1) P1: A = A – Qe, 2) P2: A = Qe then renormalized subsequently, 3) P3: A = A – Qe then renormalized subsequently, for the lower bound value is (Figure 4): 1) P1: C = C + Qe, 2) P2: C = C then renormalized subsequently, 3) P3: C = C + Qe then renormalized subsequently. When we replace the value of the register A in the decision condition ($A \geq$ 0x8000) by A = A – Qe calculated from previous step, this will result in a new decision condition ($A < 2Qe$). Considering the value of the upper bound (A), the maximum number of renormalization is 2. This comes from the difference between Min(A) = 0x8000 and Max(Qe) = 0x5601, which equals to min (A) = 0x29FF. Since the difference value has 2 leading zeros, the prediction values of P3 can be categorized into three cases ($P3_1$, $P3_2$, and $P3_3$), which corresponds to no renormalization, renormalization with one shifting, and renormalization with two shifting respectively.

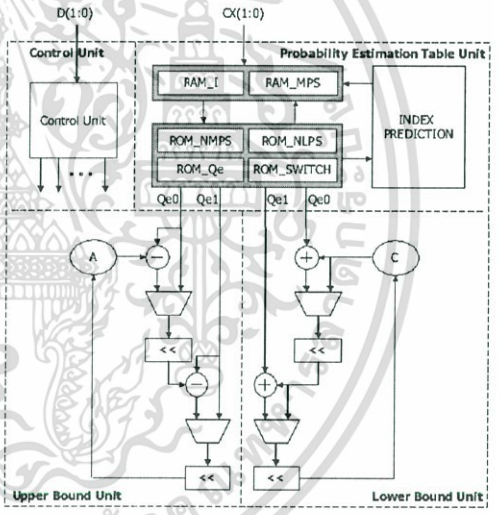


Figure 1. MQ-Coder architecture.

To compute the upper bound value and the lower bound value from a sequence of two symbols, two prediction processes must be implemented as presented in figure 5. The first prediction process aims to predict the value of register A and the value of register C resulted from the first symbol in FSEU (First Symbol Encoding Unit) while the second prediction process attempts to predict that of register A resulted and register C resulted from the second symbol in SSEU (Second Symbol Encoding Unit). Since the prediction processes of the upper bound and lower bound values are quite similar, they can be done simultaneously with the similar prediction condition. The actual value of the

register A and the register C in each prediction process depends on 4 conditions: the decision value (MPS/LPS), the number of leading zeros obtained from the leading zero detector (LZD) [7], the MSB of the result of A – Qe, and the result of the comparison of A < 2Qe. Moreover, to further reduce the computation time, the values of Qe, renormalized Qe, and 2Qe are precalculated and kept in the memory. However, to be able to predict the upper bound value, the lower bound value and the index pointed to the PET table for the second input symbol must also be estimated in advance.

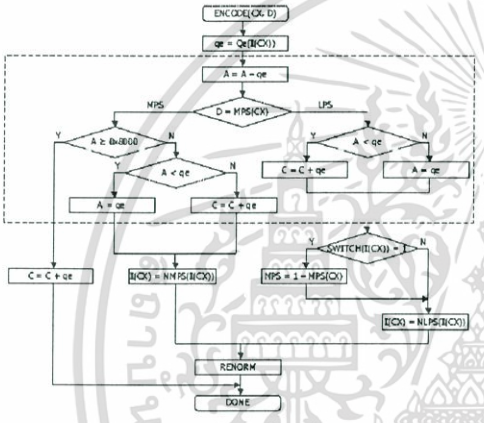


Figure 2. MQ-Coder procedure in JPEG2000.

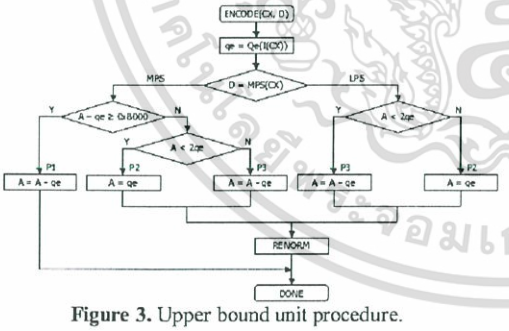


Figure 3. Upper bound unit procedure.

4. Experimental results

The proposed parallel symbol processor for MQ arithmetic coder is implemented on the Spartan-3 FPGA device using the Project Navigator 8.2.03i tool. The cost of the designed hardware for the proposed system is illustrated in table I. To evaluate the performance of the proposed system, the test vectors of input symbols obtained from images with size of 512 x 512 have been generated. The number of clock cycles used to process the sample data for an image is 2,619,383 clock cycles (excluded flush register process). The maximum frequency is 123.794 MHz.

Table II shows a comparison of the clock rate and throughput of the proposed MQ coder system and the systems proposed by [5]. The clock rate and throughput of the proposed system outperforms those of [5].

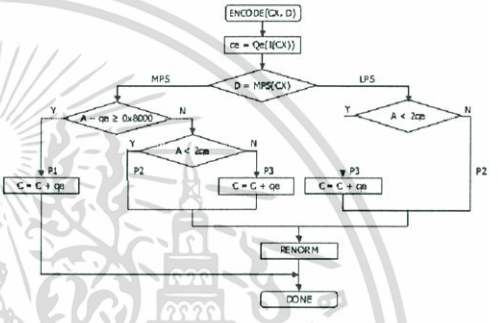


Figure 4. Lower bound unit procedure.

Table I. A hardware cost for the synthesized proposed architecture.

Technology Library	xcs3s200-5tq144
Number of Slices	901
Number of Slice Flip Flops	708
Number of 4 input LUTs	1633
Number of GClks	1
Maximum Frequency (MHz)	123.794

Table II. A comparison of clock rate and throughput.

Architecture	Clock Rate (MHz.)	Throughput (Msymbols/sec)
2 Symbol [5]	26.29	52.58
2 Symbol	123.79	61.89

5. Conclusion

This paper presents a parallel symbol encoder for arithmetic coder architecture implemented on FPGA. The proposed architecture aims to improve the performance of the JPEG2000 arithmetic encoder

(MQ-coder). Since the regular MQ-Coder is sequential, the prediction process is needed in order to be able to process two symbols in one clock cycle. The prediction processes of the upper bound value (A), the lower bound value (C) and the index value (I), are proposed. This helps to minimize the bottlenecked problem at the interface between the output of the bit-plane coder and the input of the MQ-Coder while can help to increase the performance of the JPEG2000 encoder. As a result, the dual symbol encoding speed can be greater than 100 MHz with the throughput of 61.89 Msymbols/sec. The proposed architecture could be developed to perform multiple symbol processing. However, the coding time for each image is difficult to improve since the proposed architecture is already designed to meet the critical path delay. The attempt to improve the coding system performance could be done to minimize the hardware cost or to increase the clock speed.

6. References

[1] *JPEG2000 Part 1 020719 (Final Publication Draft)*, ISO/IEC JTC1/SC29/WG1 N1636R, July, 2002.

[2] Tinku Acharya and Ping-Sing Tsai, *JPEG2000 Standard for Image Compression Concepts, Algorithms and VLSI Architectures*, John Wiley & Sons, 111 River Street, Hoboken, NJ 07030, 2005.

[3] JJ2000 website, _http://jj2000.epfl.ch/index.html_

[4] Grzegorz Pastuszak, *"A novel architecture of arithmetic coder in JPEG2000 based on parallel symbol encoding"*, The International Conference on Parallel Computing in Electrical Engineering, 2004 (PARELEC'04), Sept. 2004, pp. 303-308.

[5] M. Dyer, D. Taubman, and S. Nooshabadi, *"Improved throughput arithmetic coder for JPEG2000"*, The International Conference on Image Processing 2004 (ICIP'04), vol. 4, Oct. 2004, pp. 2817-2820.

[6] K. Mathiang and O.Chitsobhuk, *"Efficient pass-pipelined VLSI architecture for context modeling of JPEG2000"*, 2007 Asia-Pacific Conference on Communication, Bangkok, Thailand, Oct. 2007, pp 63-66.

[7] Wei He, Haiping Sun, Jinging Cai and Minglun Gao, *"Logarithmic Complexity Implementation for Leading Zeros Detector"*, The Conference on ASIC 2003, vol. 2, Oct. 2003, pp. 761-764.

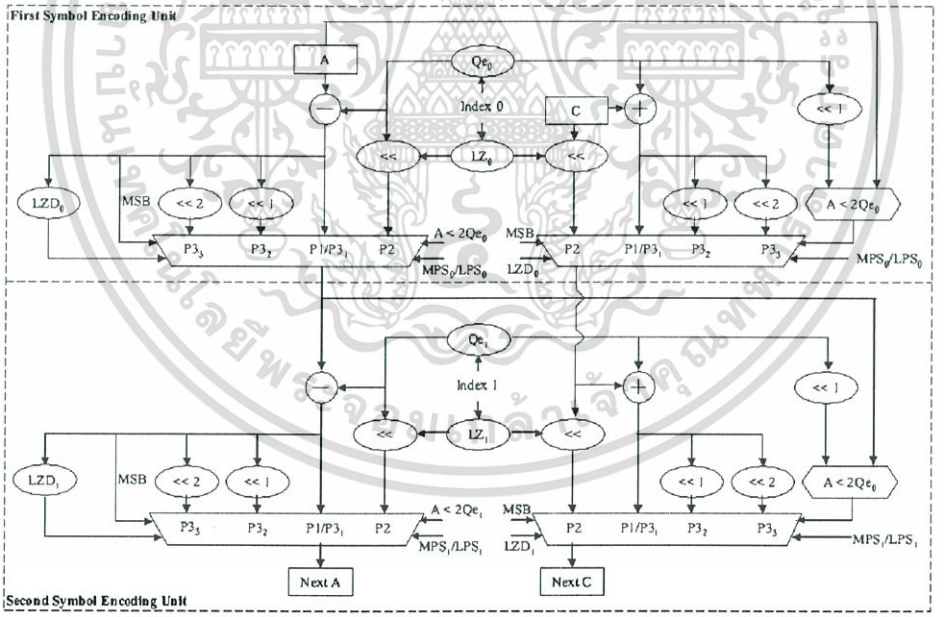


Figure 5. The proposed upper bound unit and lower bound unit.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายนพพล น้อยแก้ว
วันเดือนปีเกิด	วันที่ 9 กรกฎาคม 2524 จังหวัดสุพรรณบุรี
ที่อยู่	85 หมู่ 3 ตำบลบางตะเคียน อำเภอสองพี่น้อง จังหวัดสุพรรณบุรี 72110

ประวัติการศึกษา

พ.ศ. 2536	จบการศึกษาระดับประถมศึกษา จากโรงเรียนแม่พระประจักษ์ อำเภอสองพี่น้อง จังหวัดสุพรรณบุรี
พ.ศ. 2539	จบการศึกษาระดับมัธยมศึกษาตอนต้น จากโรงเรียนกรรณสูตศึกษาลัย อำเภอเมือง จังหวัดสุพรรณบุรี
พ.ศ. 2544	จบการศึกษาระดับประกาศนียบัตรวิชาชีพชั้นสูง สาขาอิเลคทรอนิคส์ สถาบันเทคโนโลยีราชมงคล วิทยาเขตนนทบุรี
พ.ศ. 2547	จบการศึกษาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ประสบการณ์ทำงาน

พ.ศ. 2547-2549	ผู้ช่วยวิจัย สังกัดสำนักงานกองทุนสนับสนุนการวิจัย (สกว.)