การออกแบบและสร้างวงจรบีบอัดข้อมูลตัวอักษรและภาพโดยใช้ FPGA

FPGA DESIGN FOR TEXT AND IMAGE COMPRESSION

คมศักดิ์ หาดขุนทด
KOMSAK HADKHUNTOD

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2546

ISBN 974-324-104-9

การออกแบบและสร้างวงจรบีบอัดข้อมูลตัวอักษรและภาพโดยใช้ FPGA

FPGA DESIGN FOR TEXT AND IMAGE COMPRESSION

คมศักดิ์ หาดขุนทด

KOMSAK HADKHUNTOD

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2546

ISBN 974-324-194-9

FPGA DESIGN FOR TEXT AND IMAGE COMPRESSION

KOMSAK HADKHUNTOD

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2003

ISBN 974-324-194-9

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การออกแบบและสร้างวงจรบีบอัดข้อมูลตัวอักษรและภาพโดยใช้ FPGA

FPGA DESIGN FOR TEXT AND IMAGE COMPRESSION

ชื่อนักศึกษา นายคมศักดิ์ หาดขุนทด

รหัสประจำตัว 40061083

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.ดร.สมศักดิ์ ชุมช่วย

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.สุรพันธุ์	เอื้อไพบูลย์	
ดร.สุพันธุ์	ตั้งจิตกุศลมั่น	
รศ.ดร.สุริภณ	สมควรพาณิชย์	
รศ.ดร.รัตติกร	วรากูลศิริพันธุ์	
รศ.ดร.สมศักดิ์	ชุมช่วย	

วัน/เดือน/ปี ที่สอบ 17 ตุลาคม 2545 เวลา 10.30-12.30 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-404)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่ 10 **เดือน** มกราคม **พ.ศ.** 2546

หัวข้อวิทยานิพนธ์	การออกแบบและสร้างวงจรบีบอัดข้อมูลตัวอักษรและภาพโดยใช้ FPGA
นักศึกษา	นายคมศักดิ์ หาดขุนทด
รหัสประจำตัว	40061083
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2546
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร.สมศักดิ์ ชุมช่วย

บทคัดย่อ

การบีบอัดข้อมูลถือว่าเป็นส่วนสำคัญในการออกแบบวงจรที่เกี่ยวข้องทางการจัดเก็บข้อมูลและการสื่อสารข้อมูลในปัจจุบัน จากหลักการบีบอัดข้อมูลโดย Huffman Coding ได้นำมาออกแบบวงจรเพื่อใช้ในการบีบอัดข้อมูลตัวอักษรและข้อมูลภาพที่อยู่ในไฟล์เดียวกัน แล้วทดสอบการทำงานของวงจรที่ได้ออกแบบเปรียบเทียบกับการบีบอัดข้อมูลด้วยซอร์ฟแวร์ ในส่วนของการบีบอัดข้อมูลรูปภาพอย่างเดี่ยวนั้นได้นำเสนออัลกอริทึมของ DCT ที่ทำให้ได้วงจรที่มีขนาดเล็ก แล้วสามารถนำผลลัพธ์ไปทำการควอนซ์ไตน์เซชั่นและสร้างกลุ่มของบิต(Bit Steam) ในการออกแบบได้ใช้ภาษา VHDL จำลองการทำงาน,สังเคราะห์วงจรและทดสอบเพื่อให้มีความยืดหยุ่นในการพัฒนากับอุปกรณ์ FPGA เพื่อใช้เป็นต้นแบบในการสร้างวงจรรวมขนาดใหญ่มาก(VLSI)โดยสามารถนำไปประยุกต์ใช้ได้ในระบบ Smart Card ที่จะทำให้สามารถเพิ่มสมรรถนะของระบบในการเก็บข้อมูลแบบรวม(Multidata Data) ได้มากขึ้น

Thesis Title	FPGA DESIGN FOR TEXT AND IMAGE COMPRESSION
Student	Mr. Komsak Hadkhuntod
Student ID	40061083
Degree	Master of Engineering
Programme	Electrical Engineering
Year	2003
Thesis Advisor	Assoc. Prof. Dr .Somsak Choomchuay

ABSTRACT

A method in data compression techniques has been introduced to reduce the text and image data storage and transmission costs. The hardware design of Huffman Coding for reduce of mixture data and compare compression with software . The image compression is reduced by DCT Algorithm. This thesis is proposed the mixture of prototype hardware has been implemented on a FPGA.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำสั่งสอน คำปรึกษา ตลอดจนความช่วยเหลือในด้านต่าง ๆ จาก ร.ศ. ดร. สมศักดิ์ ชุมช่วย ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่านเป็นอย่างยิ่งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณน้อง ๆ ที่ห้องปฏิบัติการ Microelectronic Devices R&D Lab , RECCIT ทุกท่านที่ช่วยเหลือให้คำแนะนำต่าง ๆ โดยเฉพาะ น้องเป็ก (นายสุรพงษ์ พงษ์ยุพินพานิช) ที่คอยช่วยเหลือแนะนำทุก ๆ อย่างจนเป็นรูปเล่ม ต้องขอขอบคุณเป็นพิเศษ

ขอขอบพระคุณ ภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และ ห้องปฏิบัติการวิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) ที่อนุเคราะห์อุปกรณ์ที่ใช้สำหรับทำการทดลอง

ขอขอบพระคุณ ท่านอธิการบดี และเพื่อนๆ อาจารย์ ข้าราชการ สถาบันราชภัฏพระนครศรีอยุธยา ที่ได้โอกาสลามาศึกษาต่อและให้กำลังใจด้วยความเป็นห่วงตลอดเวลา

ขอขอบพระคุณ บิดา มารดา และ พี่น้องทุกๆ ท่าน ที่ได้คอยให้การสนับสนุนและให้กำลังใจมาโดยตลอด

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ กระผมขอมอบแด่ผู้มีพระคุณทุกท่าน

คมศักดิ์ หาดขุนทด

สารบัญ

หน้า

บทคัดย่อภาษาไทย.....I
บทคัดย่อภาษาอังกฤษ.....II
กิตติกรรมประกาศ.....III
สารบัญ.....IV
สารบัญรูป.....VIII
สารบัญตาราง.....XI
บทที่ 1 บทนำ
1.1 ความเป็นมาและความสำคัญของปัญหา.....1
1.2 วัตถุประสงค์.....1
1.3 ทฤษฎีหรือแนวคิด.....2
1.4 ขอบเขตของวิทยานิพนธ์.....2
1.5 เนื้อหาภายในวิทยานิพนธ์.....2
บทที่ 2 ทฤษฎีข้อมูลข่าวสาร
2.1 ทฤษฎีข้อมูลข่าวสาร.....4
2.2 ความจุในการเก็บข่าวสารและการวัดข่าวสาร.....5
2.3 การวัดข่าวสารโดยวิธีลอการิทึม.....6
2.4 คำจำกัดความของความจุข่าวสาร.....6
2.5 ขบวนการสตอคาสติก.....9
2.6 ผังแสดงสภาวะ.....10
2.7 อัตราการเกิดไม่เท่ากัน.....11
2.8 การหาปริมาณข่าวสารที่ได้รับจริง.....12
2.9 ระบบสื่อสารอย่างง่าย.....13
บทที่ 3 หลักการทางคณิตศาสตร์
3.1 คณิตศาสตร์สำหรับการบีบอัดข้อมูลแลลสูญเสีย.....16
3.2 คณิตศาสตร์สำหรับการบีบอัดข้อมูลแบบไม่สูญเสีย.....18
3.2.1 Conditional Entropy.....18

สารบัญ (ต่อ)

หน้า

3.2.2 Average Mutual Information.....20
3.2.3 Differential Entropy.....21
3.2.4 หลักการของ Transform Coding.....25

บทที่ 4 ภาษาวีเอชดีแอล(VHDL)
4.1 แนะนำวีเอชดีแอล (Introduction to VHDL).....33
4.1.1 ข้อกำหนด (VHDL Requirement).....34
4.2 ความสามารถของภาษาวีเอชดีแอล (Capability).....37
4.3 หลักการสร้างโมเดลโดยใช้ภาษาวีเอชดีแอล.....38
(General VHDL Modelling Principles)
4.3.1 Top Down Design.....39
4.3.2 Modularity.....39
4.3.3 Abstraction.....41
4.3.4 Information Hiding.....42
4.3.5 Uniforminity.....43
4.4 องค์ประกอบพื้นฐานในวีเอชดีแอล (Basic concept in VHDL).....43
4.4.1 การกำหนดการเชื่อมต่อ (Interface Description).....43
4.4.2 การกำหนดรูปแบบการบรรยาย (Architecture Description).....44
4.4.3 โปรแกรมย่อย (Subprogram).....45
4.4.4 โอเปอร์เรเตอร์ (VHDL OPERATORS).....45
4.4.5 เวลาและความพร้อมเพรียง (Timing and Concurrency).....46
4.4.6 สัญญาณและตัวแปร (Signals and Variable).....46
4.5 โครงสร้างของวีเอชดีแอล.....46

บทที่ 5 เทคนิคที่ใช้ในการบีบอัดข้อมูล
5.1 การประยุกต์ใช้งานในการบีบอัดข้อมูล.....50
5.2 เทคนิคในการบีบอัดข้อมูล.....50
5.2.1 การบีบอัดข้อมูลแบบไม่สูญเสีย.....51
5.2.2 การบีบอัดข้อมูลแบบสูญเสีย.....51

สารบัญ (ต่อ)

หน้า

5.2.3 วิธีการวัดประสิทธิภพ.....51

5.2.4 โมเดลและรหัส.....52

5.3 เทคนิคการบีบอัดข้อมูลแบบไม่สูญเสีย.....53

5.3.1 การเข้ารหัสฮัฟแมน.....54

5.4 เทคนิคการบีบอัดข้อมูลแบบสูญเสีย.....57

บทที่ 6 สถาปัตยกรรมและการออกแบบ

6.1 ส่วนของการบีบอัดข้อมูลตัวอักษร (Huffman Coding).....68

6.1.1 วงจรสร้างตาราง.....68

6.1.2 วงจรสร้างรหัส Huffman Code 68

6.1.3 ส่วนของวงจรวงจรสร้าง State Machine 69

6.1.4 วงจรควบคุม.....70

6.2 ส่วนบีบอัดข้อมูลภาพโดยใช้ DCT.....71

6.2.1 DCT ทรานสฟอร์ม.....72

6.2.2 อัลกอริทึมแบบลิฟท์(Lifting Schematic).....72

6.2.3 อัลกอริทึมแบบสเกลลิฟท์(Scaled Lifting Schematic).....73

6.2.4 การทดลอง 74

6.2.5 ผลการทดลอง 76

6.2.6 สรุปในส่วนการออกแบบวงจรบีบอัดข้อมูลภาพ... 82

6.3 การประยุกต์ใช้งานวงจร DCT 83

6.4 สรุป.....87

บทที่ 7 การทดสอบและผลการทดสอบ

7.1 การทดสอบการบีบอัดข้อมูลด้วย Huffman Coding..... 88

7.1.1 การทดสอบการบีบอัดข้อมูลตัวอักษร..... 88

7.1.2 การทดสอบการบีบอัดข้อมูลภาพ.....88

7.1.3 การทดสอบการบีบอัดข้อมูลภาพและตัวอักษร.....89

7.2 สรุป.....92

สารบัญ (ต่อ)

หน้า

บทที่ 8 สรุปผลการวิจัยและข้อเสนอแนะ

8.1 สรุป..... 93

8.2 ปัญหาที่พบในการทำวิทยานิพนธ์.....95

8.3 วิธีการแก้ปัญหา 95

8.4 ข้อเสนอแนะและการพัฒนา.....96

เอกสารอ้างอิง.....97

ภาคผนวก.....98

ภาคผนวก ก. โปรแกรม VHDL 99

ภาคผนวก ข. โปรแกรมภาษา C 118

ภาคผนวก ค. ผลงานวิจัยที่ได้รับการตีพิมพ์ 134

ภาคผนวก ง. อุปกรณ์ FPGA เบอร์ EPF10K20.....140

ประวัติผู้เขียน.....141

สารบัญรูป

รูปที่ หน้า

2.1 กระบวนการสื่อสาร.....5

2.2 ตัวอย่างผังสภาวะ.....10

3.1 ค่าความสัมพันธ์ระหว่างความสูงและน้ำหนัก.....28

3.2 ลำดับในการทรานสฟอร์ม.....30

4.1 แสดงตัวอย่างการออกแบบแบบลำดับขั้น.....34

4.2 สิ่งต่างๆ ที่สามารถอธิบายได้ด้วย VHDL.....38

4.3 การแบ่งย่อยในระดับราบของการออกแบบฮาร์ดแวร์.....39

4.4 การแบ่งแบบ Hierarchy ของ VHDL Shifter Description.....40

4.5 Applying Abstraction to a ROM Description.....41

4.6 การซ่อนรายละเอียดที่ไม่จำเป็นของระดับ NAND เกท.....42

4.7 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม.....43

4.8 แสดงบล็อคไดอะแกรมและการบรรยายการเชื่อมต่อของ clock component.....44

4.9 แสดงการบรรยายเชิงพฤติกรรมของ Clock_component.....44

4.10 แสดงการใช้โพรซีเดอร์.....45

4.11 แสดงการใช้ฟังก์ชัน.....45

4.12 แสดงตัวกระทำใน VHDL.....46

4.13 รูปแสดงโครงสร้างการออกแบบ VHDL.....47

4.14 แสดงขั้นตอนการออกแบบระบบดิจิตอล.....48

4.15 แสดงการออกแบบระบบเส้นทางของข้อมูล.....48

5.1 กระบวนการบีบอัดข้อมูล.....51

5.2 นิยามการบีบอัดข้อมูล.....52

5.3 ตัวอย่างอัลกอริทึม.....53

5.4 ภาพขนาด 8x8.....61

5.5 การทำซิกแซก(Zigzag Scanning).....63

6.1 แสดงส่วนประกอบต่างๆ เพื่อจำลองการทำงานของวงจรการบีบอัดข้อมูล.....67

6.2 แสดงบล็อคของวงจรวงจรสร้างตาราง.....68

6.3 แสดง Timing Diagram ของวงจรสร้างตาราง.....68

6.4 แสดงบล็อคของวงจรสร้างรหัส Huffman Code 69

6.5 แสดง Timing Diagram ของวงจรสร้างสร้างรหัส Huffman Code 69

สารบัญรูป(ต่อ)

รูปที่ หน้า

6.6 แสดงบล็อคของวงจรวงจรสร้าง State Machine 69
6.7 แสดง Timing Diagram ของวงจรสร้างสร้างรหัส State Machine 70
6.8 แสดงบล็อคของวงจรวงจรควบคุม.....70
6.9 แสดง Timing Diagram ของวงจรวงจรควบคุม.....70
6.10 แสดงวงจรในส่วนต่างๆมารวมเป็นวงจรบีบอัดข้อมูลตัวอักษรHuffman.....71
6.11 แสดงชิปและจำนวนของลอจิกที่ใช้ภายในวงจร Huffman Coding.....72
6.12 รูปแบบทั่วไปของวงจร 1-D DCT 72
6.13 (a) รูป butterfly ทั่วไป (b) Plane rotation (c),(d) Scaled lifting Structure แบบ 3 ลิฟท์ (e),(f) Scaled lifting Structure แบบ 2 ลิฟท์ 73
6.14 วงจร DCT ทรานสฟอร์มของอัลกอริทึม A,B แบบ 2 ลิฟท์ (Lift) 75
6.15 วงจร DCT อินเวอร์สทรานสฟอร์มของอัลกอริทึม A,B แบบ 2 ลิฟท์ (Lift).....75
6.16 วงจร DCT ทรานสฟอร์มของอัลกอริทึม C,D แบบ 3 ลิฟท์ (Lift).....76
6.17 วงจร DCT อินเวอร์สทรานสฟอร์มของอัลกอริทึม C,D แบบ 3 ลิฟท์ (Lift).....76
6.18 แสดงผลการบีบอัดข้อมูลภาพตามอัลกอริทึมที่A ก) ทศนิยม1ตำแหน่งข) ทศนิยม3 ตำแหน่ง.....78
6.19 แสดงผลการบีบอัดข้อมูลภาพตามอัลกอริทึมที่ B ก) ทศนิยม 1 ตำแหน่ง ข) ทศนิยม 3 ตำแหน่ง.....78
6.20 แสดงผลการบีบอัดข้อมูลภาพตาม อัลกอริทึมที่ C ก) ทศนิยม 1 ตำแหน่ง ข) ทศนิยม 3 ตำแหน่ง 79
6.21 แสดงผลการบีบอัดข้อมูลภาพตาม อัลกอริทึมที่ D ก) ทศนิยม 1 ตำแหน่ง ข) ทศนิยม 3 ตำแหน่ง... 79
6.22..แสดงกราฟเปรียบเทียบค่า A) MSE B)ค่าจำนวนเต็มสูงสุด(Int.Max) C) ค่าจำนวนเต็มต่ำสุด(Int.Min)ของอัลกอริทึม A, B, C และ D.....80
6.23.แสดงผลการทดลองบีบอัดข้อมูลตามภาพอัลกอริทึม B.....81
6.24 แสดงสถาปัตยกรรมในการออกแบบวงจร DCT.....83
6.25 แสดงการออกแบบวงจร DCT แบบ Pipeline A) Forward B) Inverts.....84
6.26 การออกแบบวงจรบวก และลบ 85
6.27 การออกแบบวงจรบวกแบบต่างๆ 85
6.28 การออกแบบวงจรหารแบบต่างๆ.....86

สารบัญรูป(ต่อ)

รูปที่ **หน้า**

6.29 การประยุกต์วงจรบวก และลบในวงจร DCT.....86
7.1 แสดงข้อมูลตัวอักษร Test_t256.txt.....90
7.2 แสดงข้อมูลตัวอักษร Test_t256.txt.....90
7.3 แสดงข้อมูลรูปภาพขนาด 128 x128 พิกเซล.....92
7.4 แสดงข้อมูลรูปภาพขนาด 256 x256 พิกเซล.....92
7.5 แสดงข้อมูลรูปภาพขนาด 512 x512 พิกเซล.....92

สารบัญตาราง

ตารางที่ หน้า

2.1 แสดงความสัมพันธ์ระหว่าง I_i กับ S ; ($I_i = \log_2 S$) ; บิท.....8

3.1 Original sequence.....27

3.2 Transform sequence.....30

3.3 Reconstructed sequence.....31

5.1 ตัวอย่างและอัตราการเกิดของข้อมูล.....55

5.2 สร้างข้อมูล a_4'.....55

5.3 สร้างข้อมูล a_3'.....56

5.4 สร้างข้อมูล a_3''.....57

5.5 กำหนดรหัสฮัฟแมน.....57

5.6 การกำหนดบิทให้กับภาพในรูปที่ 5.4.....61

5.7 ภาพขนาด 8x8.....61

5.8 ค่า สัมประสิทธิ ที่ได้จากการทรานสฟอร์มโดย DCT($c_{i,j}$).....62

5.9 ตารางควอนไตเซชั่น.....62

5.10 ผลจากการการควอนไตเซชั่น ($I_{i,j}$).....63

5.11 ตารางการเข้ารหัสสัมประสิทธิ DC.....64

5.12 ตัวอย่างตารางที่ใช้สร้างรหัสฮัฟแมน.....64

6.1 แสดงผลการทดสอบการบีบอัดข้อมูลรูปภาพขนาด 512 x 512 พิกเซล.....77

6.2 แสดงคุณภาพของอัลกอริทึมA,B,C และ D.....82

6.3 แสดงขนาดของวงจรอัลกอริทึม A,B,C และ D.....82

7.1 แสดงการทดสอบการบีบอัดข้อมูลตัวอักษร Huffman Coding.....88

7.2 แสดงการทดสอบการบีบอัดข้อมูลตัวภาพ Huffman Coding ชนิด .PPM.....89

7.3 แสดงการทดสอบการบีบอัดข้อมูลตัวภาพ Huffman Coding ชนิด .JPG.....89

7.4 แสดงการทดสอบการบีบอัดข้อมูลตัวภาพ Huffman Coding ชนิด .BMP.....89

7.5 แสดงการทดสอบการบีบอัดข้อมูลตัวภาพ Huffman Coding ชนิด .BMP.....90

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การบีบอัดข้อมูลถือว่าเป็นส่วนสำคัญในการออกแบบวงจรที่เกี่ยวข้องทางการจัดเก็บและในการสื่อสารข้อมูลในปัจจุบัน ในวิทยานิพนธ์นี้ได้ประยุกต์การบีบอัดข้อมูลโดยใช้อัลกอริทึมของ Huffman Coding และ DCT เพื่อทำการบีบอัดข้อตัวอักษรและภาพ โดยในการออกแบบต้องการให้สามารถนำไปประยุกต์ใช้กับระบบสมาร์ทการ์ดได้ ดังนั้นจึงต้องทำการออกแบบวงจรให้มีขนาดเล็ก ระบบการบีบอัดข้อมูลแบบเดิมของระบบสมาร์ทการ์ดจะใช้วิธีการของ Hash Function สำหรับบีบอัดข้อมูลที่เป็นตัวอักษรและข้อมูลภาพ ซึ่งประสิทธิภาพในการบีบอัดข้อมูลภาพนั้นยังทำได้ไม่ดี

การบีบอัดข้อมูลตัวอักษรได้ใช้อัลกอริทึมของ Huffman coding ส่วนการบีบอัดข้อมูลภาพนั้นได้ใช้วิธีการ JPEG ทำการลดขนาดของข้อมูลภาพ โดย DCT จะทำการทรานสฟอร์มข้อมูลจากโดเมนเวลา(Time Domain) เป็นโดเมนความถี่(Frequency Domain) โดยที่ผลลัพธ์นั้นสามารถนำไปประมวลผลเพื่อลดขนาดของข้อมูลภาพ ด้วยวิธีการควอนไตเซชั่น(Quantization) , การซิแซก (Zig Zag) และวิธีการเอ็นโทรปี(Entropy Coding) เพื่อให้ได้ข้อมูลที่เป็นกลุ่มข้อมูลไบนารี(Bit Stream) ต่อไป

วิทยานิพนธ์นี้จึงได้ทำการออกแบบและพัฒนาอัลกอริทึมดังกล่าวซึ่งแบ่งออกเป็น 2 ส่วน คือ ส่วนของวงจร Huffman Coding และส่วนของวงจร DCT โดยทำการออกแบบวงจรด้วยภาษา VHDL หลังจากนั้นจึงนำมาสร้างเป็น ฮาร์ดแวร์ ด้วยอุปกรณ์ FPGA เพื่อจำลองการทำงานและทดสอบความสามารถในการบีบอัดข้อมูลของวงจร ในการทดสอบวงจรนั้นได้ทำการทดสอบโดยการนำข้อมูลตัวอักษรมาทำการบีบอัดด้วยวงจร Huffman Coding และได้ทำการนำข้อมูลภาพจะมาทำการลดขนาดของข้อมูลโดยใช้หลักการ DCT

1.2 วัตถุประสงค์ของการวิจัย

1) เพื่อศึกษาประสิทธิภาพของการบีบอัดข้อมูลตัวอักษรด้วยวิธีการ Huffman Coding ว่ามีอัตราการลดข้อมูลตัวอักษรและข้อมูลภาพได้ดีเพียงใด
2) เพื่อศึกษาการออกแบบ ฮาร์ดแวร์ (Hardware) ด้วยภาษา VHDL และศึกษาถึงวิธีการตรวจสอบสัญญาณและทำการสังเคราะห์เป็นวงจรระดับเกต
3) เพื่อศึกษาประสิทธิภาพและวิธีการของ DCT ในการทรานสฟอร์มข้อมูลภาพจาก Time Domain ไปเป็น Frequency Domain

4) เพื่อให้สามารถนำไปประยุกต์ใช้ร่วมกับชิป Smart Card และทำงานได้อย่างมีประสิทธิภาพ

1.3 ทฤษฎีหรือแนวคิด

ปัจจุบันการสื่อสารและการจัดเก็บข้อมูลถือว่ามีส่วนสำคัญมากในระบบสื่อสารข้อมูลเช่น การสื่อสารข้อมูลผ่านดาวเทียม, การเก็บข้อมูลในฮาร์ดดิสก์, การจัดเก็บข้อมูลบนสมาร์ทการ์ด ฯลฯ จะมีข้อมูลมากขึ้นทั้งข้อมูลภาพและข้อมูลตัวอักษร การบีบอัดข้อมูลเป็นขบวนการหนึ่งที่จะทำให้ข้อมูลมีขนาดเล็กลง ทำให้อุปกรณ์ที่มีขนาดเท่าเดิมแต่จัดเก็บข้อมูลได้มากขึ้น หรือทำให้การส่งข้อมูลทำได้เร็วขึ้น ในวิทยานิพนธ์จะได้ทำการออกแบบวงจรบีบอัดข้อมูลอักษรและข้อมูลภาพ เพื่อให้มีขนาดของวงจรมีขนาดเล็กเหมาะสมกับสมาร์ทการ์ด ซึ่งในปัจจุบันการบีบอัดข้อมูลที่ใช้เป็นวิธีการ Hash Function สามารถบีบอัดข้อมูลได้น้อยมาก จึงได้เสนอแนวความคิดที่จะพัฒนาขีดความสามารถในการบีบอัดข้อมูลให้สูงขึ้น จึงได้ใช้ Huffman Coding ในการบีบอัดข้อมูลตัวอักษร และใช้ DCT ในการบีบอัดข้อมูลภาพ โดยได้ออกแบบเป็นวงจรทั้ง 2 ส่วน

1.4 ขอบเขตของวิทยานิพนธ์

ในงานวิจัยนี้ต้องการพัฒนาและจำลองการทำงานวงจรบีบอัดข้อมูล เพื่อที่จะทำการบีบอัดข้อมูลตัวอักษรและข้อมูลภาพ เพื่อใช้สำหรับศึกษาและเหมาะสมกับการใช้งานเพื่อเป็นต้นแบบที่สมบูรณ์และสามารถนำไปประยุกต์ใช้กับส่วนบีบอัดข้อมูลบนสมาร์ทการ์ดต่อไป

1.5 เนื้อหาภายในวิทยานิพนธ์

เนื้อหาวิทยานิพนธ์จะอธิบายถึงขั้นตอนและรายละเอียดต่างๆในการทดลอง โดยแบ่งออกเป็นบทต่างๆได้ดังนี้

บทที่ 1 ทำการอธิบายความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์ของการวิจัย ทฤษฎีหรือแนวคิด ขอบเขตของวิทยานิพนธ์และเนื้อหาภายในวิทยานิพนธ์

บทที่ 2 อธิบายถึงหลักการของทฤษฎีข้อมูลข่าวสาร(Information Theory) ที่จำเป็นต่อการทำความเข้าใจในเรื่องการบีบอัดข้อมูล

บทที่ 3 อธิบายถึงหลักการและทฤษฎีทางคณิตศาสตร์ที่ใช้ในการบีบอัดข้อมูล ทั้งข้อมูลตัวอักษรและข้อมูลภาพ

บทที่ 4 อธิบายถึงการทำงานและการออกแบบ Hardware ด้วยภาษา VHDL ซึ่งประกอบด้วยความสามารถของภาษา VHDL หลักการการสร้างโมเดล โครงสร้างทางภาษา และอื่นๆ

บทที่ 5 อธิบายถึงหลักการที่ใช้ในการบีบอัดข้อมูล ทั้งข้อมูลตัวอักษรและข้อมูลภาพ

บทที่ 6 อธิบายถึงอัลกอริทึมที่ได้นำมาใช้ในการออกแบบโครงสร้างทางฮาร์ดแวร์ของวงจร และส่วนประกอบภายใน

บทที่ 7 อธิบายถึงการทดสอบและผลการทดสอบ

บทที่ 8 สรุปผลการวิจัย ปัญหาที่พบและข้อเสนอแนะ

บทที่ 2
ทฤษฎีข้อมูลข่าวสาร

ในบทนี้จะกล่าวถึงทฤษฎีข้อมูลข่าวสารที่ใช้เกี่ยวข้องกับการบีบอัดข้อมูล โดยในเบื้องต้นจะกล่าวถึงทฤษฎีข้อมูลข่าวสาร เพื่อให้ทราบถึงนิยามและคำจำกัดความต่าง ๆ ที่ต้องใช้เกี่ยวกับการสื่อสารข้อมูลรวมถึงคณิตศาสตร์พื้นฐานที่ใช้สำหรับทฤษฎีการบีบอัดข้อมูล

2.1 ทฤษฎีข้อมูลข่าวสาร (Information Theory)

ข่าวสารที่ส่งออกมาจากแหล่งกำเนิด ข่าวสาร(Information Source) ในระบบสื่อสารจะมีรูปแบบต่าง ๆ กัน เช่น

1.) ฟังก์ชั่นง่าย ๆ ของเวลา เช่น ความถี่ที่เป็นไซน์นูซอลดอล(sinusodal) ; $\sin \omega t, \cos \omega t$

2.) คอมเพล็กซ์ฟังก์ชั่นของเวลา และ ตัวแปรอีกตัวหนึ่ง เช่น สัญญาณ เอ. เอ็ม. $A\cos\omega_c t = I(1+m_a \cos pt)\cos\omega_c t$

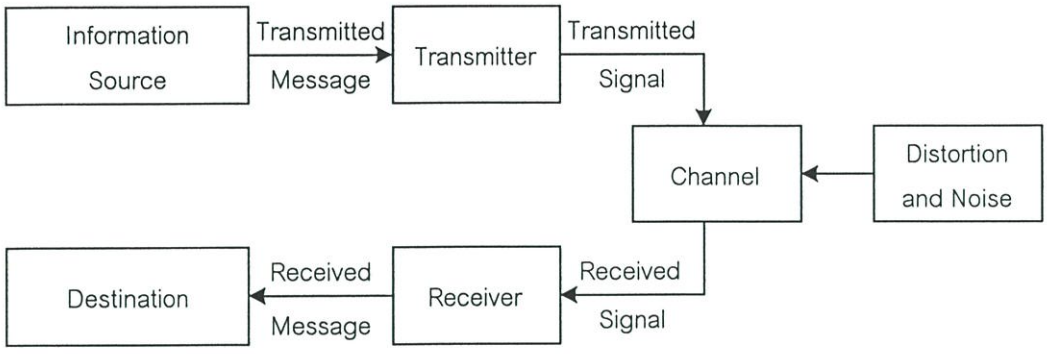
3.) ฟังก์ชั่นของเวลา ซึ่งมีตัวแปรอีกสองตัว เช่น สัญญาณโทรทัศน์

4.) ฟังก์ชั่นของเวลาหลาย ๆ ฟังก์ชั่น ซึ่งมีตัวแปรหนึ่งตัว เช่น สัญญาณเสียงระบบสเตอริโอ

5.) ฟังก์ชั่นของเวลาหลาย ๆ ฟังก์ชั่น และตัวแปรอีกหลายตัว เช่น สัญญาณโทรทัศน์สี เป็นต้น

ทฤษฎีข้อมูลข่าวสารนั้น พิจารณาแต่สัญญาณดีสครีท(Discrete Signal) คือ สัญญาณที่มีขนาดคงที่แน่นอนเป็นช่วง ๆ เช่น สัญญาณพัลซ์(Pulse Signal) แต่เมื่อต้องพิจารณาสัญญาณต่อเนื่อง (Continuously variable Signal) ก็จำเป็นต้องเปลี่ยนรูปให้เป็นรูปใกล้เคียงกับสัญญาณพัลซ์เสียก่อนแล้วจึงสามารถนำมาพิจารณาได้

ข่าว(message) ที่ป้อนเข้ามาจากที่ต่าง ๆ จะทำให้เครื่องส่ง(Transmitter) ผลิตสัญญาณ(Signal) ที่เหมาะสม ผ่านช่องสื่อสาร(channel) ซึ่งเป็นอุปกรณ์ในการนำเอาสัญญาณจากเครื่องส่งไปสู่เครื่องรับ (Receiver) โดยช่องสื่อสารอาจเป็น เส้นลวด, อากาศ เป็นต้น ดังแสดงในรูปที่ 2.1



รูปที่ 2.1 กระบวนการสื่อสาร

2.2 ความจุในการเก็บข่าวสารและการวัดข่าวสาร

วิธีแสดงจำนวนข่าวสาร(information) ที่มีอยู่ เมื่อรับเอาข่าวสาร(message) หนึ่ง ๆ มา สมมุติให้ S เป็นจำนวนแตกต่างของสัญลักษณ์และข่าวที่ได้รับประกอบด้วยจำนวนลำดับของสัญลักษณ์ M ตัว และ D เป็นจำนวนข่าวแตกต่างกันที่ได้รับ

S = จำนวนแตกต่างของสัญลักษณ์

M = จำนวนลำดับของสัญลักษณ์

D = จำนวนข่าวสารที่แตกต่าง

$$D = S^M \tag{2.1}$$

ตัวอย่าง เช่น ให้ S เป็นระบบสัญญาณดิจิตอล คือ 0,1 และ จำนวนลำดับของสัญลักษณ์ $M = 3$ ดังนั้น จำนวนแตกต่างของข่าวสารที่ได้รับ $D = 2^3 = 8$ ข่าวสาร ได้แก่ 000,001,010, 011,100,101,110,111 หรือให้ $M = 8$ ก็จะได้ จำนวนข่าวสารแตกต่าง(D) เท่ากับ 256 ข่าวสาร จากสมการที่ (2.1) เมื่อใส่ลอการิทึม (Logarithm) ฐาน 2 จะได้

$$\log_2 D = \log_2 S^M = M \log_2 S \tag{2.2}$$

ซึ่งสมการที่ 2.2 เป็นสมการที่ใช้ในการวัดความจุข่าวสาร(Information Content)ของข่าวสารหนึ่งๆ

2.3 การวัดข่าวสารโดยวิธีลอการิทึม

ให้ M เป็นจำนวนลำดับของสัญลักษณ์หรือลำดับความยาว(จำนวน)ของข่าว หรือจำนวนสัญลักษณ์ต่อหนึ่งข่าวสาร ถ้าให้ $M = 1$, $S = 2$ จากสมการ (2.1) จะได้ $D = 2$ ดังนั้นความจุของข่าวสารเท่ากับ $M \log_2 S = \log_2 D = \log_2 2 = 1$ ให้ S คงที่ แต่เปลี่ยน $M = 2$; $D = 2^2 = 4$ ความจุของข่าวสารเท่ากับ $M \log_2 S = \log_2 D = \log_2 4 = 2$ และถ้าให้ S คงที่ แต่เปลี่ยน $M = 3$; $D = 2^3 = 8$ ความจุของข่าวสารเท่ากับ $M \log_2 S = \log_2 D = \log_2 8 = 3$ จะเห็นได้ว่าการวัดข่าวสารแบบลอการิทึมความจุของข่าวสาร จะเป็นสัดส่วนกับจำนวนความยาวของสัญลักษณ์(M) เมื่อจำนวนแตกต่างของสัญลักษณ์(S) คงที่

ตัวอย่างที่ 2.1 กรณีที่ 1 ให้ $S = 4$ คือ a,b,c,d

ให้ $M = 2$

∴ ข่าว(message) ที่เกิดขึ้นได้ คือ

aa	ba	ca	da
ab	bb	cb	db
ac	bc	cc	dc
ad	bd	cd	dd

∴ จะได้ข่าวสารต่างๆกัน = 16 แบบ ; หรือ $D = S^M = 4^2 = 16$ แบบ

กรณีที่ 2 ให้ $S = 2$ คือ (0,1)

ให้ $M = 4$

∴ ข่าว(message) ที่เกิดขึ้นได้

0000	0100	1000	1100
0001	0101	1001	1101
0010	0110	1010	1110
0011	0111	1011	1111

จะได้ข่าวสารต่าง ๆ กัน 16 แบบเช่นกัน ; หรือ $D = S^M = 2^4 = 16$ แบบโดยการใช้วิธีวัดความจุข่าวสาร(Information Content)

$\log_2 D = M \log_2 S = 4 \log_2 2 = 4$; $\log_2 D = M \log_2 S = 2 \log_2 4 = 4$

จะเห็นได้ว่าทั้ง 2 ค่าให้ความจุข่าวสารเท่ากันคือ 4 โดยที่ข่าว(message)แต่ละข่าวจะมีโอกาสเกิดขึ้นได้ $\frac{1}{16}$ เท่ากัน และถ้าเราเลือก cc จากตารางกรณีที่ 1 จะบรรจุข่าวสารจำนวนเดียวกันกับ 1010 ของตารางกรณีที่ 2

2.4 คำจำกัดความของความจุข่าวสาร

ถ้าแต่ละข่าว(message)ประกอบไปด้วยจำนวน M สัญลักษณ์ซึ่งเลือกจากจำนวนของสัญลักษณ์ที่มีใช้ในระบบ เช่นจำนวนแตกต่างของสัญลักษณ์ในภาษาอังกฤษ มี 26 ตัวหรือมี S เท่ากับ 26 ตัว และไม่จำกัดจำนวนอาจมีตัวเดียวหรือสิบตัวก็ได้ คำจำกัดความของความจุข่าวสารของแต่ละข่าวคือ

$$\text{ความจุข่าวสารต่อข่าว} = M \log_2 S \qquad (2.3)$$

เมื่อใช้ลอการิทึมฐาน 2 ค่าความจุข่าวสารของแต่ละข่าวจะมีหน่วยเป็น ไบนารีดิจิท(binary digits) หรือ บิต การคำนวณลอการิทึมฐาน 2 ถึงแม้จะมีตารางสำหรับการคำนวณโดยตรงอยู่แล้ว แต่บางครั้งสามารถหาลอการิทึมฐาน 2 ได้จากลอการิทึมฐาน 10 คือ

$$y = \log_2 x$$

$$\therefore \quad x = 2^y$$

$$\log_{10} x = \log_{10} 2^y$$

$$= y \log_{10} 2$$

แทน $y = \log_2 x$;

$$= \log_2 x \log_{10} 2$$

$$\therefore \log_2 x = \frac{\log_{10} x}{0.3010}$$

ในทำนองเดียวกันเราสามารถหาได้จาก log ฐาน e ได้เช่นเดียวกัน

$$\log_2 e = \frac{1}{\log_e 2}$$

$$\therefore \log_2 e = \frac{\log e}{\log 2} = \frac{1}{\log_e 2}$$

ถ้าให้สัญลักษณ์ที่ใช้แทนความจุข่าวสารต่อข่าวคือ I_m จากสมการที่ (1.3) จะได้

$$I_m = M \log_2 S \qquad (2.4)$$

จากสมการที่ (2.4) เป็นความจุข่าวสารต่อข่าว โดยในแต่ละข่าวยังประกอบด้วยสัญลักษณ์ M ตัว ถ้าให้สัญลักษณ์ที่ใช้แทนความจุข่าวสารต่อสัญลักษณ์ คือ I_i

$$I_i = \frac{M \log_2 S}{M} = \log_2 S \quad \text{บิต} \qquad (2.5)$$

จากตัวอย่างที่ 1 ทั้งสองตารางมีความจุข่าวสารต่อข่าวเท่ากับ 4 เท่ากันคือ

$I_{M2} = M \log_2 S = 2 \log_2 4 = 4$; $I_{M4} = M \log_2 S = 4 \log_2 2 = 4$

ความจุข่าวสารต่อสัญลักษณ์ $I_{i2} = = \log_2 S = \log_2 4 = 2$; $I_{i4} = \log_2 S = \log_2 2 = 1$

ตารางที่ 2.1 แสดงความสัมพันธ์ระหว่าง I_i กับ S ; ($I_i = \log_2 S$) ; บิต

ความจุข่าวสารต่อสัญลักษณ์ (I_i)	จำนวนของสัญลักษณ์ (S)
1	2
2	4
3	8
4	16

หรือ

S = 0,1 ; I_i = 1 คือ 1 ข่าวสารใช้ 1 สัญลักษณ์

S = 00,01,10,11 ; I_i = 2 คือ 1 ข่าวสารใช้ 2 สัญลักษณ์

S = 000,001,010,011,100,101,110,111 ; I_i = 3 คือ 1 ข่าวสารใช้ 3 สัญลักษณ์

จากสมการที่ (2.1) จำนวนข่าวสาร $D = S^M$, สมการที่ (2.2) ความจุข่าวสารต่อข่าว; $I_m = \log_2 S$ และสมการที่ (2.3) ความจุข่าวสารต่อสัญลักษณ์ ; $I_i = \log_2 S$ บิต/สัญลักษณ์ ถ้าสัญลักษณ์มีอัตราการเกิดขึ้นได้เท่าๆกัน อัตราการเกิด(probability) ของแต่ละสัญลักษณ์หรืออัตราการเกิดเอพริโอไร(a priori probability)

$$P(i) = \frac{1}{S} \tag{2.6}$$

จากสมการที่ (2.6) เป็นอัตราการเกิดของสัญลักษณ์ i โดยที่ S เป็นจำนวนของสัญลักษณ์หรือเหตุการณ์ ที่สามารถเกิดขึ้นได้เท่า ๆ กันดังนั้นสมการที่ (2.3) ความจุข่าวสารต่อสัญลักษณ์ ; $I_i = \log_2 S$ เท่ากับ

$$I_i = \log_2 \frac{1}{P(i)} = -\log_2 P(i) \quad \text{บิต/สัญลักษณ์} \tag{2.7}$$

จะเห็นได้ว่าถ้า probability มาก I_i น้อย หรือความจุข่าวสาร (บิต/สัญลักษณ์) ยิ่งน้อยๆจะมี probability มากนั่นเอง ความจุข่าวสารในเทอมของอัตราการเกิด(Probability) หรือกล่าวได้ว่า ความจุข่าวสาร

ของสัญลักษณ์หนึ่ง ๆ จะสามารถแสดงในเทอมของ อัตราการเกิด (probability) ของตัวมันเอง จาก สมการ (2.7) สามารถเขียนได้เป็น

$$I_i = \log_2 \frac{1}{\text{a priori probability}} \qquad (2.8)$$

จากสมการที่ (2.8) เป็นความจุข่าวสารของข่าวซึ่งประกอบจากสัญลักษณ์ตัวเดียว ในกรณีของข่าว สารที่ประกอบไปด้วยสัญลักษณ์ที่เป็นเลขฐาน 2 จำนวน n บิต การประกอบกันเป็นข่าวสารของ สัญลักษณ์ก็จะมีความแตกต่างกัน 2^n จำนวน ซึ่งแต่ละสัญลักษณ์จะมีอัตราการเกิดเอพริโอไร เท่ากับ $1/2^n$ ดังนั้นความจุของข่าวสารจะเท่ากับ $\log_2 1/(1/2^n) = n$ บิต

2.5 ขบวนการสตอคาสติก (Stocastic Process)

เป็นขบวนการซึ่งผลิตลำดับตัวอักษรหรือสัญลักษณ์โดยขึ้นอยู่กับอัตราการเกิดคงที่จะเรียกกันว่า ขบวนการสตอคาสติก (Stocastic Process) ซึ่งมีความคงที่ทางสถิติ (statistically stationary) ตัวอย่างเช่นภาษาอังกฤษ เพราะว่าตัวอักษรต่าง ๆ และลำดับของคำต่าง ๆ สามารถกำหนดอัตราการเกิดให้มันได้ ในภาษาอังกฤษหรือภาษาอื่นใดก็ตาม มักจะมีวิวัฒนาการไปอย่างช้า ๆ โดยเพิ่มคำศัพท์ใหม่ ๆ หรือตัดคำศัพท์เก่า ๆ ที่ไม่ใช้แล้วทิ้งไป แต่ในช่วงระยะเวลาสั้นแล้วเราถือว่าอัตราการเกิดคงที่

ขบวนการสตอคาสติกซึ่งกำหนดอัตราการเกิดของการเปลี่ยน(transition)จากสัญลักษณ์หนึ่งไปยังอีกสัญลักษณ์หนึ่งเรียกว่า " ขบวนการมาร์คอฟ (Markoff Process)" หรือ "Markoff Chain" โดยอัตราการเกิดของสัญลักษณ์ใดๆขึ้นอยู่กับ ลักษณะของตัวอักษรก่อนหน้านี้ ถ้าขึ้นอยู่กับจำนวนสัญลักษณ์ M ตัวก่อนหน้านี้เราเรียกแหล่งกำเนิดสัญลักษณ์เช่นนี้ว่า "แหล่งกำเนิดมาร์คอฟ ชั้นที่ M (M^{th} order)" ส่วนแหล่งกำเนิดซึ่งผลิตสัญลักษณ์ซึ่งเป็นอิสระจากสัญลักษณ์ตัวก่อนหน้านี้ เรียก "แหล่งกำเนิดความจำศูนย์ (Zero – memory Source)" เช่น แหล่งกำเนิดมาร์คอฟจะผลิตลำดับของสัญลักษณ์ดังนี้คือ

. . . dddaaabbbaaacccbbbdddccc . . .

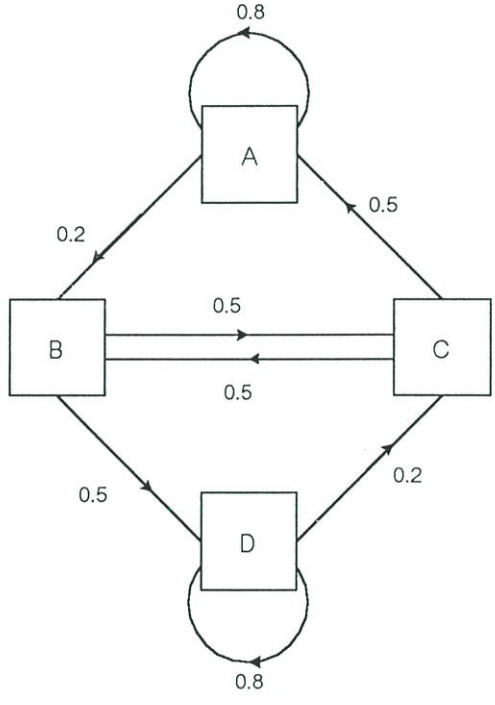
เมื่อ p(a) = p(b) = p(c) = p(d)

จะเห็นว่าอัตราการเกิดของ a,b,c,d เท่ากันในลำดับของข่าวทั้งหมด 24 ตัว

2.6 ผังแสดงสภาวะ (State Diagram)

ขบวนการมาร์คอฟมักแสดงโดยการเขียนผังแสดงสภาวะ (State Diagram) ของอัตราการเกิดการเปลี่ยนแปลงสภาวะของระบบ

ถ้ามีสัญลักษณ์ที่แตกต่างกันทั้งหมด S ตัว ขบวนการมาร์คอฟชั้นที่ M ก็จะผลิตสภาวะได้ S^M จำนวน จากรูปที่ 2.2 แสดงตัวอย่างของระบบซึ่งมีอัตราการเกิด(probability)ของการเปลี่ยนแปลงไม่เท่ากัน แต่ละสภาวะ ของ S^M จะถูกแทนด้วยบล็อกๆหนึ่ง การเปลี่ยนแปลงระหว่างสภาวะจะแทนด้วยเส้นตรงจากรูป อัตราการเกิดการเปลี่ยนแปลงจะเป็นดังนี้ $P_A(A) = P_D(D) = 0.8$; $P_A(B) = P_D(C) = 0.2$; $P_B(C) = P_B(D) = P_C(A) = P_C(B) = 0.5$ ในระบบ Binary สภาวะอาจแทนด้วยเลขฐาน 2 เช่น A = 00, B = 01, C = 10, D = 11 การคำนวณอัตราการเกิดของสัญลักษณ์จะทำได้เมื่อทราบอัตราการเกิดของการเปลี่ยนแปลง เช่น



รูปที่ 2.2 ตัวอย่างผังสภาวะ

$$P(A) = P_A(A)\,P(A) + P_C(A)\,P(C)$$

$\therefore$ จากตัวอย่าง $P(A) = 0.8P(A) + 0.5P(C)$

$$P(B) = 0.2P(A) + 0.5P(C)$$

$$P(C) = 0.5P(B) + 0.2P(D)$$

$$P(D) = 0.8P(D) + 0.5P(B)$$

เมื่อแก้สมการจะได้

$$P(A) = P(D) = 2.5P(B) = 2.5P(C)$$

แต่เรารู้ว่า $P(A) + P(B) + P(C) + P(D) = 1$

$\therefore$ $P(A) = P(D) = \frac{5}{14}$; $P(B) = P(C) = \frac{2}{14} = \frac{1}{7}$

$\therefore$ ข่าวสารในกรณีนี้จึงอาจอยู่ในรูปแบบ

– AAAAABCBDDDDDC –

2.7 อัตราการเกิดไม่เท่ากัน

อัตราการเกิดของสัญลักษณ์ ในกรณีที่สัญลักษณ์ต่างมีอัตราการเกิดเอพริโอไรไม่เท่ากันเช่น สัญลักษณ์ A และ B

$$P(A) = \frac{m}{S}, \quad P(B) = \frac{n}{S} \tag{2.9}$$

และ $P(A) + P(B) = \frac{m}{s} + \frac{n}{s} = \frac{m+n}{s} = 1$ หรือ $m + n = s$

ดังนั้น จาก (2.9) สามารถเขียนแทน

$$P(A) = \frac{m}{m+n}, \quad P(B) = \frac{n}{m+n} \tag{2.10}$$

ถ้าเราพิจารณาข่าวสารยาวๆของสัญลักษณ์ A และ B ซึ่งข่าวสารอาจเป็น

$$A_1, A_2, \ldots, A_x, \ldots, A_m, B_1, B_2, \ldots, B_y, \ldots, B_n$$

แต่ละสัญลักษณ์ก็มีอัตราการเกิดเท่าๆกันคือ

$$P(A_x) = P(B_y) = \frac{1}{m+n} = \frac{1}{s} \tag{2.11}$$

ความจุข่าวสารที่ประกอบด้วยสัญลักษณ์ $A_1, \ldots, B_n$ ตัวใดตัวหนึ่งเท่ากับ

$$I_I = \log_2 s = \log_2 \frac{1}{P(Ax)} = \log_2(m+n) \qquad (2.12)$$

โดยปริมาณ $\log_2 m + n$ ถือว่าเป็นความจุข่าวสารสูงสุดต่อสัญลักษณ์ P(A) = ความน่าจะเป็นของสัญลักษณ์ A ทั้งหมด m ตัว

P(Ax) = ความน่าจะเป็นของสัญลักษณ์ A ในแต่ละตัว

จาก $P(A) = \frac{m}{m+n}$; ทำให้เราทราบว่าสัญลักษณ์จำนวน m ตัวใน $m+n$ ตัว

และ $P(Ax) = \frac{1}{m+n}$ จะเป็น A

$$mP(Ax) = m \cdot \frac{1}{m+n} = \frac{m}{m+n} = P(A) \qquad (2.13)$$

เมื่อ $\log_2 m$ เป็นความจุข่าวสารที่รู้ก่อนว่าจะเป็นสัญลักษณ์ A

$$mP(Ax)\log_2 m = \frac{m}{m+n}\log_2 m = P(A)\log_2 m \text{ บิต/สัญลักษณ์} \qquad (2.14)$$

ในทำนองเดียวกันเมื่อรู้อัตราการเกิดเอพริโอไรทางด้านรับว่าใน $m+n$ ตัวนั้นมี B อยู่ n ตัว

$$mP(By)\log_2 n = \frac{n}{m+n}\log_2 n = P(B)\log_2 n \text{ บิต/สัญลักษณ์} \qquad (2.15)$$

2.8 การหาปริมาณข่าวสารที่ได้รับจริง

จะต้องเอาข่าวสารที่รู้ก่อนไปลบออกจากข่าวสารรวมต่อสัญลักษณ์

I_i = ข่าวสารรวม – ข่าวสารที่รู้ก่อน

= $\log_2(m+n)$ – ข่าวสารที่รู้ก่อน

$$I_i = \log_2(m+n) - \left[\frac{m}{m+n}\log_2 m + \frac{n}{m+n}\log_2 n\right] \text{ บิท/สัญลักษณ์} \qquad (2.16)$$

จากสมการ (2.16) จัดรูปใหม่โดยแยก

$$\log_2(m+n) = \frac{m}{m+n}\log_2(m+n) + \frac{n}{m+n}\log_2(m+n)$$

$$I_i = \left(\frac{m}{m+n}\right)\log_2(m+n) + \left(\frac{n}{m+n}\right)\log_2(m+n) - \left[\left(\frac{m}{m+n}\right)\log_2(m+n) + \left(\frac{n}{m+n}\right)\log_2(m+n)\right]$$

$$= -\left(\frac{m}{m+n}\right)\log_2 m + \left(\frac{m}{m+n}\right)\log_2(m+n) - \left[\left(\frac{n}{m+n}\right)\log_2 n + \left(\frac{n}{m+n}\right)\log_2(m+n)\right]$$

$$= \left[-\left(\frac{m}{m+n}\right)\log_2 m - \left(\frac{m}{m+n}\right)\log_2 \frac{1}{(m+n)}\right] + \left[-\left(\frac{n}{m+n}\right)\log_2 n - \left(\frac{n}{m+n}\right)\log_2 \frac{1}{m+n)}\right.$$

$$= \left[-\left(\frac{m}{m+n}\right)\log_2\left(\frac{m}{m+n}\right)\right] + \left[-\left(\frac{n}{m+n}\right)\log_2\left(\frac{n}{m+n}\right)\right]$$

เมื่อ $P(A) = \frac{m}{m+n}$ และ $P(B) = \frac{n}{m+n}$

$$I_i = -P(A)\log_2 P(A) - P(B)\log_2 P(B)$$

$$I_i = -\sum_{i=A}^{B} P(i)\log_2 P(i) \text{ บิต/สัญลักษณ์} \qquad (2.17)$$

ในสมการ (2.17) เรียกว่า "กฎของ แชนนอนและไวเนอร์" ส่วนค่าข่าวสารเฉลี่ยต่อสัญลักษณ์นี้นิยมเรียกว่า "เอนโทรปี (Entropy)"

2.9 ระบบสื่อสารอย่างง่าย

ในระบบสื่อสารอย่างง่ายๆที่มีสัญลักษณ์ Aและ B เท่านั้นและอัตราการเกิดของ A และ B คือ P(A) และ P(B) ไม่เท่ากัน แต่ P(A) รวมกับ P(B) แล้วจะต้องเท่ากับหนึ่งคือ P(B) = 1 – P(A) ในกรณีข่าวสารเฉลี่ยต่อสัญลักษณ์จะเป็น

$$I_i = -\sum_{i=A}^{B} P(i)\log_2 P(i)$$

$$= -P(A)\log_2 P(A) - P(B)\log_2 P(B) \text{ บิต/สัญลักษณ์}$$

สมมุติว่า $P(A) = 0.2$ และ $P(B) = 0.8$

$$I_i = -0.2\log_2 0.2 - 0.8\log_2 0.8$$
$$= 0.72 \text{ บิต/สัญลักษณ์}$$

ถ้านำมาพล็อตกราฟจะได้ดังรูป จะเห็นว่า I_i สูงสุดจะอยู่ที่ $P(A) = 0.5$ (สัญลักษณ์ A และ B เกิดเท่ากัน)

จะได้ $I_i = -0.5\log_2 0.5 - 0.5\log_2 0.5 = -\log_2 0.5 = -\log_2 \frac{1}{2}$

$$= 1$$

ถ้า $P(A) = 1$ จะได้ $I_i = -P(A)\log_2 P(A) - P(B)\log_2 P(B)$

$$= -\log_2 1 = 0 = I_A$$

จะไม่มีข่าวสารเกิดขึ้นทางด้านรับและเมื่อ $P(A) = 0$; $P(B) = 1$ จะไม่มีข่าวสารเกิดขึ้นเช่นเดียวกัน ข่าวสารเฉลี่ยต่อสัญลักษณ์ เมื่อมีสัญลักษณ์มากกว่า 2 ตัวในระบบ เช่น a,b,c,…,n การหาข่าวสารเฉลี่ยก็ยังคงใช้สูตรเดิมคือ

$$I_i = -\sum_{i=a}^{n} P(i)\log_2 P(i) \text{ บิต/สัญลักษณ์}$$

ซึ่งก็คือค่าข่าวสารเฉลี่ยต่อสัญลักษณ์ เรียกว่า "ENTROPY" และเมื่อพิจารณาลำดับของสัญลักษณ์ซึ่งยาวมากและมีจำนวนสัญลักษณ์เท่ากับ M จะได้ข่าวสารรวม

$$I_i = -M\sum_{i=a}^{n} P(i)\log_2 P(i) \text{ บิต}$$

<u>ตัวอย่างที่ 2.2</u> ในการโยนเหรียญให้ออกหัวหรือก้อย จะมีข่าวสารเท่าไรบรรจุอยู่

<u>วิธีทำ</u>

$$I_i = -M \sum_{i=head}^{tail} P(i)\log_2 P(i)$$

$$= -P(head)\log_2 P(head) - P(tail)\log_2 P(tail)$$

$$= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}$$

$\therefore$ จะมีข่าวสารบรรจุอยู่ = 1 บิต/สัญลักษณ์

ข้อสังเกตุ ตัวอย่างนี้มีอัตราการเกิดหัวเท่ากับก้อย $P(head) = P(tail) = 0.5 = \frac{1}{2}$

ตัวอย่างที่ 2.3 ของอัตราการเกิดไม่เท่ากัน

เมื่ออัตราการเกิดไม่เท่ากัน โดยพิจารณาที่ข่าวยาวๆของสัญลักษณ์จำนวน M ตัว ซึ่งมีอัตราการเกิด A = 0.51 ; B = 0.25 ; C = 1.1 ; D = 0.1 ; E = 0.05 ซึ่งรูปแบบของข่าวอาจเกิดขึ้นดังนี้

... ADBAAEBACBAAACABDAAB ...

ข่าวสารรวม

$$I = -M \sum_{i=A}^{E} P(i)\log_2 P(i)$$

$$= -M(0.5\log_2 0.5 + 0.25\log_2 0.25 + 0.1\log_2 0.1 + 0.05\log_2 0.05)$$

$$= +M(0.5 + 0.5 + 0.3321 + 0.3321 + 0.216)$$

$$= 1.88\,M \text{ บิต}$$

บทที่ 3
หลักการทางคณิตศาสตร์

ในบทนี้จะกล่าวถึงหลักการทางคณิตศาสตร์ที่ใช้เกี่ยวข้องกับการบีบอัดข้อมูลทั้งการบีบอัดข้อมูลแบบสูญเสียและการบีบอัดข้อมูลแบบไม่สูญเสีย เพื่อให้ทราบถึงนิยามและคำจำกัดความต่าง ๆ ที่ต้องใช้เกี่ยวข้องกับคณิตศาสตร์ในการบีบอัดข้อมูล

1. คณิตศาสตร์สำหรับการบีบอัดข้อมูลแบบสูญเสีย (Mathmatical for Lossless Compression)
2. คณิตศาสตร์สำหรับการบีบอัดข้อมูลแบบไม่สูญเสีย (Mathmatical for Lossy Compression)

3.1 คณิตศาสตร์สำหรับการบีบอัดข้อมูลแบบสูญเสีย(Mathmatical for Lossless ompression)

สมมติว่าเรามีเหตุการณ์ A ซึ่งคือชุดของ outcomes ของการทดลองแบบสุ่มจำนวนหนึ่ง ถ้า P (A) คือความน่าจะเป็นที่เกิดเหตุการณ์ A ค่าความจุข่าวสารของเหตุการณ์ A

$$i(A) = \log_b \frac{1}{P(A)} = -\log_b P(A) \tag{3.1}$$

สมมุติให้เหตุการณ์ A และ B เป็นอิสระต่อกัน ความจุข่าวสารของเหตุการณ์ A และ B สามารถแทนด้วยสมการ

$$i(AB) = \log_b \frac{1}{P(AB)} \tag{3.2}$$

เมื่อ A และ B เป็นอิสระต่อกัน

$$P(AB) = P(A)P(B) \tag{3.3}$$

และ

$$\begin{aligned} i(AB) &= \log_b \frac{1}{P(A)P(B)} \\ &= \log_b \frac{1}{P(A)} + \log_b \frac{1}{P(B)} \\ &= i(A) + i(B) \end{aligned}$$

ถ้ามีเซ็ต A มีสมาชิกที่เป็นอิสระต่อกันและ A_i เป็นสมาชิกที่ได้จากการสุ่มข้อมูลกำหนดให้เป็นเซ็ต

$$\bigcup A_i = S$$

เมื่อ S เป็นกลุ่มข้อมูลตัวอย่าง ซึ่งค่าเฉลี่ยความจุของข้อมูลข่าวสาร(self-information) จะขึ้นอยู่กับผลของการสุ่ม (random) ตัวอย่างข้อมูล ค่าเฉลี่ยความจุของข้อมูลข่าวสารของสมาชิก A_i หรือ ค่าเอนโทรปี สามารถเขียนได้เป็น

$$H = \sum P(A_i) i(A_i) = -\sum P(A_i) \log_b P(A_i)$$

และค่าเอนโทรปีนี้จะขึ้นอยู่กับการสุ่มข้อมูล แชนนอน (Shannon) ได้แสดงให้เห็นว่าข้อมูลที่สุ่ม A_i จาก เซ็ต A นั้นค่าเอนโทรปีเป็นค่าความจุข่าวสารเฉลี่ยของเลขฐานสองต่อสัญลักษณ์ของแหล่งกำเนิด และสามารถแสดงให้เห็นวิธีสร้างรหัสเลขฐานสองที่ดีที่สุด ในการบีบอัดข้อมูลแบบสูญเสีย (lossless compression) จะมีค่าเท่ากับค่าเอนโทรปีของแหล่งข้อมูลนั่นด้วย

ถ้ามีกลุ่มสัญลักษณ์ A ซึ่งนิยมเรียกว่า "alphabet" ที่เป็นแหล่งข้อมูล สมาชิก A = {1,2,...,m } ซึ่งแทนด้วยข้อมูล $X_i = \{X_1, X_2, \ldots\}$ เอนโทรปีของข้อมูลนี้คือ

$$H(S) = \lim_{n \to \infty} \frac{1}{n} G_n \tag{3.4}$$

ขณะที่

$$G_n = -\sum_{i_1=1}^{i_1=m} \sum_{i_2=1}^{i_2=m} \cdots \sum_{i_1=1}^{i_n=m} P(X_1 = i_1, X_2 = i_2, \cdots, X_n = i_n) \log P(X_1 = i_1, X_2 = i_2, \cdots, X_n = i_n)$$

และ

$\{X_1, X_2, \ldots, X_n\}$ เป็นขนาดความยาวของข้อมูล n ข้อมูล ถ้าสมาชิกแต่ละตัวเป็นอิสระต่อกัน

$$G_n = -n \sum_{i_1}^{i_1=m} P(X_1 = i_1) \log P(X_1 = i_1) \tag{3.5}$$

ฉะนั้นค่าเอนโทรปีจะเขียนแทนด้วย

$$H(S) = -\sum P(X_i = i_1) \log_b P(X_i) \tag{3.6}$$

โดยปกติแล้วค่าเอนโทรปีจะขึ้นอยู่กับโครงสร้างของลำดับข้อมูลด้วย เช่น ถ้าเรามีลำดับข้อมูลดังนี้

1 2 3 2 3 4 5 4 5 6 7 8 9 8 9 10

จากลำดับข้อมูลจะได้ n = 16 และค่าความถี่หรือความน่าจะเป็นที่จะเกิดขึ้นของข้อมูลแต่ละตัว

$$P(1) = P(6) = P(7) = P(10) = \frac{1}{16}$$
$$P(2) = P(3) = P(4) = P(5) = P(8) = P(9) = \frac{2}{16}$$

เราสามารถคำนวณหาค่าเอนโทรปีของข้อมูลชุดนี้ได้คือ

$$H = -\sum_{i=1}^{10} P(i) \log_2 P(i)$$
$$= -(\frac{1}{16}\log_2\frac{1}{16} + \frac{1}{16}\log_2\frac{1}{16} + \frac{1}{16}\log_2\frac{1}{16} + \frac{1}{16}\log_2\frac{1}{16} + \frac{1}{16}\log_2\frac{1}{16} +$$
$$\frac{2}{16}\log_2\frac{2}{16} + \frac{2}{16}\log_2\frac{2}{16} + \frac{2}{16}\log_2\frac{2}{16} + \frac{2}{16}\log_2\frac{2}{16} + \frac{2}{16}\log_2\frac{2}{16} + \frac{2}{16}\log_2\frac{2}{16})$$
$$= 3.25 \text{ บิต/สัญลักษณ์}$$

ดังนั้น จากตัวอย่างเราสามารถแทนจำนวน 3.25 บิตต่อสัญลักษณ์

3.2 คณิตศาสตร์สำหรับการบีบอัดข้อมูลแบบไม่สูญเสีย(Mathmatical for Lossy Compression)

3.2.1 Conditional Entropy

กำหนดให้ X เป็นแหล่งกำเนิดข้อมูล โดยที่ $X = \{x_0, x_1, \ldots, x_{N-1}\}$ และให้ Y เป็นข้อมูลที่ถูกสร้างใหม่ จาก X และ $Y = \{y_0, y_1, \ldots, y_{M-1}\}$ เราจะได้ค่าเอนโทรปีของทั้ง แหล่งกำเนิดข้อมูล และข้อมูลที่ถูกสร้างใหม่โดย

$$H(X) = -\sum_{i=0}^{N-1} P(x_i) \log_2 P(x_i) \tag{3.7}$$

และ $$H(Y) = -\sum_{i=0}^{M-1} P(y_i) \log_2 P(y_i) \quad (3.8)$$

การวัดความสัมพันธ์ระหว่าง H(X \Y) เรียกว่า " Condition Entropy " ถ้ามีข้อมูล A มีโอกาสจะเกิดเท่ากับ P(A) ความจุข่าวสารของข้อมูล A (self-information)สามารถเขียนได้คือ

$$i(A) = \log\left(\frac{1}{P(A)}\right) = -\log P(A) \quad (3.9)$$

ในทำนองเดียวกันถ้ามีข้อมูล A มีโอกาสจะเกิดเท่ากับ P(A) และข้อมูล B มีโอกาสจะเกิดเท่ากับ P(B) ความจุข่าวสารของข้อมูล A\B สามารถเขียนได้คือ

$$i(A \setminus B) = \log\left(\frac{1}{P(A \setminus B)}\right) = -\log P(A \setminus B) \quad (3.10)$$

แต่สิ่งที่สนใจคือค่าเฉลี่ยความจุข่าวสารที่เรียกว่า Condition Entropy ของแหล่งกำเนิดข้อมูล (X) และ ข้อมูลที่ถูกสร้างใหม่(Y)

$$H(X \setminus Y) = -\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} P(x_i \setminus y_i) P(y_i) \log_2 P(x_i \setminus y_i) \quad (3.11)$$

และ $$H(Y \setminus X) = -\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} P(x_i \setminus y_i) P(y_i) \log_2 P(y_i \setminus x_i) \quad (3.12)$$

ซึ่งจะได้ความสัมพันธ์ของ Condition Entropy กับ Entropy ของแหล่งกำเนิดข้อมูลดังสมการที่ (3.13) หรือ Condition Entropy จะน้อยกว่าเอนโทรปีของแหล่งกำเนิดข้อมูล

$$H(X \setminus Y) \leq H(X) \quad (3.13)$$

ตัวอย่างในการทำความเข้าใจพื้นฐานการบีบอัดข้อมูลแบบสูญเสีย(lossy compression) ก็คือการลดจำนวนบิตที่มีนัยสำคัญต่ำกว่าของข้อมูลเอาท์พุทลงเช่นการสร้างข้อมูลภาพสีเดียวซึ่งมีขนาด 8 บิตต่อพิกเซลซึ่งจะนำไปแสดงบนจอภาพขาวดำขนาด 6 บิตต่อพิกเซล นั้นทำได้โดยการลดจำนวนบิตที่มีนัยสำคัญต่ำกว่าในแต่ละพิกเซลลง 2 บิตก่อนทำการส่งภาพไปแสดงซึ่งยังมีวิธีการอื่นที่จะสามารถสร้างให้มีประสิทธิภาพได้ดีกว่า

สมมุติใช้ข้อมูลเลขไบนารี 4 บิตแทนข้อมูลสัญลักษณ์หนึ่งหรือจะสร้างข้อมูลได้ {0,1,2,...,15} และกำหนดว่าต้องบีบอัดข้อมูลให้มีขนาด 3 บิตหรือจะสร้างข้อมูลได้ {0,1,2,...,7}จะเห็นได้ว่าข้อมูลเอาท์พุทไม่สามารถสร้างข้อมูลใหม่ให้เหมือนข้อมูลอินพุทได้ ดังนั้นเราจะประมาณข้อมูลเอาท์พุทโดยการเลื่อนบิต ในตำแหน่งบิตที่มีความสำคัญต่ำขึ้น 1 บิตหรือก็คือการคูณข้อมูลเอาท์พุทด้วย 2 ดังนั้นข้อมูลใหม่ที่ได้ {0,2,4,...,14}

จากแหล่งกำเนิดข้อมูล $P(X=i)=\frac{1}{16}; i\in\{0,1,2,...,15\}$

ค่าเอนโทรปีของแหล่งกำเนิดข้อมูลนี้คือ $H(X)=-\sum_i \frac{1}{16}\log\frac{1}{16}=\log 16=4$ บิต

ความน่าจะเป็นในการสร้างข้อมูลใหม่ $P(Y=j)=P(X=j)+P(X=j+1)=\frac{1}{16}+\frac{1}{16}=\frac{1}{8}$

และ ค่าเอนโทรปีของข้อมูลใหม่ H(Y) เท่ากับ 3 บิต การหา Condition Entropy ; $H(X \setminus Y)$ จำเป็นต้องทราบค่าของ $P(x_i \setminus y_i)$ จากการเข้ารหัสของแหล่งกำเนิดข้อมูลเพื่อสร้างข้อมูลใหม่

$$P(X=i|Y=j)=\begin{cases}\frac{1}{2} & \text{if } i=j \text{ or } i=j+1 \text{ for } j=0,2,4,...,14\\ 0 & \text{otherwise}\end{cases}$$

ดังนั้นจากสมการ (3.9) เราสามารถหา Condition Entropy ; $H(X \setminus Y)$ ได้

$$\begin{aligned}H(X \setminus Y)&=-\sum_i\sum_j P(X=i|Y=j)P(Y=j)\log P(X=i|Y=j)\\ H(X \setminus Y)&=-\sum_j\Big[P(X=j|Y=j)P(Y=j)\log P(X=j|Y=j)\\ &\quad+P(X=j+1|Y=j)P(Y=j)\log P(X=j+1|Y=j)\Big]\\ &=-8[\tfrac{1}{2}\cdot\tfrac{1}{8}\log\tfrac{1}{2}+\tfrac{1}{2}\cdot\tfrac{1}{8}\log\tfrac{1}{2}]\\ &=1\end{aligned}$$

3.2.2 Average Mutual Information

เราสามารถหาจำนวนของบิตต่อสัญลักษณ์ (Entropy) ของตัวแปรสุ่ม 2 ตัวแปรที่เรียกว่า " mutual information " ได้คือ

$$i(x_k;y_j)=\log\left[\frac{P(x_k \setminus y_j)}{P(x_k)}\right] \qquad (3.14)$$

โดยที่ค่าเฉลี่ยของจำนวนนี้เราเรียกว่า " average mutual information " สามารถกำหนดได้โดย

$$I(X;Y)=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_i)\log\left[\frac{P(x_i|y_i)}{P(x_i)}\right] \tag{3.15}$$

$$=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i|y_i)P(y_i)\log\left[\frac{P(x_i|y_i)}{P(x_i)}\right] \tag{3.16}$$

จากสมการ (3.16) เราสามารถเขียนอยู่ในรูปของค่า Entropy และ Condition Entropy

$$I(X;Y)=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_i)\log\left[\frac{P(x_i|y_i)}{P(x_i)}\right] \tag{3.17}$$

$$=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_i)\log P(x_i|y_i)-\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_i)\log P(x_i) \tag{3.18}$$

$$=H(X)-H(X|Y) \tag{3.19}$$

จากสมการ (3.18) ค่า average mutual information ก็คือ ผลต่างของค่าเอนโทรปีของแหล่งกำเนิดกับ Condition Entropy นั่นเอง หรือสามารถเขียนใหม่ได้เป็น

$$I(X;Y)=H(Y)-H(Y|X)=I(Y;X) \tag{3.20}$$

3.2.3 Differential Entropy

จากที่ผ่านมาเป็นการศึกษาข้อมูลอินพุทหรือแหล่งกำเนิดข้อมูลที่ไม่ต่อเนื่อง(discrete alphabet) แต่ในความเป็นจริงแล้ว การศึกษาเทคนิคการบีบอัดข้อมูลแบบสูญเสีย แหล่งกำเนิดข้อมูลส่วนใหญ่จะไม่เป็นอย่างนั้น ในหัวข้อนี้จะแสดงถึงแหล่งกำเนิดข้อมูลแบบต่อเนื่อง (Continuous) แล้วทำการสุ่ม(random)เลือกข้อมูลหรือตัวแปรมาทำการวิเคราะห์

ในหัวข้อที่ผ่านมาเราทราบว่าความจุข่าวสาร(self-informtion) ของตัวแปร x_i ที่มีโอกาสที่จะเกิด $P(x_i)$ มีค่าเป็น $\log\frac{1}{p(x_i)}$ แต่ในกรณีของตัวแปรสุ่ม(random variable) ของข้อมูลแบบต่อ

เนื่องแล้ว $P(x_i) = 0$ ดังนั้นความจุข่าวสารในกรณีนี้จะเท่ากับอนันต์(∞) หรือเราไม่สามารถหาค่าความจุข่าวสารได้นั่นเอง

วิธีในการนิยาม Entropy ซึ่งจะทำให้เราหาค่าเฉลี่ยของ ค่าความจุข่าวสารได้นั้น เราจะใช้หลักการของฟังก์ชันต่อเนื่อง(continuouse function) โดยทำการกำหนดลิมิต(limit) ของฟังก์ชันนั่นให้ X เป็นตัวแปรสุ่มแบบต่อเนื่อง(continuouse random variable)ใด ๆ ความน่าจะเป็นของฟังก์ชันจะเรียกว่า pdf(probability density function) ; $f_x(x)$

ให้ Δ เป็นช่วงระหว่างค่าของตัวแปรสุ่ม โดยในแต่ละช่วงของตัวแปรสุ่ม x_i คือ $[(i-1)\Delta, i\Delta)$ ดังนั้น

$$f_x(x_i)\Delta = \int_{(i-1)\Delta}^{i\Delta} f_x(x)dx \tag{3.21}$$

กำหนดให้ X_d เป็นค่าตัวแปรสุ่มแบบไม่ต่อเนื่อง(discrete random variable)

$$P(X_d = x_i) = f_x(x_i)\Delta \tag{3.22}$$

ดังนั้นเราจะได้ค่าเอนโทรปีของตัวแปรสุ่ม; $H(X_d)$

$$H(X_d) = -\sum_{i=-\infty}^{\infty} P(x_i)\log P(x_i) \tag{3.23}$$

$$= -\sum_{i=-\infty}^{\infty} f_x(x_i)\Delta \log f_x(x_i)\Delta \tag{3.24}$$

$$= -\sum_{i=-\infty}^{\infty} f_x(x_i)\Delta \log f_x(x_i) - \sum_{i=-\infty}^{\infty} f_x(x_i)\Delta \log\Delta \tag{3.25}$$

$$= -\sum_{i=-\infty}^{\infty} [f_x(x_i)\log f_x(x_i)]\Delta - \log\Delta \tag{3.26}$$

ใส่ลิมิต $\Delta \to 0$ ในสมการ (3.26) ซึ่งเรียกว่า "differential entropy" ของแหล่งกำเนิดข้อมูลแบบต่อเนื่อง(continuouse source) เขียนแทนด้วย h(X) จะได้

$$h(X) = -\int_{-\infty}^{\infty} f_x(x)\log f_x(x)dx \tag{3.27}$$

ตัวอย่าง สมมุติตัวแปรสุ่ม X มีการกระจายแบบแบบสม่ำเสมอในช่วง [a,b) ค่า differential entropy ของตัวแปรสุ่มหาได้จากสมการ (3.27);

$$h(X) = -\int_{-\infty}^{\infty} f_x(x)\log f_x(x)dx$$
$$= \log(b-a)$$

ตัวอย่าง สมมุติตัวแปรสุ่ม X มีการกระจายแบบ Gaussian *pdf*

$$f_x(X) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp -\frac{(x-\mu)^2}{2\sigma^2}$$

ดังนั้น diiferential entropy สามารถหาได้จาก

$$h(X) = -\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}}\exp -\frac{(x-\mu)^2}{2\sigma^2} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp -\frac{(x-\mu)^2}{2\sigma^2}\right]dx$$
$$= -\log\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} f_x(x)dx + \int_{-\infty}^{\infty}\frac{(x-\mu)^2}{2\sigma^2}\log e f_x(x)dx$$
$$= \frac{1}{2}\log 2\pi\sigma^2 + \frac{1}{2}\log e$$
$$= \frac{1}{2}\log 2\pi e\sigma^2$$

ค่าของ diffential entropy สำหรับการกระจายของข้อมูลแบบ Gaussian นั้นจะมีสูงกว่าการกระจายของตัวแปรสุ่มแบบต่อเนื่อง ดังแสดงในสมการ 3.28

$$h(x) \le \frac{1}{2}\log 2\pi e\sigma^2 \tag{3.28}$$

การพิสูจน์ทำได้โดยการกำหนดการกระจายของตัวแปรสุ่มแบบต่อเนื่อง 2 ตัวคือ $f_x(x)$ และ $g_x(x)$

$$-\int_{-\infty}^{\infty} f_x(x)\log f_x(x)xdx \leq -\int_{-\infty}^{\infty} f_x(x)\log g_x(x)xdx \tag{3.29}$$

เราจะไม่พิสูจน์สมการ (3.29) ว่าสมการซ้ายน้อยกว่าสมการขวามืออย่างไร ซึ่งจะพิสูจน์โดยจะกำหนดให้ $g_x(x)$ มีการกระจายของข้อมูลแบบ Gaussian ดังนั้น

$$\begin{aligned} h(X) &\leq -\int_{-\infty}^{\infty} f_x(x)\log\frac{1}{\sqrt{2\pi\sigma^2}}\exp-\frac{(x-\mu)^2}{2\sigma^2}dx \\ &= \frac{1}{2}\log(2\pi\sigma^2)+\log e\int_{-\infty}^{\infty} f_x(x)\frac{(x-\mu)^2}{2\sigma^2}dx \\ &= \frac{1}{2}\log(2\pi\sigma^2)+\frac{\log e}{2\sigma^2}\int_{-\infty}^{\infty} f_x(x)(x-\mu)^2dx \\ &= \frac{1}{2}\log(2\pi e\sigma^2) \end{aligned} \tag{3.30}$$

ถ้าให้ Y_d เป็นค่าตัวแปรสุ่มแบบไม่ต่อเนื่อง(discrete random variable)เหมือน X_d แล้วเราสามารถแสดงให้เห็นได้ว่า

$$H(X_d \setminus Y_d) = -\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}[f_{x/y}(x_i \setminus y_i)f_y(y_i)\log f_{x/y}(x_i \setminus y_i)]\Delta\Delta - \log\Delta \tag{3.31}$$

และค่า averrage mutual information จะกำหนดได้เป็น

$$I(X_d;Y_d) = H(Y_d) - H(X_d \setminus Y_d) \tag{3.32}$$

$$= -\sum_{i=-\infty}^{\infty} f_x(x_i)\Delta\log f_x(x_i) \tag{3.33}$$

$$-\sum_{i=-\infty}^{\infty}\left[\sum_{j=-\infty}^{\infty} f_{x\setminus y}(x_i \setminus y_i)f_y(y_i)\log f_{x\setminus y}(x_i \setminus y_i)\Delta\right]\Delta \tag{3.34}$$

ถ้าให้ $\Delta \to 0$ $I(X_d;Y_d)$ ก็จะเขียนได้เป็น $I(X;Y)$, เทอมในสมการที่ (3.33) ก็จะมีค่าเป็น $h(X)$ และ เทอมในสมการที่ (3.34) ก็จะมีค่าเป็น $h(X \setminus Y)$

$$I(X;Y) = h(X) - h(X \setminus Y) \tag{3.35}$$

3.2.4 หลักการของ Transform Coding

การทรานสฟอร์มส่วนใหญ่จะเป็นการทรานสฟอร์มเชิงเส้น (linear transforms) โดย $\{\theta_n\}$ เป็นข้อมูลที่ได้จากการทรานสฟอร์มข้อมูลอินพุท $\{x_n\}$ เขียนเป็นความสัมพันธ์ดังสมการ

$$\theta_n = \sum_{i=0}^{N-1} x_i a_{n,i} \tag{3.36}$$

ซึ่งเรียกว่า "forward transform" และข้อมูลอินพุท $\{x_n\}$ นี้สามารถสร้างกลับมาโดยการทรานสฟอร์มกลับ(inverse transform) ดังสมการ

$$x_n = \sum_{i=0}^{N-1} \theta_i b_{n,i} \tag{3.37}$$

การทรานสฟอร์มสามารถเขียนแทนด้วยเมทริกซ์

$$\theta = Ax \tag{3.38}$$

$$x = B\theta \tag{3.39}$$

ขณะที่ A และ B เป็นเมทริกซ์ขนาด $N \times N$ และ (i, j) เป็นตำแหน่งของสมาชิกภายในเมทริกซ์ และกำหนดโดย

$$[A]_{i,j} = a_{i,j} \tag{3.40}$$

$$[B]_{i,j} = b_{i,j} \tag{3.41}$$

การทรานสฟอร์มและการทรานสฟอร์มกลับของเมทริกซ์ A และ B นั้นเป็นอินเวอร์ส ซึ่งกันและกันโดย AB = BA = I ขณะที่ I เป็นเมทริกซ์เอกลักษ์(identity matrix) ในสมการที่ (3.36) และ (3.37) เป็นการ ทรานสฟอร์มของข้อมูล 1 มิติ(one-dimension)

เช่น ข้อมูลจำพวกเสียง(speech ,audio)และการทรานสฟอร์ม มีความจำเป็นมากในการบีบอัดข้อมูลภาพ(image compression)ซึ่งจะเป็นการทรานสฟอร์ม ของข้อมูล 2 มิติ(two-dimension)

กำหนด $X_{i,j}$ โดยที่ (i , j) เป็นตำแหน่งของพิกเซล(pixel) ในรูปภาพ โดยทั่วไป การทรานสฟอร์ม แบบ 2 มิติ ของบล็อกขนาด $N \times N$ สามารถเขียนแทนด้วยสมการ (3.42)

$$\Theta_{k,l} = \sum_{i=0}^{N-1} \sum_{J=0}^{N-1} X_{i,j} a_{i,j,k,l} \tag{3.42}$$

การทรานสฟอร์มสามารถแยกทำครั้งละ 1 มิติโดย การทรานสฟอร์มจากแถวให้ครบทุกแถวแล้วถึงมาทำในด้านของคอลัมภ์ ซึ่งผลที่ได้เท่ากับการทำ 2 มิติ เมื่อต้องการทำการทรานสฟอร์มกลับ ก็จะทำจากทุกคอลัมภ์ แล้วถึงมาทำในด้านของแถว การทรานสฟอร์ม ลักษณะนี้สามารถเขียนแทนด้วยสมการ

$$\Theta_{k,l} = \sum_{i=0}^{N-1} \sum_{J=0}^{N-1} a_{k,i} X_{i,j} a_{i,j} \tag{3.43}$$

โดยทฤษฎีของเมทริกซ์ แล้ว จะเขียนแทนด้วย

$$\Theta = AXA^{T} \tag{3.44}$$

และ inverse transform จะเขียนแทนด้วย

$$X = B\Theta B^{T} \tag{3.45}$$

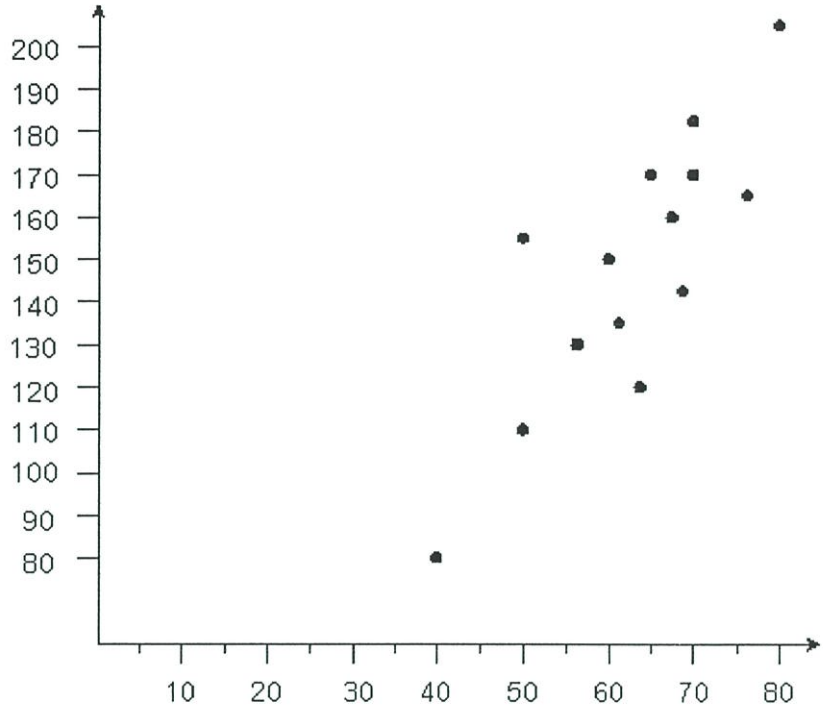
การ transform ข้างต้นที่กล่าวมา เรียกว่า “orthonormal transform “ ในรูปแบบการ transform ลักษณะนี้ จะมีคุณสมบัติ ดังสมการข้างล่างนี้

$$B = A^{-1} = A^{T} \tag{3.46}$$

ตัวอย่างที่ 3.1 สมมุติมีความสัมพันธ์ของข้อมูลระหว่างความสูง(Height)และน้ำหนัก(Weight) ดังแสดงในตารางที่ 3.1

ตารางที่ 3.1 Original sequence

Height	Weight
65	170
75	188
60	150
70	170
56	130
80	203
68	160
50	110
40	80
50	153
69	148
62	140
76	164
64	120



รูปที่ 3.1 ค่าความสัมพันธ์ระหว่างความสูงและน้ำหนัก

เมื่อนำข้อมูลในตารางที่ 3.1 ระหว่างความสูงและน้ำหนักมาเขียนกราฟจะได้ดังแสดงในรูปที่ 3.1 ซึ่งแนวโน้มของข้อมูลจะกระจายรอบ ๆ เส้น y = 2.5x ทำให้สามารถทำการแปลง (Transformation) ได้เป็น

$$\theta = Ax \tag{3.47}$$

โดยที่ x เป็นข้อมูลเวคเตอร์ 2 มิติ สามารถเขียนแทนได้เป็น

$$x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \tag{3.48}$$

ขณะที่ x_0 และ x_1 เป็นข้อมูลเวคเตอร์ 1 มิติที่แทนความสูงและน้ำหนักตามลำดับ ให้ A เป็นเมทริกซ์ที่ทำการแปลง(Rotation matrix)

$$A = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \tag{3.49}$$

ϕ เป็นมุมระหว่างแกน x , เส้น y = 2.5x และ

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \tag{3.50}$$

ในกรณีนี้การคำนวณหาเมทริกซ์ A สามารถคำนวณได้จาก y = 2.5x หรือ $\tan\phi = 2.5$ เมื่อหาค่ามุมจะได้ $\phi = 68.198$, หาค่า $\cos 68.198 = 0.37139068$, $\sin 68.198 = 0.92847669$ จะได้เมทริกซ์ A คือ

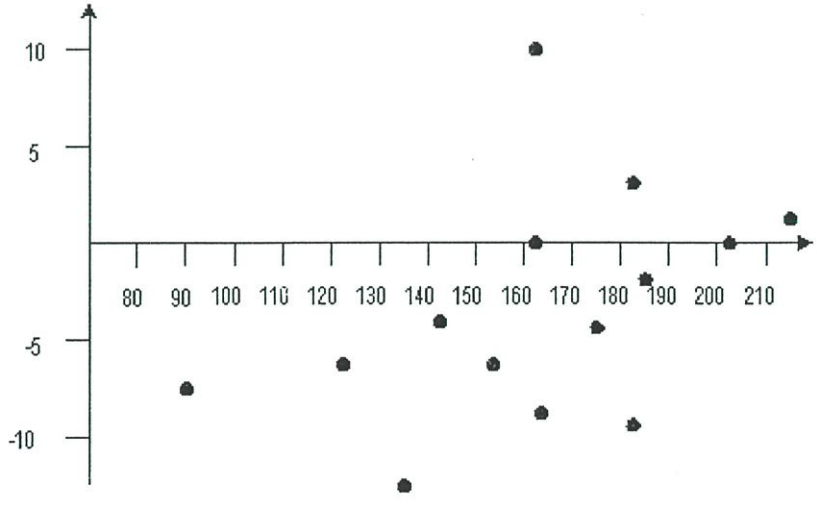
$$A = \begin{bmatrix} 0.37139068 & 0.92847669 \\ -0.92847669 & 0.37139068 \end{bmatrix} \tag{3.51}$$

ต่อไปก็จะเป็นขบวนการในการแปลง(Transfromed sequence) ดังแสดงในตารางที่ 3.2 แล้วนำข้อมูลที่ได้มาพล็อตลงกราฟดังแสดงในรูปที่ 3.2 เพื่อหาลำดับการแปลง ข้อสังเกตุจากตารางที่ 3.2 เมื่อทำการแปลงแล้วมีข้อมูลบางค่ามีค่าเป็นศูนย์ 2 ค่า ทำให้เราลดจำนวนของข้อมูลไปครึ่งหนึ่ง สิ่งที่มีผลต่อการลดจำนวนของข้อมูลคืออินเวิร์สของการแปลง(inverse transform) ซึ่งแทนด้วย A^{-1}

$$A^{-1} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \tag{3.52}$$

ตารางที่ 3.2 Transform sequence

First Coordinate	Second Coordinate
182	3
202	0
162	0
184	-2
141	-4
218	1
174	-4
121	-6
90	-7
161	10
163	-9
153	-6
181	-9
135	-15



รูปที่ 3.2 ลำดับในการทรานสฟอร์ม

การสร้างข้อมูลกลับคืน(Reconstruced sequence) แสดงดังในตารางที่ 3.3

ตารางที่ 3.3 Reconstructed sequence

Height	Weight
68	169
75	188
60	150
68	171
53	131
81	203
65	162
45	112
34	84
60	150
61	151
57	142
67	168
50	125

จากตารางจะเห็นว่ามีความผิดพลาดของข้อมูลเกิดจากการแปลงขึ้น ซึ่งแสดงความสัมพันธ์ได้ดังสมการ

$$\sum_{i=0}^{N-1} (x - \hat{x}_i)^2 = \sum_{i=0}^{N-1} (\theta_i - \hat{\theta}_i)^2 \qquad (3.53)$$

โดยที่ $\{\hat{x}_n\}$ เป็นข้อมูลที่สร้างกลับคืน

$$\hat{\theta}_i = \begin{cases} \theta_i & i = 0,2,4,.... \\ 0 & \text{otherwise} \end{cases} \qquad (3.54)$$

ในเรื่อง Transform Coding มี 3 ขั้นตอนหลัก คือข้อมูล $\{x_n\}$ จะถูกแบ่งออกเป็นบล็อกที่มีขนาด N โดยในแต่ละบล็อกจะถูกทำการทรานสฟอร์มไปตามลำดับ;$\{\theta_n\}$ขั้นตอนที่สองคือจะทำการควอนไตซ์$\{\theta_n\}$และขั้นตอนสุดท้ายคือนำข้อมูลจากขั้นตอนที่สองไปทำการเข้ารหัสเลขไบนารี่(Entropy Coding)

บทที่ 4
ภาษาวีเอชดีแอล (VHDL)

บทนำ

วีเอชดีแอล (VHDL) ย่อมาจากคำว่า VHSIC Hardware Description Language (VSHIC : Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Language) VHDL เป็นภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ทั้งในระดับเกทจนถึงระดับดิจิตอลที่ซับซ้อน ทั้งนี้เนื่องจาก VHDL เป็นภาษาที่ที่เหมาะในการนำมาใช้ในการเลียนแบบการทำงานของอุปกรณ์ อีกทั้งยังมีความยืดหยุ่นและไม่ถูกจำกัดโดยความสามารถทางเทคโนโลยีใดๆ ทำให้ประหยัดเวลาและค่าใช้จ่ายในการออกแบบ จึงได้มีการนำมาใช้กันอย่างกว้างขวางในวงการอุตสาหกรรม รูปแบบของภาษาวีเอชดีแอลประกอบด้วย 2 ส่วนใหญ่ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล(Sequential Language) และภาษาคอนเคอร์เรนท์ (Concurrent Language) การโปรแกรมด้วยภาษาวีเอชดีแอลสามารถเขียนได้ทั้งสองรูปแบบรวมกัน นอกจากนี้ตัวภาษายังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยเข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้และสามารถกำหนดรูปแบบไวยากรณ์(Syntex) อีกทั้งยังมีการตรวจสอบความหมายของภาษาว่าจะซิมูเลท(Simulate) ได้หรือไม่ เพราะโปรแกรมที่เขียนโดยวีเอชดีแอลต้องผ่านการซิมูเลทเพื่อตรวจสอบการทำงาน ฉะนั้นในการคอมไพล์(Compile) จะมีการตรวจสอบทั้งวงจรและซีมูเลชั่นซีแมนติก(Semantic) อย่างไรก็ตามแม้ตัวภาษาจะมีความซับซ้อนในรูปแบบและกฎเกณฑ์ของภาษา แต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำไปใช้งานโดยไม่จำเป็นต้องศึกษารายละเอียดทั้งหมด

4.1 แนะนำวีเอชดีแอล (Introduction to VHDL)

ในช่วงฤดูร้อนของปี 1981 สถาบันเพื่อป้องกัน (The Institute for Defence Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่ง เพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้น เรียกว่า VHDL (VHSIC Hardware Description Language) โดย VHISC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวกับวงจรรวมที่มีความเร็วสูงมาก (VERY High Speed Intergrated Circuit) ต่อมาในปี 1985 IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL ในด้านของการกำหนดพฤติกรรม

การทำงานของวงจร ทำให้นักออกแบบสามารถกำนดรูปแบบพฤติกรรมการทำงานได้ทั้งวงจรของดิจิตอลทั่วๆ ไป และในระบบที่แตกต่างกันออกไป เช่น พฤติกรรมการทำงานของระบบเรดาร์หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในสมองมนุษย์ได้ ข้อดีหลักที่สำคัญของ VHDL ก็คือภาษานี้จะสามารถถูกใช้ได้ตลอดในทุกๆ ระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูง(System Level) จนถึงระดับที่ต่ำกว่า(Lower hardware level) สามารถใช้ภาษาเดียวกันได้โดยตลอด ทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

4.1.1 ข้อกำหนด (VHDL Requirement)

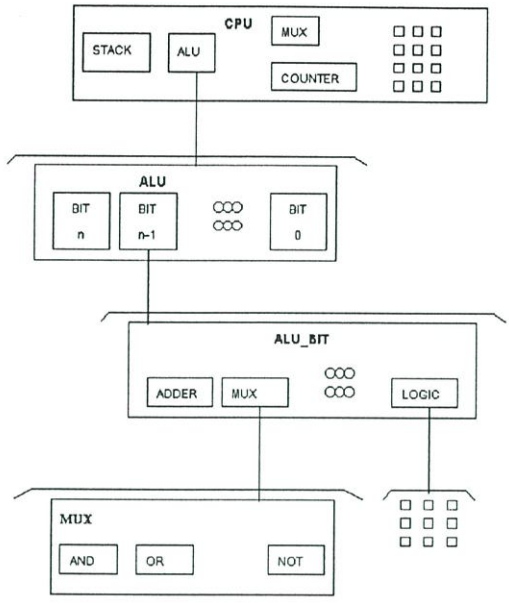
ในเอกสารของ DoD (Department of Defense Requriment for Hardware Description Language) ซึ่งออกมาในเดือนมกราคมปี 1983 ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ไว้ดังนี้

4.1.1.1 ลักษณะทั่วไป (Generation Features)

เอกสารของ DoD กำหนดไว้ว่า VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง ความสามารถในการเลียนแบบ(Simulation) การสังเคราะห์(Synthesis) และการทดสอบ(Testing) นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์คือ ระบบจนถึงระดับเกทอีกด้วย เนื่องจากในการทำงานของระบบดิจิตอลจริงๆ ทุกๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อมๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ถือว่าเป็นข้อกำหนดที่สำคัญอย่างหนึ่งใน VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้ว ความพร้อมเพรียงจะหมายถึงทุกๆ คำสั่ง องค์ประกอบเกท หรือวงจรต่างๆ จะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายแล้วก็จะดูเหมือนว่า ได้มีการปฏิบัติไปพร้อมๆ กัน)

4.1.1.2 สนับสนุนการออกแบบแบบลำดับขั้น (Support for Design Hierarchy)

การออกแบบลำดับขั้น เป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบที่มีหลายๆ ระดับในการออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบก็สามารถกำหนดได้ด้วยตนเองหรือถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อยๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้ ดังรูปที่ 4.1



รูปที่ 4.1 แสดงตัวอย่างการออกแบบแบบลำดับขั้น

4.1.1.3 ไลบรารี (Library Support)

VHDL ได้สนับสนุนการมีไลบราลีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารี หลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้ว เพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไปใช้ได้ด้วย

4.1.1.4 ลำดับคำสั่ง (Sequential Statement)

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case if-then-eles และ loop ทั่วๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานพร้อมเพรียงกันเช่นเดิม

4.1.1.5 การกำหนดคุณสมบัติ (Generic Design)

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่นๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน สิ่งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้นๆ ภาษาสำหรับการออกแบบที่ดีควรจะสามารถให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาด ลักษณะทางกายภาพ เวลา โหลดและเงื่อนไขทางสภาพแวดล้อมอื่นๆ ความสามารถในการกำหนดคุณสมบัติก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

4.1.1.6 ชนิดของข้อมูล (Type Declaration and usage)

VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิด BIT และ BOOLEAN เท่านั้น แต่ยังสามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยมและชนิดลำดับการนับ (Enumerate Type) หรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นใส่เองก็ได้

4.1.1.7 โปรแกรมย่อย (Use of Subprogram)

ความสามารถในการใช้ฟังก์ชันและโพรซีเยอร์(Procedure) เป็นข้อกำหนดอีกอย่างหนึ่งในVHDL เราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก(Logic) การกำหนดตัวกระทำต่างๆ ทั้งเก่าและใหม่หรืออะไรก็ตามได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

4.1.1.8 การควบคุมเวลา (Timing Control)

VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการการตรวจสอบ การออกแบบเกต หรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์(Event) นอกจากนี้ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

4.1.1.9 การกำหนดแบบโครงสร้าง (Structural specification)

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุกๆ ระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวม ที่เกิดจากองค์ประกอบย่อยที่ต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่งเช่นกัน

4.2 ความสามารถของภาษาวีเอชดีแอล (Capability)

- ตัวภาษา VHDL สามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิพกับผู้ออกแบบ (CAD Tools)

- ใช้เป็นสื่อกลางในการแลกเปลี่ยนสื่อสารระหว่างซีเออี(CAE) และซีเอดีทูล(CAD Tools) เช่นตัวภาษาซ้อสโค้ด(SOURCE CODE) ของ VHDL สามารถคอมไพล์โดยใช้คอมไพล์เลอร์ (Compiler) และซิมูเลเตอร์(Simulator) ได้หลายตัวแตกต่างกัน

- ภาษา VHDL สนับสนุนการออกแบบ แบบท๊อปดาวน์(Top Down Design) และแบบบัททอมอัพ(Bottom Up Design) หรือผสมกันทั้งสองแบบ

- ตัวภาษา VHDL เป็นแบบทั่วไป(Generic)ไม่อิงเทคโนโลยีอันใดอันหนึ่ง ในขณะเดียวกันก็สนับสนุนหลายๆ เทคโนโลยี

- ตัวภาษา VHDL สามารถอ่านและทำความเข้าใจได้โดยมนุษย์

-สนับสนุนการออกแบบทั้งระบบซิงโครนัส(Synchronous) และอะซิงโครนัส (Asynchronous)

- ตัวภาษา VHDL เป็นภาษามาตรฐานรับรองโดย IEEA และ ANSI ทำให้โมเดลที่ออกแบบโดยภาษา VHDL สามารถเคลื่อนย้ายไปยังระบบใดๆ ก็ได้ และสามารถนำกลับมาใช้ใหม่ได้

- สามารถเขียนโมเดลได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของโมเดล (ขึ้นอยู่กับซอฟแวร์)

-ภาษา VHDL สนับสนุนการเขียนถึง 3 รูปแบบ ได้แก่ แบบบีเฮฟวีเออร์(Behavioral Style) แบบสตรัคเจอรัล(Structural Style) แบบดาต้าโฟลว์ (Data Flow) หรือสามารถเขียนรวมกันได้ทั้ง 3 รูปแบบ

-สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของส่วนประกอบ(Component) ฟังก์ชั่นโพรซีเดอร์(Function Procedure) และแพคเกต(Package)

- สามารถอธิบายตัวแปรที่เกี่ยวกับฟังก์ชันทางด้านเวลา เช่น Propagation delay , Min-Max Delay , Setup , Holding Time สามารถอธิบายได้โดยตัวภาษา

-ภาษา VHDL เป็นมาตรฐานที่ใช้โดยบริษัทและผู้ออกแบบหลายๆ แห่ง ฉะนั้นจึงง่ายที่จะทำความเข้าใจถึงแม้ว่าจะมาจากแหล่งต่างๆ

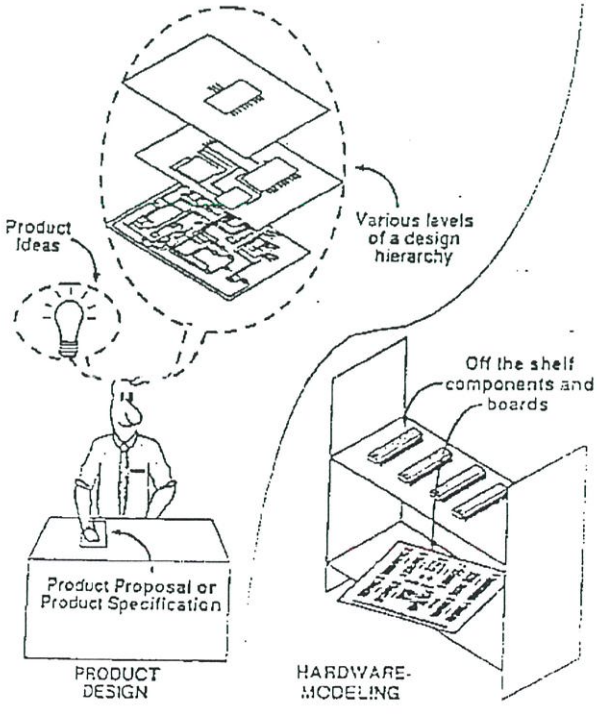
-โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะว่าตัวแปรภาษาได้ตรวจสอบตัวแปรทางซิมูเลชั่นซีแมนติกไว้ด้วย

4.3 หลักการสร้างโมเดลโดยใช้ภาษาวีเอชดีแอล

(General VHDL Modelling Principles)

วีเอชดีแอลเป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้ ซึ่งจะช่วยในการสร้างและออกแบบวงจรรวมดิจิตอลและส่วนประกอบต่างๆ อาจใช้อธิบายระบบทั้งระบบหรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของ (Component Block) จากนั้นก็ทำการจำลองการทำงาน(Simulate) โดยที่รูปแบบนั้นยังไม่ได้สร้างขึ้นจริงหรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น(Textual Format) หลังจากจำลองการทำงานจนได้ตามที่ต้องการจึงนำไปทำการ Synthesis เพื่อให้ได้วงจรเกทเลเวลต่อไป ประโยชน์จริงของการใช้วงจร วีเอชดีแอลเป็น Design Tools แทนการสร้างต้นแบบ (Prototype) ขึ้นมาจริง คือเราสามารถอธิบาย Product Idea , Product Proposal, Product Specification ในรูปของ Text จากนั้นก็นำคอมไพล์เพื่อดู Timing การทำงาน แล้วแก้ไข (Refine) จนกว่าจะได้ Specification ตามต้องการ เมื่อ product ได้ผลตามที่ต้องการแล้วจึงนำไปสู่การสังเคราะห์(Synthesis) เพื่อให้ได้เกทเลเวล Schematic เพื่อนำไปสร้างเป็นต้นแบบจริงต่อไป ซึ่งต้นแบบที่สร้างนั้นทำงานได้จริงเพราะได้ทำการ Simulate เรียบร้อยแล้ว เป็นการลดเวลาและค่าใช้จ่ายในการสร้างต้นแบบได้มาก

ตัวภาษา VHDL สนับสนุนหลักการต่างๆ ให้เขียนแก้ไขและบำรุงรักษาวงจรดิจิตอลที่มีความซับซ้อนให้เป็นไปอย่างรวดเร็วและมีประสิทธิภาพโดยมีหลักการดังนี้



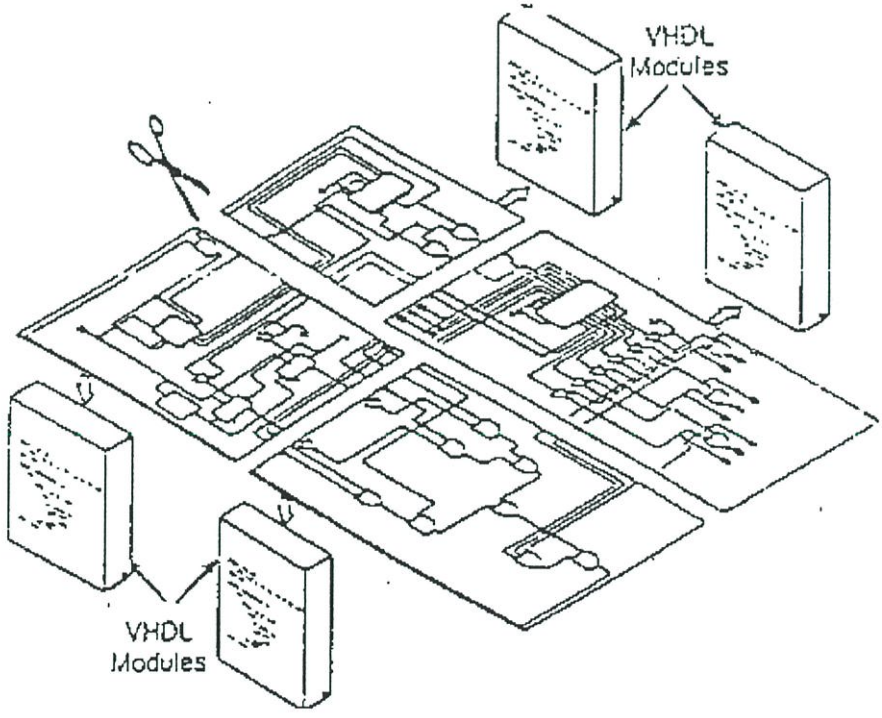
รูปที่ 4.2 สิ่งต่างๆ ที่สามารถอธิบายได้ด้วย VHDL

4.3.1 Top Down Design

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน เช่น ASIC (Application Specific Integrated Circuit) วิศวกรหรือผู้ออกแบบมักจะมองรูปแบบให้อยู่ในรูปของ Block Diagram เสียก่อน ก่อนที่จะย่อยรูปแบบให้ถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาติให้อธิบายการทำงานของแต่ละ Block วิเคราะห์การทำงาน จัดการแก้ไขและปรับปรุงการทำงานจากการวิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ ก่อนที่จะทำการออกแบบให้ละเอียดลึกลงไปในขั้นตอนต่อไป การแก้ไขในขั้นตอนนี้จะทำให้ลดค่าใช้จ่ายกว่าการแก้ไขในช่วงของการพัฒนาในระดับสร้างซิลิกอนชิพ

4.3.2 Modularity

Modularity คือ หลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ ออกเป็นส่วนย่อยเล็กลงไป ซึ่งปกติการทำงานของฮาร์ดแวร์ใหญ่ๆ ต้องประกอบด้วยฮาร์ดแวร์ย่อยๆ ลงไปดังรูปที่ 4.3

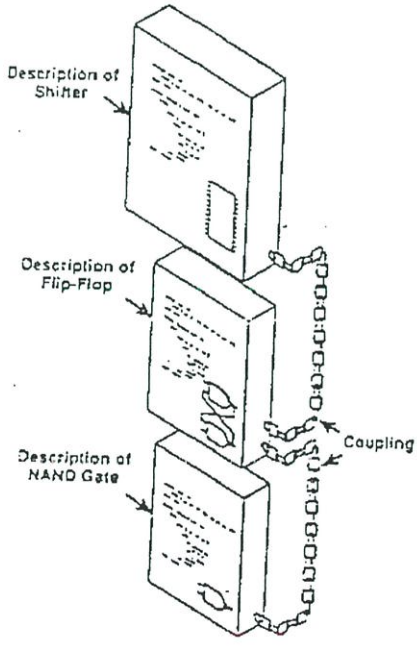


รูปที่ 4.3 การแบ่งย่อยในระดับราบของการออกแบบฮาร์ดแวร์

จากรูปได้แสดงวงจรทั้งหมดในรูปเดียว (Flatten Design) หลังจากนั้นตัดเป็นส่วนย่อยๆ เล็กลงมา เมื่อเราออกแบบโดยใช้ภาษา VHDL หน้าที่การทำงานของแต่ละส่วนต้องสามารถ

อธิบายได้โดยโมดูลของโค้ด(คล้ายฟังก์ชั่นหรือโพรซีเดอร์) ซึ่งแสดงการทำงานของส่วนย่อยนั้นอย่างชัดเจน ซึ่งการแยกรูปใหญ่ๆ ออกเป็นส่วนย่อยๆ นี้ ทำให้ง่ายต่อการจัดการและง่ายต่อการทำความเข้าใจ

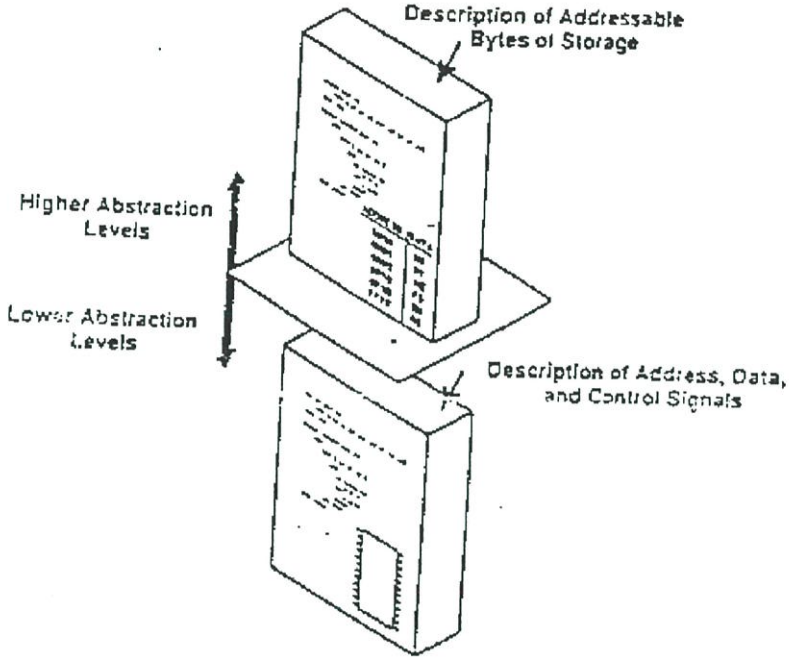
รูปที่ 4.4 แสดง Hierarchy Method โดยการแยกส่วนรูปแบบออกเป็นส่วนย่อยๆ ส่วนบนสุดอธิบายการทำงานของ Shifter ส่วนล่างๆ ลงมา คือการแยกส่วนของ Shifter ออกเป็นฟลิปฟลอป จากฟลิปฟลอปแยกเป็น NAND เกท ภายใน Shifter ได้อธิบายการทำงานโดยใช้การต่อกันของฟลิปฟลอป ในระดับที่ต่ำลงมา ฟลิปฟลอปก็เกิดจาก NAND เกท ต่อกัน 2 ตัวในระดับที่ต่ำลงมาอีกก็เป็น NAND เกท ซึ่งมีการอธิบายการทำงานอยู่ภายใน โดยแต่ละโมดูลจะมีคำอธิบายการทำงานในตัวของมันเองอยู่แล้วคำอธิบายในแต่ละโมดูลมีไว้เพื่อให้สามารถใช้ฟลิปฟลอปโมดูลได้ ส่วนฟลิปฟลอปโมดูลก็อธิบายการเชื่อมต่อไว้อย่างดีทำให้สามารถเชื่อมต่อกับ NAND เกท ในระดับล่างสุดได้ ประโยชน์อย่างหนึ่งของการแยกส่วนฟลิปฟลอปและ NAND เกท ออกจากกัน เนื่องจากทำให้ง่ายในการใช้ NAND เกท ตัวนี้ในรูปแบบไฮเลเวลตัวอื่นๆ ทำให้นำออกไปใช้งานได้อีกและลดความซับซ้อนในการใช้อุปกรณ์เพื่อแก้ไขการทำงานของ Shifter ง่ายขึ้น โดยปราศจากการแก้ไขฟลิปฟลอปและ NAND เกท ประโยชน์ที่ได้จากการทำ Modularity นี้ทำให้รูปแบบที่ออกแบบง่ายต่อการเข้าใจและแก้ไขได้เสมอ



รูปที่ 4.4 การแบ่งแบบ Hierarchy ของ VHDL Shifter Description

4.3.3 Abstraction

คำนิยามของรูปแบบ จะอธิบายการทำงานของตัวรูปแบบมากกว่าจะอธิบายว่าพัฒนาตัวรูปแบบนั้นได้อย่างไร หลักการนี้มีความสำคัญอย่างใกล้ชิดกับหลักการของ Modularity ในรูปที่ 4.4 ฟลิปฟลอป เป็นการนิยามในการใช้ NAND เกท และ Shifter เป็นนิยามในการใช้ฟลิปฟลอป

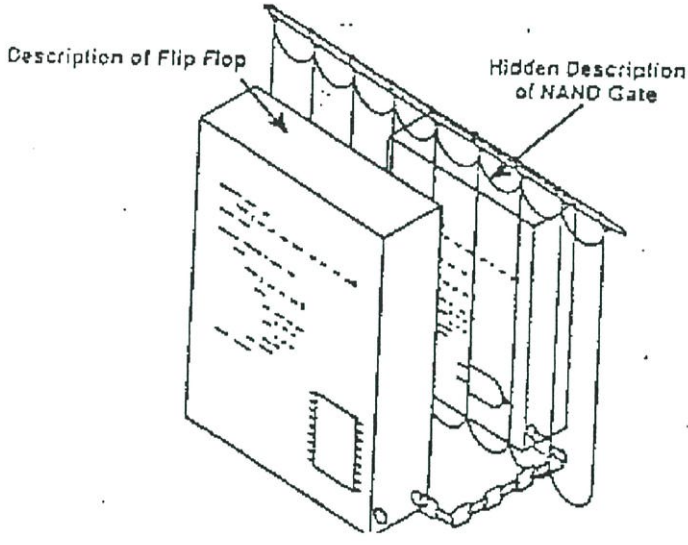


รูปที่ 4.5 Applying Abstraction to a ROM Description

รูปที่ 4.5 แสดงถึงการอธิบายการทำงานของรูปแบบโดยใช้ VHDL ในหลายๆ ระดับของการนิยาม ROM (Read Only Memory) อธิบายโดยใช้ภาษาระดับสูง แสดงถึงตำแหน่งต่างๆ ซึ่งเก็บข้อมูลไว้ ในตำแหน่งนั้นๆ ที่ระดับนี้ไม่ต้องสนใจถึง Address Line , Data Line และ Control Line เราสามารถเพ่งจุดสนใจไปที่ขนาดของข้อมูลโดยไม่ต้องคำนึงถึงสัญญานควบคุมต่างๆ ภายในเพราะว่าส่วนนั้นจะถูกจัดการเองในระดับที่ต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ในการจัดการสัญญาณภายในทุกเส้นภายใน การที่จะอ่านข้อมูลหรือโปรแกรมข้อมูลใน ROM ถ้าต้องการเปลี่ยนค่าข้อมูลภายใน ROM ควรแก้ไขในระดับที่สูงขึ้นมาจะง่ายกว่าการควบคุมสัญญาณภายใน จะเห็นว่าแต่ละระดับมีความเหมาะสมแตกต่างกันออกไปทำให้รูปแบบที่เราออกแบบไปง่ายต่อการแก้ไขโดยการใช้ประโยชน์ของ Abstuction

4.3.4 Information Hiding

เมื่อทำการเขียน VHDL Code ขึ้นมาเพื่ออธิบายการทำงานของฮาร์ดแวร์ตัวหนึ่ง บางครั้งอาจต้องการที่จะซ่อนรายละเอียดการพัฒนาโมดูล นั้นโดยไม่ต้องการให้ส่วนโมดูลอื่นๆ รู้การทำงานภายใน Information Hiding มีประโยชน์คือ ทำให้รูปแบบภาษา VHDL นั้นสามารถจัดการและอ่านเข้าใจได้ง่าย หลักการนี้จะสนับสนุนหลักการ Abstraction คือสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่ารูปแบบนั้นจะถูกสร้างขึ้นมาอย่างไร เป็นต้น การซ่อนรายละเอียดภายในโมดูลทำให้ความสนใจของผู้ออกแบบนั้นสนใจไปในส่วนที่สำคัญมากกว่า ในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ ดังรูปที่ 4.6



รูปที่ 4.6 การซ่อนรายละเอียดที่ไม่จำเป็นของระดับ NAND เกท

อธิบายการทำงานของฟลิปฟลอปไม่ต้องสนใจว่า NAND เกท จะทำงานอย่างไร จะต่อกันภายในอย่างไร โดย NAND เกท สามารถเขียนขึ้นมาแล้วคอมไพล์เก็บไว้ในไลบรารี ผู้ที่ออกแบบฟลิปฟลอประดับสูงขึ้นมา เพียงแต่ต้องรู้ว่า จะเชื่อมต่ออินพุท/เอาท์พุทของ NAND เกท มาใช้งานได้อย่างไร โดยไม่ต้องสนใจว่า NAND เกท จะถูกสร้างและพัฒนามาอย่างไร ประโยชน์อีกอย่างหนึ่งคือ ป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย VHDL โมเดลไปยังที่อื่นทำให้เราป้องกันทรัพย์สินทางปัญญาได้อีกในระดับหนึ่ง

4.3.5 Uniforminity

Uniforminity เป็นหลักการอีกอย่างหนึ่งที่ช่วยในการอธิบายฮาร์ดแวร์ด้วยภาษา VHDL หมายถึง การสร้างโมดูลของรหัส ในลักษณะคล้ายกัน โดยใช้ตัวภาษา VHDL Building Block ทำให้เกิดการเขียนรหัสที่ดีอย่างเช่น มีการใช้ย่อหน้า มีการใช้คำอธิบาย(Comment) เป็นต้น ทำให้การพัฒนาโมดูลทำความเข้าใจง่าย

4.4 องค์ประกอบพื้นฐานในวีเอชดีแอล (Basic concept in VHDL)

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบใน VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ(Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม(Architecture) ดังแสดงในรูปที่ 4.7 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำ ENTITY ตามด้วยชื่อขององค์ประกอบและคำ IS ภายในบรรยายถึงพอร์ตการติดต่อ อินพุท/เอาท์พุท พอร์ตขององค์ประกอบ ส่วนลักษณะกายภาพภายนอกอื่นๆ เช่น เวลา อุณหภูมิ ก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณ อินพุท/เอาท์พุท และพารามิเตอร์อื่นๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 4.7 การบรรยายหน้าที่ขององค์ประกอบจะเริ่มต้นหลังคำว่า BEGIN เป็นต้นไป

```
ENTITY component_name IS
        Input and output ports.
        Physical and other parameters.
END component_name;
```

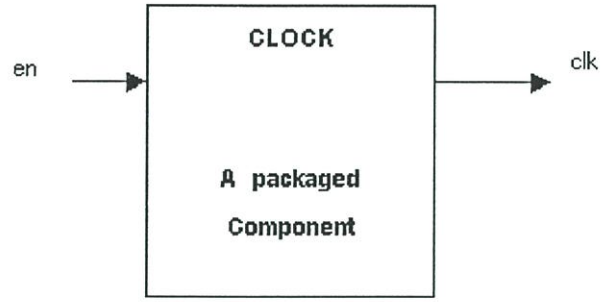
```
ARCHITECTURE identifier OF compionent_name IS
Declaratuons.
BEGIN
        Specification of functionality of the component
        In terms of its input lines and as influenced
        By physical and other parameters.
END identifier;
```

รูปที่ 4.7 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม

4.4.1 การกำหนดการเชื่อมต่อ (Interface Description)

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ต สำหรับการติดต่อกับองค์ประกอบภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 4.8 บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็นชื่อ clock component ตามด้วยคำว่า PORT และชื่อ

ของพอร์ตอยู่ในวงเล็บ IN และ OUT กำหนดโหมดของสัญญาณเป็นอินพุทหรือเอาท์พุท BIT แสดงชนิดของข้อมูล




```
ENTITY clock_component IS
        PORT (en: IN BIT; ck: OUT BIT);
END clock_component;
```

รูปที่ 4.8 แสดงบล็อคไดอะแกรมและการบรรยายการเชื่อมต่อของ clock component

4.4.2 การกำหนดรูปแบบการบรรยาย (Architecture Description)

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้การบรรยายสามารถกำหนดค่าของสัญญานเอาท์พุทในเทอมของ clock component ในรูปที่ 4.9 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมมี en เป็นอินพุทและมี clk เป็นเอาท์พุท PROCESS เป็นค่าเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม ภายในโพรเซสกำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น "0" ถ้าสัญญาน en มีค่าเป็น "1" ค่าของ periodic จะถูกคอมพลีเมนต์และส่งค่าให้กับ clk ซึ่งเป็นสัญญานเอาท์พุท คำสั่ง WAIT กำหนดให้สัญญานมีคาบเป็นเวลา 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock_component IS
BEGIN
        PROCESS
                VARIABLE periodic : BIT := " 0 ";
BEGIN
  IF en = " 1 " THEN
                periodic := NOT periodic;
  END IF;
                ck <= periodic;
                WAIT FOR 1US;
 END PROCESS;
END behavioral;
```

รูปที่ 4.9 แสดงการบรรยายเชิงพฤติกรรมของ Clock_component

4.4.3 โปรแกรมย่อย (Subprogram)

การใช้ฟังก์ชันและโพรซีเดอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆ ไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่น ถ้าเราให้ฟังก์ชั่นแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรตรรกะจริงๆ ในขณะที่เราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวนค่าหน่วงเวลา แล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

รูปที่ 4.10 แสดงการใช้ฟังก์ชั่นโพรซีเดอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิต เป็นค่าจำนวนเต็ม รูปที่ 4.11 แสดงการใช้ฟังก์ชันโดยกำหนดให้ X เป็นตัวแปรชนิดบิตแทนการกระทำในสมการบูลีน

```
TYPE byte IS ARRAY (7 DOWN TO  ) OF BIT
...
PROCEDURE byte_to_integer (ib: IN byte; oi: OUT INTEGER) IS
        VARIABLE result : INTERGER :=0;
BEGIN
        FOR I IN 0 TO 7 LOOP
                IF ib(I) = "1" THEN
                        Result := result+2**I;
                END IF;
END LOOP;
Oi := result;
END byte_to_interger;
```

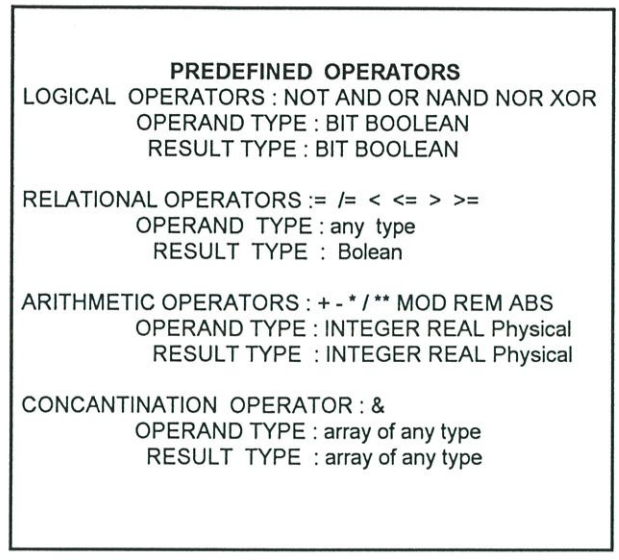
รูปที่ 4.10 แสดงการใช้โพรซีเดอร์

```
FUNCTION f ( a, b, c : BIT ) RETURN  BIT  IS
        VARIABLE  x : BIT;
BEGIN
        X := ((NOT a) AND (NOT b )  AND c);
        RETURN  x ;
END f;
```

รูปที่ 4.11 แสดงการใช้ฟังก์ชัน

4.4.4 โอเปอร์เรเตอร์ (VHDL OPERATORS)

การบรรยายเชิงพฤติกรรมใน VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกันกับภาษาซอฟแวร์ทั่วไปดังรูปที่ 4.12



รูปที่ 4.12 แสดงตัวกระทำใน VHDL

4.4.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)

ในวงจรอิเล็กทรอนิกส์ อุปกรณ์ทุกตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ(always active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วยเสมอในทุกๆ เหตุการณ์ที่เกิดขึ้น VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงาน ที่อยู่ภายในส่วนของสถาปัตยกรรมการบรรยายจะมีการทำงานที่พร้อมเพรียงกันเสมอหรือแม้แต่โพรเซสที่มีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายๆโพรเซสอยู่ภายในโครงการสร้างเดียวกันทุกๆ โพรเซสก็จะทำงานไปพร้อมๆกันด้วย

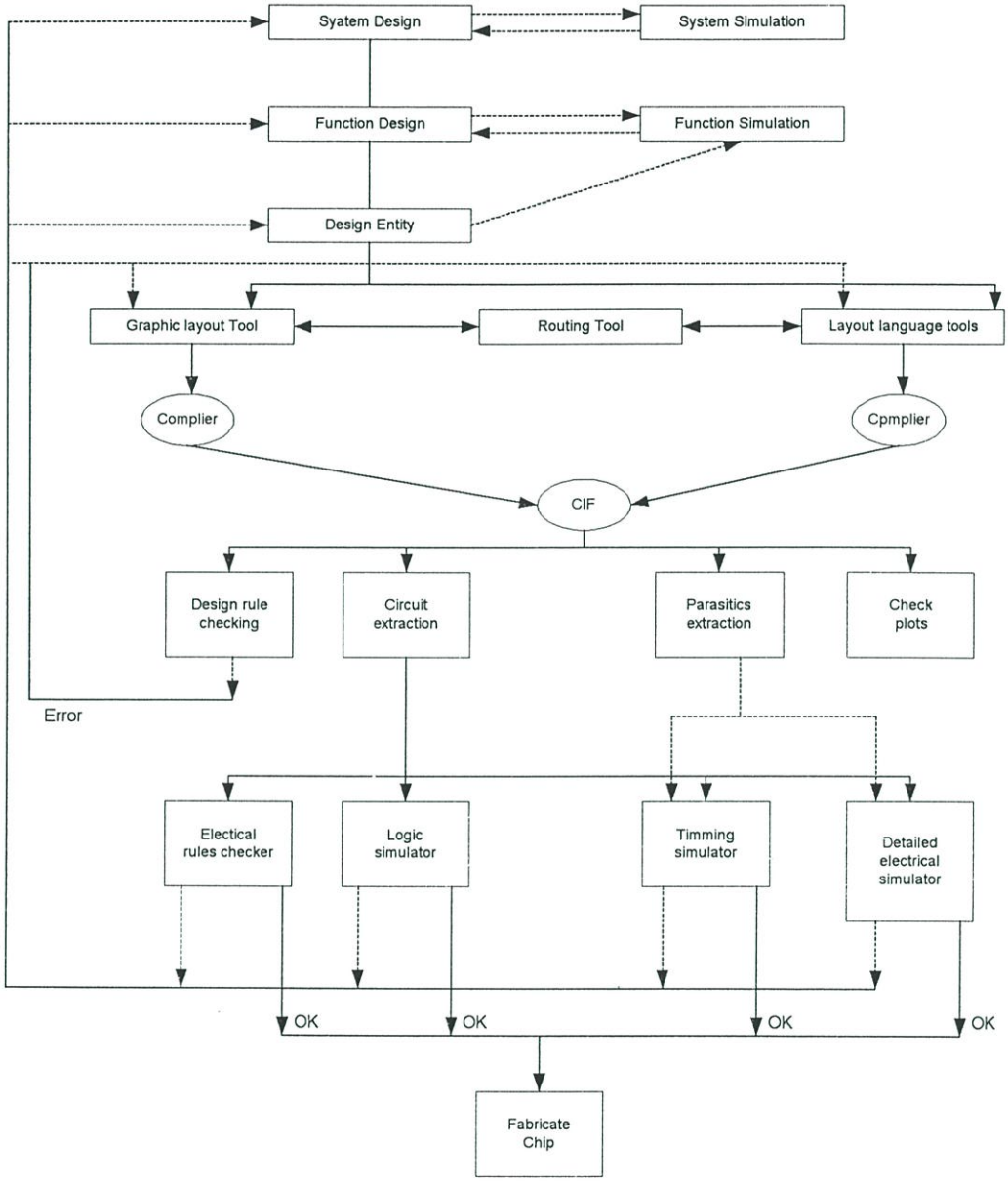
4.4.6 สัญญาณและตัวแปร (Signals and Variable)

สัญญาณที่เป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กันสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น W <= AFTER 12 ns หมายถึง กำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 ns.

4.5 โครงสร้างของวีเอชดีแอล

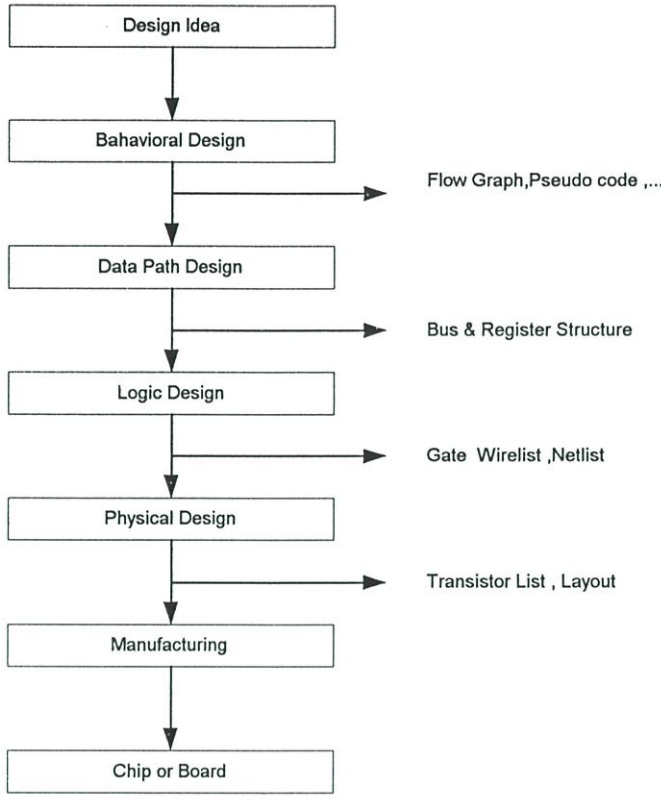
ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็น อุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้ จะต้องผ่านขั้นตอนต่างๆ มากมายและในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้น ทำการเพิ่มเติมตามความจำเป็นและ

เข้ากระบวนการออกแบบในขั้นต่อไป โครงสร้างการออกแบบ VHDL สามารถเขียนเป็นแผนผังได้ดังรูปที่ 4.13



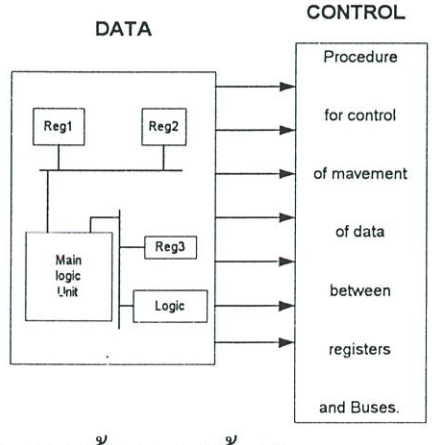
รูปที่ 4.13 รูปแสดงโครงสร้างการออกแบบ VHDL

รูปที่ 4.14 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วๆ ไป ซึ่งขั้นแรกผู้ออกแบบกำหนดแนวความคิดในการออกแบบเสียก่อนและทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นในขั้นตอนนี้จึงมีความจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจเป็นผังงาน ผังแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo Code)



รูปที่ 4.14 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะต้องกำหนดส่วนประกอบของรีจีสเตอร์ ผู้ออกแบบจะกำหนดส่วนของรีจีสเตอร์(Register) และวงจรตรรกะ (Logic) ที่จำเป็นทั้งหมดที่ประกอบกันเป็นระบบที่สมบูรณ์ แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง(Unidirectional or Bidirectional Bus) กระบวนการควบคุมในการเคลื่อนย้ายข้อมูลระหว่างรีจีสเตอร์และวงจรตรรกะจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ ดังรูปที่ 4.15



รูปที่ 4.15 แสดงการออกแบบระบบเส้นทางของข้อมูล

การออกแบบวงจรตรรกะจะเป็นขั้นตอนต่อไป การออกแบบในขั้นนี้เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิปฟลอปเป็นส่วนของอุปกรณ์แยกต่างๆ ได้แก่ รีจีสเตอร์เก็บข้อมูลจงจรตรรกะ และส่วนควบคุมฮาร์ดแวร์ ซึ่งในขั้นสุดท้ายจะได้ออกมาเป็นเครือข่ายของการโยงใยระหว่างเกทและ ฟลิปฟลอปนั่นเอง

ต่อมาเป็นขั้นตอนของการเปลี่ยนเครือข่ายการโยงใยในขั้นตอนที่แล้วให้เป็นทรานซิสเตอร์และเลย์เอาท์(Transistor list and layout) ในขั้นตอนนี้เกี่ยวข้องกับการจัดวางเกทและฟลิปฟลอปแทนด้วยทรานซิสเตอร์หรือไลบรารีเซล ขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

บทที่ 5
เทคนิคที่ใช้ในการบีบอัดข้อมูล

การบีบอัดข้อมูลเป็นวิธีการลดขนาดของข้อมูล โดยข้อมูลอาจอยู่ในรูปของตัวอักษร,ข้อมูลเสียง,ข้อมูลภาพ การบีบอัดข้อมูลมีความสำคัญต่อหลายๆสาขา เช่น การส่งสัญญาณโทรทัศน์แบบดิจิตอล ถ้าต้องการส่งสัญญาณ HDTV (High Definition TeleVision) โดยไม่ทำการบีบอัดข้อมูล จะต้องทำการส่งข้อมูลด้วยความเร็ว 884 เมกะบิต/วินาที แต่ถ้ามีการบีบอัดข้อมูลก่อนส่งสามารถส่งได้น้อยกว่า 20 เมกะบิต/วินาที ข้อมูลเสียงก็มีความจำเป็นที่ต้องมีการบีบอัดข้อมูลเช่นเดียวกัน ในระบบสื่อสารข้อมูลทางคอมพิวเตอร์ หลายประเภทเช่น โมเด็ม,การเก็บข้อมูลในหน่วยความจำสำรอง ก็จำเป็นต้องใช้การบีบอัดข้อมูลมาช่วยลดขนาดของข้อมูล ก่อนทำการส่งหรือจัดเก็บข้อมูลเช่นเดียวกัน บทนี้จะแสดงให้เห็นรายละเอียด ในการบีบอัดข้อมูล เช่น เทคนิคที่ใช้ ,วิธีการหรือ อัลกอลิทึมที่ใช้ในการบีบอัดข้อมูล, การค้นหาเทคนิคการบีบขนาดที่เหมาะสมมากที่สุดสำหรับข้อมูลทางกายภาพ เป็นต้น

5.1 การประยุกต์ใช้งานในการบีบอัดข้อมูล(Application of data compression)

ตัวอย่างแรกของการบีบขนาดข้อมูลคือ รหัส Morse, พัฒนาโดย Samuel Morse ในกลางศตวรรษที่ 19 จดหมายที่ส่งทางโทรเลขถูกเข้ารหัสด้วยจุด(.) และขีด(-) Morse สังเกตุเห็นว่าจดหมายจะมีการใช้ตัวอักษรซ้ำ ๆ บ่อยให้ใช้สัญลักษณ์สั้นเพื่อที่จะลดเวลาเฉลี่ยในการส่งข้อความ เช่น a (. -) และe (.) และ ถ้าตัวอักษรไม่ค่อยได้ใช้ เช่น q (- - . -) หรือ j (. - - -) จะใช้รหัสที่ยาวกว่าเป็นต้น จะเห็นได้ว่าสิ่งที่ใช้เป็นเครื่องมือในการบีบอัดข้อมูลใน รหัส Morse คือโครงสร้างสถิติของข้อความนั่นเอง ในปัจจุบันมีการใช้การบีบขนาดข้อมูลในการสื่อสารและการจัดเก็บมากมาย เช่น การส่งข้อมูลภาพในระบบอินเตอร์เน็ต , การส่งข้อมูลภาพถ่ายในระยะทางไกลในระบบดาวเทียม รวมทั้งงานประยุกต์ต่าง ๆ เกี่ยวกับการประมวลผลทางไกลโดยเชื่อมต่อกันผ่านระบบการสื่อสาร ในการลดความต้องการเกี่ยวกับการเก็บข้อมูลหรือค่าใช้จ่ายในการสื่อสารจำเป็นต้องลดความซ้ำซ้อน(redundancy),การจัดเก็บข้อมูลภาพหรือเสียงเพลงในซีดีรอม

นอกจากนี้การบีบอัดข้อมูล จะทำให้ความลับของข้อมูล(data security) ดีขึ้น เพราะข้อมูลที่ถูกบีบอัดจะไม่สามารถอ่านรู้เรื่อง

5.2 เทคนิคในการบีบอัดข้อมูล (Compression techniques)

เทคนิคในการบีบอัดข้อมูลหรืออัลกอลิทึมที่ใช้ในการบีบอัดข้อมูล จะประกอบด้วย 2 ขั้นตอนได้แก่ ขั้นตอนแรกได้แก่ การนำอินพุท X มาทำการบีบอัดข้อมูลและแทนด้วย X_C ซึ่งจะมี

จำนวนบิทที่น้อยกว่าอินพุท X ขั้นตอนที่สอง นำค่าที่ได้ X_c มาทำการสร้างข้อมูล(reconstruction) ให้ได้ข้อมูลเดิมหรือใกล้เคียงกับข้อมูลเดิม Y ดังแสดงในรูปที่ 2.1 โดยหลักการในการบีบอัดข้อมูลแบ่งออกเป็น 2 แบบ ได้แก่

-การบีบอัดข้อมูลแบบไม่สูญเสีย (Lossless) ในกรณีนี้ Y จะมีค่าเหมือน X ทุกประการ

-การบีบอัดข้อมูลแบบสูญเสีย(Lossy) ซึ่งวิธีนี้จะมีเทคนิคในการบีบอัดข้อมูลสูงกว่าแบบแรก แต่ค่า Y ที่ได้จะมีความแตกต่างจากค่าของ X

รูปที่ 5.1 กระบวนการบีบอัดข้อมูล

5.2.1 การบีบอัดข้อมูลแบบไม่สูญเสีย (Lossless compression)

เทคนิคในการบีบอัดข้อมูลในวิธีนี้จะไม่มีการสูญเสียของข้อมูลเกิดขึ้น ถ้ามีการบีบอัดข้อมูลเกิดขึ้น ข้อมูลต้นฉบับจะสามารถได้คืนมาเหมือนต้นฉบับร้อยเปอร์เซ็น วิธีนี้จะใช้ในการบีบอัดข้อมูลหรือสัญญาณที่ไม่ต่อเนื่อง (discrete signal) เช่น ข้อความ, ข้อมูลที่คอมพิวเตอร์สร้างขึ้น

5.2.2 การบีบอัดข้อมูลแบบสูญเสีย (Lossy compression)

เทคนิคการบีบอัดข้อมูลแบบนี้ ข้อมูลบางส่วนจะมีการสูญเสีย ดังนั้นในการคืนสภาพข้อมูลจะไม่สามารถได้ข้อมูลครบถ้วนอย่างต้นฉบับอย่างแน่นอนและวิธีนี้จะมีอัตราส่วนในการบีบอัดสูงกว่า การบีบอัด lossless วิธีนี้จะใช้ในการบีบอัดข้อมูลหรือสัญญาณที่ต่อเนื่อง (Continuouse signal) เช่น ข้อมูลเสียง, ข้อมูลรูปภาพ

5.2.3วิธีการวัดประสิทธิภาพ (Measures of performance)

อัลกอลิทึมในการบีบอัดข้อมูลมีแตกต่างหลายวิธีแต่ละวิธีก็มีข้อดีและไม่ดีแตกต่างกันไป สิ่งที่จะบอกว่าดีหรือไม่นั้นอาจจะดูจากประสิทธิภาพของอัลกอลิทึม ที่พอจะสามารถวัดได้ เช่น ความสามารถในการนำไปใช้งานจริง, ความเร็วของอัลกอลิทึมในการประมวลผลเมื่อนำไปสร้างฮาร์ดแวร์หรือซอฟแวร์, ความสามารถในการคืนสภาพข้อมูลให้เหมือนต้นฉบับเดิมเป็นต้น มีหลายๆ วิธีการที่จะวัดความสามารถของอัลกอลิทึมที่ใช้ในการบีบอัดข้อมูล เช่น อัตราส่วนจำนวนบิตก่อนทำการบีบอัดเมื่อเทียบกับจำนวนบิตหลังการบีบอัดที่เรียกว่า “ อัตราส่วนในการบีบอัดข้อมูล (Compression ratio)“ ตัวอย่าง เช่น การที่จัดเก็บข้อมูลภาพที่อยู่ในรูปของอาร์เรย์ ขนาด 256x 256 8 บิตต่อพิกเซล (pixel) ซึ่งต้องใช้พื้นที่ในการจัดเก็บ 64 กิโลไบต์ ถ้าทำการบีบอัดข้อมูลภาพ

นี้และมีขนาด 16 กิโลไบต์แสดงว่าประหยัดพื้นที่ในการจัดเก็บถึง 48 กิโลไบต์ และอัตราส่วนในการบีบอัดข้อมูลเท่ากับ 4

วิธีการอื่นที่จะวัดประสิทธิภาพในการบีบอัดข้อมูล ก็คือค่าเฉลี่ยของจำนวนบิตที่ใช้ต่อหนึ่งตัวอย่าง ซึ่งโดยทั่วไปก็หมายถึงอัตราส่วน ตัวอย่างเช่น ข้อมูลภาพที่กล่าวมาแล้วข้างต้น ค่าเฉลี่ยของจำนวนบิตต่อพิกเซลในการบีบอัดข้อมูลแทนด้วย 2 ดังนั้น อัตราส่วนก็คือ 2 บิตต่อพิกเซล

ในการบีบอัดข้อมูลแบบสูญเสียการคืนสภาพข้อมูลจะแตกต่างจากข้อมูลต้นฉบับเช่นนี้ สิ่งที่จะชี้วัดประสิทธิภาพของการบีบอัดข้อมูลก็คือปริมาณความแตกต่างของข้อมูลต้นฉบับและข้อมูลที่ทำการขยายกลับ ซึ่งนิยมเรียกว่า "การสูญเสีย(distortion) " ค่านี้สามารถคำนวนเป็นตัวเลขหรือเปอร์เซ็นความแตกต่างระหว่างข้อมูลก่อนการบีบอัดและหลังการบีบอัดได้

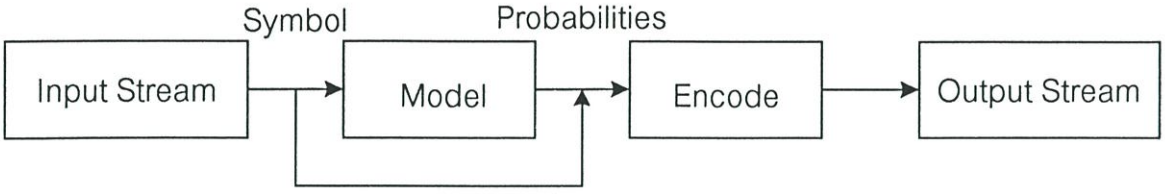
5.2.4 โมเดลและรหัส (Modelling and Codeing)

โดยทั่วไปการบีบอัดข้อมูลประกอบด้วย การอ่านสายข้อมูลเข้ามาแล้วทำการแปลงข้อมูลเหล่านี้ให้อยู่ในรูปแบบโค้ดต่างๆ ถ้าหากการบีบอัดข้อมูลมีประสิทธิผล โค้ดที่ได้ต้องมีขนาดสั้นกว่าข้อมูลเดิม โดยโค้ดที่ได้จะเป็นอย่างไรนั้น ก็ขึ้นกับโมเดล (Model) ที่ใช้ซึ่งโมเดลคือ กลุ่มของข้อมูลที่เลือกไว้และกฎที่ใช้ในการประมวลผลข้อมูลเหล่านั้นเพื่อที่จะหาโค้ดที่สามารถแทนข้อมูลเหล่านั้นได้ โปรแกรมที่จะใช้โมเดลในการคำนวณหาความน่าจะเป็นของการซ้ำซ้อนของข้อมูลแต่ละตัว เพื่อทำการสร้างโค้ดสำหรับข้อมูลนั้นๆ ในด้านการบีบอัดข้อมูลโดยส่วนใหญ่เรามักจะใช้คำว่า Coding เพื่อหมายถึงขบวนการบีบอัดข้อมูลทั้งขบวนการ แทนที่จะหมายถึงส่วนหนึ่งของขบวนการ เช่น เราอาจจะได้ยินคำว่า Huffman Coding จะถูกใช้ในการอธิบายเทคนิคการบีบอัดข้อมูล แต่ในความเป็นจริงมันเป็นเพียงวิธีการเข้ารหัสข้อมูลที่ถูกนำมาใช้ร่วมกับโมเดลในการบีบอัดข้อมูลเท่านั้น ดังนั้นจึงสามารถนิยามการบีบอัดข้อมูลได้ดังนี้ในรูปที่ 2.2

Data Compression = Modeling + Coding

รูปที่ 5.2 นิยามการบีบอัดข้อมูล

ถ้าเราดูตัวอย่างจากอัลกอลิทึมของ Huffman Coding ขบวนการบีบอัดข้อมูลสามารถแบ่งออกได้เป็นขั้นตอนย่อย ๆ ได้ดังรูปที่ 5.3 ต่อไปนี้



รูปที่ 5.3 ตัวอย่างอัลกอริทึม

Coding

ในการบีบอัดข้อมูลเราต้องการสร้างโค้ดสำหรับแต่ละตัวอักษรโดยใช้จำนวนบิตตามความเป็นจริง เช่น ตัวอักษร e ใช้ 4 บิต หรือตัวอักษร a ใช้ 6 บิต เป็นต้น แต่ในความเป็นจริงถ้าหากเราเข้ารหัสตัวอักษรโดยใช้ ASCII เราก็ไม่สามารถที่จะเข้ารหัสทุก ๆ ตัวอักษรได้อย่างกะทัดรัดที่สุด ซึ่งในการแก้ไขปัญหาเหล่านี้จึงเป็นที่นิยมในการใช้เทคนิค Shannon-Fano Coding และ Huffman Coding ในการสร้างโค้ดที่มีความยาวแปรเปลี่ยนได้ โดยที่จำนวนบิตของโค้ดแต่ละตัวก็ขึ้นกับความน่าจะเป็นของการซ้ำซ้อนของข้อมูลในกลุ่มนั้น

Modeling

ถ้าหากมองการบีบอัดข้อมูลที่เป็นรถจักรยานยนต์ Coding ก็จะเป็นส่วนวงล้อและ Modeling ก็จะเป็นส่วนเครื่องยนต์ ดั้งนั้นถ้าหากเราเลือกโมเดลที่ไม่ดีพอ ในการป้อนข้อมูลให้กับส่วนเข้ารหัส อาจจะทำให้การบีบอัดข้อมูลมีประสิทธิภาพที่ต่ำก็ได้ เช่น การบีบอัดข้อมูลในตระกูล Lossless Data Compression โดยทั่วไปจะมีการเลือกใช้เพียง 2 โมเดลคือ Statistical และ Dictionary-Based โดยที่โมเดล Statistical จะอ่านข้อมูลและทำการเข้ารหัสแต่ละตัวอักษร ส่วนโมเดล Dictionary-Based จะใช้โค้ดตัวหนึ่งที่มีในคำศัพท์ (Dictionary) ในการแทนที่สตริงชุดหนึ่ง ๆ โดยถ้าหากพบสตริงดังกล่าวผลลัพธ์ที่ได้ก็จะเป็นค่าอินเด็กซ์หรือค่าพอยเตอร์สำหรับตำแหน่งที่พบในคำศัพท์แทนที่จะเป็นโค้ดสำหรับสตริงนั้น ๆ

5.3 เทคนิคการบีบอัดข้อมูลแบบไม่สูญเสีย(Lossless Compression techniques)

ในหัวข้อนี้จะอธิบายรายละเอียดของเทคนิคการบีบอัดข้อมูลแบบสูญเสีย ซึ่งอัลกอริทึมที่นิยมนำมาใช้ในการบีบอัดได้แก่ Huffman Coding, run length encoding, dictionary techniques โดยในหัวข้อนี้จะอธิบายเฉพาะส่วนของ Huffman Coding

5.3.1 การเข้ารหัสฮัฟแมน(Huffman Coding)

เป็นวิธีการเข้ารหัสข้อมูลที่มีประสิทธิภาพวิธีหนึ่ง ซึ่งถูกนำเสนอโดย D.A. Huffman และถูกตีพิมพ์ครั้งแรกในปี ค.ศ. 1952 ในบทความ "A Method for the Construction of Minimum Redundancy Codes" ซึ่งคุณสมบัติหลายอย่างที่เหมือนกับ Shannon-Fano coding คือการสร้างโค้ดที่มีความยาว(จำนวนบิต)ที่แตกต่างกันได้ ข้อมูลที่มีความน่าจะเป็นสูงกว่าก็จะถูกเข้ารหัสด้วยโค้ดที่มีจำนวนบิตน้อยกว่า และที่สำคัญข้อมูลที่ได้จะมีแอตทริบิวต์ทำหน้าที่ไม่เหมือนกัน ดังนั้นโค้ดที่ได้จึงสามารถถูกถอดรหัสได้อย่างถูกต้อง ขั้นตอนการถอดรหัสสายโค้ดโดยวิธีนี้ส่วนใหญ่มักกระทำโดยใช้แผนภูมิต้นไม้(Binary Decoder Tree)

การสร้างแผนภูมิต้นไม้โดยวิธี Huffman นั้นจะสร้างจากล่างขึ้นบน ดังนั้นโค้ดทุกตัวจะเริ่มถูกสร้างจากกิ่งก้าน(left node) และ ไปสิ้นสุดที่ราก (root node) โดยมมีขั้นตอนในการสร้างโดยสังเขปดังนี้

1.) ทำการนับค่าความน่าจะเป็นของข้อมูลทุกตัวและเรียงลำดับข้อมูลตามค่าความน่าจะเป็นจากน้อยไปหามาก
2.) โหนด(node) จะถูกสร้างทีละคู่จากคู่ความน่าจะเป็นน้อยไปยังคู่ความน่าจะเป็นมาก แต่ถ้าหากข้อมูลตัวสุดท้ายไม่สามารถจัดเข้าคู่ได้ก็จะแยกเป็นอิสระที่มีกิ่งเดียว
3.) สร้างโหนดแม่(Parent Node)ของโหนดคู่ที่ได้จากขั้นตอนแรกและจะมีค่าน้ำหนักเท่ากับผลรวมของค่าความน่าจะเป็นของโหนดลูก(Child Node)ทั้งสอง
4.) โหนดแม่ที่ได้ก็จะถูกเพิ่มเข้ามาเป็นโหนดอิสระและโหนดลูกก็จะถูกยกเลิก
5.) โหนดลูกโหนดหนึ่งจะถูกกำหนดให้เป็นไนารี '0' ส่วนอีกโหนดจะถูกกำหนดให้เป็น '1'
6.) ให้ทำซ้ำขั้นตอนทั้งหมดจนเหลือโหนดอิสระเพียงโหนดเดียวซึ่งก็คือโหนดราก

ตัวอย่าง สมมุติมีข้อมูล 5 ข้อมูลได้แก่ a_1,a_2,a_3,a_4 และ a_5 และมีค่าความน่าจะเป็นดังนี้ $P(a_1)=0.2$, $P(a_2)=0.4$, $P(a_3)=0.2$, $P(a_4)=0.1$, $P(a_5)=0.1$ สิ่งแรกที่ต้องทำคือการเข้ารหัส; $c(a_I)$ ของข้อมูล a_I โดยเรียงข้อมูลจากค่าความน่าจะเป็นมากไปหาน้อย

ตารางที่ 5.1 ตัวอย่างและอัตราการเกิดของข้อมูล

Data	Probability	Code
a_2	0.4	$c(a_2)$
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$
a_4	0.1	$c(a_4)$
a_5	0.1	$c(a_5)$

ค่าความน่าจะเป็นน้อยที่สุด 2 ค่าจะถูกสร้างรหัสก่อนดังนี้

$$c(a_4) = \propto_1 * 0 \tag{5.1}$$

$$c(a_5) = \propto_1 * 1 \tag{5.2}$$

โดย $\propto_1$ เป็น generic binary string

* แทน concatenation ระหว่าง 2 ข้อมูล

ให้ a_4' เป็นข้อมูลที่เกิดจากความสัมพันธ์ $P(a_4') = P(a_4) + P(a_5) = 0.2$ ดังแสดงข้อมูลในตารางที่ 5.2

ตารางที่ 5.2 สร้างข้อมูล a_4'

Data	Probability	Code
a_2	0.4	$c(a_2)$
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$
a_4'	0.2	α_1

ในตารางนี้ค่าความน่าจะเป็นต่ำ 2 ค่าคือ a_3 และ a_4' จึงสามารถสร้างรหัสได้เป็น

$$c(a_3) = \propto_2 * 0 \tag{5.3}$$

$$c(a_4') = \propto_2 * 1 \tag{5.4}$$

กำหนดให้ $c(a_4') = \propto_1$ จากตารางที่ 5.2 และสมการที่ (5.4) แสดงว่า $\propto_1 = \propto_2 * 1$ ดังนั้นสมการที่ (5.1) และ สมการที่ (5.2) จะเขียนได้เป็น

$$c(a_4) = \propto_2 * 10 \tag{5.5}$$

$$c(a_5) = \propto_2 * 11 \tag{5.6}$$

จากตารางที่ 5.2 ค่าที่จะเข้ามาสร้างรหัสต่อไปคือ a_3 และ a_4' เพราะเป็นคู่ที่มีอัตราการเกิดน้อยที่สุด กำหนดให้ a_3' เป็นข้อมูลที่เกิดจากความสัมพันธ์ $P(a_3') = P(a_3) + P(a_4') = 0.4$ ดังแสดงข้อมูลในตารางที่ 5.3

ตารางที่ 5.3 สร้างข้อมูล a_3'

Data	Probability	Code
a_2	0.4	$c(a_2)$
a_3'	0.4	α_2
a_1	0.2	$c(a_1)$

จากตารางที่ 5.3 ค่าที่จะเข้ามาสร้างรหัสต่อไปคือ a_3' และ a_1 จึงสามารถสร้างรหัสได้เป็น

$$c(a_3') = \propto_3 * 0 \tag{5.7}$$

$$c(a\,1) = \propto_3 * 1 \tag{5.8}$$

การกำหนด $c(a_3') = \propto_2$ จากตารางที่ 5.3 ดังนั้นในสมการที่ (5.7) สามารถเขียนใหม่ได้เป็น $\propto_2 = \propto_3 * 0$ จึงทำให้สมการที่ (5.3),(5.5) และ (5.6) เข้ารหัสได้เป็น

$$c(a_3) = \propto 3 * 00 \tag{5.9}$$

$$c(a_4) = \propto_3 * 010 \tag{5.10}$$

$$c(a_5) = \alpha_3 * 011 \tag{5.11}$$

กำหนดให้ $a_3^{/\prime}$ เป็นข้อมูลที่เกิดจากความสัมพันธ์ $P(a_3^{//}) = P(a_3^{/\prime}) + P(a_1) = 0.6$ ดังแสดงข้อมูลในตารางที่ 5.4

ตารางที่ 5.4 สร้างข้อมูล $a_3^{//}$

Data	Probability	Code
$a^{//}_3$	0.6	α_3
a_2	0.4	$c(a_2)$

ในขั้นตอนสุดท้ายจะเหลือข้อมูลให้สร้างรหัส 2 ตัว และจะได้

$$c(a_3^{//}) = 0 \tag{5.12}$$

$$c(a_2) = 1 \tag{5.13}$$

ดังนั้นจะได้ Huffman Code ดังตารางที่ 5.5

ตารางที่ 5.5 กำหนดรหัสฮัฟแมน

Data	Probability	Code
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011

5.3 เทคนิคการบีบอัดข้อมูลแบบสูญเสีย

ในการบีบอัดข้อมูลแบบสูญเสีย มีหลายวิธี ได้แก่ Wavelet transform , Subband Coding ,Transform Coding โดยในหัวข้อนี้จะอธิบายเฉพาะ Transform Coding

Transform Codig เป็นวิธีที่ใช้ในการเปลี่ยนรูปแบบของข้อมูลจากโดเมนหนึ่งไปสู่ข้อมูลอีกโดเมนหนึ่ง เช่น เปลี่ยนจากโดเมนของเวลา(time domain)เป็นโดเมนของความถี่(frequency domain) ซึ่งในการ ทรานสฟอร์มจะขึ้นอยู่กับรูปแบบของข้อมูลและส่วนที่ไม่จำเป็นหรือส่วนที่ซ้ำซ้อน(redundancy)ของข้อมูลอินพุทที่จะนำมาบีบอัดด้วย โดย อัลกอริทึมจะมี 3 ขั้นตอน

1.) การทรานสฟอร์ม(transform) ถ้ามีข้อมูลอินพุท $\{s_n\}$ และจะทำการแบ่งข้อมูลอินพุทออกเป็นส่วนย่อย ๆ ที่เรียกว่า บล็อก(block) จำนวน N บล็อก แล้วในแต่ละบล็อก ก็ถูกทรานสฟอร์มที่เป็นอิสระต่อกัน และจะได้ข้อมูล $\{c_n\}$

2.) การควอนไตเซชั่น(quantization) จะนำผลที่ได้จากการทรานสฟอร์ม $\{c_n\}$ โดยการนำตัวอย่างข้อมูลมาทำการจัดระดับเพื่อกำหนดแทนด้วยเลขไบนารี่

3.) การสร้างรหัส(coding) นำข้อมูลที่ได้จากขั้นตอนการควอนไตเซชั่น มาทำการเข้ารหัส เช่น Huffman Coding

หลักการนี้สามารถอธิบายในทางคณิตศาสตร์ได้ โดยกำหนดข้อมูลอินพุท $\{s_n\}$ และทำการแบ่งข้อมูลอินพุทออกเป็นบล็อกย่อย ๆ จำนวน N บล็อก และ จะถูกทรานสฟอร์มกลับด้วยเมทริกซ์ A โดยเขียนแทนด้วยรูปแบบของเมทริกซ์ คือ

$$c = As \tag{5.14}$$

หรืออยู่ในรูปสมการทางคณิตศาสตร์ ดังสมการที่ (5.15)

$$c_n = \sum_{i=0}^{N-1} s_i a_{n,i} \tag{5.15}$$

โดย A เป็นเมทริกซ์ $[A]_{i,j} = a_{i,j}$

ส่วนในขั้นตอนการควอนไตเซชั่นและการสร้างรหัสไบนารี่ โดยการนำเซ็ต $\{c_n\}$ ซึ่งจะทำให้การบีบอัดข้อมูลนั้นเหมาะสมที่สุด

ในส่วนของขั้นตอน การขยายข้อมูลกลับ (decompression) จะทำได้โดยการนำอินเวอร์สทรานสฟอร์มของเมทริกซ์ A มาคูณ เซ็ต $\{c_n\}$ เพื่อให้ได้ เซ็ต $\{s_n\}$ กลับคืนมา ให้ $B = A^{-1}$

$$s = Bc \tag{5.16}$$

หรืออยู่ในรูปสมการทางคณิตศาสตร์ ดังสมการที่ (5.17)

$$s_n = \sum_{i=0}^{N-1} s_i b_{n,i} \tag{5.17}$$

โดย B เป็นเมทริกซ์ $[B]_{i,j} = b_{i,j}$

หลักการในการทรานสฟอร์มนี้สามารถใช้ได้กับข้อมูลอินพุทแบบ 2 มิติ (bi-dimension) เช่น ข้อมูลภาพ (image data) ได้ สมมุติมีข้อมูภาพ (digital images) S ที่มีขนาด $N \times N$ โดยสมาชิกในแต่ละสมาชิกแทนด้วย $S_{i,j}$ โดยที่ (i,j)เป็นตำแหน่งของแต่ละพิกเซล(pixel) ของข้อมูลภาพ จะนำเมทริกซ์ A ที่มีขนาด $N \times N$ โดยที่ เมทริกซ์ A จะต้องมีคุณสมบัติในการหาอินเวอร์สได้มาทำการคูณ ซึ่ง $a_{i,j}$ เป็นสมาชิกของทรานสฟอร์มของเมทริกซ์ A และ $C_{i,j}$ โดยที่ (i,j)เป็นตำแหน่งของแต่ละพิกเซลที่ได้จากการบีบอัดข้อมูลภาพแล้ว ดังสมการ

$$C_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} s_{i,j} a_{i,j} a_{k,l} \tag{5.18}$$

ในการทรานสฟอร์มจะขึ้นกับการแบ่งขนาดของบล็อก และยังสามารถแบ่งทำการทรานสฟอร์ม 2 มิติ(2D) ออกเป็นการทรานสฟอร์มครั้งละ 1 มิติ (1D) ได้ โดยทำการทรานสฟอร์ม 1 มิติไปตามแถวจำนวน N แถวหลังจากนั้นจึงทำการทรานสฟอร์ม 1 มิติไปตามคอลัมภ์จำนวน N คอลัมภ์ จากสมการ (5.18) จึงเขียนแทนได้ด้วยสมการ (5.19)

$$C_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} s_{i,j} a_{k,i} a_{l,j} \tag{5.19}$$

จากสมการ (5.18) และ (5.19) เขียนแทนด้วยสมการ (5.20),(5.21) และ (5.22)

$$C = ASA^T \tag{5.20}$$

และ อินเวอร์สทรานสฟอร์มเขียนแทนด้วยสมการ (5.21)

$$S = BCB^T \tag{5.21}$$

โดยที่ $B = A^{-1} = A^{T}$ ดังนั้นสมการ (5.21) สามารถเขียนแทนด้วยสมการ (5.22)

$$S = A^{T}CA \tag{5.22}$$

สำหรับตัวอย่างการประยุกต์ Transfom Coding ที่สำคัญคือ การบีบอัดข้อมูลภาพตามมาตรฐานของ JPEG โดยใช้ Discrete Cosine Transform (DCT) เป็นการทรานสฟอร์มที่มีความสำคัญต่อการบีบอัดข้อมูลภาพที่ได้รับความนิยมอย่างแพร่หลาย ทั้งนี้เพราะค่าสัมประสิทธิ์ในโดเมนของความถี่ ที่ได้จะเป็นเทอมของค่าจริง (real time) เท่านั้น อีกทั้งสามารถคำนวณได้อย่างรวดเร็ว (fast algorithms) และยังสามารถใช้งานจริงในลักษณะ real time โดยใช้ ฮาร์ดแวร์ ได้ไม่ยาก ในปัจจุบันหลักการของ DCT ยังคงมีการวิจัยกันอยู่ต่อไปเพื่อให้สามารถลดขนาดของข้อมูลให้ได้มากที่สุด พร้อมทั้งหาวิธีที่จะลดขนาดของวงจร และเพิ่มความเร็วในการคำนวณให้มากขึ้น

ระบบการบีบอัดข้อมูลโดยใช้เทคนิค DCT ภาพอินพุทที่เข้ามาจะถูกแยกออกเป็นบล็อกเล็ก ๆ โดยเราสามารถกำหนดขนาดของบล็อกได้ว่าเป็นขนาดเท่าไร ขนาดของบล็อกที่เหมาะสมจะเป็นตัวเพิ่มประสิทธิภาพในการรวมพลังงานของการทรานสฟอร์มซึ่งจะทำให้ภาพที่ได้มีรายละเอียดที่ดี ความเหมาะสมของขนาดของบล็อกค่าหนึ่ง ๆ จะเหมาะสมที่ค่าบิตเรท (bit rate) ค่าหนึ่ง ๆ แต่อย่างไรก็ตามขนาดของบล็อกที่เหมาะสมสามารถคำนวณได้ด้วยคณิตศาสตร์ที่ยุ่งยากพอ ๆ กับการทรานสฟอร์มโดยส่วนมากแล้วขนาดของบล็อกจะมีค่าเป็นเลขยกกำลังของเลข 2 เช่น 4, 8, 16, หลังจากการแยกข้อมูลออกเป็นบล็อกย่อย ๆ แล้ว จะทำการทรานสฟอร์มไปโดยอิสระของแต่ละบล็อก โดยมีสมการของการทรานสฟอร์มแบบ 2 มิติ

$$[A]_{i,j} = \omega(i)\cos\left(\frac{(2j+1)i\pi}{2N}\right) \quad j = 0,1,\ldots,N-1 \tag{5.23}$$

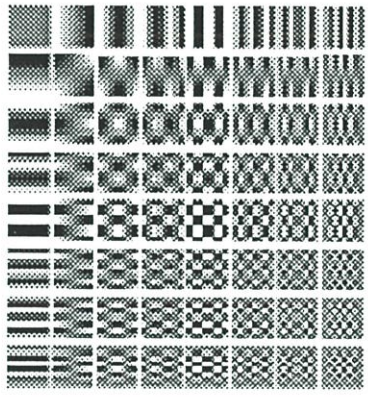
$$\omega(i) = \begin{cases} \sqrt{\frac{1}{N}} & i = 0 \\ \sqrt{\frac{2}{N}} & i = 1,\ldots,N-1 \end{cases}$$

สำหรับการบีบอัดข้อมูลตามมาตรฐาน JPEG ค่า N = 8 ดังนั้นสมการของการทรานสฟอร์มจะสามารถแสดงได้ดังสมการ (5.24)

$$[A]_{i,j} = \omega(i)\cos\left(\frac{(2j+1)i\pi}{16}\right) \quad j = 0,1,\ldots,7 \tag{5.24}$$

$$\omega(i) = \begin{cases} \sqrt{\frac{1}{8}} & i = 0 \\ \sqrt{\frac{1}{4}} & i = 1,\ldots,7 \end{cases}$$

ในรูปที่ 5.4 เป็นรูปภาพที่แสดงขนาดของบล็อก 8x8 ที่ได้จากการทรานสฟอร์มตามสมการที่ (5.24) หลังจากนั้นจะนำไปทำการควอนไตเซชั่นและการเข้ารหัสเพื่อกำหนดเป็นรหัสเลขฐานสอง ดังตารางที่ 5.6 ข้อสังเกต ค่าของสัมประสิทธิ์จะมากที่สุดที่มุมบนด้านซ้ายและจะลดลงมาทั้งในด้านของแถวและคอลัมภ์



รูปที่ 5.4 ภาพขนาด 8x8

ตารางที่ 5.6 การกำหนดบิตให้กับภาพในรูปที่ 5.4

8	7	5	3	1	1	0	0
7	5	3	2	1	0	0	0
4	3	2	1	1	0	0	0
3	3	2	1	1	0	0	0
2	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

ตัวอย่าง ถ้ามีข้อมูลภาพขนาดบล็อกละ 8x8 ดังแสดงในตารางที่ 5.7

ตารางที่ 5.7 ภาพขนาด 8x8

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156

ค่าของแต่ละพิกเซลในบล็อกจะถูกแปลงเปลี่ยนตามค่าของ 2^{p-1} ซึ่ง p หมายถึงจำนวนบิตต่อพิกเซล ในกรณีนี้ p = 8 ทำให้ค่าในแต่ละพิกเซลมีค่าระหว่าง 0-255 แล้วในแต่ละพิกเซลจะลบด้วย 128 ดังนั้นจะทำให้ค่าของแต่ละพิกเซลจะมีค่าอยู่ระหว่าง –128 ถึง 127 แล้วนำค่าแต่ละบล็อกไปทำในขั้นตอนที่หนึ่ง คือ การ ทรานสฟอร์ม ผลที่ได้ แสดง ในตารางที่ 5.8 และค่าแต่ละค่าแทนด้วย $c_{i,j}$

ตารางที่ 5.8 ค่า สัมประสิทธิ ที่ได้จากการทรานสฟอร์มโดย DCT($c_{i,j}$)

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

จากตารางที่ 5.8 จะเห็นได้ว่าค่าสูงที่สุดจะอยู่ที่มุมซ้ายด้านบน (upper-left corner) และค่าที่มีความถี่สูง ๆ จะอยู่ที่มุมขวาด้านล่าง หลังจากนั้นจะทำในขั้นตอนที่สอง โดย นำค่าที่ได้ มาทำการควอนไตเซชั่น(quantization) กับตารางควอนไตเซชั่น (quantization table) ; $Q^t_{i,j}$ โดยที่ (i , j) เป็นตำแหน่งของสมาชิกใน ตารางดังแสดงในตารางที่ 5.9

ตารางที่ 5.9 ตารางควอนไตเซชั่น

2	1	0	0	0	0	0	0
-9	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

ในการควอนไตเซชั่นนั่นจะทำตามสมการที่ (5.25)

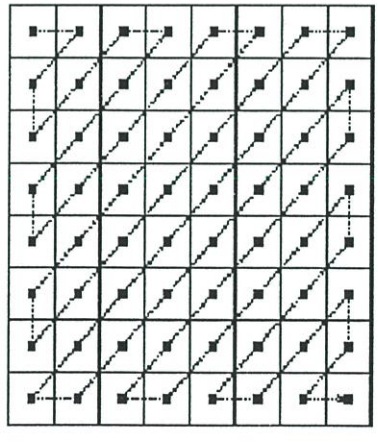
$$I_{i,j} = \left[\frac{c_{i,j}}{Q^t_{i,j}} + 0.5 \right] \qquad (5.25)$$

และ $[x]$ หมายถึง ค่าที่มากที่สุดของจำนวนเต็มที่น้อยกว่า x เช่น $I_{0,0} = \left[\frac{39.88}{16} + 0.5\right] = 2.9925$ ดังนั้นค่าที่ได้ $I_{0,0} = 2$ ค่าทั้งหมดที่ได้จากการควอนไตเซชั่นแสดงในตารางที่ 5.10 จากขั้นตอนนี้จะเห็นได้ว่าจากค่าข้อมูล 64 พิกเซลเราสามารถลดได้เหลือเพียง 4 พิกเซลเท่านั้น

ตารางที่ 5.10 ผลจากการการควอนไตเซชั่น ($I_{i,j}$)

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.13	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.051	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

ในขั้นตอนที่สามเป็นการนำมาเพื่อสร้างรหัสเลขฐานสอง (Coding) โดยข้อมูลจากตารางที่ 5.10 นั้นจะเห็นได้ว่าค่าข้อมูลที่ตำแหน่งบนสุดด้านซ้ายเป็นค่าสัมประสิทธิ DC (DC coefficient) ซึ่งเป็นค่าเฉลี่ยของข้อมูลทั้ง 64 พิกเซล ส่วนค่าที่เหลือจะเป็นค่าสัมประสิทธิ AC (AC coefficient) ซึ่งสามารถแยกพิจารณาได้ ในการเข้ารหัสก่อนอื่นต้องนำค่าตารางที่ 5.10 มาทำการซิกแซก(Zigzag Scanning) ดังรูปที่ 5.5 เสียก่อน



รูปที่ 5.5 การทำซิกแซก(Zigzag Scanning)

จากตารางที่ 5.9 เมื่อทำการซิกแซกจะได้ข้อมูลคือ

สัมประสิทธิ DC : 2

สัมประสิทธิ AC : 1 , -9 , 3 , 0 , 0 , 0 , ..., 0

ในการเข้ารหัสจะใช้วิธี Huffman Code และจะมีตารางที่ช่วยในการเข้ารหัส 2 ตารางดังแสดงในตารางที่ (5.11) และ (5.12)

ตารางที่ 5.11 ตารางการเข้ารหัสที่ใช้กับสัมประสิทธิ์ DC

0				0			
1			-1		1		
2		-3	-2		2	3	
3	-7	...	-4		4	...	7
4	-15	...	-8		8	...	15
5	-31	...	-16		16	...	31
6	-63	...	-32		32	...	63
7	-127	...	-64		64	...	127
8	-255	...	-128		128	...	255
9	-511	...	-256		256	...	511
10	-1,023	...	-512		512	...	1,023
11	-2,047	...	-1,024		1,024	...	2,047
12	-4,095	...	-2,048		2,048	...	4,095
13	-8,191	...	-4,096		4,096	...	8,191
14	-16,383	...	-8,192		8,192	...	16,383
15	-32,767	...	-16,384		16,384	...	32,767
16				32,768			

ตารางที่ 5.12 ตัวอย่างตารางที่ใช้ในการสร้างรหัส Huffman

Z/C	Codeword	Z/C	Codeword	...	Z/C	Codeword
0/0 (EOB)	1010		1010	½	F/0 (ZRL)	11111111001
0/1	00	1/1	1100	...	F/1	1111111111110101
0/2	01	1/2	11011	...	F/2	1111111111110110
0/3	100	1/3	1111001	...	F/3	1111111111110111
0/4	1011	1/4	111110110	...	F/4	1111111111111000
0/5	11010	1/5	11111110110	...	F/5	1111111111111001
.	.	.	.		.	

ตารางที่ 5.12 (ต่อ)

.	.	.	.		.	
.	.	.	.		.	

ในขั้นตอนเข้ารหัสไบนารีหรือที่เรียกว่า Entropy Coding นั่น ขั้นตอนแรก นำค่า DC มาทำการเข้ารหัสก่อนโดยทำการเทียบค่ากับค่า DC ในบล็อกก่อนหน้านี้ว่ามีค่าแตกต่างกันเท่าไร แล้วนำไปเทียบกับค่าในตารางที่ 5.11 เช่น กรณีนี้ถ้า DC ในบล็อกก่อนหน้ามีค่า –1 แต่บล็อกนี้มีค่า DC 2 ดังนั้นแสดงว่า DC มีค่าแตกต่างเท่ากับ 3 แล้วนำค่า 3 ซึ่งเป็นค่าแอมปริจูด (Amplitude) ว่าจะใช้ขนาด (SIZE) กี่บิต จากตารางที่ 5.11 ค่าแอมปริจูดเท่ากับ 3 จะใช้ขนาดข้อมูล 2 บิต สามารถเขียนได้ว่า

DC ; (2) (3) จะได้ Huffman Codeword เป็น 11

ขั้นตอนต่อมานำค่า AC มาทำการเข้ารหัสทีละค่าตามการซิกแซก ค่าแรกมีแอมปริจูดเท่ากับ 1 แล้วดูจากการซิกแซกว่ามีค่าศูนย์นำหน้ากี่ตัว และนำ แอมปริจูด 1 ไปเปิดตารางที่ 5.11 ว่าใช้ขนาดกี่บิต แล้วเขียนคู่ลำดับ

(number Zero , Size),(Amplitude)

ในกรณีตามตัวอย่างนี้ แอมปริจูดเท่ากับ 1 จะได้ (0,1) (1) แล้วนำค่า (0,1) ไปเปิดตารางที่ 5.12 เพื่อหา Codeword ต่อไป

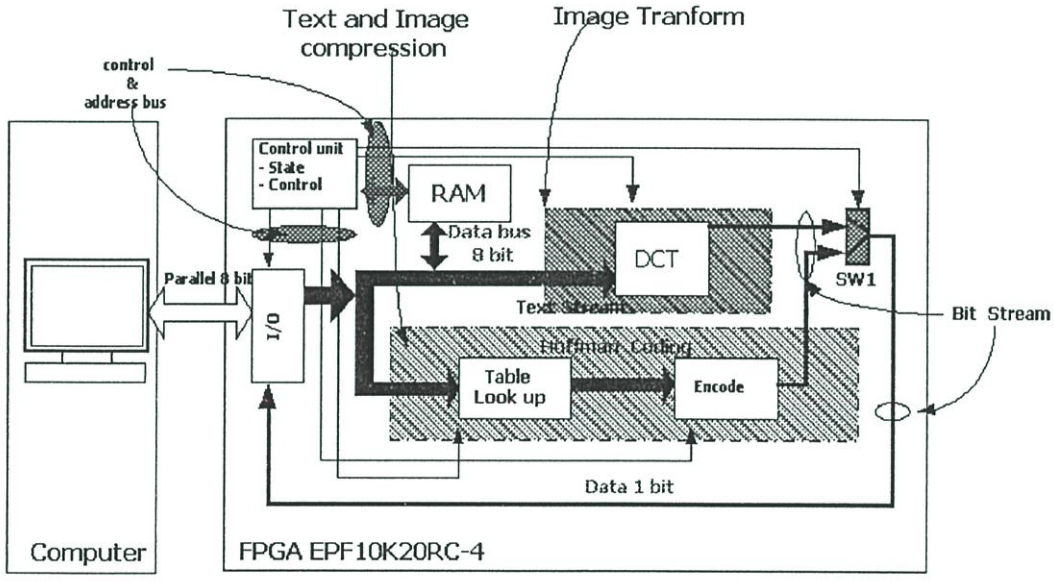
AC Coefficient	(number Zero , size)	(Amplitude)	Z/C	Huffman Codewode	Bit Stream
1	(0,1)	1	0/1	00	00*1
-9	(0,4)	-9	0/4	1011	1011*0110
3	(0,2)	3	0/2	01	01*11

* แยกระหว่าง โค็ดเวิร์ด กับ ค่าแอมปริจูด

ดังนั้นกลุ่มของบิต(Bit Stream)ที่จะทำการส่งคือ 110011011011001111010 รวมทั้งหมด 21 บิต ใน 2 บิตแรกจะเป็นค่าสัมประสิทธิ์ DC , 4 บิตสุดท้ายจะเป็นรหัสปิดท้ายในแต่ละบล็อก(EOB ; End Of Block) ส่วนบิตที่เหลือเป็นสัมประสิทธิ์ AC

บทที่ 6
สถาปัตยกรรมและการออกแบบ

ในบทนี้ได้กล่าวถึงโครงสร้างและสถาปัตยกรรมของวงจรการบีบอัดข้อมูลรูปภาพและข้อมูลตัวอักษร โดยแบ่งออกเป็น 2 ส่วนใหญ่ ๆ ด้วยกันคือ ส่วนของวงจร Huffman Coding ซึ่งทำการบีบอัดข้อมูลตัวอักษร และส่วนของ DCT ซึ่งทำการบีบอัดข้อมูลรูปภาพ โดยวงจรทั้งหมดจะถูกรวมในชิป FPGA เพียงตัวเดียว สามารถแสดงส่วนประกอบต่าง ๆ ดังแสดงในรูปที่ 6.1



รูปที่ 6.1 แสดงส่วนประกอบต่างๆ เพื่อจำลองการทำงานของวงจรการบีบอัดข้อมูล

การออกแบบนั้นได้ทำการออกแบบวงจรและทดสอบเป็นส่วน ๆ โดยแบ่งตามข้อมูลที่ทำการบีบอัดในส่วนวงจรบีบอัดข้อมูลตัวอักษรตามอัลกอริทึม Huffman Coding นั้นจะประกอบไปด้วย 4 วงจร ได้แก่ วงจรสร้างตาราง, วงจรสร้างหรัส Huffman Code วงจรสร้าง State Machine และ วงจรควบคุม ในส่วนวงจรบีบอัดข้อมูลภาพนั้นจะทำการออกแบบวงจรในส่วนของ DCT ตามอัลกอริทึมของ bin_DTC(Binary Discrete Cosine Transform)

ในการทดสอบวงจรทำได้โดยส่งข้อมูลจากคอมพิวเตอร์ผ่านพอร์ตขนาน โดยใช้ซอฟต์แวร์ในการอ่านข้อมูลจากคอมพิวเตอร์ทั้งข้อมูลภาพและข้อมูลตัวอักษรแล้วทำการบีบอัดข้อมูลผ่าน FPGA เมื่อทำการบีบอัดข้อมูลเสร็จแล้วนำข้อมูลผ่านพอร์ตขนาน เพื่อดูประสิทธิภาพในการบีบอัดข้อมูลตามอัลกอริทึมทั้งข้อมูลภาพและข้อมูลตัวอักษร

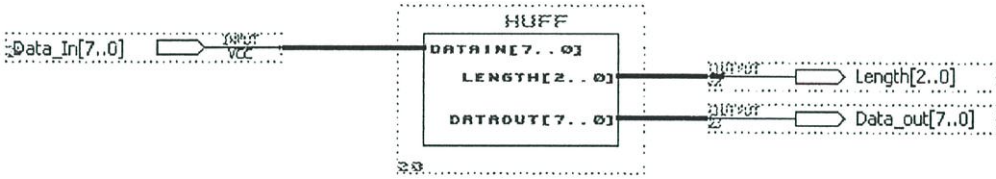
6.1 ส่วนของการบีบอัดข้อมูลตัวอักษร (Huffman Coding)

ในส่วนของการบีบอัดข้อมูลตัวอักษรนั้นได้ออกแบบวงจรออกเป็น 4 ส่วนด้วยกันคือ

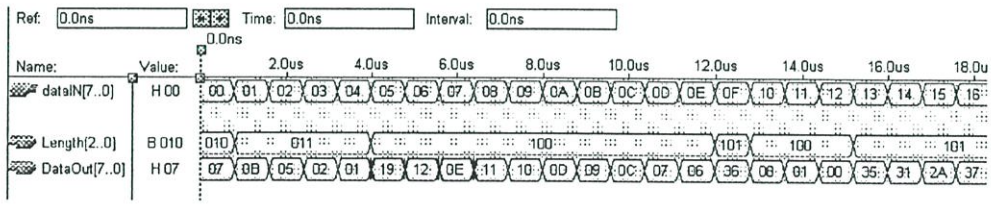
- วงจรสร้างตาราง
- วงจรสร้างรหัส Huffman Code
- วงจรสร้าง State Machine
- วงจรควบคุม

6.1.1 วงจรสร้างตาราง

ในส่วนของวงจรวงจรสร้างตาราง เป็นวงจรที่ใช้ในการสร้างตาราง Look up table เพื่อใช้สำหรับเปรียบเทียบกับไฟล์ข้อมูลอินพุทโดยประกอบด้วย 2 ส่วนหลัก คือ หมายเลขตาราง และส่วนที่เป็นรหัส Huffman Code ดังแสดงเป็น บล็อกของวงจรได้ดังรูปที่ 6.2 และแสดงผลการจำลองการทำงานของวงจรในส่วนนี้ดังรูปที่ 6.3



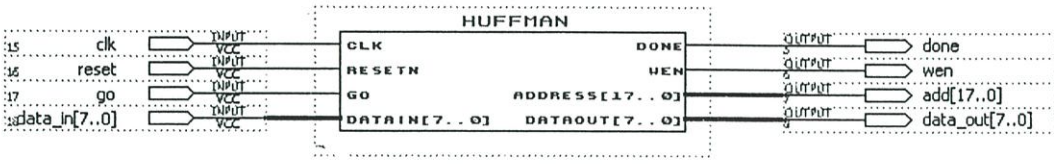
รูปที่ 6.2 แสดงบล็อกของวงจรวงจรสร้างตาราง



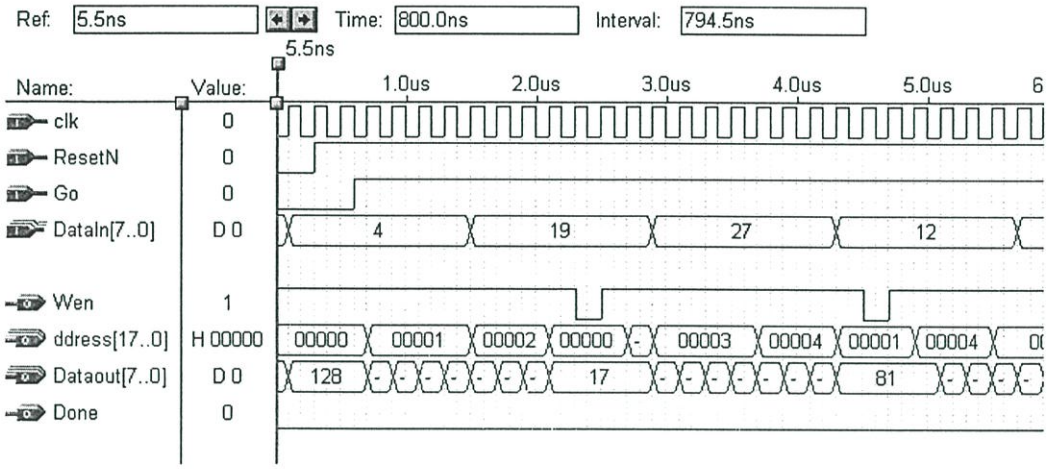
รูปที่ 6.3 แสดง Timing Diagram ของวงจรสร้างตาราง

6.1.2 วงจรสร้างรหัส Huffman Code

วงจรสร้างรหัส Huffman Code เป็นวงจรที่นำไฟล์ข้อมูลตัวอักษร มาคำนวณหาค่าความถี่ของจำนวนตัวอักษร โดยเปรียบเทียบกับผลลัพท์ที่ได้จากวงจรในรูปที่ 6.1 เพื่อที่จะสร้างรหัส Huffman Code ดังแสดงเป็น บล็อกของวงจรได้ดังรูปที่ 6.4 แสดงผลการจำลองการทำงานของวงจรในส่วนนี้ดังรูปที่ 6.5



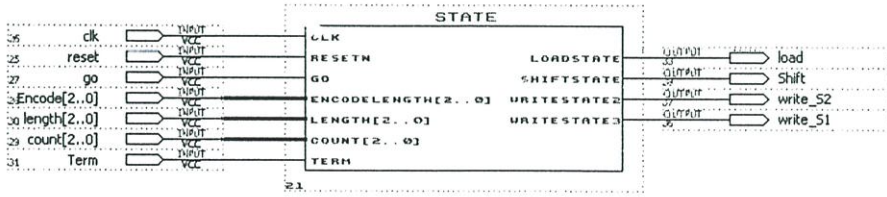
รูปที่ 6.4 แสดงบล็อกของวงจรสร้างรหัส Huffman Code



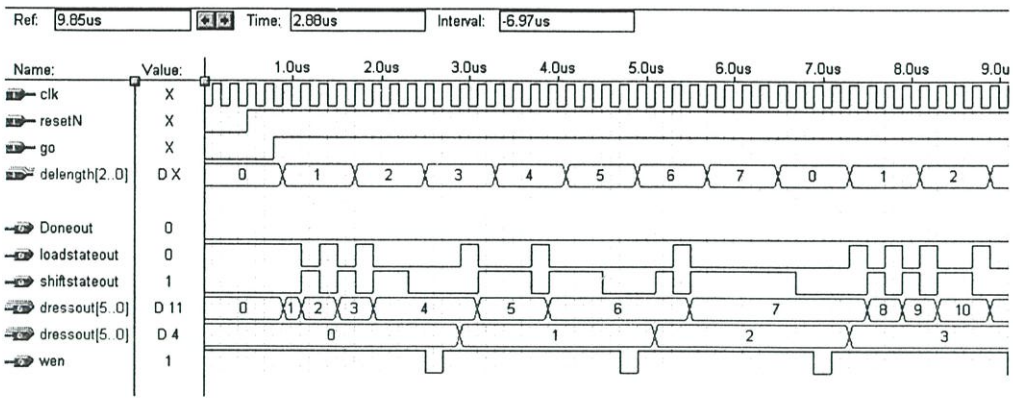
รูปที่ 6.5 แสดง Timing Diagram ของวงจรสร้างสร้างรหัส Huffman Code

6.1.3 ส่วนของวงจรวงจรสร้าง State Machine

วงจรสร้าง State Machine เป็นวงจรที่ใช้สำหรับเขียนข้อมูลที่เป็น Huffman code ลงบนหน่วยความจำทำหน้าที่โหลดและควบคุมจังหวะการทำงานในส่วนต่างๆ เพื่อให้ทำงานของระบบถูกต้อง สามารถแสดงเป็นบล็อกของวงจรได้ดังรูปที่ 6.6 และ แสดงผลการจำลองการทำงานของวงจรในส่วนนี้ดังรูปที่ 6.7



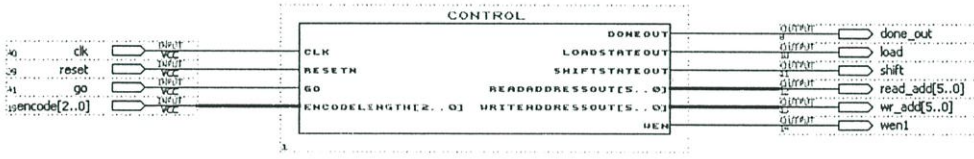
รูปที่ 6.6 แสดงบล็อคของวงจรวงจรสร้าง State Machine



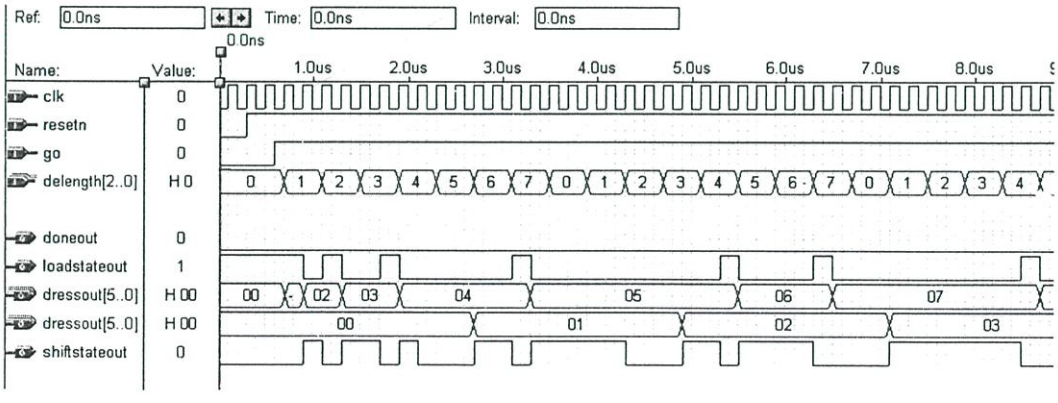
รูปที่ 6.7 แสดง Timing Diagram ของวงจรสร้างสร้างรหัส State Machine

6.1.4 วงจรควบคุม

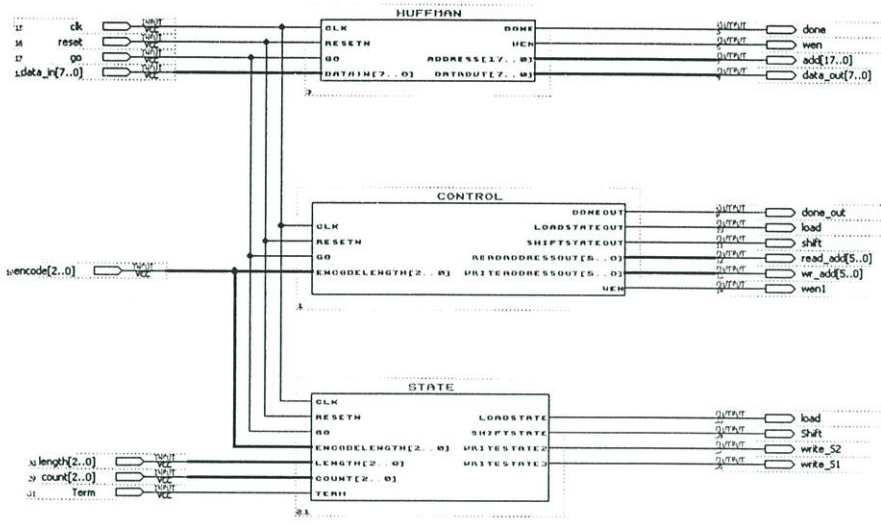
วงจรควบคุมเป็นวงจรที่ใช้สำหรับควบคุมการอ่านและเขียนข้อมูลออกจากหน่วยความจำ ซึ่งแสดงเป็นบล็อกไดอะแกรมได้ดังรูปที่ 6.8 และ แสดงผลการจำลองการทำงานของวงจรในส่วนนี้ดังรูปที่ 6.9



รูปที่ 6.8 แสดงบล็อคของวงจรวงจรควบคุม



รูปที่ 6.9 แสดง Timing Diagram ของวงจรวงจรควบคุม



รูปที่ 6.10 แสดงวงจรในส่วนต่าง ๆ มารวมเป็นวงจรบีบอัดข้อมูลตัวอักษรตามอัลกอริทึมของ Huffman

```
** DEVICE SUMMARY **

Chip/                                Input  Output  Bidir  Memory  Memory              LCs
POF          Device                  Pins   Pins    Pins   Bits %  Utilized  LCs  % Utilized

comp1        EPF10K20RC240-3         21     48      0      0         0 %     245      21 %

User Pins:                           21     48      0
```

รูปที่ 6.11 แสดงชิปและจำนวนของลอจิกที่ใช้ภายในวงจร Huffman Coding

จากการสังเคราะห์วงจรในรูปที่ 6.10 สามารแสดงชิปและจำนวนของลอจิกที่ใช้ภายในวงจร Huffman Coding ได้ดังรูปที่ 6.11 โดยคิดเป็นจำนวนลอจิกได้เท่ากับ 2,100 เกต

6.2 ส่วนบีบอัดข้อมูลภาพโดยใช้ DCT (Discrete Cosine Transform)

การบีบอัดข้อมูลภาพโดยใช้เทคนิค DCT ภาพอินพุทที่เข้ามาจะถูกแยกออกเป็นบล็อกเล็ก ๆ โดยเราสามารถกำหนดขนาดของบล็อกได้ว่าเป็นขนาดเท่าไร ขนาดของบล็อกที่เหมาะสมจะเป็นตัวเพิ่มประสิทธิภาพในการรวมพลังงานของการทรานสฟอร์มซึ่งจะทำให้ภาพที่ได้มีรายละเอียดที่ดี ความเหมาะสมของขนาดของบล็อกค่าหนึ่ง ๆ จะเหมาะสมที่ค่าบิทเรท (bit rate) ในมาตรฐานการบีบอัดข้อมูลแบบ JPEG จะมีขนาดของบล็อกเป็น 8x8 พิกเซล

การออกแบบในส่วนนี้ได้ทำการใช้อัลกอริทึมของ DCT โดยทำการทรานสฟอร์มครั้งละ 1 แถวหรือเรียกว่าเป็นการทรานสฟอร์มแบบ 1 มิติ (1-D) จากสมการของ

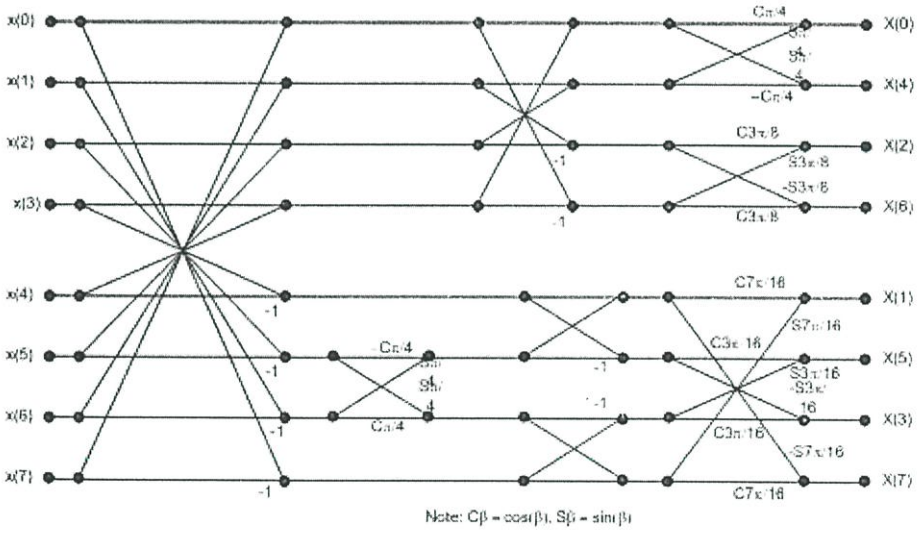
$$X_i = c_i \sum_{k=0}^{7} x_k \cos(\frac{(2k+1)i\pi}{16}) \quad ; 0 \le i \le 7 \tag{6.1}$$

$$c_i = \begin{cases} \sqrt{2} \; ; i = 0 \\ 1 \quad ; other \end{cases} \tag{6.2}$$

โดยที่ x_k เป็นข้อมูลอินพุทขนาด 8 บิต , X_i เป็นข้อมูลเอาท์พุทที่ได้จากการทรานสฟอร์ม

6.2.1 DCT ทรานสฟอร์ม (DCT Transform)

จากสมการทรานสฟอร์ม ในสมการที่ (6.1) เป็นสมการทรานสฟอร์มแบบ 1 มิติ (1-D DCT) สามารถเขียนแทนด้วยไดอะแกรมแบบทั่วไปของวงจร 1-D DCT ในรูปที่ 6.12 โดยอินพุทจะทำการ ทรานสฟอร์มครั้งละ 1 แถวขนาด 8 พิกเซล ไปจนครบตามขนาดของรูปภาพ

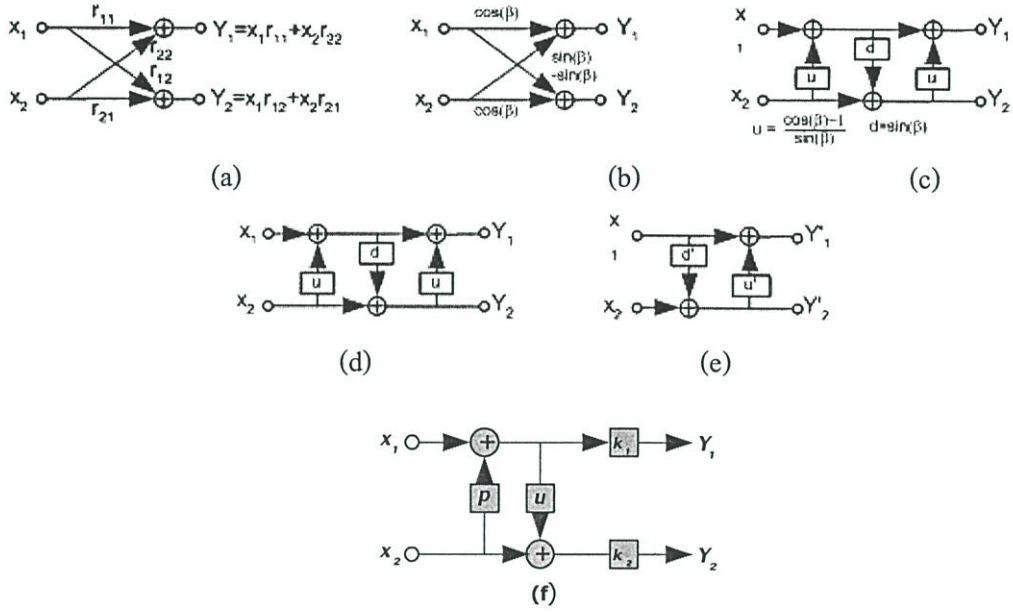


รูปที่ 6.12 รูปแบบทั่วไปของวงจร 1-D DCT

6.2.2 อัลกอริทึมรูปแบบลิฟท์ (Lifting Schematic)

อัลกอริทึมรูปแบบลิฟท์พัฒนามาจากอัลกอริทึมของ Chain Factorization ได้นำเสนอวิธีการที่จะทำให้การคูณเพื่อทำการทรานสฟอร์มทำได้เร็วขึ้น โดยวิธีการนี้จะแทนการคูณและการหาร ด้วยวงจรบวกและวงจรเลื่อนบิต โดยในแต่ละขั้นตอน จะสามารเขียนแทนด้วยเทอมของ $k/2^m$; $k,m \in Z$ ในกรณีการหารจะใช้การเลื่อนบิต $1/2^m$ จะเป็นการเลื่อนไปเท่ากับ m บิต ซึ่งจะทำให้ตัวคูณน้อยลงทำให้ง่ายต่อการ ทรานสฟอร์ม และขนาดของวงจรก็จะมีขนาดเล็กด้วย

6.2.3 อัลกอริทึมแบบสเกล (Scaled Lifting Schematic)



รูปที่ 6.13 (a) รูป butterfly ทั่วไป (b) Plane rotation (c),(d) Scaled lifting Structure แบบ 3 ลิฟท์ (e),(f) Scaled lifting Structure แบบ 2 ลิฟท์

เป็นอัลกอริทึมที่จะช่วยลดความซับซ้อนของ lift schematic โดยทั่วไปแล้วรูป butterfly จะแทนด้วย Scaled lifting schematic 2 ตัว และอีก 2 Scaling factor โดยที่ 2 Scaling factor นี้สามารถใช้เป็นส่วนหนึ่งในการทำงานในส่วนของขั้นตอนการควอนไตซ์เซชั่นได้ ดังนั้นจึงทำให้เหลือเพียง 2 lifting Step ด้านซ้ายเท่านั้นที่จะถูกทรานสฟอร์ม ทำให้การทรานสฟอร์ม มีประสิทธิภาพมากกว่าการทรานฟอร์มแบบปกติ

จากรูปที่ 6.13 (a) เราสามารถหาพารามิเตอร์สำหรับ Scaled lifting ได้ ดังสมการที่ 6.3 และ 6.4

$$Y_1 = r_{11}X_1 + r_{12}X_2 \tag{6.3}$$

$$Y_2 = r_{21}X_1 + r_{22}X_2 \tag{6.4}$$

ในรูปที่ 6.13 (b) เป็นการแทนรูปที่ 6.13 (a) ด้วยมุมระหว่าง X_1, X_2

ในทำนองเดียวกับScaled lifting ในรูปที่ 6.13 (f) จะสามารถคำนวณค่าของเอาท์พุท ได้ตามสมการที่ 6.5 ถึงสมการที่ 6.10

$$Y_1 = k_1(x_1 + pX_2) = k_1X_1 + k_1pX_2 \tag{6.5}$$

$$Y_2 = k_2(u(x_1 + pX_2) + X_2) = k_2uX_1 + k_2(1 + pu)X_2 \tag{6.6}$$

จากสมการจะมีตัวแปรที่ยังไม่ทราบค่า 4 ตัวแปร คือ

$$p = \frac{r_{12}}{r_{11}} \tag{6.7}$$

$$u = \frac{r_{12}r_{21}}{r_{11}r_{22} - r_{21}r_{12}} \tag{6.8}$$

$$k_1 = r_{11} \tag{6.9}$$

$$u = \frac{r_{11}r_{22} - r_{21}r_{12}}{r_{11}} \tag{6.10}$$

6.2.4 การทดลอง

ในการทดลองจะทำการทดลองเปรียบเทียบทั้ง 4 อัลกอริทึม ได้แก่อัลกอริทึม A,B,C และ D แสดงในรูปที่ 6.14 ถึง รูปที่ 6.17 แสดงทั้งวงจรทรานส์ฟอร์มและอินเวอร์สทรานสฟอร์มโดยจะทำการเปรียบเทียบทั้งจำนวนของวงจรบวก (Add) และ วงจรเลื่อนข้อมูล(Shift) ว่ามีจำนวนแตกต่างกันอย่างไร รวมทั้งหาความผิดพลาด(Mean Square Error) ของรูปภาพที่เกิดจากอัลกอริทึมทั้งสี่แล้วนำผลที่ได้มาเปรียบเทียบหาข้อดีข้อเสียของแต่ละวงจร สมการการหาค่า Mean Square Error (MSE) แสดงดังสมการที่ (6.12)

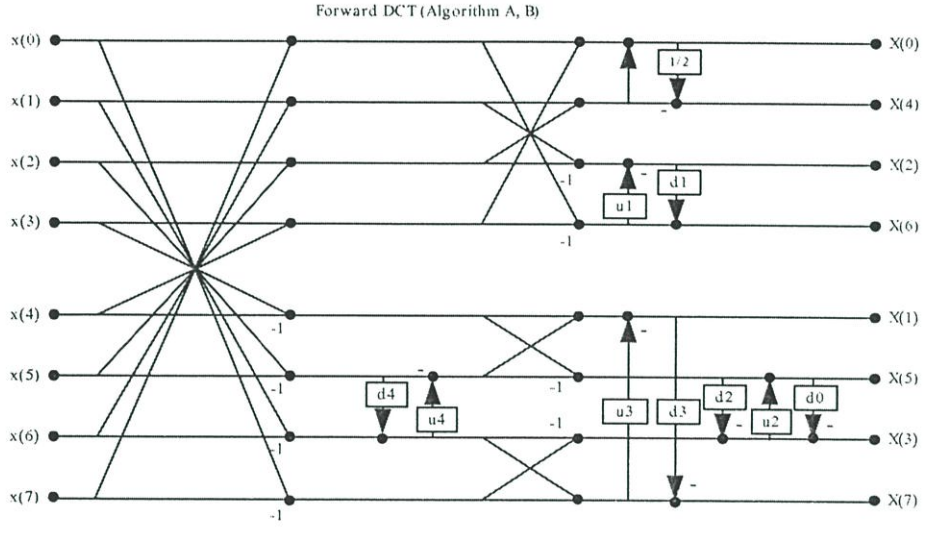
$$MSE = \frac{1}{Size^2}\sum_{row}\sum_{col}\left\{\left(x_i - X_i\right)^2\right\} \tag{6.12}$$

โดยที่ x_i เป็นสัญญาณอินพุท

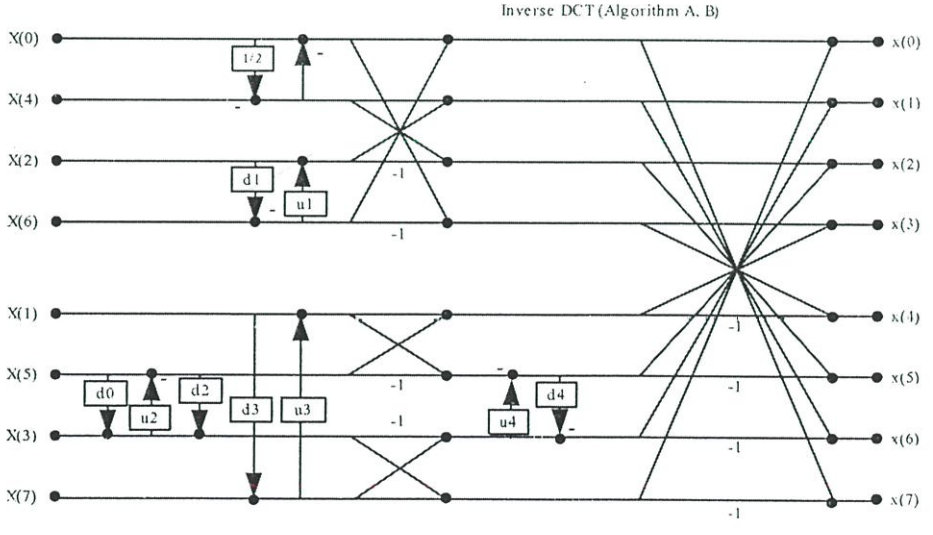
X_i เป็นสัญญาณเอาท์พุท

$$i = 0,1,...,7$$

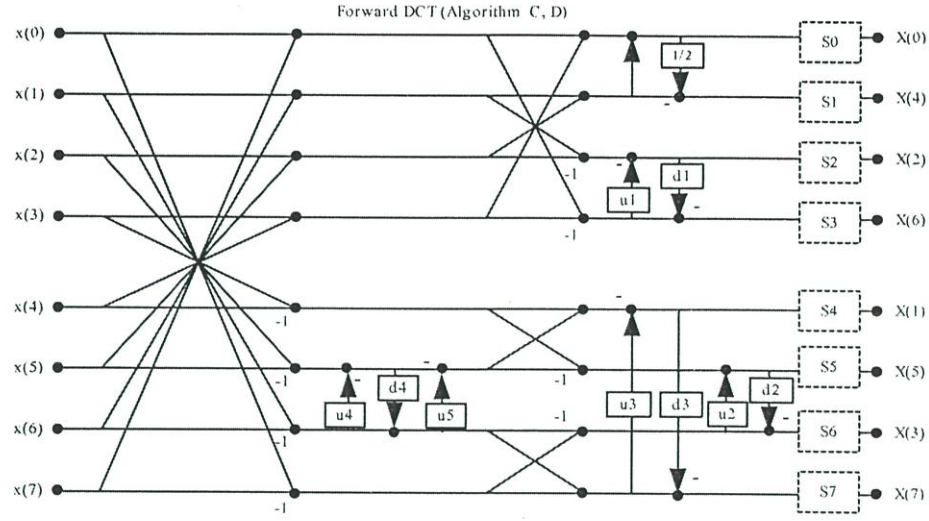
ในการทดลองเพื่อหาค่า MSE โดยจะแบ่งการทดลองจากผลที่เกิดจากการเลื่อน (shift) บิตและเมื่อทำการเพิ่มบิตขยาย (Extend Bit) ออกเป็น 5 กรณี ได้แก่ ไม่มีการเพิ่ม , เพิ่ม 1 บิต , เพิ่ม 2 บิต ,เพิ่ม 3 บิต และ เพิ่ม 4 บิต และในการทดลองจะใช้รูปภาพของ Lena ทีมีขนาดของพิกเซลที่แตกต่างกัน คือขนาด 512 , 256 ,126 และ 64 พิกเซล



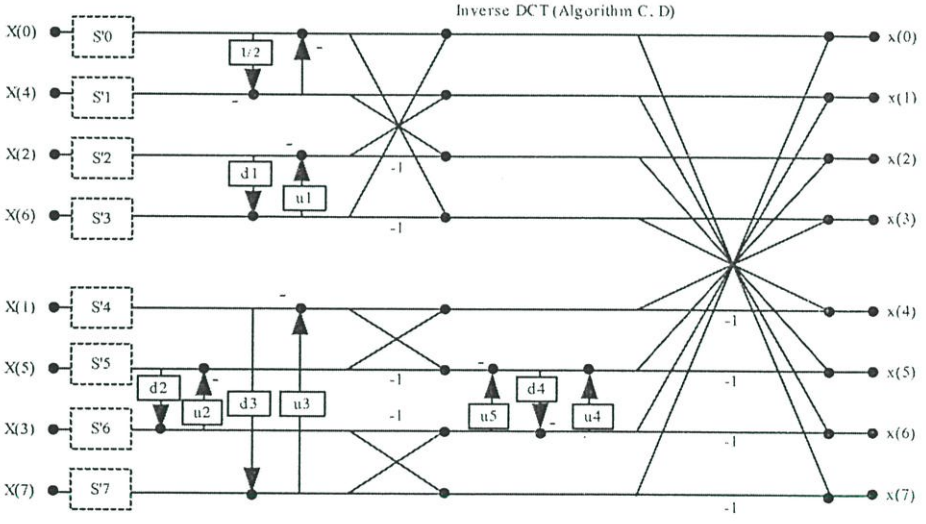
รูปที่ 6.14 วงจร DCT ทรานสฟอร์มของอัลกอริทึม A,B แบบ 2 ลิฟท์ (Lift)



รูปที่ 6.15 วงจร DCT อินเวอร์สทรานสฟอร์มของอัลกอริทึม A,B แบบ 2 ลิฟท์ (Lift)



รูปที่ 6.16 วงจร DCT ทรานสฟอร์มของอัลกอริทึม C,D แบบ 3 ลิฟท์ (Lift)



รูปที่ 6.17 วงจร DCT อินเวอร์สทรานสฟอร์มของอัลกอริทึม C,D แบบ 3 ลิฟท์ (Lift)

6.2.5 ผลการทดลอง

ในการทดลองตามอัลกอริทึมทั้ง 4 นั้น ได้ทำการทดลองในแต่ละอัลกอริทึมโดยการนำรูปภาพขาวดำขนาด 512 x 512 พิกเซล มาทำการบีบอัดข้อมูล โดยทดสอบกับอัลกอริทึม A ,B ,C และ D ตามลำดับ แสดงผลการทดลอง ดังตารางที่ 6.1 และแสดงผลการบีบอัดข้อมูลภาพ เมื่อคำนวณโดยการเพิ่มทศนิยม 1 ตำแหน่งและ 3 ตำแหน่ง ของทั้ง 4 อัลกอริทึมได้ดังรูปที่ 6.18 , 6.19 , 6.20 และ 6.21 พร้อมกับแสดง กราฟเปรียบเทียบค่า MSE ของแต่ละอัลกอริทึมได้ดังรูปที่ 6.22 , 6.23 และ 6.24

ตารางที่ 6.1 แสดงผลการทดสอบการบีบอัดข้อมูลรูปภาพขนาด 512 x 512 พิกเซล

ALG_A						
	Size	512				
	ext_b	0	1	2	3	4
	Mse	229.04	106.096	45.87	17.269	6.447
	Int.Max	112	113	113.5	114	114
	Int.Min	-27	-27.5	-27.75	-27.75	-27.75
ALG_B						
	Size	512				
	ext_b	0	1	2	3	4
	Mse	227.94	103.176	43.293	15.798	5.893
	Int.Max	112	113	113.5	114	114
	Int.Min	27	-27.5	-27.75	-27.75	-27.75
ALG_D(simplify)						
	Size	512				
	ext_b	0	1	2	3	4
	Mse	183.64	85.416	35.163	9.79	3.685
	Int.Max	112	113	113.5	114	114
	Int.Min	-24	-25.5	-26	-26	-26.063
ALG_C(full)						
	size	512				
	ext_b	0	1	2	3	4
	mse	177.53	82.4317	33.509	9.2	3.363
	Int.Max	112	113	113.5	114	114
	Int.Min	-25	-26.5	-26.75	-27	-27.063

ก) ทศนิยม 1 ตำแหน่ง

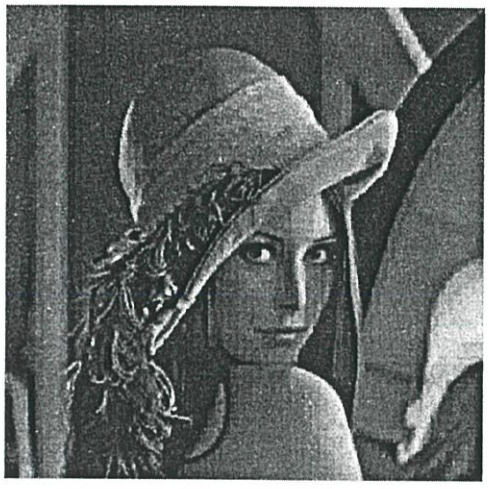
ข) ทศนิยม 3 ตำแหน่ง

รูปที่ 6.18 แสดงผลการบีบอัดข้อมูลภาพตามอัลกอริทึม A

ก) ทศนิยม 1 ตำแหน่ง

ข) ทศนิยม 3 ตำแหน่ง

รูปที่ 6.19 แสดงผลการบีบอัดข้อมูลภาพตามอัลกอริทึม B



ก) ทศนิยม 1 ตำแหน่ง

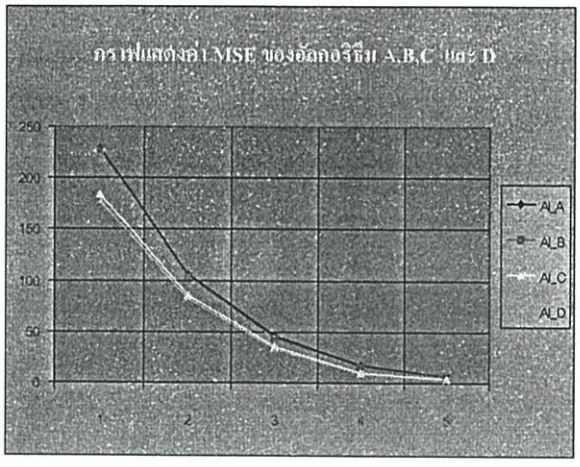
ข) ทศนิยม 3 ตำแหน่ง

รูปที่ 6.20 แสดงผลการบีบอัดข้อมูลภาพตามอัลกอริทึม C

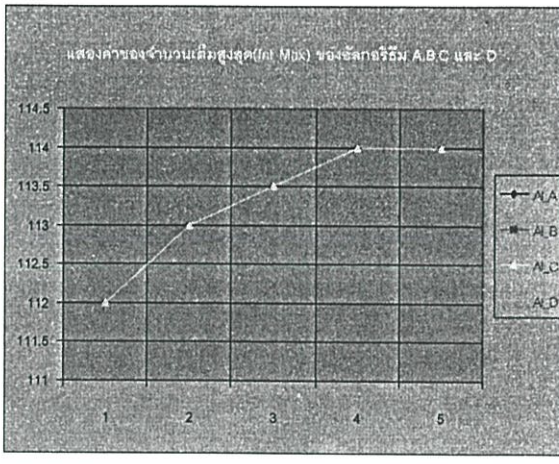
ก) ทศนิยม 1 ตำแหน่ง

ข) ทศนิยม 3 ตำแหน่ง

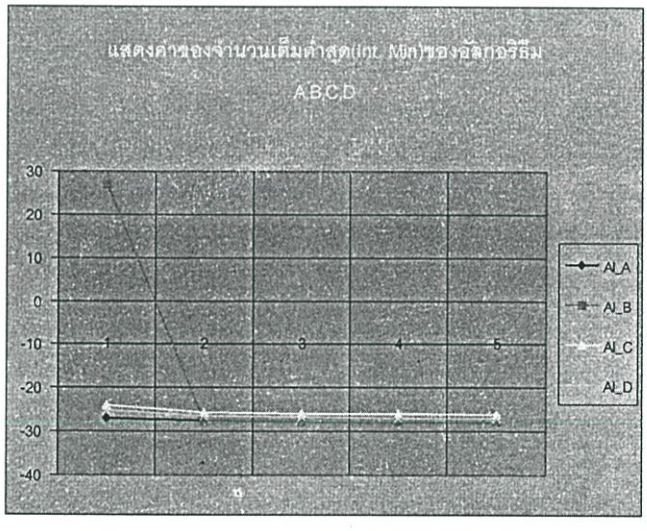
รูปที่ 6.21 แสดงผลการบีบอัดข้อมูลภาพตามอัลกอริทึม D



ก) กราฟเปรียบเทียบค่า MSE



ข) กราฟเปรียบเทียบค่า Int.Max



ค) กราฟเปรียบเทียบค่า Int.Min

รูปที่ 6.22 แสดงกราฟเปรียบเทียบค่า MSE , ค่าจำนวนเต็มสูงสุด(Int.Max) และ ค่าจำนวนเต็มต่ำสุด (Int. Min) ของอัลกอริทึม A, B, C และ D เมื่อแกนนอนหมายถึงจำนวนบิตทศนิยมที่เพิ่มขึ้น

ในการทดสอบทั้ง 4 อัลกอริทึม พบว่าอัลกอริทึม A และ B เป็นวงจรแบบ 2 ลิฟท์ และอัลกอริทึม C และ D เป็นวงจรแบบ 3 ลิฟท์ ในการเลือกอัลกอริทึมที่นำมาใช้ออกแบบวงจรจะเลือกอัลกอริทึมที่มีขนาดเล็กที่สุดและมีประสิทธิภาพในการบีบอัดข้อมูลที่ดีพอใช้ได้มาทำการออกแบบ คืออัลกอริทึม A ,B แต่อัลกอริทึม B นั้นมีจำนวนของวงจรบวกน้อยกว่าอัลกอริทึม A ดังนั้นจึงเลือกอัลกอริทึม B มาประยุกต์ใช้งาน สามารถแสดงผลการทดสอบการบีบอัดข้อมูลของอัลกอริทึม B ได้ดังรูปที่ 6.23 ซึ่งประกอบด้วยการเลือกจำนวน ทศนิยมที่ 0 , 1 , 2 และ 3 ตำแหน่ง

ก) ไม่มีการเพิ่มบิตทศนิยม

ข) เพิ่มบิตทศนิยม 1 ตำแหน่ง

ค) เพิ่มบิตทศนิยม 2 ตำแหน่ง

ง) เพิ่มบิตทศนิยม 3 ตำแหน่ง

รูปที่ 6.23 แสดงผลการทดลองบีบอัดข้อมูลภาพตามอัลกอริทึม B

6.2.6 สรุปในส่วนการออกแบบวงจรบีบอัดข้อมูลภาพ

จากผลการทดลองในส่วนวงจรบีบอัดข้อมูลภาพ สามารถนำมาวิเคราะห์และสรุปได้ดังนี้

1. ในการทดลองเพื่อต้องการตรวจสอบอัลกอริทึมไหนมีค่า MSE ดีที่สุด โดยที่ค่า MSE จะเป็นค่าที่บอกถึงประสิทธิภาพของภาพที่ได้จากการขยายภาพของวงจร โดยค่า MSE น้อยจะดีกว่าค่า MSE มาก
2. ในทุกอัลกอริทึมจะเห็นได้ว่าในทุกขนาดไฟล์ภาพ เมื่อมีการเพิ่มบิตขยายมากขึ้น (Extend Bit) ค่า MSE ก็จะลดลง แสดงว่าเมื่อทำการเพิ่มบิตขยายมากขึ้น ก็จะทำให้ภาพที่ได้มีคุณภาพดีขึ้นด้วย ดังแสดงในกราฟรูปที่ 6.22 (ก)
3. ในรูปภาพขนาดเท่ากัน อัลกอริทึม C มี MSE น้อยที่สุด และ อัลกอริทึม A จะมี MSE มากที่สุด ดังนั้นแสดงว่าอัลกอริทึม C เป็นอัลกอริทึมที่ดีที่สุด
4. การพิจาณาในส่วนของวงจรนั้น อัลกอริทึม A,B เป็นวงจรแบบ 2 ลิฟท์ ส่วนอัลกอริทึม C,D เป็นวงจรแบบ 3 ลิฟท์ ดังนั้นอัลกอริทึม A, B จะมีขนาดของวงจรเล็กกว่า และเมื่อเปรียบเทียบเฉพาะอัลกอริทึม A,B พบว่าวงจรภายในของอัลกอริทึม A มีจำนวนวงจรบวกอยู่มากกว่าอัลกอริทึม B แสดงว่า ขนาดของวงจรในอัลกอริทึม B มีขนาดเล็กที่สุดเมื่อเปรียบเทียบทั้ง 4 อัลกอริทึม

ตารางที่ 6.2 แสดงคุณภาพของอัลกอริทึม A,B,C และ D

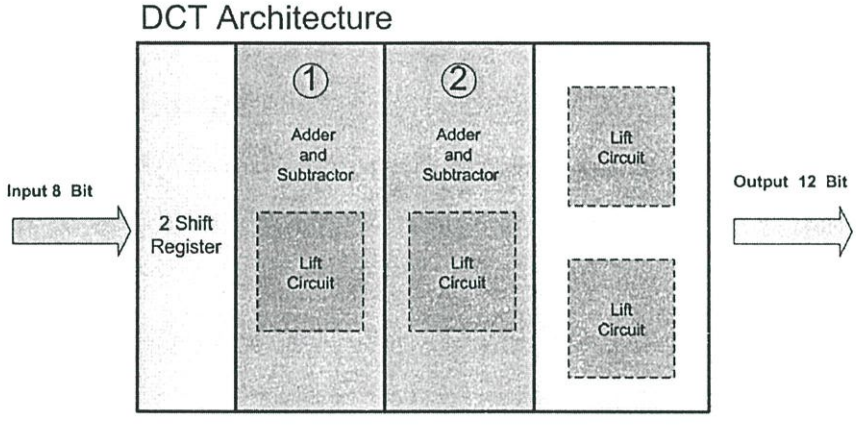
	คุณภาพของภาพ
สูงสุด ↑ ต่ำสุด	Algorithm C
	Algorithm D
	Algorithm B
	Algorithm A

ตารางที่ 6.3 แสดงขนาดของวงจรของอัลกอริทึม A,B,C และ D

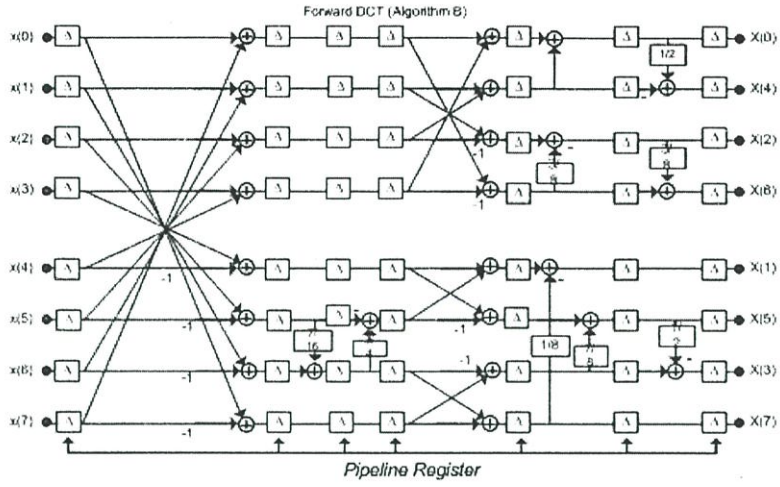
	ขนาดของวงจร
สูงสุด ↑ ต่ำสุด	Algorithm D
	Algorithm C
	Algorithm A
	Algorithm B

6.3 การประยุกต์ใช้งานวงจร DCT

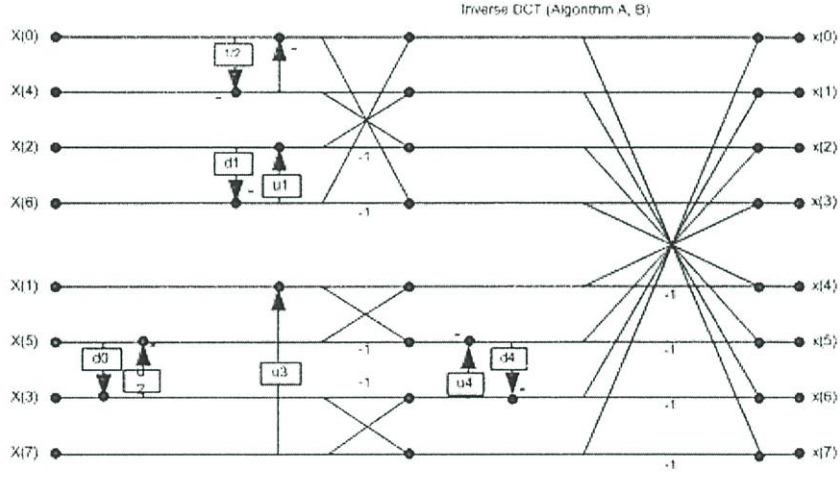
จากผลการทดสอบในหัวข้อที่ 6.25 นั้นทำให้เลือกอัลกอริธึม B มาทำการออกแบบวงจรในส่วนของ DCT แสดงวงจรที่ทำการออกแบบดังรูปที่ 6.25 โดยทำการออกแบบโดยใช้เทคนิค Pipeline ซึ่งทำให้ได้ผลการคำนวณที่รวดเร็ว สามารถแสดงสถาปัตยกรรมสำหรับออกแบบวงจร DCT ตามอัลกอริทึม B ได้ดังรูปที่ 6.24 โดยการรับข้อมูลอินพุทขนาด 8 บิต หลังจากทำการทรานสฟอร์มโดย DCT จะได้เอาท์พุทขนาด 12 บิตโดยบิตที่เพิ่มขึ้นขนาด 3 บิตจะช่วยในการลดค่า MSE ของข้อมูลภาพนั้นเอง ในวิทยานิพนธ์นี้ ได้ออกแบบโอเปอร์เรชั่นทั้งหมดนั้นเป็นแบบ Bit Serial เพื่อที่จะให้ได้วงจรที่มีขนาดเล็กแต่วงจรจะใช้จำนวน Clock Cycle สำหรับการประมวลผลสูง ซึ่งเหมาะกับการใช้งานในลักษณะที่ไม่ต้องการความเร็วในการประมวลผลสูงมากแต่ต้องการวงจรที่มีขนาดเล็กนั่นเอง



รูปที่ 6.24 แสดงสถาปัตยกรรมในการออกแบบวงจร DCT



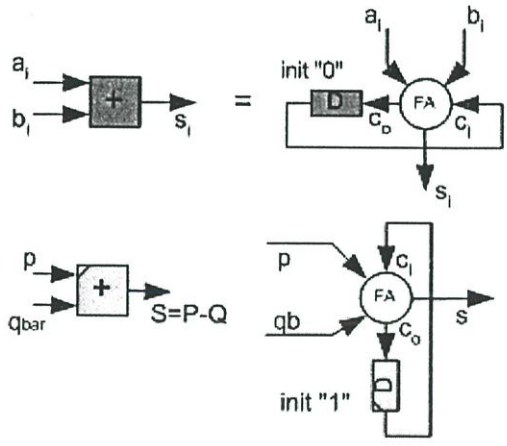
A) Forward



B) Inverts

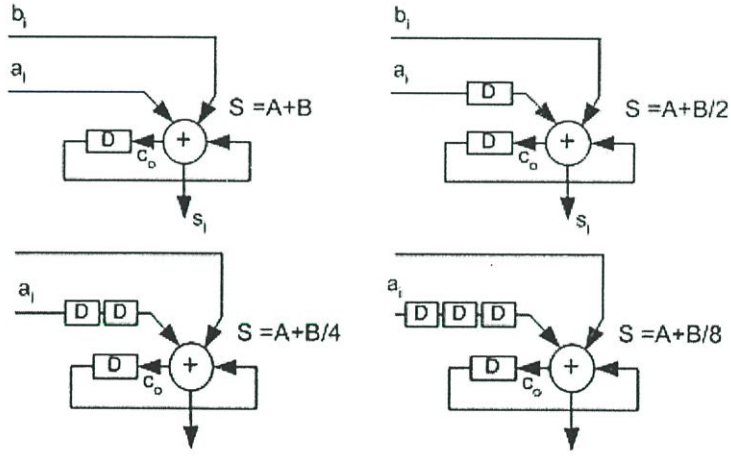
รูปที่ 6.25 แสดงการออกแบบวงจร DCT แบบ Pipeline A) Forward B) Inverts

รูปที่ 6.26 แสดงวงจรการบวกและลบที่ใช้ในการออกแบบวงจร DCT โดยทำการรับข้อมูลแบบอนุกรมมาทีละตัวเพื่อกระทำการบวกและลบตามโอเปอร์เรชั่นดังกล่าวโดยในการวกนั้น เมื่อมีการบวกเกินต้องทำการส่งค่าที่เกินนั้นมาเก็บไว้ในฟลิบฟลอบ D เพื่อที่จะเอาไว้สำหรับบวกกับค่าใน State ต่อไป ซึ่งในการบวกและลบนั้นสามารถที่ใช้วงจรร่วมกันได้เพียงแต่ในต่อนเริ่มต้นนั้นต้องทำการ initial ค่าของ ฟลิบฟลอบ D ต่างกันคือ ถ้าทำการลบนั้นต้องให้ค่าในฟลิบฟลอบ D เท่ากับ “1” ส่วนในการบวกนั้นต้องให้ค่าในฟลิบฟลอบ D เท่ากับ “0”



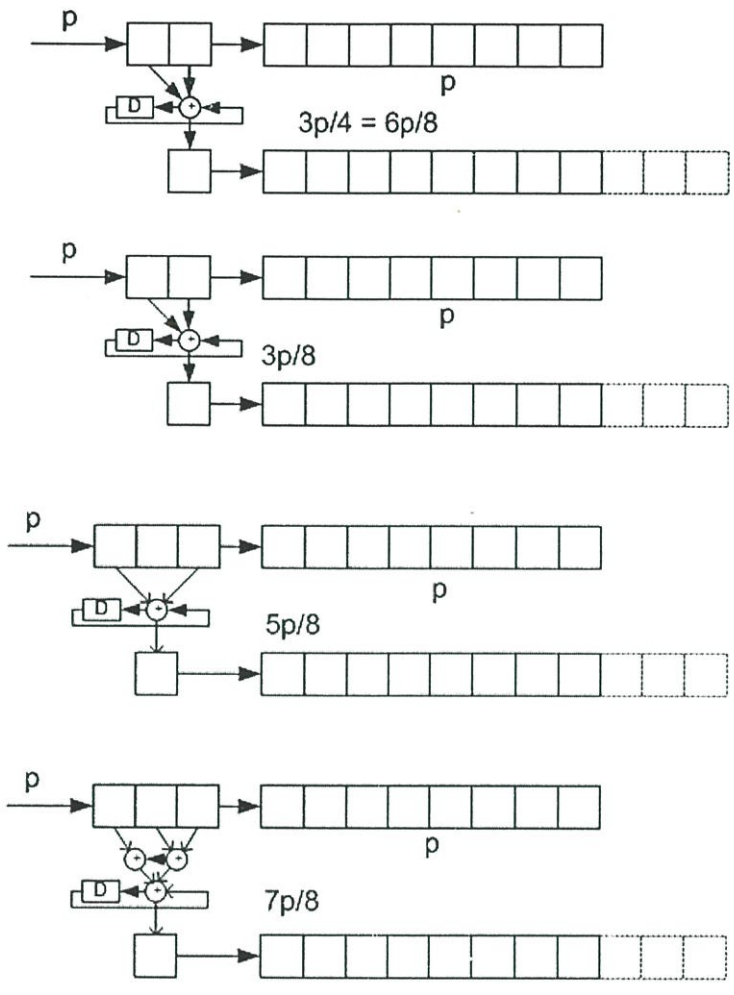
รูปที่ 6.26 การออกแบบวงจรบวก และลบ

ในรูปที่ 6.27 แสดงวงจรการบวกแบบต่างๆที่ใช้สำหรับการออกแบบวงจร DCT โดยการรับข้อมูลอินพุท 2 ค่าและซึ่งถ้าเป็นวงจรหาร 2 นั้นต้องทำการดีเลย์ค่าของอินพุทจำนวน 1 ค่า วงจรหาร 4 นั้นต้องทำการดีเลย์ค่าของอินพุทจำนวน 2 ค่า และวงจรหาร 8 นั้นต้องทำการดีเลย์ค่าของอินพุทจำนวน 3 ค่า



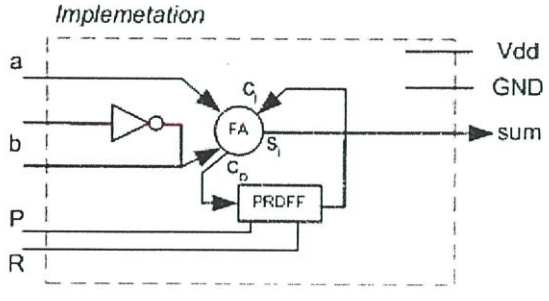
รูปที่ 6.27 การออกแบบวงจรบวกแบบต่างๆ

รูปที่ 6.28 แสดงการออกแบบวงจรหารต่างๆ โดยการใชหลักการของการ Shift Register โดยมีหลักการในการออกแบบดังนี้ เช่นในวงจรหาร $\frac{3}{4}, \frac{3}{8}$ และ $\frac{6}{8}$ มีหลักการในการออกแบบเหมือนกันคือ นำค่า 2 ค่าที่อยู่ติดกันมารวมกันจึงจะได้เป็นผลลัพธ์



รูปที่ 6.28 การออกแบบวงจรหารแบบต่างๆ

สำหรับรูปที่ 6.29 แสดงการประยุกต์วงจร โดยการรวมวงจรบวกและลบเข้ามาไว้เป็นวงจรเดียวกัน เพื่อนำมาประยุกต์ใช้งานเป็นโอเปอร์เรชั่นสำหรับการบวกและลบในวงจรต่อไป



รูปที่ 6.29 การประยุกต์วงจรบวก และลบในวงจร DCT

6.4 สรุป

จากการออกแบบวงจรได้ทำการออกแบบและทำการจำลองการทำงานทีละส่วน วงจรบีบอัดข้อมูลตัวอักษรนั้นสามารถบีบอัดข้อมูลได้ถูกต้องตามอัลกอริทึมของ Huffman Coding และจากสังเคราะห์วงจรใช้เกตไปจำนวน 2,100 เกต และส่วนวงจรบีบอัดข้อมูลภาพนั้นจะทำการออกแบบและทดสอบเฉพาะส่วนวงจรของ DCT อัลกอริทึม A,B,C และ D ในทุกอัลกอริทึมจะเห็นได้ว่าในทุกไฟล์ภาพ เมื่อมีการเพิ่มบิตขยายมากขึ้น (Extend Bit) ค่า MSE ก็จะลดลง นั่นหมายความว่า ยิ่งเพิ่มบิตขยายมากขึ้น ก็จะทำให้ภาพที่ได้ดีขึ้นด้วย ในขนาดรูปภาพขนาดเท่ากัน อัลกอริทึม C มี MSE น้อยที่สุด และ อัลกอริทึม A จะมี MSE มากที่สุด แสดงว่าอัลกอริทึม C เป็นอัลกอริทึมที่ดีกว่าอัลกอริทึม A,B และ D

ในส่วนขนาดของวงจรอัลกอริทึม A,B เป็นวงจรแบบ 2 ลิฟท์ อัลกอริทึม C,D เป็นวงจรแบบ 3 ลิฟท์ ดังนั้นอัลกอริทึม A,B จะมีขนาดของวงจรเล็กกว่า แต่เมื่อทำการเปรียบเทียบเฉพาะ อัลกอริทึม A,B นั้น วงจรภายในของอัลกอริทึม A มีวงจรบวกมากกว่าอัลกอริทึม B ดังนั้นจึงเลือกอัลกอริทึม B มาทำการออกแบบเป็นวงจร DCT โดยทำการออกแบบโดยใช้เทคนิคไปล์ไลท์ Pipeline ซึ่งทำให้ได้ผลการคำนวณที่รวดเร็ว สามารถแสดงสถาปัตยกรรมสำหรับออกแบบวงจร DCT ตามอัลกอริทึม B ซึ่งในวิทยานิพนธ์นี้ ได้ออกแบบโอเปอร์เรชั่นทั้งหมดนั้นเป็นแบบ Bit Serial เพื่อที่จะให้ได้วงจรที่มีขนาดเล็กแต่วงจรจะใช้จำนวน Clock Cycle สำหรับการประมวลผลสูง ซึ่งเหมาะกับการใช้งานในลักษณะที่ไม่ต้องการความเร็วในการประมวลผลสูงมากแต่ต้องการวงจรที่มีขนาดเล็กนั่นเอง

บทที่ 7

การทดสอบและผลการทดสอบ

ในบทนี้ได้กล่าวถึงการทดสอบและผลการทดสอบการบีบอัดข้อมูลตัวอักษรและข้อมูลภาพด้วยวงจรที่ทำการออกแบบ โดยนำวงจรที่ทำการออกแบบมาเชื่อมต่อกับเครื่องคอมพิวเตอร์ทางพอร์ตขนาน และส่งไฟล์ข้อมูลที่ต้องการทดสอบจากเครื่องคอมพิวเตอร์ไปทำการบีบอัดข้อมูลในวงจรที่ออกแบบบน FPGA นำผลของการบีบอัดข้อมูลเก็บไว้ใน RAM ของ FPGA แล้วจึงใช้ซอฟท์แวร์อ่านมาเก็บไว้ในรูปแบบของไฟล์บนคอมพิวเตอร์อีกครั้ง ทำการทดสอบขนาดของไฟล์ข้อมูลที่บีบอัดแล้ว ในการทดสอบได้แบ่งการทดสอบออกเป็น 3 ส่วนคือส่วนของข้อมูลตัวอักษร, ส่วนของข้อมูลภาพ และส่วนของข้อมูลภาพรวมกับข้อมูลตัวอักษร

7.1 การทดสอบการบีบอัดข้อมูลด้วย Huffman Coding

7.1.1 การทดสอบการบีบอัดข้อมูลตัวอักษร

ทำการส่งไฟล์ข้อมูลตัวอักษรขนาด 256ไบต์ ,512ไบต์ และ 1028 ไบต์ เพื่อแสดงให้เห็นประสิทธิภาพการบีบอัดข้อมูลตัวอักษรของ Huffman Coding ได้ผลการทดสอบดังตารางที่ 7.1

ตารางที่ 7.1 แสดงการทดสอบการบีบอัดข้อมูลตัวอักษร Huffman Coding

In File	Input (Byte)	Output (byte)	Compression ratio
Test_t256.txt	256	234	9%
Test_t512.txt	512	400	22%
Test_t1K.txt	1,028	761	26%

จากตารางที่ 7.1 พบว่าเมื่อทำการทดสอบการบีบอัดข้อมูลตัวอักษรนั้น ถ้าไฟล์มีใหญ่ประสิทธิภาพในการบีบอัดข้อมูล(Compression ratio) สูงกว่าไฟล์ที่มีขนาดเล็ก ด้วยเหตุผลที่ว่าถ้าไฟล์ที่มีขนาดเล็กจะสิ้นเปลืองพื้นที่ในการสร้างตารางความถี่ของข้อมูลเพิ่มขึ้น

7.1.2 การทดสอบการบีบอัดข้อมูลภาพ

ทำการส่งไฟล์ข้อมูลภาพขาวดำขนาด 128ไบต์ ,256ไบต์ และ 512 ไบต์เพื่อแสดงให้เห็นประสิทธิภาพการบีบอัดข้อมูลตัวอักษรของ Huffman Coding โดยได้แบ่งการทดสอบออกเป็น 3 ส่วนคือส่วนข้อมูลชนิด . PPM .JPG และ .BMPได้ผลการทดสอบดังตารางที่ 7.2 , 7.3 และ 7.4

ตารางที่ 7.2 แสดงการทดสอบการบีบอัดข้อมูลภาพ Huffman Coding ชนิด .PPM

In File	Input (Byte)	Output (byte)	Compression ratio
img_128.ppm	65,559	24,331	63%
img_256.ppm	269,877	105,801	61%
img_512.ppm	1,048,598	403,702	62%

ตารางที่ 7.3 แสดงการทดสอบการบีบอัดข้อมูลภาพ Huffman Coding ชนิด .JPG

In File	Input (Byte)	Output (byte)	Compression ratio
img_128.jpg	3002	3237	-7%
img_256.jpg	11,546	11,789	-2%
img_512.jpg	28,490	28,558	0%

ตารางที่ 7.4 แสดงการทดสอบการบีบอัดข้อมูลภาพ Huffman Coding ชนิด .BMP

In File	Input (Byte)	Output (byte)	Compression ratio
img_128.bmp	17,462	14,431	18%
img_256.bmp	66,614	60,346	10%
img_512.bmp	263,222	247,032	7%

จากตารางที่ 7.2 , 7.3 และ 7.4 นั้นได้ทำการบีบอัดข้อมูลรูปภาพ .PPM .JPG และ .BMP ขนาด 128 x128 , 256 x 256 และ 512 x512 พิกเซล โดยตารางที่ 7.2 นั้นได้ทำการบีบอัดข้อมูลรูปภาพ .PPM พบว่าอัลกอริทึมของ Huffman สามารถบีบอัดข้อมูลภาพของไฟล์ประเภทนี้ได้ดี จากตารางที่ 7.3 ได้ทำการบีบอัดข้อมูลรูปภาพ .JPG พบว่าไม่สามารถบีบอัดข้อมูลให้เล็กลงได้เนื่องจากไฟล์ชนิดนี้นั้นได้ถูกบีบอัดข้อมูลเป็นทีเรียบร้อยแล้ว จากตารางที่ 7.2 นั้นได้ทำการบีบอัดข้อมูลรูปภาพ .BMP พบว่าข้อมูลรูปภาพยิ่งมีขนาดเล็กจะทำให้สามารถบีบอัดข้อมูลได้ดีเช่นกัน

7.1.3 การทดสอบการบีบอัดข้อมูลภาพและตัวอักษร

ทำการส่งไฟล์ข้อมูลตัวอักษรและภาพขาวดำขนาด 65,668ไบต์ , 270,114ไบต์ และ 1,049,091 ไบต์ ในการสร้างไฟล์ผสมทำได้โดยการนำไฟล์รูปภาพ . PPM มาทำการแปลงเป็น

ไฟล์ตัวอักษร(.TXT) แล้วทำการรวมกับไฟล์ข้อมูลตัวอักษรที่ได้เตรียมไว้ และทำการเก็บข้อมูลเหล่านั้นในรูปของไฟล์ .TXT แล้วจึงบีบอัดข้อมูล ได้ผลการทดสอบดังตารางที่ 7.5

ตารางที่ 7.5 แสดงการทดสอบการบีบอัดข้อมูลภาพ Huffman Coding ชนิด .TXT

In File	Input (Byte)	Output (byte)	Compression ratio
text_im_128.txt	65,668	24,668	63%
text_im_256.txt	270,114	107,579	61%
text_im_512.txt	1,049,091	410,703	61%

จากตารางที่ 7.5 แสดงให้เห็นว่าสามารถลดขนาดข้อมูลรูปภาพและข้อมูลตัวอักษรที่รวมกันได้เป็นอย่างดีโดยการแปลงข้อมูลภาพให้เป็นตัวอักษรแล้วจึงรวมข้อมูลเหล่านั้นเป็นไฟล์เดียวกัน แล้วจึงนำไปเข้าบีบอัดข้อมูลเพื่อลดขนาดข้อมูลต่อไป

เราได้เห็นการประยุกต์ใช้งานไปบ้างแล้ว ได้แก่ การทำตัวกรองแอนะลอกจากตัวกรองดิจิตอล การเปลี่ยนแปลงอัตราการสุ่ม และการวิเคราะห์ทาสเปกตรัมโดยการใช้การแปลง FFT การประยุกต์ใช้ตัวกรองดิจิตอลในงานต่างๆ รวมทั้ง แนะนำการประยุกต์ใช้งานในขั้นสูงขึ้นได้แก่ ตัวกรองแบบปร

รูปที่ 7.1 แสดงข้อมูลตัวอักษร Test_t256.txt

เราได้เห็นการประยุกต์ใช้งานไปบ้างแล้ว ได้แก่ การทำตัวกรองแอนะลอกจากตัวกรองดิจิตอล การเปลี่ยนแปลงอัตราการสุ่ม และการวิเคราะห์ทาสเปกตรัมโดยการใช้การแปลง FFT การประยุกต์ใช้ตัวกรองดิจิตอลในงานต่างๆ รวมทั้ง แนะนำการประยุกต์ใช้งานในขั้นสูงขึ้นได้แก่ ตัวกรองแบบปร เราได้เห็นการประยุกต์ใช้งานไปบ้างแล้ว ได้แก่ การทำตัวกรองแอนะลอกจากตัวกรองดิจิตอล การเปลี่ยนแปลงอัตราการสุ่ม และการวิเคราะห์ทาสเปกตรัมโดยการใช้การแปลง FFT การประยุกต์ใช้ตัวกรองดิจิตอลในงานต่างๆ รวมทั้ง แนะนำการประยุกต์ใช้งานในขั้นสูงขึ้นได้แก่ ตัวกรองแ

รูปที่ 7.2 แสดงข้อมูลตัวอักษร Test_t256.txt

รูปที่ 7.3 แสดงข้อมูลรูปภาพขนาด 128 x128 พิกเซล

รูปที่ 7.4 แสดงข้อมูลรูปภาพขนาด 256 x256 พิกเซล

รูปที่ 7.5 แสดงข้อมูลรูปภาพขนาด 512 x512 พิกเซล

จากรูปที่ 7.1 และ 7.2 เป็นข้อมูลตัวอักษรที่ใช้นำมาทำการทดสอบ ในส่วนรูปที่ 7.4 , 7.5 และ 7.6 เป็นข้อมูลรูปภาพที่มีขนาดต่าง ๆ ที่ใช้ในการทดสอบ ในส่วนของข้อมูลผสมระหว่างข้อมูล ตัวอักษรและข้อมูลรูปภาพ ก็นำข้อมูลในรูปเหล่านี้มารวมกัน

7.2 สรุป

จากการทดสอบการบีบอัดข้อมูลภาพและตัวอักษรโดยใช้ Huffman Coding ผลจากการทดสอบจะพบว่าเมื่อทำการทดสอบการบีบอัดข้อมูลตัวอักษรนั้น ถ้าไฟล์มีขนาดใหญ่ขึ้นจะสามารถบีบอัดข้อมูลได้มากขึ้น ส่วนข้อมูลรูปภาพได้ทำการบีบอัดข้อมูลรูปภาพ .PPM, .JPG และ .BMP ขนาด 128 x128 , 256 x 256 และ 512 x512 พิกเซล โดยข้อมูลรูปภาพ .PPM นั้นถ้ามีขนาดเล็กทำให้สามารถบีบอัดข้อมูลได้ดี ข้อมูลรูปภาพ .JPG ไม่สามารถบีบอัดข้อมูลให้เล็กลงได้เนื่องจากไฟล์ชนิดนี้ได้ถูกบีบอัดข้อมูลแล้ว ส่วนข้อมูลรูปภาพ .BMP พบว่าข้อมูลรูปภาพยิ่งมีขนาดเล็กจะทำให้สามารถบีบอัดข้อมูลได้ดีเช่นกัน ส่วนการลดขนาดข้อมูลรูปภาพและข้อมูลตัวอักษรที่รวมกันนั้นวิธีการนี้สามารถบีบอัดข้อมูลได้ดีเช่นกัน ซึ่งสามารถนำไปใช้เป็นเทคนิคในการบีบอัดข้อมูลอีกวิธีหนึ่ง จึงกล่าวได้ว่าวิธีการของ Huffman Coding สามารถบีบอัดข้อมูลตัวอักษรและภาพได้ดีซึ่งถ้าภาพมีขนาดเล็ก ยิ่งทำความสามารถในการบีบอัดข้อมูลดีขึ้น

บทที่ 8
สรุปผลการวิจัยและข้อเสนอแนะ

8.1 สรุป

การบีบอัดข้อมูลถือว่าเป็นส่วนสำคัญในการออกแบบวงจรที่เกี่ยวข้องทางการจัดเก็บและการสื่อสารข้อมูลในปัจจุบัน ในวิทยานิพนธ์นี้ได้ทำการออกแบบวงจรโดยใช้ FPGA เพื่อเป็นต้นแบบของวงจรรวมขนาดใหญ่มาก (VLSI) สำหรับบีบอัดข้อมูลโดยใช้หลักการ Huffman Coding และ JPEG เพื่อทำการบีบอัดข้อมูลตัวอักษรและบีบอัดข้อมูลภาพ โดยต้องการให้ขนาดของวงจรทั้งหมดของวงจรบีบอัดข้อมูลมีขนาดเล็ก เพื่อนำไปประยุกต์ใช้งานร่วมกับวงจร การจัดเก็บข้อมูล,วงจรสื่อสาร หรือวงจรอื่น ๆ ที่ต้องใช้การบีบอัดข้อมูลก่อนทำการจัดเก็บหรือทำการส่ง

การบีบอัดข้อมูลเป็นวิธีการลดขนาดของข้อมูล โดยข้อมูลอาจอยู่ในรูปของ ตัวอักษร, ข้อมูลเสียง,ข้อมูลภาพ การบีบอัดข้อมูลมีความสำคัญต่อหลายๆสาขา เช่น การส่งสัญญาณโทรทัศน์แบบดิจิตอล ถ้าต้องการส่งสัญญาณ HDTV (High Definition Tele Vision) โดยไม่ทำการบีบอัดข้อมูล จะต้องทำการส่งข้อมูลด้วยความเร็ว 884 เมกะบิตต่อวินาที แต่ถ้ามีการบีบอัดข้อมูลก่อนทำการส่ง สามารถส่งได้น้อยกว่า 20 เมกะบิตต่อวินาที ข้อมูลเสียงก็มีความจำเป็นที่ต้องมีการบีบอัดข้อมูลเช่นเดียวกัน ในระบบสื่อสารข้อมูลทางคอมพิวเตอร์ หลายประเภทเช่น โมเด็ม,การเก็บข้อมูลในหน่วยความจำสำรอง ก็จำเป็นต้องใช้การบีบอัดข้อมูลมาช่วยลดขนาดของข้อมูล ก่อนทำการส่งหรือจัดเก็บข้อมูลเช่นเดียวกัน

การออกแบบวงจร ได้ทำการออกแบบด้วยภาษา VHDL ซึ่งเป็นภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ ทั้งในระดับเกตจนถึงระดับดิจิตอลที่ซับซ้อน ทั้งนี้เนื่องจาก VHDL เป็นภาษาที่ที่เหมาะในการนำมาใช้ในการเลียนแบบการทำงานของอุปกรณ์ อีกทั้งยังมีความยืดหยุ่นและไม่ถูกจำกัดโดยความสามารถทางเทคโนโลยีใดๆ ทำให้ประหยัดเวลาและค่าใช้จ่ายในการออกแบบ จึงได้มีการนำมาใช้กันอย่างแพร่หลายในวงการอุตสาหกรรม โดยลักษณะรูปแบบของภาษาวีเอชดีแอลประกอบด้วย 2 ส่วนใหญ่ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล(Sequential Language) และภาษาคอนเคอร์เรนท์ (Concurrent Language) การโปรแกรมด้วยภาษาวีเอชดีแอลสามารถเขียนได้ทั้งสองรูปแบบรวมกัน นอกจากนี้ตัวภาษายังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยเข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้ และสามารถกำหนดรูปแบบไวยากรณ์(Syntax) อีกทั้งยังมีการตรวจสอบความหมายของภาษาว่าจะซิมูเลท(Simulate) ได้หรือไม่ เพราะโปรแกรมที่เขียนโดยวีเอชดีแอลต้องผ่านการซิมูเลทเพื่อตรวจสอบการทำงาน ฉะนั้นในการคอมไพล์(Compile) จะมีการตรวจสอบทั้งวงจรและซีมูเลชั่นซีแมนติก(Semantic) จึงทำให้สะดวกต่อการใช้งาน

การออกแบบนั้นได้ทำการออกแบบให้วงจรบีบอัดข้อมูลรับส่งข้อมูลจากคอมพิวเตอร์ผ่านพอร์ตขนาน โดยใช้ซอฟท์แวร์ในการอ่านข้อมูลจากคอมพิวเตอร์ทั้งข้อมูลภาพและข้อมูลตัวอักษร แล้วทำการบีบอัดข้อมูลผ่าน FPGA เมื่อทำการบีบอัดข้อมูลเสร็จแล้วนำข้อมูลผ่านพอร์ตขนาน เพื่อดูประสิทธิภาพในการบีบอัดข้อมูลตามอัลกอริทึมทั้งข้อมูลภาพและข้อมูลตัวอักษร

จากการออกแบบวงจร ได้ทำการออกแบบและทำการจำลองการทำงานของวงจรทีละส่วน วงจรบีบอัดข้อมูลตัวอักษรนั้นสามารถบีบอัดข้อมูลได้ถูกต้องตามอัลกอริทึมของ Huffman Coding และ ใช้เกต(gate)รวม 2,100 เกต โดยผลการทำงานของวงจรเมื่อทำการทดสอบเปรียบเทียบกับการบีบอัดด้วยซอฟท์แวร์ถูกต้องทุกประการ และวงจรบีบอัดข้อมูลจะทำงานได้เร็วกว่าการบีบอัดข้อมูลด้วยซอฟท์แวร์

ส่วนวงจรบีบอัดข้อมูลภาพนั้นจะทำการออกแบบและทดสอบการทำงาน เฉพาะส่วนวงจรของ DCT โดยจากอัลกอริทึมที่ทำการออกแบบมี 4 อัลกอริทึมได้แก่ A,B,C และ D เมื่อนำข้อมูลภาพมาทำการทดสอบทุกอัลกอริทึมจะเห็นได้ว่า เมื่อมีการเพิ่มบิตขยายมากขึ้น (Extend Bit) ค่า MSE ก็จะลดลง นั่นหมายความว่า ยิ่งเพิ่มบิตขยายมากขึ้น ก็จะทำให้ภาพที่ขยายกลับคืนมีคุณภาพดีขึ้นด้วย ในขนาดรูปภาพขนาดเท่ากัน อัลกอริทึม C มี MSE น้อยที่สุด และ อัลกอริทึม A จะมี MSE มากที่สุด และจากการทดลองทุกอัลกอริทึม จะเห็นได้ว่าว่าอัลกอริทึม C เป็นอัลกอริทึมที่ดีกว่าทั้ง 3 อัลกอริทึม

การพิจาณาในส่วนขนาดของวงจรพบว่า อัลกอริทึม A, B นั้นได้ออกแบบวงจรแบบ 2 ลิฟท์ ส่วนอัลกอริทึม C, D ได้ออกแบบวงจรแบบ 3 ลิฟท์ ซึ่งจะเห็นได้ว่าอัลกอริทึม A, B จะมีขนาดของวงจรเล็กกว่า อัลกอริทึม C, D และในส่วนของอัลกอริทึม A, B วงจรภายในของอัลกอริทึม A มีจำนวนวงจรบวกอยู่มากกว่าอัลกอริทึม B ดังนั้นขนาดของวงจรในอัลกอริทึม B ขนาดเล็กกว่า ดังนั้นในวิทยานิพนธ์นี้จึงเลือกอัลกอริทึม B ในการนำมาออกแบบเป็นวงจรในส่วน DCT โดยทำการออกแบบจะใช้เทคนิคไปไปล์ไลน์ (Pipeline) ซึ่งทำให้ได้ผลการคำนวณของวงจรที่รวดเร็ว ซึ่งในวิทยานิพนธ์นี้ได้ออกแบบโอเปอร์เรชั่นทั้งหมดนั้นเป็นแบบอนุกรมบิต (Bit Serial) เพื่อจะทำให้วงจรที่มีขนาดเล็ก แต่วงจรจะใช้จำนวนสัญญาณนาฬิกา(Clock Cycle) สำหรับการประมวลผลสูง ทำให้วงจรเหมาะกับการใช้งานในลักษณะที่ไม่ต้องการความเร็วในการประมวลผลสูงมากแต่จะได้วงจรที่มีขนาดเล็ก

การทดลองการบีบอัดข้อมูลภาพและตัวอักษรโดยใช้ Huffman Coding ผลจากการทดลองปรากฏว่าเมื่อทำการทดสอบการบีบอัดข้อมูลตัวอักษรนั้น ถ้าไฟล์ข้อมูลมีขนาดใหญ่จะสามารถบีบอัดข้อมูลได้มากกว่าการบีบอัดไฟล์ข้อมูลมีขนาดเล็ก ส่วนข้อมูลรูปภาพได้ทำการบีบอัดข้อมูลรูปภาพประเภท .PPM, .JPG และ .BMP ขนาด 128 x 128 , 256 x 256 และ 512 x 512 พิกเซล โดยข้อมูลรูปภาพ .PPM นั้นถ้ามีขนาดเล็กทำให้สามารถบีบอัดข้อมูลได้ดี ข้อมูลรูปภาพ .JPG ไม่สามารถ

บีบอัดข้อมูลให้เล็กลงได้เนื่องจากไฟล์ชนิดนี้ได้ถูกบีบอัดข้อมูลแล้ว ส่วนข้อมูลรูปภาพ .BMP พบว่าข้อมูลรูปภาพยิ่งมีขนาดเล็กจะทำให้ประสิทธิภาพในการบีบอัดสูง(Compression ratio)

ในส่วนการทดลองลดขนาดข้อมูลรูปภาพและข้อมูลตัวอักษรที่รวมในไฟล์เดียวกัน วิธีการนี้สามารถบีบอัดข้อมูลได้ดีเช่นกัน ซึ่งสามารถนำไปใช้เป็นเทคนิคในการบีบอัดข้อมูลอีกวิธีหนึ่ง จึงกล่าวได้ว่าวิธีการของ Huffman Coding สามารถบีบอัดข้อมูลตัวอักษรร่วมกับข้อมูลภาพในไฟล์เดียวกันได้ดี ถ้าภาพมีขนาดเล็ก ยิ่งทำให้ความสามารถในการบีบอัดข้อมูลดีขึ้น การทดสอบและผลการทดสอบการบีบอัดข้อมูลตัวอักษร ตามอัลกอริทึมของ Huffman Coding ผลการทดลองการบีบอัดข้อมูลด้วยฮาร์ดแวร์เปรียบเทียบกับการบีบอัดข้อมูลด้วยซอฟท์แวร์ ผลปรากฏเป็นไปตามอัลกอริทึม แสดงว่าส่วนของวงจรทำงานได้อย่างถูกต้อง

8.2 ปัญหาที่พบในการทำวิทยานิพนธ์

ปัญหาที่พบในการทำวิทยานิพนธ์

1. จากความตั้งใจที่จะทำการศึกษาวิจัยการบีบอัดข้อมูลตัวอักษร โดยใช้ Huffman Coding และข้อมูลภาพโดยใช้ JPEG ทุกขั้นตอน ซึ่งทำให้งานวิจัยมีขนาดใหญ่เกินไป ทำให้เสียเวลาในการศึกษาและแจกแจงปัญหาของการบีบอัดข้อมูลไปมาก
2. หลังจากเข้าใจอัลกอริทึมในการบีบอัดข้อมูลพอสมควรแล้ว ยังต้องทำการศึกษาถึงวิธีการที่จะหาแนวทางในการที่จะทำให้อัลกอริทึมที่เหมาะสมที่จะนำไปใช้ออกแแบบวงจรรวมซึ่งจะทำให้วงจรมีขนาดเล็กอีกด้วย
3. ในส่วนของการบีบอัดข้อมูลรูปภาพนั้นจะเสียเวลาในการออกแบบวงจรในแต่ละอัลกอริทึม รวมทั้งเวลาในการทดสอบประสิทธิภาพในแต่ละอัลกอริทึม เพื่อเปรียบเทียบผลในการบีบอัดว่าอัลกอริทึมไหนดีที่สุดและมีขนาดของวงจรเล็กที่สุดด้วย

8.2 วิธีการแก้ปัญหา

1. ถ้าต้องการศึกษาวิจัยการบีบอัดข้อมูลตัวอักษร และ ข้อมูลภาพ ต้องมั่นใจว่ามีเวลามากเพียงพอต่อการทำการวิจัย
2. ต้องเริ่มศึกษาอย่างเป็นขั้นตอน ตั้งแต่ภาพรวมทั้งหมดในการบีบอัดข้อมูลว่ามีขั้นตอนและวิธีการอย่างไร เพื่อให้เห็นภาพรวมของงาน แล้วถึงลงในรายละเอียดของแต่ละส่วน เพื่อเจาะลึกลงไปว่า ถ้าต้องการนำไปออกแบบวงจรรวม แล้วจะมีแนวทางอะไรที่จะช่วยทำให้สามารถออกแบบได้ง่ายและเร็วขึ้น

8.3 ข้อเสนอแนะและการพัฒนา

ในการออกแบบวงจรนั้นค่อนข้างยุ่งยากมากเนื่องจากจะต้องมีแยกการออกแบบวงจรออกเป็น 2 ส่วน คือส่วนที่เป็นข้อมูลตัวอักษร และ ส่วนที่เป็นข้อมูลภาพ ดังนั้นเพื่อลดความซับซ้อนดังกล่าวจึงควรทำการออกแบบวงจรที่สามารถบีบอัดได้ทั้งข้อมูลรูปภาพและตัวอักษร แล้วศึกษาถึงผลลัพท์ที่ได้จากการบีบอัดแล้วเปรียบเทียบกับการแยกบีบอัดในแต่ละส่วนโดยดูประสิทธิภาพในการบีบอัด (Compression ratio) ว่าแตกต่างกันอย่างไร และ ทำการเปรียบเทียบขนาดของวงจรว่าแตกต่างกันอย่างไร เหมาะกับงานประเภทไหน เช่น ในระบบสมาร์ทการ์ด ต้องการวงจรมีขนาดเล็กและประสิทธิภาพของการบีบอัดต้องสูงด้วย จะต้องทำอย่างไร

เอกสารอ้างอิง

[1] T.A.Welch, " A technique for high-performance data compression", Computer,vol. 17,pp. 8-19,1984.

[2] K. Parhi, "High-speed VLSI Architectures for Huffman and Viterbi decoders," IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 29, no. 6,pp. 385-391, June,1992.

[3] W.H. Chen, C.H. Smith, and S.C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Tranform", IEEE Trans.Commun.,Vol. COM-25, pp. 1004-1009, Sept. 1977.

[4] B.G. Lee, "A New Algorithm to Compute the Discrete Cosine Transform", IEEE Trans Acoust., Speech, and Signal Processing , Vol. ASSP-32, pp. 1243-1245, Dec. 1984.

[5] N.I Cho and S.u. Lee, "Fast Algorithm and Implementation of 2-D Discrete Cosine Transform",IEEE Trans .Circuits and system, Vol. 38, No. 3, pp. 297-305, Mar. 1991

[6] K. Ogawa et al., "A single Chip Compression/Decompression LSI Based On JPEG",IEEE Trans. On Consumer Electronics,vol. 38, pp. 703-710, Aug. 1992.

[7] M. Nakagawa et al., "DCT-Based still Image Compression Ics with Bit-Rate Control",IEEE Trans. On Consumer Electronics, vol. 38, pp. 711-717, Aug. 1992.

[8] Trac D. Tran, "The BinDCT: Fast Multiplierless Approximation of the DCT",IEEE signal processing letters, vol. 7, No. 6 , June. 2000.

[9] Jie Liang and Trac D. Tran, "Fast Multiplierless Approximation of the DCT with the Lifting Scheme",IEEE Transaction on signal processing , vol. 49, No. 12 , December. 2001.

[10] S. Choomchuay, "On the Implementation of Finite Field Basis Conversions", EECON-17, King Mongkut's Institute of Technology, Nort Bangkok,1994,pp. 482-486.

[11] K. Hadkhuntod , S. Pongyupinnpanich and S. Choomchuay, "An Application of ElGamall Elliptic Curve Cryptosystem to Smart Card", การประชุมวิชากาทางวิศวกรรมไฟฟ้า ครั้งที่ 25 (EECON-25) ,มหาวิทยาลัยสงขลานครินทร์ , หน้า 46-50.

ภาคผนวก

ภาคผนวก ก
โปรแกรม VHDL

1. Top level module support code to interface with lower level module

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY huffman IS
  PORT (
      Clk          :        IN     STD_LOGIC;
      Resetn       :        IN     STD_LOGIC;
      Go           :        IN     STD_LOGIC;
      Done         :        OUT    STD_LOGIC;
      WEn          :        OUT    STD_LOGIC;
      Address      :        OUT    STD_LOGIC_VECTOR (17 DOWNTO 0);
      DataIn       :        IN     STD_LOGIC_VECTOR (7 DOWNTO 0);
      DataOut      :        OUT    STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END huffman;

ARCHITECTURE huffman_arch OF huffman IS

COMPONENT control
 PORT(
    Clk              :   IN     STD_LOGIC;
    Resetn           :   IN     STD_LOGIC;
    Go               :   IN     STD_LOGIC;
    EncodeLength     :   IN     STD_LOGIC_VECTOR(2 DOWNTO 0);
    DoneOut          :   OUT    STD_LOGIC;
    LoadStateOut     :   OUT    STD_LOGIC;
    ShiftStateOut    :   OUT    STD_LOGIC;
    ReadAddressOut   :   OUT    STD_LOGIC_VECTOR (5 DOWNTO 0);
    WriteAddressOut  :   OUT    STD_LOGIC_VECTOR (5 DOWNTO 0);
    WEn              :   OUT    STD_LOGIC
```

```
);
END COMPONENT;

COMPONENT huff
 PORT(
        DataIn      :   IN      STD_LOGIC_VECTOR (7 DOWNTO 0);
        Length      :   OUT     STD_LOGIC_VECTOR (2DOWNTO 0);
        DataOut     :   OUT     STD_LOGIC_VECTOR (7 DOWNTO 0);
END COMPONENT;

SIGNAL EncodeLength         :       STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL dgo                  :       STD_LOGIC ;
SIGNAL Count                :       STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL LoadState, ShiftState :      STD_LOGIC;

--**previously commented out


SIGNAL UncodeData, EncodeData   :       STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL ShiftData                :       STD_LOGIC_VECTOR (6 DOWNTO 0);
SIGNAL WriteAddress, ReadAddress :      STD_LOGIC_VECTOR (5 DOWNTO 0);
SIGNAL TempDataOut              :       STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN

CTRL : control  PORT MAP(
                Clk             =>      Clk,
                Resetn          =>      Resetn,
                Go              =>      Go,
                EncodeLength    =>      EncodeLength,
                DoneOut         =>      one,
                LoadStateOut    =>      LoadState,
                ShiftStateOut   =>      ShiftState,
                ReadAddressOut  =>      ReadAddress,
                WriteAddressOut =>      WriteAddress,
                WEn             =>      WEn);
```

```
ENC : huff PORT MAP(
                    DataIn          =>      UncodeData,
                    Length          =>      EncodeLength,
                    DataOut         =>      EncodeData);

                    DataOut         <=      TempDataOut;

PROCESS(Clk)
 BEGIN
   IF( Clk'event   AND     Clk='1') THEN
        IF     (LoadState =         '1')  THEN
               ShiftData  <=        EncodeData(7 DOWNTO 1);
        ELSIF (ShiftState =         '1')  THEN
               ShiftData  <=        '-' & ShiftData(6 DONTO 1);
        ENDIF;

        IF  (LoadState              =       '1')  THEN
               TempDataOut(7)  <=   EncodeData(0);
        ELSEIF  (ShiftState         =       '1') THEN
               TempDataOut(7)  <=   ShiftData(0);
        END IF;

        IF ((LoadState    =   '1'    AND    Go = '1')  OR  ShiftState   = '1' ) THEN
               TempDataOut(6 DOWNTO 0<= TempDataOut(7 DOWNTO 1);
        END IF;

        IF (LoadState = '1')  THEN
            UncodeData  <=      DataIn;
        END IF;
   END IF;
END PROCESS;
```

```
ADDRESS(17 DOWNTO 6) <= "000000000000";

PROCESS   (LoadState, ShiftState, WriteAddress, ReadAddress)
 BEGIN
    IF     (LoadState = '1'  OR  ShiftState = '1') THEN
           ADDRESS(5 DOWNTO 0)              <=          ReadAddress;
    ELSE
           ADDRESS(5 DOWNTO 0)              <=          WriteAddress;
    END IF;
END PROCESS;

END  huffman_arch;

CONFIGURATION config_huffman OF huffman IS
  FOR huffman_arch
  END FOR;
END config_huffman;
```

2 *The control system ; keep track of memory address to use when reading and writing data to memory*

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC _ARITH.ALL;
USE IEEE.STD_LOGIC _UNSIGNED.ALL;

ENTITY control IS
  PORT (
      Clk                 :   IN    STD_LOGIC;
      Resetn              :   IN    STD_LOGIC;
      Go                  :   IN    STD_LOGIC;
      EncodeLength        :   IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
      DoneOut             :   OUT   STD_LOGIC;
      LoadStateOut        :   OUT   STD_LOGIC ;
      ShIFtStateOut       :   OUT   STD_LOGIC ;
      ReadAddressOut      :   OUT   STD_LOGIC_VECTOR(5 DOWNTO 0);
      WriteAddressOut     :   OUT   STD_LOGIC_VECTOR(5 DOWNTO 0);
      WEn                 :   OUT   STD_LOGIC
  );
END control;

ARCHITECTURE control_arch OF control IS

COMPONENT state
 PORT( Clk          :    IN   STD_LOGIC;
       Resetn       :    IN   STD_LOGIC;
       Go           :    IN   STD_LOGIC;
    EncodeLength    :    IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
```

```
    Length      :   IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
    Count       :   IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
    Term        :   IN        STD_LOGIC;
    LoadState   :   OUT       STD_LOGIC;
    ShIFtState  :   OUT       STD_LOGIC;
    WriteState2 :   OUT       STD_LOGIC;
    WriteState3 :   OUT       STD_LOGIC
);
END COMPONENT;

SIGNAL Length       :       STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL Count        :       STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL LoadState    :       STD_LOGIC;
SIGNAL ShIFtState   :       STD_LOGIC;
SIGNAL WriteState2  :       STD_LOGIC;
SIGNAL WriteState3  :       STD_LOGIC;
SIGNAL WriteAddress, ReadAddress : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL Done         :       STD_LOGIC;



SIGNAL wrt2 : STD_LOGIC ;

BEGIN

        LoadStateOut        <=      LoadState;
        ShIFtStateOut       <=      ShIFtState;
        WriteAddressOut     <=      WriteAddress;
        ReadAddressOut      <=      ReadAddress;
        DoneOut             <=      Done;

STATE_MACH : state PORT MAP(Clk => Clk,
```

```
                Resetn          =>      Resetn,
                Go              =>      Go,
                EncodeLength    =>      EncodeLength,
                Length          =>      Length,
                Count           =>      Count,
                Term            =>      Done,
                LoadState       =>      LoadState,
                ShIFtState      =>      ShIFtState,
                WriteState2     =>      WriteState2,
                WriteState3     =>      WriteState3);

PROCESS(WriteState2)

 BEGIN
    wrt2 <= WriteState2;
    IF   (wrt2 = '1')       THEN
                WEn <= '0';
    ELSE
                WEn <= '1';
    END IF;
END PROCESS;

PROCESS(Clk,Resetn)
 BEGIN
    IF    (Resetn = '0')    THEN
                ReadAddress     <=      (others => '0');
                WriteAddress    <=      (others => '0');
                Count           <=      (others => '0');
                Length          <=      (others => '0');
                Done            <=      '0';
    ELSIF (Clk'event and Clk='1') THEN
                IF          (Go = '1')          THEN
```

```
        IF      (LoadState = '1')  THEN
                ReadAddress   <=      ReadAddress + '1';
        END IF;
        IF (WriteState3 = '1') THEN
                WriteAddress   <=      WriteAddress + '1';
        END IF;
        IF (ShIFtState = '1' or LoadState = '1') THEN
                Count          <=      Count + '1';
        END IF;
        IF (LoadState = '1') THEN
                Length         <=      EncodeLength;
        ELSIF (ShIFtState = '1') THEN
                Length         <=      Length - '1';
        END IF;
      END IF;
      IF (ReadAddress = "111111") THEN
              Done     <=      '1';
      END IF;
    END IF;
END process;

end control_arch;

CONFIGURATION config_control OF control IS
  FOR control_arch
  END FOR;
END config_control;
```

3. ***Simple Huffman Lookup Table***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY huff IS
  PORT (
            DataIn   :  IN     STD_LOGIC_VECTOR(7 DOWNTO 0);
            Length   :  OUT    STD_LOGIC_VECTOR(2 DOWNTO 0);
            DataOut  :  OUT    STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END huff;

ARCHITECTURE huff_arch OF huff IS
BEGIN

 PROCESS(DataIn)
   BEGIN
     CASE DataIn IS
        WHEN "00011111" =>
                Length          <=      "101";
                DataOut         <=      "00011110";
        WHEN "00011110" =>
                Length          <=      "101";
                DataOut         <=      "00011111";
        WHEN "00011010" =>
                Length          <=      "101";
                DataOut         <=      "00100110";
        WHEN "00011101" =>
                Length          <=      "101";
                DataOut         <=      "00100111";
        WHEN "00011011" =>
```

```
            Length      <=      "101";
            DataOut     <=      "00101000";
WHEN "00010111" =>
            Length      <=      "101";
            DataOut     <=      "00101001";
WHEN "00010101" =>
            Length      <=      "101";
            DataOut     <=      "00101010";
WHEN "00011100" =>
            Length      <=      "101";
            DataOut     <=      "00101011";
WHEN "00011001" =>
            Length      <=      "101";
            DataOut     <=      "00110000";
WHEN "00010100" =>
            Length      <=      "101";
            DataOut     <=      "00110001";
WHEN "00011000" =>
            Length      <=      "101";
            DataOut     <=      "00110100";
WHEN "00010011" =>
            Length      <=      "101";
            DataOut     <=      "00110101";
WHEN "00001111" =>
            Length      <=      "101";
            DataOut     <=      "00110110";
WHEN "00010110" =>
            Length      <=      "101";
            DataOut     <=      "00110111";
WHEN "00010010" =>
            Length      <=      "100";
            DataOut     <=      "00000000";
```

```
WHEN "00010001" =>
        Length      <=      "100";
        DataOut     <=      "00000001";
WHEN "00001110" =>
        Length      <=      "100";
        DataOut     <=      "00000110";
WHEN "00001101" =>
        Length      <=      "100";
        DataOut     <=      "00000111";
WHEN "00010000" =>
        Length      <=      "100";
        DataOut     <=      "00001000";
WHEN "00001011" =>
        Length      <=      "100";
        DataOut     <=      "00001001";
WHEN "00001100" =>
        Length      <=      "100";
        DataOut     <=      "00001100";
WHEN "00001010" =>
        Length      <=      "100";
        DataOut     <=      "00001101";
WHEN "00000111" =>
        Length      <=      "100";
        DataOut     <=      "00001110";
WHEN "00001001" =>
        Length      <=      "100";
        DataOut     <=      "00010000";
WHEN "00001000" =>
        Length      <=      "100";
        DataOut     <=      "00010001";
WHEN "00000110" =>
        Length      <=      "100";
```

```
                DataOut     <=      "00010010";
        WHEN "00000101" =>
                Length      <=      "100";
                DataOut     <=      "00011001";
        WHEN "00000100" =>
                Length      <=      "011";
                DataOut     <=      "00000001";
        WHEN "00000011" =>
                Length      <=      "011";
                DataOut     <=      "00000010";
        WHEN "00000010" =>
                Length      <=      "011";
                DataOut     <=      "00000101";
        WHEN "00000001" =>
                Length      <=      "011";
                DataOut     <=      "00001011";
        WHEN "00000000" =>
                Length      <=      "010";
                DataOut     <=      "00000111";
        WHEN others =>
                Length      <=      "---";
                DataOut     <=      "--------";
    END CASE;
  END PROCESS;
END huff_arch;

CONFIGURATION config_huff OF huff IS
  FOR huff_arch
  END FOR;
END config_huff;
```

4. State machine which loads and compile 8-bit sequences of huffman codes and write output to memory

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSINED.ALL;

ENTITY state IS
  PORT (
        Clk             :       IN    STD_LOGIC;
        Resetn          :       IN    STD_LOGIC;
        Go              :       IN    STD_LOGIC;
        EncodeLength    :       IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
        Length          :       IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
        Count           :       IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
        Term            :       IN    STD_LOGIC;
        LoadState       :       OUT   STD_LOGIC;
        ShIFtState      :       OUT   STD_LOGIC;
        WriteState2     :       OUT   STD_LOGIC;
        WriteState3     :       OUT   STD_LOGIC);
END state;

ARCHITECTURE state_arch OF state IS
TYPE states IS (load, shIFt, write1, write2, write3, done);
SIGNAL current_state, next_state : states;
SIGNAL WriteState1, DoneState : STD_LOGIC;

BEGIN
statetrans: PROCESS(Clk,Resetn)
```

```
BEGIN
    IF (Resetn = '0') THEN
                current_state    <=      load;
    ELSIF       (Clk'event  AND  Clk='1') THEN
        IF (Go = '1') THEN
                current_state    <=      next_state;
        END IF;
    END IF;
END PROCESS;

state: PROCESS(current_state, EncodeLength, count, length, term)
BEGIN
        LoadState       <=      '0';
        ShIFtState      <=      '0';
        WriteState1     <=      '0';
        WriteState2     <=      '0';
        WriteState3     <=      '0';
        DoneState       <=      '0';
CASE  (current_state) IS
  WHEN   load =>
    IF (Count = "111"   OR  (EncodeLength = "000" AND Term = '1')) THEN
            next_state          <=      write1;
    ELSIF (EncodeLength = "000") THEN
            next_state          <=      load;
    ELSE
            next_state          <=      shIFt;
    END IF;
          LoadState             <=      '1';
  WHEN    shIFt =>
    IF (Count = "111" OR (Length = "001" AND Term = '1')) THEN
            next_state          <=      write1;
    ELSIF (Length = "001") THEN
```

```
                next_state          <=      load;
    ELSE
                next_state          <=      shIFt;
    END IF;
                ShIFtState          <=      '1';
  WHEN    write1 =>
                next_state          <=      write2;
                WriteState1         <=      '1';
  WHEN     write2 =>
                next_state          <=      write3;
                WriteState2         <=      '1';
  WHEN     write3 =>
    IF (Term = '1'  AND  Length = "000") THEN
                next_state          <=      done;
    ELSIF (Length = "000") THEN
                next_state          <=      load;
    ELSE
                next_state          <=      shIFt;
    END IF;
                WriteState3         <=      '1';
  WHEN    done =>
                next_state          <=      done;
                DoneState           <=      '1';
  WHEN    others =>
                next_state          <=      done;
  END CASE;
END PROCESS;
END state_arch;
CONFIGURATION config_state OF state IS
  FOR state_arch
  END FOR;
END config_state;
```

5. ไฟล์ที่ใช้ทดสอบการทำงานของวงจร

```
USE STD.TEXIO.ALL;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;

ENTITY tb_huffman IS

END tb_huffman;

ARCHITECTURE structure OF tb_huffman IS

 SIGNAL Resetn     :  STD_LOGIC;
 SIGNAL Clk        :  STD_LOGIC;
 SIGNAL DataIn     :  STD_LOGIC_VECTOR(7 DOWNTO 0);
 SIGNAL Go         :  STD_LOGIC;
 SIGNAL DataOut    :  STD_LOGIC_VECTOR(7 DOWNTO 0);
 SIGNAL Address    :  STD_LOGIC_VECTOR(17 DOWNTO 0);
 SIGNAL Wen        :  STD_LOGIC;
 SIGNAL Done       :  STD_LOGIC;

 CONSTANT   PERIOD          :      time := 0.00000005 ns;
 CONSTANT   HALF_PERIOD     :      time := PERIOD / 2;
 CONSTANT   SETTLING_TIME:         time := PERIOD / 5;
 COMPONENT  huffman
 PORT( Resetn     : IN  STD_LOGIC;
       Clk        : IN  STD_LOGIC;
       DataIn     : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
       Go         : IN  STD_LOGIC;
       DataOut    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
       Address    : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);
       Wen        : OUT STD_LOGIC;
```

```
        Done        : OUT STD_LOGIC);
END COMPONENT;
BEGIN

UUT   :        huffman
 PORT MAP(Resetn, Clk, DataIn, Go, DataOut, Address, Wen, Done);

STIMULUS_RESPONSE :        PROCESS

file TV : TEXT is in "huffman.tv";
 VARIABLE    L              :        LINE;
 VARIABLE    RESETN_IN      :        STD_LOGIC;
 VARIABLE    GO_IN          :        STD_LOGIC;
 VARIABLE    DATAIN_IN      :        STD_LOGIC_VECTOR(7 DOWNTO 0);

 BEGIN

 --Get a vector
  READLINE    (TV,L);
  READ        (L,RESETN_IN);
  READ        (L,GO_IN);
  READ        (L,DATAIN_IN);
  IF endfile(TV) THEN   wait;
  END IF;

  CLK         <=       '0';
  WAIT FOR HALF_PERIOD - SETTLING_TIME;

 --Assign input values
  RESETN      <=       RESETN_IN;
  GO          <=       GO_IN;
  DATAIN      <=       DATAIN_IN;


 --Clock edge
  WAIT FOR SETTLING_TIME;
```

```
    CLK          <=       '1';

    WAIT  FOR  HALF_PERIOD;

  END PROCESS;
END  STRUCTURE;

CONFIGURATION  CFG_tb_huffman_structure  OF  tb_huffman  IS

  FOR  STRUTURE
    FOR  UUT  :  huffman
      USE configuration WORK.config_huffman;
    END  FOR;
  END  FOR;

END  CFG_tb_huffman_structure;
```

ภาคผนวก ข

โปรแกรมภาษา C

```
/* dct_33b.c DCT & IDCT lift-off structure, Scaling-Descaling, 1-2-3-4 bit trucation
(extension) */
/* 1-array, 7 September 2002, run argument*/
#include <stdio.h>
#include <math.h>
#include <malloc.h>
FILE *fp_img, *fp_opt;
char img_file[15],opt_file[15];
float  *creat_array(int);
float  *f,*g;
int  length,size,trc=3;                    /* extension_bit 0->4; 9= no truncation */
float dmax=(float)0,dmin=(float)0;
void mag_rec(float);
float truncat(float,int);

void main(argc,argv)
int argc;
char *argv[];
{
 float p[8],q[8];   /* DCT Size */
 float mse,diff,sf=(float)1, dsf=(float)16;   /* external caling & Deacaling factor */
 int i,j,k,m,n,length;
 void  argument_chk(int,char **);
 int  file_read(FILE *);
 void forward_b(float *,float*),forward_c(float *,float*),forward_d(float *,float*);
 void invert_b(float *,float*),invert_c(float *,float*),invert_d(float *,float*);
   argument_chk(argc,argv);
   length = file_read(fp_img);
   printf("\n Data from %s as %d pixel data read in",img_file,length);
   for(i=0;i<length;i++){ f[i]=f[i]/sf;}
   printf("\n - Data pre-scaling PASS\n");
         g=creat_array(length);for(i=0;i<length;i++){g[i]=(float)0;}
       k=0;
       while(k<length) {
        i=k%8; m=(k-1)/8; p[i]=f[k];
```

```
 if(i==7){ forward_d(p,q);     for(j=0;j<8;j++){n=8*m+j; g[n]=q[j];}}
 k++;
 }
printf("\n - Row Transform PASS\n");

 m=size/8;
 for(n=0;n<size;n++){
  for(i=0;i<m;i++) {
                    for(j=0;j<8;j++) {k=(8*i*size)+(j*size)+n; p[j]=g[k];}
                    forward_d(p,q);
   for(j=0;j<8;j++) {k=(8*i*size)+(j*size)+n; g[k]=q[j];}
                   }
                  }
 printf("\n - Column Transform PASS\n");

 m=size/8;
 for(n=0;n<size;n++){
  for(i=0;i<m;i++) {
                    for(j=0;j<8;j++) {k=(8*i*size)+(j*size)+n; p[j]=g[k];}
                    invert_d(p,q);
   for(j=0;j<8;j++) {k=(8*i*size)+(j*size)+n; g[k]=q[j];}
                   }
                  }
 printf("\n - Column Inverse Transform PASS\n");

 k=0;
 while(k<length) {
  i=k%8; m=(k-1)/8; p[i]=g[k];
  if(i==7){ invert_d(p,q);       for(j=0;j<8;j++){n=8*m+j; g[n]=q[j];}}
 k++;
 }
 printf("\n - Row Inverse Transform PASS\n");
for(i=0;i<length;i++){ g[i]=g[i]*dsf;}
printf("\n - Output Descaling PASS\n");
 for(i=0;i<length;i++){
```

```
        diff=f[i]-g[i];
        mse=mse+diff*diff;
      }
      mse=mse/(size*size);
     printf("\n - MSE Calculation PASS\n");
     if((fp_opt = fopen(opt_file,"w"))==NULL) {
       printf("\n-Imaga data file not be open"); exit();
     }
            fprintf(fp_opt,"P2\n# File: %s with %d extended bit(s) \n%d
%d\n255\n",img_file,trc,size, size);
     for(i=0;i<length;i++){fprintf(fp_opt,"%d ",(int)g[i]); if(i%size==size-1){fprintf
(fp_opt,"\n");} }
     fclose(fp_opt);
           printf("\n Scaling factor (sf) = %f",sf);
           printf("\n Descaling factor (dsf) = %f",dsf);
    printf("\n Max. intermediate data = %f",dmax);
           printf("\n Min. intermediate data = %f",dmin);
           printf("\n Number of Extension bits (trc) = %d",trc);
           printf("\n Mean Square Errors (mse) = %f",mse);
}
/************/
void argument_chk(int gc,char **gv)
 {
          char  chk[1]=".",iext[4]=".txt",exten[4]=".out";
          char  blk[15]="";
            if(gc==1) {
                              printf("\n Usage :");
                              printf("\n    dct_nn (inputfile  outputfile) or");
                              printf("\n    dct_nn (inputfile )\n");
                              exit(00);
                             }
          else if(gc==2) {
                                    strcpy(img_file,gv[1]);
                                    if((fopen(img_file,"r"))==NULL) {
                                      printf("\n%s -file not found!!",img_file);
```

```
                                        printf("\nLooking for %s.txt",img_file);
                                        strncat(blk,img_file,strcspn(img_file,chk));
                                        strncat(blk,iext,4);
                                        strcpy(img_file,blk);
                                      }
                                      strcpy(blk,"");
                                      strncat(blk,img_file,strcspn(img_file,chk));
                                      strncat(blk,exten,4);
                                      strcpy(opt_file,blk);
                                     }
            else if(gc==3) {
                        strcpy(img_file,gv[1]);
                        strcpy(opt_file,gv[2]); }
            else {printf("\nformat specification error!!"); exit(0);}
  }
 /*****************/
 int file_read (FILE *fp_img)
 {
   int k,dat;
     if((fp_img = fopen(img_file,"r"))==NULL) {
       printf("\n-Imaga data file not found"); exit();
     }
       k=0;
       while(fscanf(fp_img,"%d",&dat)!=EOF) k++;
              size=(int)sqrt(k);
                    if(size*size!=k) {printf("\n%d wrong number of data to process!"); exit();}
                    printf("\n Image frame size = %d",size);
           f=creat_array(k);
           fseek(fp_img,0,0);
       k=0;
       while(fscanf(fp_img,"%d",&dat)!=EOF) {f[k]=(float)dat;k++;}
           fclose(fp_img);
           return(k);
 }
 /*************/
```

```
float truncat(float u,int m)
{
 float w,z;
 if(u<0){z=(int)u-u;}
 else {z=u-(int)u;}
 if (m==9) /* no trucation; unlimited extension */
 {       w=z;}
 else if (m==4) /* 4 bits extension bit */
 {
  if(z<0.0625) w=(float)0;
        else if((z>=0.0625)&&(z<0.125))   w=(float)0.0625;
        else if((z>=0.125)&&(z<0.1875))   w=(float)0.125;
        else if((z>=0.1875)&&(z<0.25))    w=(float)0.1875;
        else if((z>=0.25)&&(z<0.3125))    w=(float)0.25;
        else if((z>=0.3125)&&(z<0.375))   w=(float)0.3125;
        else if((z>=0.375)&&(z<0.4375))   w=(float)0.375;
        else if((z>=0.4375)&&(z<0.50))    w=(float)0.4375;
        else if((z>=0.50)&&(z<0.5625))    w=(float)0.50;
        else if((z>=0.5625)&&(z<0.625))   w=(float)0.5625;
        else if((z>=0.625)&&(z<0.6875))   w=(float)0.625;
        else if((z>=0.6875)&&(z<0.75))    w=(float)0.6875;
        else if((z>=0.75)&&(z<0.875))     w=(float)0.75;
        else if((z>=0.875)&&(z<0.9375))   w=(float)0.875;
        else w=(float)0.9375;
}
else if (m==3) /* 3 bits extension bit */
{
 if(z<0.125) w=(float)0;
        else if((z>=0.125)&&(z<0.25))     w=(float)0.125;
        else if((z>=0.25)&&(z<0.375))     w=(float)0.25;
        else if((z>=0.375)&&(z<0.5))      w=(float)0.375;
        else if((z>=0.5)&&(z<0.625))      w=(float)0.5;
        else if((z>=0.625)&&(z<0.75))     w=(float)0.625;
        else if((z>=0.75)&&(z<0.875))     w=(float)0.75;
        else w=(float)0.875;
```

```
  }
  else if(m==2)     /* 2 bits extension bit */
  {
    if(z<0.25) w=(float)0;
          else if((z>=0.25)&&(z<0.50))        w=(float)0.25;
          else if((z>=0.50)&&(z<0.75))        w=(float)0.50;
          else w=(float)0.75;
  }
  else if(m==1)     /* 1 bit extension bit */
  {
          if(z<0.5) w=(float)0;
          else w=(float)0.5;
  }
  else      /* zero extension bit */
  {w=(float)0;}
  if(u<0){w=(int)u-w;}
  else {w=(int)u+w;}
  return (w);
 }
/**************/
/**************/
void forward_d (float *x,float *y)
{
 float a,b;
 float p1=(float)0.5, u1=(float)0.375;
 float p2=(float)0.875, u2=(float)0.5;
 float p3=(float)0.1875, u3=(float)0.25;
 float p4=(float)0.4375,u4=(float)0.75;
 /*float p3=(float)0.25, u3=(float)0.25;
 float p4=(float)0.5,u4=(float)0.75;*/
 float p5=(float)0.375;
 int i;
 for(i=0;i<8;i++){x[i]=x[i]/4;} /* input data scaling followed by bty scaling*/
 a=x[0]+x[7]; b=x[0]-x[7];x[0]=a/2; x[7]=b/2; x[0]=truncat(x[0],trc); x[7]=truncat(x[7],trc);
mag_rec(x[0]); mag_rec(x[7]);
```

```
 a=x[1]+x[6]; b=x[1]-x[6];x[1]=a/2; x[6]=b/2; x[1]=truncat(x[1],trc); x[6]=truncat(x[6],trc);
mag_rec(x[1]); mag_rec(x[6]);
 a=x[2]+x[5]; b=x[2]-x[5];x[2]=a/2; x[5]=b/2; x[2]=truncat(x[2],trc); x[5]=truncat(x[5],trc);
mag_rec(x[2]); mag_rec(x[5]);
 a=x[3]+x[4]; b=x[3]-x[4];x[3]=a/2; x[4]=b/2; x[3]=truncat(x[3],trc); x[4]=truncat(x[4],trc);
mag_rec(x[3]); mag_rec(x[4]);
 /* insert pipeline state here */
 x[5]=x[5]-p4*x[6];          x[5]=truncat(x[5],trc); mag_rec(x[5]);
 x[6]=x[6]+u4*x[5];          x[6]=truncat(x[6],trc); mag_rec(x[6]);
 x[5]=p5*x[6]-x[5];          x[5]=truncat(x[5],trc); mag_rec(x[5]);
 /*****/
 a=x[0]+x[3]; b=x[0]-x[3];x[0]=a/2; x[3]=b/2; x[0]=truncat(x[0],trc); x[3]=truncat(x[3],trc);
mag_rec(x[0]); mag_rec(x[3]);
 a=x[1]+x[2]; b=x[1]-x[2];x[1]=a/2; x[2]=b/2; x[1]=truncat(x[1],trc); x[2]=truncat(x[2],trc);
mag_rec(x[1]); mag_rec(x[2]);
 a=x[4]+x[5]; b=x[4]-x[5];x[4]=a/2; x[5]=b/2; x[4]=truncat(x[4],trc); x[5]=truncat(x[5],trc);
mag_rec(x[4]); mag_rec(x[5]);
 a=x[7]+x[6]; b=x[7]-x[6];x[7]=a/2; x[6]=b/2; x[7]=truncat(x[7],trc); x[6]=truncat(x[6],trc);
mag_rec(x[7]); mag_rec(x[6]);
 /*insert pipelinehere */
 x[4]=p3*x[7]-x[4];          x[4]=truncat(x[4],trc); mag_rec(x[4]);
 x[7]=x[7]-u3*x[4];          x[7]=truncat(x[7],trc); mag_rec(x[7]);
 x[5]=x[5]+p2*x[6];          x[5]=truncat(x[5],trc); mag_rec(x[5]);
 x[6]=x[6]-u2*x[5];          x[6]=truncat(x[6],trc); mag_rec(x[6]);
x[0]=x[0]+x[1];   x[0]=truncat(x[0],trc); mag_rec(x[0]);
x[1]=x[0]/2-x[1]; x[1]=truncat(x[1],trc); mag_rec(x[1]);
x[2]=p1*x[3]-x[2];          x[2]=truncat(x[2],trc); mag_rec(x[2]);
x[3]=x[3]-u1*x[2];          x[3]=truncat(x[3],trc); mag_rec(x[3]);
/**o/p map **/
y[0]=x[0];
y[4]=x[1];
y[6]=x[2];
y[2]=x[3];
y[7]=x[4];
y[5]=x[5];
```

```
 y[3]=x[6];
 y[1]=x[7];
}
/***********/
void invert_d(float *y, float *x)
{
 float a,b;
 float p1=(float)0.5, u1=(float)0.375;
 float p2=(float)0.875, u2=(float)0.5;
 float p3=(float)0.1875, u3=(float)0.25;
 float p4=(float)0.4375,u4=(float)0.75;
 /*float p3=(float)0.25, u3=(float)0.25;
 float p4=(float)0.5,u4=(float)0.75;*/
 float p5=(float)0.375;

y[4]=y[0]/2-y[4];  x[4]=truncat(x[4],trc); mag_rec(x[4]);
y[0]=y[0]-y[4];       x[0]=truncat(x[0],trc); mag_rec(x[0]);
y[2]=y[2]+u1*y[6]; x[2]=truncat(x[2],trc); mag_rec(x[2]);
y[6]=p1*y[2]-y[6]; x[6]=truncat(x[6],trc); mag_rec(x[6]);
y[3]=y[3]+u2*y[5]; x[3]=truncat(x[3],trc); mag_rec(x[3]);
y[5]=y[5]-p2*y[3]; x[5]=truncat(x[5],trc); mag_rec(x[5]);
y[1]=y[1]+u3*y[7]; x[1]=truncat(x[1],trc); mag_rec(x[1]);
y[7]=p3*y[1]-y[7]; x[7]=truncat(x[7],trc); mag_rec(x[7]);
/*pipe line
y[4]=2*y[4]; y[6]=2*y[6]; y[0]=2*y[0]; y[2]=2*y[2];
y[7]=2*y[7]; y[5]=2*y[5]; y[1]=2*y[1]; y[3]=2*y[3];    */
a=y[4]+y[6]; b=y[4]-y[6];y[4]=a; y[6]=b; x[4]=truncat(x[4],trc); x[6]=truncat(x[6],trc);
mag_rec(x[4]); mag_rec(x[6]);
a=y[0]+y[2]; b=y[0]-y[2];y[0]=a; y[2]=b; x[0]=truncat(x[0],trc); x[2]=truncat(x[2],trc);
mag_rec(x[0]); mag_rec(x[2]);
a=y[7]+y[5]; b=y[7]-y[5];y[7]=a; y[5]=b; x[7]=truncat(x[7],trc); x[5]=truncat(x[5],trc);
mag_rec(x[7]); mag_rec(x[5]);
a=y[1]+y[3]; b=y[1]-y[3];y[1]=a; y[3]=b; x[1]=truncat(x[1],trc); x[3]=truncat(x[3],trc);
mag_rec(x[1]); mag_rec(x[3]);
/***/
```

```
 y[5]=p5*y[3]-y[5]; x[5]=truncat(x[5],trc); mag_rec(x[5]);
 y[3]=y[3]-u4*y[5]; x[3]=truncat(x[3],trc); mag_rec(x[3]);
 y[5]=y[5]+p4*y[3]; x[5]=truncat(x[5],trc); mag_rec(x[5]);
 /**/
 a=y[2]+y[7]; b=y[2]-y[7];y[2]=a; y[7]=b; x[2]=truncat(x[2],trc); x[7]=truncat(x[7],trc);
 mag_rec(x[2]); mag_rec(x[7]);
 a=y[6]+y[5]; b=y[6]-y[5];y[6]=a; y[5]=b; x[6]=truncat(x[6],trc); x[5]=truncat(x[5],trc);
 mag_rec(x[6]); mag_rec(x[5]);
 a=y[4]+y[3]; b=y[4]-y[3];y[4]=a; y[3]=b; x[4]=truncat(x[4],trc); x[3]=truncat(x[3],trc);
 mag_rec(x[4]); mag_rec(x[3]);
 a=y[0]+y[1]; b=y[0]-y[1];y[0]=a; y[1]=b; x[0]=truncat(x[0],trc); x[1]=truncat(x[1],trc);
 mag_rec(x[0]); mag_rec(x[1]);
 /* o/p map */
 x[0]=y[0];
 x[1]=y[4];
 x[2]=y[6];
 x[3]=y[2];
 x[4]=y[7];
 x[5]=y[5];
 x[6]=y[3];
 x[7]=y[1];
 }
 /**********/
 void forward_c (float *x,float *y)
 {
  int i;
  float a,b;
  for(i=0;i<8;i++){x[i]=x[i]/4;} /* input data scaled by 4 followed by bty scaling of2*/
  a=x[0]+x[7]; b=x[0]-x[7];x[0]=a/2; x[7]=b/2; x[0]=truncat(x[0],trc); x[7]=truncat(x[7],trc);
 mag_rec(x[0]); mag_rec(x[7]);
  a=x[1]+x[6]; b=x[1]-x[6];x[1]=a/2; x[6]=b/2; x[1]=truncat(x[1],trc); x[6]=truncat(x[6],trc);
 mag_rec(x[1]); mag_rec(x[6]);
  a=x[2]+x[5]; b=x[2]-x[5];x[2]=a/2; x[5]=b/2; x[2]=truncat(x[2],trc); x[5]=truncat(x[5],trc);
 mag_rec(x[2]); mag_rec(x[5]);
```

```
 a=x[3]+x[4]; b=x[3]-x[4];x[3]=a/2; x[4]=b/2; x[3]=truncat(x[3],trc); x[4]=truncat(x[4],trc);
mag_rec(x[3]); mag_rec(x[4]);
 /*insert pipeline state here */
 x[6]=x[6]+3*x[5]/8; x[6]=truncat(x[6],trc);  mag_rec(x[6]);
 x[5]=5*x[6]/8-x[5]; x[5]=truncat(x[5],trc);  mag_rec(x[5]);
 /*****/
 a=x[0]+x[3]; b=x[0]-x[3];x[0]=a/2; x[3]=b/2; x[0]=truncat(x[0],trc); x[3]=truncat(x[3],trc);
mag_rec(x[0]); mag_rec(x[3]);
 a=x[1]+x[2]; b=x[1]-x[2];x[1]=a/2; x[2]=b/2; x[1]=truncat(x[1],trc); x[2]=truncat(x[2],trc);
mag_rec(x[1]); mag_rec(x[2]);
 a=x[4]+x[5]; b=x[4]-x[5];x[4]=a/2; x[5]=b/2; x[4]=truncat(x[4],trc); x[5]=truncat(x[5],trc);
mag_rec(x[4]); mag_rec(x[5]);
 a=x[7]+x[6]; b=x[7]-x[6];x[7]=a/2; x[6]=b/2; x[7]=truncat(x[7],trc); x[6]=truncat(x[6],trc);
mag_rec(x[7]); mag_rec(x[6]);
 /*insert pipelinehere */
 x[4]=x[4]-x[7]/8;   x[4]=truncat(x[4],trc);  mag_rec(x[4]);
 x[0]=x[0]+x[1];     x[0]=truncat(x[0],trc);  mag_rec(x[0]);
 x[1]=x[0]/2-x[1];   x[1]=truncat(x[1],trc);  mag_rec(x[1]);
 x[2]=x[2]-3*x[3]/8;  x[2]=truncat(x[2],trc);  mag_rec(x[2]);
 x[3]=x[3]+3*x[2]/8;  x[3]=truncat(x[3],trc);  mag_rec(x[3]);
 x[5]=x[5]+7*x[6]/8;  x[5]=truncat(x[5],trc);  mag_rec(x[5]);
 x[6]=x[6]-x[5]/2;   x[6]=truncat(x[6],trc);  mag_rec(x[6]);
 /**o/p map **/
 y[0]=x[0];
 y[4]=x[1];
 y[6]=x[2];
 y[2]=x[3];
 y[7]=x[4];
 y[5]=x[5];
 y[3]=x[6];
 y[1]=x[7];
}
/***********/
void invert_c(float *y, float *x)
{
```

```
float a,b;
y[4]=y[0]/2-y[4];  y[4]=truncat(y[4],trc);   mag_rec(x[4]);
y[0]=y[0]-y[4];              y[0]=truncat(y[0],trc);   mag_rec(x[0]);
y[2]=y[2]-3*y[6]/8;          y[2]=truncat(y[2],trc);   mag_rec(x[2]);
y[6]=y[6]+3*y[2]/8;          y[6]=truncat(y[6],trc);   mag_rec(x[6]);
y[3]=y[3]+y[5]/2; y[3]=truncat(y[3],trc);   mag_rec(x[3]);
y[5]=y[5]-7*y[3]/8;          y[5]=truncat(y[5],trc);   mag_rec(x[5]);
y[7]=y[7]+y[1]/8; y[7]=truncat(y[7],trc);   mag_rec(x[7]);
/*pipe line */
a=y[4]+y[6]; b=y[4]-y[6];y[4]=a; y[6]=b; x[4]=truncat(x[4],trc); x[6]=truncat(x[6],trc);
mag_rec(x[4]); mag_rec(x[6]);
a=y[0]+y[2]; b=y[0]-y[2];y[0]=a; y[2]=b; x[0]=truncat(x[0],trc); x[2]=truncat(x[2],trc);
mag_rec(x[0]); mag_rec(x[2]);
a=y[7]+y[5]; b=y[7]-y[5];y[7]=a; y[5]=b; x[7]=truncat(x[7],trc); x[5]=truncat(x[5],trc);
mag_rec(x[7]); mag_rec(x[5]);
a=y[1]+y[3]; b=y[1]-y[3];y[1]=a; y[3]=b; x[1]=truncat(x[1],trc); x[3]=truncat(x[3],trc);
mag_rec(x[1]); mag_rec(x[3]);
/***/
y[5]=5*y[3]/8-y[5];          y[5]=truncat(y[5],trc);    mag_rec(x[5]);
y[3]=y[3]-3*y[5]/8;          y[3]=truncat(y[3],trc);    mag_rec(x[3]);
/**/
a=y[2]+y[7]; b=y[2]-y[7];y[2]=a; y[7]=b; x[2]=truncat(x[2],trc); x[7]=truncat(x[7],trc);
mag_rec(x[2]); mag_rec(x[7]);
a=y[6]+y[5]; b=y[6]-y[5];y[6]=a; y[5]=b; x[6]=truncat(x[6],trc); x[5]=truncat(x[5],trc);
mag_rec(x[6]); mag_rec(x[5]);
a=y[4]+y[3]; b=y[4]-y[3];y[4]=a; y[3]=b; x[4]=truncat(x[4],trc); x[3]=truncat(x[3],trc);
mag_rec(x[4]); mag_rec(x[3]);
a=y[0]+y[1]; b=y[0]-y[1];y[0]=a; y[1]=b; x[0]=truncat(x[0],trc); x[1]=truncat(x[1],trc);
mag_rec(x[0]); mag_rec(x[1]);
/* o/p map */
x[0]=y[0];
x[1]=y[4];
x[2]=y[6];
x[3]=y[2];
x[4]=y[7];
```

```
x[5]=y[5];
x[6]=y[3];
x[7]=y[1];
}
/******************/
void forward_b (float *x,float *y)
{
 int i;
 float a,b;
 for(i=0;i<8;i++){x[i]=x[i]/4;} /*input data scaled by 4 followed by bty scaling of2*/
 a=x[0]+x[7]; b=x[0]-x[7];x[0]=a/2; x[7]=b/2; x[0]=truncat(x[0],trc); x[7]=truncat(x[7],trc);
mag_rec(x[0]); mag_rec(x[7]);
 a=x[1]+x[6]; b=x[1]-x[6];x[1]=a/2; x[6]=b/2; x[1]=truncat(x[1],trc); x[6]=truncat(x[6],trc);
mag_rec(x[1]); mag_rec(x[6]);
 a=x[2]+x[5]; b=x[2]-x[5];x[2]=a/2; x[5]=b/2; x[2]=truncat(x[2],trc); x[5]=truncat(x[5],trc);
mag_rec(x[2]); mag_rec(x[5]);
 a=x[3]+x[4]; b=x[3]-x[4];x[3]=a/2; x[4]=b/2; x[3]=truncat(x[3],trc); x[4]=truncat(x[4],trc);
mag_rec(x[3]); mag_rec(x[4]);
 /*insert pipeline state here */
 x[6]=x[6]+3*x[5]/8; x[6]=truncat(x[6],trc);  mag_rec(x[6]);
 x[5]=5*x[6]/8-x[5]; x[5]=truncat(x[5],trc);  mag_rec(x[5]);
 /*****/
 a=x[0]+x[3]; b=x[0]-x[3];x[0]=a/2; x[3]=b/2; x[0]=truncat(x[0],trc); x[3]=truncat(x[3],trc);
mag_rec(x[0]); mag_rec(x[3]);
 a=x[1]+x[2]; b=x[1]-x[2];x[1]=a/2; x[2]=b/2; x[1]=truncat(x[1],trc); x[2]=truncat(x[2],trc);
mag_rec(x[1]); mag_rec(x[2]);
 a=x[4]+x[5]; b=x[4]-x[5];x[4]=a/2; x[5]=b/2; x[4]=truncat(x[4],trc); x[5]=truncat(x[5],trc);
mag_rec(x[4]); mag_rec(x[5]);
 a=x[7]+x[6]; b=x[7]-x[6];x[7]=a/2; x[6]=b/2; x[7]=truncat(x[7],trc); x[6]=truncat(x[6],trc);
mag_rec(x[7]); mag_rec(x[6]);
 /*insert pipelinehere */
 x[4]=x[4]-x[7]/8;   x[4]=truncat(x[4],trc);  mag_rec(x[4]);
 x[0]=x[0]+x[1];     x[0]=truncat(x[0],trc);  mag_rec(x[0]);
 x[1]=x[0]/2-x[1];   x[1]=truncat(x[1],trc);  mag_rec(x[1]);
 x[2]=x[2]-3*x[3]/8; x[2]=truncat(x[2],trc);  mag_rec(x[2]);
```

```
 x[3]=x[3]+3*x[2]/8; x[3]=truncat(x[3],trc); mag_rec(x[3]);
 x[6]=x[6]-3*x[5]/4; x[6]=truncat(x[6],trc); mag_rec(x[6]);
 x[5]=x[5]+x[6]/2;   x[5]=truncat(x[5],trc); mag_rec(x[5]);
 x[6]=x[6]-x[5]/8;   x[6]=truncat(x[6],trc); mag_rec(x[6]);
 /**o/p map **/
 y[0]=x[0];
 y[4]=x[1];
 y[6]=x[2];
 y[2]=x[3];
 y[7]=x[4];
 y[5]=x[5];
 y[3]=x[6];
 y[1]=x[7];
}
/***********/
void invert_b(float *y, float *x)
{
float a,b;
y[4]=y[0]/2-y[4]; y[4]=truncat(y[4],trc);  mag_rec(x[4]);
y[0]=y[0]-y[4];          y[0]=truncat(y[0],trc);  mag_rec(x[0]);
y[2]=y[2]-3*y[6]/8;      y[2]=truncat(y[2],trc);  mag_rec(x[2]);
y[6]=y[6]+3*y[2]/8;      y[6]=truncat(y[6],trc);  mag_rec(x[6]);
y[3]=y[3]+y[5]/8; y[3]=truncat(y[3],trc);  mag_rec(x[3]);
y[5]=y[5]-1*y[3]/2;      y[5]=truncat(y[5],trc);  mag_rec(x[5]);
y[3]=y[3]+3*y[5]/4;      y[3]=truncat(y[3],trc);  mag_rec(x[3]);
y[7]=y[7]+y[1]/8; y[7]=truncat(y[7],trc);  mag_rec(x[7]);
/*pipe line */
a=y[4]+y[6]; b=y[4]-y[6];y[4]=a; y[6]=b; x[4]=truncat(x[4],trc); x[6]=truncat(x[6],trc);
mag_rec(x[4]); mag_rec(x[6]);
a=y[0]+y[2]; b=y[0]-y[2];y[0]=a; y[2]=b; x[0]=truncat(x[0],trc); x[2]=truncat(x[2],trc);
mag_rec(x[0]); mag_rec(x[2]);
a=y[7]+y[5]; b=y[7]-y[5];y[7]=a; y[5]=b; x[7]=truncat(x[7],trc); x[5]=truncat(x[5],trc);
mag_rec(x[7]); mag_rec(x[5]);
a=y[1]+y[3]; b=y[1]-y[3];y[1]=a; y[3]=b; x[1]=truncat(x[1],trc); x[3]=truncat(x[3],trc);
mag_rec(x[1]); mag_rec(x[3]);
```

```
/***/
y[5]=5*y[3]/8-y[5];          y[5]=truncat(y[5],trc);   mag_rec(x[5]);
y[3]=y[3]-3*y[5]/8;          y[3]=truncat(y[3],trc);   mag_rec(x[3]);
/**/
a=y[2]+y[7]; b=y[2]-y[7];y[2]=a; y[7]=b; x[2]=truncat(x[2],trc); x[7]=truncat(x[7],trc);
mag_rec(x[2]); mag_rec(x[7]);
a=y[6]+y[5]; b=y[6]-y[5];y[6]=a; y[5]=b; x[6]=truncat(x[6],trc); x[5]=truncat(x[5],trc);
mag_rec(x[6]); mag_rec(x[5]);
a=y[4]+y[3]; b=y[4]-y[3];y[4]=a; y[3]=b; x[4]=truncat(x[4],trc); x[3]=truncat(x[3],trc);
mag_rec(x[4]); mag_rec(x[3]);
a=y[0]+y[1]; b=y[0]-y[1];y[0]=a; y[1]=b; x[0]=truncat(x[0],trc); x[1]=truncat(x[1],trc);
mag_rec(x[0]); mag_rec(x[1]);
/* o/p map */
x[0]=y[0];
x[1]=y[4];
x[2]=y[6];
x[3]=y[2];
x[4]=y[7];
x[5]=y[5];
x[6]=y[3];
x[7]=y[1];
}
/**********/
void mag_rec(float z)
{
 if(z>dmax) dmax=(float)z;
 if(z<dmin) dmin=(float)z;
}
/*********/
float *creat_array(int n)
{
  float *plag;
  if(n!=0) {
    plag = (float*)calloc(n,sizeof(float));
    if(plag==(float*)NULL) {printf("\nFlag error!! Out of memory"); exit(0); }
```

```
  }
  return (plag);
 }
/*****************/
```

ภาคผนวก ค
ผลงานที่ได้รับการตีพิมพ์

K. Hadkhuntod S. Pongyupinpanich and S. Choomchuay . "An Application of ELGamal Elliptic Curve Cryptography to Smart Card", การประชุมวิชาการทางไฟฟ้า ครั้งที่ 25 ,157

An Application of ElGamal Elliptic Curve Cryptosystems to Smart Card

K. Hadkhuntod S. Pongyupinpanich and S. Choomchuay
Department of Electronics, Faculty of Engineering
Microelectronic Devices R&D Lab, Research Centre for Communications and Information Technology
King Mongkut's Institute of Technology Ladkrabang (KMITL)
Chalongkrung Road, Ladkrabang, Bangkok 10520, Thailand.
Tel: (66) 2 326 4222 Ext. 114, Fax: (66) 2 739 2429, E-mail: kchsomsa@kmitl.ac.th

Abstract

To enhance the applications to modern smart card system and on the fly data encryption/decryption system, an On-chip Data Storage with ECC cryptosystem was designed. The ElGamal Elliptic Curve Cryptosystems with the key size of 176 bits was chosen to ensure the security level and with short key size. The design can be configured to be either for encryption or the decryption with a single control signal. To minimize the hardware cost, size and complexity, the slightly modified fix-coefficient bit-serial finite field multiplier has been employed. The prototype implementation on FPGA proved the concept idea and the mathematics behind.

Keywords: Elliptic Curve Cryptography, ECC Cryptosystems, Smart Card, Finite Field

1. Introduction

One most important application of a portable on-chip data storage and corresponding processor is smart card. A class of contact smart card can incorporate a tiny microprocessor and a set of memory. Data stored are divided into zones according to their accessibility, i.e., open zone, secret zone and user record. Firstly developed in 1950 by Diners Club, the nowadays smart card is much more smart. It can accommodate more information and covers most area of applications; ticket, phone card, credit card, medical card and so on. Despite the operation of smart cards that quite varies, the common issue is data security. Data stored as well as data transmitted or received are in the form that are not easy to be guessed or understood. Generally those texts are ciphered or encrypted by using a special class of mathematics.

The cryptosystems can be categorised into two groups, symmetric cryptography and asymmetric cryptography. DES (Data Encryption Standard) is a well-known and well-accepted class of the first one. The proposing FIP PUB 46-3 (Triple DES), 64 bits key size is shared between the sender and the receiver. Although DES seems to be able to guarantee a good level of security, its weakness is a single key system. The encrypted word can be easily cracked if the key and the procedure are known. An alternative class of cryptosystems, asymmetric type, provides better security according its 2-key system, private key and public key. RSA developed by Rivest Chamir and Aldenman based on integer factorization. The security relies on the integer size. Modulo operation of those big numbers is fairly clumsy and time-consuming. Discrete Logarithm Cryptosystems (DLC) based on finite field of $GF(2^m)$[3] offers better computation and storage efficiency. However, its key size is considerably long ie., 512-1024 bits. Rather than elements are defined only in the finite field of $GF(2^m)$, they can be also member of points defined by elliptic curve equation. ElGamal Cryptosystems under elliptic curve therefore requires shorter key compared to its DLC version. For comparable level of security, the ECC may require the key size of 155-201 bits. For a general purpose, cryptosystem processor, the ElGamal technique can be implemented as a processor-based hardware [1].

In this paper we propose an implementation of data encryption/decryption scheme that could be either applied to smart card system or transmitting and receiving data with ciphering and deciphering capability. Mathematical background of cryptosystem, in particular ElGamal Elliptic Curve Cryptosystems as well as the corresponding example are given in the following section 2. A discussion of hardware for implementing such an algorithm is also discussed. In section 3, the architecture that emphasizes the working in a pair of the encrypter and the decrypter is elaborated. Several mathematic operations can be computed off line. As a result, this piece of hardware just have to compute only a few simple multiplications. Bit serial design has further reduced the multiplier size. The architecture given in section 2 is then implemented with FPGA as detailed in section 3 before the conclusion of our works.

2. Elliptic Curve Cryptography

2.1 ElGamal Public Key Cryptography

A standard ElGamal public-Key cryptography is based on the large integer prime field Z_P of which α is a root, for instant $P = 2357$ and $\alpha = 2$. A private key a, can be chosen arbitrarily, $a \in Z_P$. The public key R is defined such that $R(P, \alpha, \alpha^a)$. The sender chooses k as an arbitrary to compute $\gamma = \alpha^k \bmod p$. A message $m \in Z_P$ is encrypted as $\delta = m\left(\alpha^a\right)^k \bmod p$. Both γ and δ are then sent through the communication channel. Once the receiver has received the messages, he

computes $\beta = \gamma^{(P-1-a)} \bmod p$. The sent message can be decrypted as $m = \beta \cdot \delta \bmod p$. It should be noted that $\alpha^{kP} = \alpha^k$.

The polynomial version operates in the similar manner. The degree n irreducible polynomial chosen such that the Galois field GF(2^n) of which α is a root is defined. Text message is then described as a polynomial of degree n-1, ie., $m(x) = \sum_{i=0}^{n-1} m_i x^i$ where m_i is a member of GF(2).

2.2 Elliptic Curves Mathematics

Elliptic curves cryptography is preferably implemented using non-supersingular curve because of its better security. The underline field of characteristic 2, GF(2^m) is then the set of solution to the equation

$$y^2 + xy = x^3 + Ax^2 + B \qquad (1)$$

where $A, B \in GF(2^m)$ and $B \neq 0$. Number of points on an elliptic curve, order of the curve, is denoted by #E.

Let $P \in (x_1, y_1)$ be a member of those points, then $-P \in (x_1, y_1 + x_1)$. For all $Q \in (x_2, y_2)$ with $P, Q \neq o$ (where o denotes the point at infinity) and $Q \neq -P$, the resulted $P + Q \in (x_3, y_3)$ are as follows:

$$x_3 = \begin{cases} \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)^2 + \dfrac{y_1 + y_2}{x_1 + x_2} + (x_1 + x_2) + A, & P \neq Q \\ x_1^2 + \dfrac{B}{x_1^2}, & P = Q \end{cases}$$

and

$$y_3 = \begin{cases} \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_2) + x_3 + y_1, & P \neq Q \\ x_1^2 + \left(x_1 + \dfrac{y_1}{x_1}\right)x_3 + x_3, & P = Q \end{cases}$$

2.3 Elliptic Curve Public-Key Cryptosystems

Let P be a fix point as a member of #E(Set of Elliptic Curve). The sender A can choose randomly an integer a as a private key where $a \in GF(2^m)$. With this private key a, the public key R can be computed from $R = aP$. With another randomly chosen k, $k \in GF(2^m)$, the fix point Q also can be computed from $Q = kP$. The sender A encrypts his massage M by making use of $kR = akP = (x', y')$ as $m = MkR$, preferably, $m_1 = M_1 x'$ and $m_2 = M_2 y'$, where $M = M_1 + M_2$. The encrypted words then sent along with Q.

Once the encrypted words received by the receiver B, with A's private key, he computes $aQ = akP = (x', y')$. Subsequently $m_1 (x')^{-1} = M_1 x' (x')^{-1}$ and $m_2 (y')^{-1} = M_2 y' (y')^{-1}$ are computed and the original messages M_1 and M_2 are obtained respectively. It should be noted that, firstly, the private key a, must be known to both the sender and the receiver in prior. Secondly, in obtaining the original message, two inversions are required.

2.4 Illustrative Example

The Elliptic Curve Cryptosystems (ECC) with the key size of 155 to 201 bits can yield the similar level security as Discrete Logarithm Cryptosystems (DLC) with the key size of 512 to 1024 bits. The ECC is therefore superior to DLC in terms of block length and storage requirement. Our implementation is based on the key size of 176 bits where GF(2^{176}) is the underline field.

1) To minimize the hardware complexity (wiring complexity according to the feedback path and number of XOR gates), the irreducible polynomial $I(x) = x^{176} + x^{11} + x^3 + x^2 + 1$ was chosen.

2) A and B, $A, B \in GF(2^{176})$ are chosen as arbitraries:

A = 6f4e a37c cb7c 6443 44de 2368 dba3 c524 bd4a e585 bb6c
B = 8a38 e3e4 1595 f58f a35e c0a2 dd61 d80a 8984 bc08 4f91

3) With such A and B in 2), there could be possible solutions to the elliptic curve equation;

$$y^2 + xy = x^3 + Ax^2 + B \qquad (2)$$

Subsequently we yield a fix point $P(x,y)$ of the curve

x = 9e15 2cda 8c22 e6e8 23d7 674e 72e5 a195 44b2 35e7 c3d1
y = 8bf3 e7ed dbe0 3681 aa7b c488 1815 dba6 c633 569a 43b3.

4) Regardless of the computing complexity at this step, A private key a is freely chosen and similar as k.
As an example, $a = 75$ and $k = 100$ were arbitrarily chosen.

5) The encrypter has to compute $kR = akP = (x', y')$, R denotes the public key and we yield

x' = f874 8697 a7d1 73d5 da23 9933 615f 89d1 aef7 93af c1e8
y' = 34ab ecbb a95f 327f ffb2 50f2 c7a8 65d5 dd5a 471e 8422

6) The encrypted messages are computed as $m_1 = M_1 x'$ and $m_2 = M_2 y'$ where M_1 and M_2 are consecutive blocks of which the data length is m bits (176 bits).

7) Encrypted messages are fed into and stored in the array of Flash ROMs which can be on chip or can be a separate module.

Operation 6) and 7) can repeat many times until all the data is encrypted and stored.

To read out the encrypted messages, operations are as follows:

1) $Q = kP$ is pre-computed and $aQ = (x', y')$
2) Step 1) could be omitted since the encrypter and the decrypter are in the same module. From 5) above $(x')^{-1}$ and $(y')^{-1}$ can be computed.

$(x')^{-1}$ = 78c4 f832 64fe 4305 7a4c 2f3e 9cd2 88bd 8d1c e7ee be32
$(y')^{-1}$ = f64c b612 cb58 5156 292d f8d1 0e60 6c8a a4c9 677a a0b9

3) Subsequently;

$$M_1 = m_1(x')^{-1} \text{ and } M_2 = m_2(y')^{-1} \quad (3)$$

Operation in 3) is to be repeated until all the data are decrypted.

3. Architecture

ElGamal Elliptic Curve Public-Key Encryption discussed in section 2 above can be more simplified when the encrypter circuit and the decrypter circuits are sitting aside. That is, x', y', $(x')^{-1}$ and $(y')^{-1}$ can be pre-computed and kept in the registers. The encrypter then just have to compute

$$m_1 = M_1 x' \text{ and } m_2 = M_2 y' \quad (4)$$

At the receiving end, the decrypter has to compute

$$m_1(x')^{-1} = M_1 x'(x')^{-1} \quad (5.1)$$

and

$$m_2(y')^{-1} = M_2 y'(y')^{-1} \quad (5.2)$$

The operation is illustrated in Fig.1. The advantage of this architecture is the minimum hardware requirement. Only a merely fix-coefficient multiplier can be used when the hardware is programmed to be just only for encryption or decryption. Such a multiplier can be also used alternately for encryption and decryption in a half duplex operation.

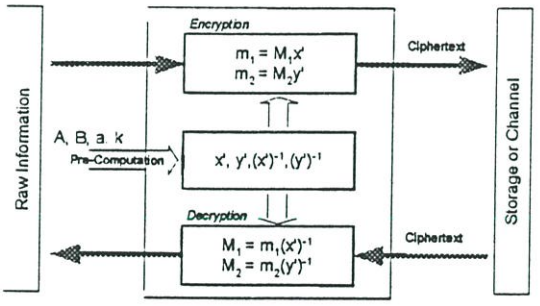


Fig. 1 Encryption and decryption scheme when the encrypter and the decrypter are sitting aside

For encryption, the multiplier has to compute M_1x' and M_2x'. For decryption, a similar multiplier has to compute $m_1(x')^{-1}$ and $m_2(x')^{-1}$. A single multiplier can be used be modified to cover 2-4 coefficients. Even so, with careful selection of the multiply end, the complexity of this merely bit-serial fix-coefficient multiplier is much simpler than a general purpose one [2,3]. The architecture is depicted in Fig 2.

Note that, the multiplication is performed over the Galois field, selection of the irreducible polynomial can effect the wiring complexity. In practical, select one that has small number of terms (i.e. 5 terms). The proposed architecture also has some drawbacks, for instants, it is not flexible to change the private key and/or public key. If one wants to do so, this piece of hardware then requires the capability of elliptic equation solving, inversion and multiplication. The circuit size then cannot be kept minimum. For current chip design and VLSI technology, the hard-wire program as our approach, it is not very difficult.

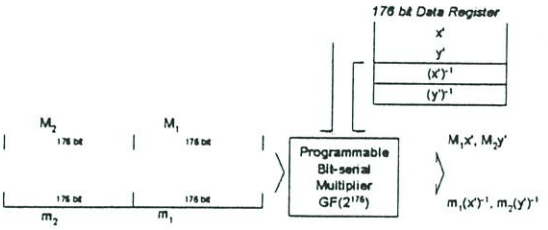


Fig. 2 An architecture when a single fix-coefficient bit-serial multiplier is used

In the case that the multiplier has to be the general purpose one, the number of required gate is fairly big. Multiplication in the composite field can be alternatively performed in the composite filed, $GF(2^n)^m$ [4]. For instant, 176 can be factorised as 16×11; ie., $GF(2^{16})^{11}$ or $GF(2^{11})^{16}$. Paar [5,6] has worked out a good deal of composite field multiplier and its variations. Composite field inversion is subsequently applicable. Guajardo and Paar inversion scheme could be considered [7].

4. Implementation

The purpose of our implementation is to design a tiny piece of hardware with the following features:

- Can be configured to work as an encrypter or decrypter while incorporating ElGamal ECC scheme.
- Encrypted data are kept on chip (smart card application)
- RS-232 interface standard (Baud rate =9600)

The prototype design was implemented with FPGA since the design time is fairly low and the good degree of flexibility is allowed. We first focused on text data, however, with memory expansion image data can be also accommodated. The functional blocks are shown below in Fig. 3.

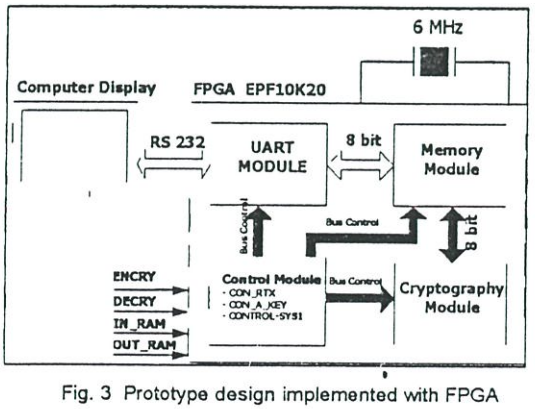


Fig. 3 Prototype design implemented with FPGA

Control lines control 4 functions of the chip which are: IN_RAM – Read data from the port and write to the memory, OUT_RAM – Read stored data and write to the port, ENCRY – Data in the memory are encrypted and write back, and DECRY – Data in the memory are decrypted and write back. These features may not quite necessary in the real application but at the state of verification they are quite important since the contents of memory can be simply dumped.

We have employed FPGA *EPF10K20RC* (Altera, 20,000 gates) in our design. The chip operates at 6.00 MHz clock frequency. Gate utilization can be detailed as:

UART Module	1600 gates
Control & Clock	2752 gates
Encrypter/Decrypter	5000 gates
Memory (256 Bytes)	2048 gates

Note that just about 60% of the available gates are used. In fact in the final design, the capacity of the memory has been increased to cover wider need of data storing. However for such a FPGA, the maximum number of gates that could be allowed for memory is 12288 gates (about 1500 bytes). In practical, on-chip flash ROM (smart card) or external Flash ROM modules (other applications) can be employed.

5. Result

The prototype designed function have been confirmed by the experiment results of which to be elaborated in the section. The perform of a bi-Serial multiplier show as a block in Fig.4 is draw as show in Fig.4 Basically. The multiplier has to perform $M_1 x'$ and $M_1 y'$ in the process of decryption. It also performs $m_1(x')^{-1}$ and $m_1(y)^{-1}$ in the process of decryption. Timing diagram of the encryption and decryption is shown in Fig 5 and 6 representatively that to be encrypted texts and their encrypted version are shown in Fig.7 and Fig.8. We also have input the 128 x 128 image into the encrypter. The encrypted image is shown in Fig.10

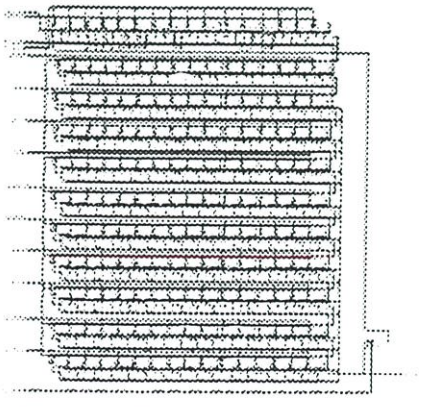


Fig. 4 A portion of 176 bit Bit-Serial multiplier Module

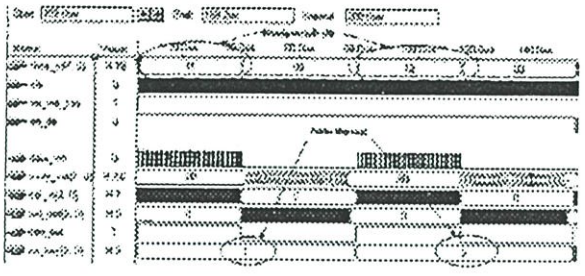


Fig.5 Timing Diagram of Encryptor on FPGA

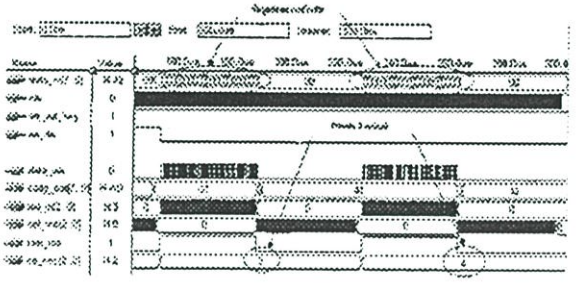


Fig. 6 Timing Diagram of Decryptor on FPGA

The Computational cost of encryption is a barrier to wider appli cation of a variety of data security protocols. Virtually all resear ch on Elliptic Curve Cryptography(ECC) provides evidence to s uggest that ECC can provide family of encryption

Fig. 6 Plaintext for encryption (*m*)

ä*‡□"D ,ø*Ûäã¯V□"Å□□ÃÙ□□+'%H£ƒ□O□!hw-Ñ-'"Ab ßì—Vµèç½□jÚ¹Gðßu□Ùj□□£□é□□□ìúÅø¯□ ¯□□IяÀ…Ce© ïa4□»dz□□□¾ð□UßPm²RŠ—□›ı'_#ª-‡é«□Ö{ëqÂ¯zÅÒX?□,, □³_ésx□yÊ¶¤cœåŸš"(„P— sIO-ý9ÛÙ¢Î□□Ù–q÷{.µçF*ÁD □¹wŸ!Vu×[Ð¢Ù□□×□c(□¹ŠðHØƒÍY7·°{Ðäè□øFàÀpð·JK□Ï. □pC=²Á□

Fig. 7 Chiphertext form encryption

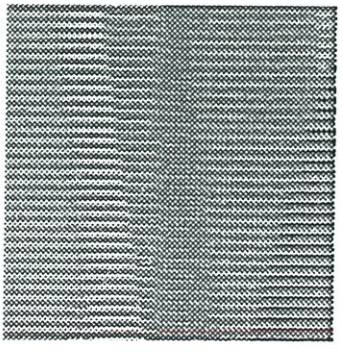


Fig. 8 Input for encryption (128 x 128)

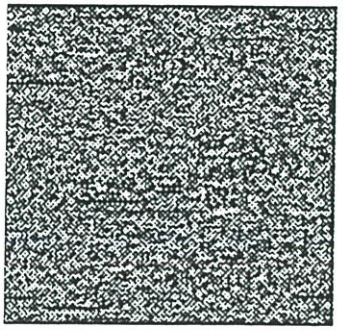


Fig. 9 Result encrypted Image(128 x 128)

In Fig. 8 and 9 was presented the Image encrytpion grayscale size 128 x 128

6. Conclusions

In this work we have proposed the data encrypter and decrypter that have to work in a pair. Elliptic curve cryptosystems with ElGamal public key system provides good security while the key size can be kept small, 176 bits in our case. Several computations, including inversion can be computed of line. This enables the small hardware size. Careful selection of x', y', (1/x') and (1/y') can make the multiplier slightly more compact. With the proposed concept, on the fly encryption/decryption could be possibly developed to enhance data security during its transmission without the requirement of extra computation or additional software. Real time application is also promise provided that the multiplication speed can be configured to match the data speed. The concept of on-chip storage can be extended for smart card application. The prototype design finished in FPGA has demonstrated very well, the basic concept as well as the future possibility.

References

[1] S. Sutikno, A. Surya and R. Effendi, "An Implementation of ElGamal Elliptic Curves Cryptosystems", Int. Symposium on Intelligent Signal Processing and Communication Systems (ISPACS '99), Phuket, Thailand, Dec.1999.

[2] E.R. Berlekamp, "Bit-serial Reed-Solomon Encoder", IEEE Trans. on Information Theory, Vol. IT-28, Nov. 1982, pp. 869-874.

[3] S. Choomchuay, "On the Implementation of Finite Field Basis Conversions", EECON-17, King Mongkut's Institute of Technology, North Bangkok, 1994, pp. 482-486.

[4] S. Pongyupinpanich and S.Choomchuay, "Composite Field Bit Serial-Parallel Multiplier, Proc. of Information and Computer Engineering Workshop 2002 (ICEP2002), Prince of Songkla University, Jan. 2002, pp.65-69.

[5] Christof Paar, "A New Architecture for a Parallel Finite Field Multiplier With Low Complexity Based on Composite Field", IEEE Trans. On Computers, Vol. 45, No. 7, July 1996.

[6] G. Orlando and C. Paar, "A Super-serial Galois Field Multiplier for FPGAs and its Application to Public-Key Algorithms", 7th Annual IEEE Symp. on Field-Programmable Custom Computing Machines, Napa, CA, Apr. 1999.

[7] J. Guajardo and C. Paar, "Fast Inverse Inversion in Composite Galois Filed $(2^n)^m$), ISIT 1998, Cambridge USA.

ภาคผนวก ง

อุปกรณ์ FPGA เบอร์ EPF10K20

Includes
FLEX 10KA

FLEX 10K

Embedded Programmable Logic Device Family

March 2001, ver. 4.1 Data Sheet

Features...

- The industry's first embedded programmable logic device (PLD) family, providing System-on-a-Programmable-Chip (SOPC) integration
  - Embedded array for implementing megafunctions, such as efficient memory and specialized logic functions
  - Logic array for general logic functions
- High density
  - 10,000 to 250,000 typical gates (see Tables 1 and 2)
  - Up to 40,960 RAM bits; 2,048 bits per embedded array block (EAB), all of which can be used without reducing logic capacity
- System-level features
  - MultiVolt™ I/O interface support
  - 5.0-V tolerant input pins in FLEX® 10KA devices
  - Low power consumption (typical specification less than 0.5 mA in standby mode for most devices)
  - FLEX 10K and FLEX 10KA devices support peripheral component interconnect Special Interest Group (PCI SIG) *PCI Local Bus Specification, Revision 2.2*
  - FLEX 10KA devices include pull-up clamping diode, selectable on a pin-by-pin basis for 3.3-V PCI compliance
  - Select FLEX 10KA devices support 5.0-V PCI buses with eight or fewer loads
  - Built-in Joint Test Action Group (JTAG) boundary-scan test (BST) circuitry compliant with IEEE Std. 1149.1-1990, available without consuming any device logic

Table 1. FLEX 10K Device Features

Feature	EPF10K10 EPF10K10A	EPF10K20	EPF10K30 EPF10K30A	EPF10K40	EPF10K50 EPF10K50V
Typical gates (logic and RAM) *(1)*	10,000	20,000	30,000	40,000	50,000
Maximum system gates	31,000	63,000	69,000	93,000	116,000
Logic elements (LEs)	576	1,152	1,728	2,304	2,880
Logic array blocks (LABs)	72	144	216	288	360
Embedded array blocks (EABs)	3	6	6	8	10
Total RAM bits	6,144	12,288	12,288	16,384	20,480
Maximum user I/O pins	150	189	246	189	310

Table 2. FLEX 10K Device Features

Feature	EPF10K70	EPF10K100 EPF10K100A	EPF10K130V	EPF10K250A
Typical gates (logic and RAM) *(1)*	70,000	100,000	130,000	250,000
Maximum system gates	118,000	158,000	211,000	310,000
LEs	3,744	4,992	6,656	12,160
LABs	468	624	832	1,520
EABs	9	12	16	20
Total RAM bits	18,432	24,576	32,768	40,960
Maximum user I/O pins	358	406	470	470

Note to tables:

(1) The embedded IEEE Std. 1149.1 JTAG circuitry adds up to 31,250 gates in addition to the listed typical or maximum system gates.

...and More Features

- Devices are fabricated on advanced processes and operate with a 3.3-V or 5.0-V supply voltage (see Table 3)
- In-circuit reconfigurability (ICR) via external configuration device, intelligent controller, or JTAG port
- ClockLock™ and ClockBoost™ options for reduced clock delay/skew and clock multiplication
- Built-in low-skew clock distribution trees
- 100% functional testing of all devices; test vectors or scan chains are not required

Table 3. Supply Voltages for FLEX 10K & FLEX 10KA Devices

5.0-V Devices	3.3-V Devices
EPF10K10 EPF10K20 EPF10K30 EPF10K40 EPF10K50 EPF10K70 EPF10K100	EPF10K10A EPF10K30A EPF10K50V EPF10K100A EPF10K130V EPF10K250A

- Flexible interconnect
  - FastTrack® Interconnect continuous routing structure for fast, predictable interconnect delays
  - Dedicated carry chain that implements arithmetic functions such as fast adders, counters, and comparators (automatically used by software tools and megafunctions)
  - Dedicated cascade chain that implements high-speed, high-fan-in logic functions (automatically used by software tools and megafunctions)
  - Tri-state emulation that implements internal tri-state buses
  - Up to six global clock signals and four global clear signals
- Powerful I/O pins
  - Individual tri-state output enable control for each pin
  - Open-drain option on each I/O pin
  - Programmable output slew-rate control to reduce switching noise
  - FLEX 10KA devices support hot-socketing
- Peripheral register for fast setup and clock-to-output delay
- Flexible package options
  - Available in a variety of packages with 84 to 600 pins (see Tables 4 and 5)
  - Pin-compatibility with other FLEX 10K devices in the same package
  - FineLine BGA™ packages maximize board space efficiency
- Software design support and automatic place-and-route provided by Altera development systems for Windows-based PCs and Sun SPARCstation, HP 9000 Series 700/800 workstations
- Additional design entry and simulation support provided by EDIF 2 0 0 and 3 0 0 netlist files, library of parameterized modules (LPM), DesignWare components, Verilog HDL, VHDL, and other interfaces to popular EDA tools from manufacturers such as Cadence, Exemplar Logic, Mentor Graphics, OrCAD, Synopsys, Synplicity, VeriBest, and Viewlogic

Table 4. FLEX 10K Package Options & I/O Pin Count *Note (1)*

Device	84-Pin PLCC	100-Pin TQFP	144-Pin TQFP	208-Pin PQFP RQFP	240-Pin PQFP RQFP
EPF10K10	59		102	134	
EPF10K10A		66	102	134	
EPF10K20			102	147	189
EPF10K30				147	189
EPF10K30A			102	147	189
EPF10K40				147	189
EPF10K50					189
EPF10K50V					189
EPF10K70					189
EPF10K100					
EPF10K100A					189
EPF10K130V					
EPF10K250A					

Table 5. FLEX 10K Package Options & I/O Pin Count (Continued) *Note (1)*

Device	503-Pin PGA	599-Pin PGA	256-Pin FineLine BGA	356-Pin BGA	484-Pin FineLine BGA	600-Pin BGA	403-Pin PGA
EPF10K10							
EPF10K10A			150		150 *(2)*		
EPF10K20							
EPF10K30				246			
EPF10K30A			191	246	246		
EPF10K40							
EPF10K50				274			310
EPF10K50V				274			
EPF10K70	358						
EPF10K100	406						
EPF10K100A				274	369	406	
EPF10K130V		470				470	
EPF10K250A		470				470	

Notes to tables:

(1) FLEX 10K and FLEX 10KA device package types include plastic J-lead chip carrier (PLCC), thin quad flat pack (TQFP), plastic quad flat pack (PQFP), power quad flat pack (RQFP), ball-grid array (BGA), pin-grid array (PGA), and FineLine BGA™ packages.

(2) This option is supported with a 256-pin FineLine BGA package. By using SameFrame pin migration, all FineLine BGA packages are pin compatible. For example, a board can be designed to support both 256-pin and 484-pin FineLine BGA packages. The Altera software automatically avoids conflicting pins when future migration is set.

General Description

Altera's FLEX 10K devices are the industry's first embedded PLDs. Based on reconfigurable CMOS SRAM elements, the Flexible Logic Element MatriX (FLEX) architecture incorporates all features necessary to implement common gate array megafunctions. With up to 250,000 gates, the FLEX 10K family provides the density, speed, and features to integrate entire systems, including multiple 32-bit buses, into a single device.

FLEX 10K devices are reconfigurable, which allows 100% testing prior to shipment. As a result, the designer is not required to generate test vectors for fault coverage purposes. Additionally, the designer does not need to manage inventories of different ASIC designs; FLEX 10K devices can be configured on the board for the specific functionality required.

Table 6 shows FLEX 10K performance for some common designs. All performance values were obtained with Synopsys DesignWare or LPM functions. No special design technique was required to implement the applications; the designer simply inferred or instantiated a function in a Verilog HDL, VHDL, Altera Hardware Description Language (AHDL), or schematic design file.

Table 6. FLEX 10K & FLEX 10KA Performance

Application	Resources Used		Performance				Units
	LEs	EABs	-1 Speed Grade	-2 Speed Grade	-3 Speed Grade	-4 Speed Grade	
16-bit loadable counter *(1)*	16	0	204	166	125	95	MHz
16-bit accumulator *(1)*	16	0	204	166	125	95	MHz
16-to-1 multiplexer *(2)*	10	0	4.2	5.8	6.0	7.0	ns
256 × 8 RAM read cycle speed *(3)*	0	1	172	145	108	84	MHz
256 × 8 RAM write cycle speed *(3)*	0	1	106	89	68	63	MHz

Notes:

(1) The speed grade of this application is limited because of clock high and low specifications.

(2) This application uses combinatorial inputs and outputs.

(3) This application uses registered inputs and outputs.

The FLEX 10K architecture is similar to that of embedded gate arrays, the fastest-growing segment of the gate array market. As with standard gate arrays, embedded gate arrays implement general logic in a conventional "sea-of-gates" architecture. In addition, embedded gate arrays have dedicated die areas for implementing large, specialized functions. By embedding functions in silicon, embedded gate arrays provide reduced die area and increased speed compared to standard gate arrays. However, embedded megafunctions typically cannot be customized, limiting the designer's options. In contrast, FLEX 10K devices are programmable, providing the designer with full control over embedded megafunctions and general logic while facilitating iterative design changes during debugging.

Each FLEX 10K device contains an embedded array and a logic array. The embedded array is used to implement a variety of memory functions or complex logic functions, such as digital signal processing (DSP), microcontroller, wide-data-path manipulation, and data-transformation functions. The logic array performs the same function as the sea-of-gates in the gate array: it is used to implement general logic, such as counters, adders, state machines, and multiplexers. The combination of embedded and logic arrays provides the high performance and high density of embedded gate arrays, enabling designers to implement an entire system on a single device.

FLEX 10K devices are configured at system power-up with data stored in an Altera serial configuration device or provided by a system controller. Altera offers the EPC1, EPC2, EPC16, and EPC1441 configuration devices, which configure FLEX 10K devices via a serial data stream. Configuration data can also be downloaded from system RAM or from Altera's BitBlaster™ serial download cable or ByteBlasterMV™ parallel port download cable. After a FLEX 10K device has been configured, it can be reconfigured in-circuit by resetting the device and loading new data. Because reconfiguration requires less than 320 ms, real-time changes can be made during system operation.

FLEX 10K devices contain an optimized interface that permits microprocessors to configure FLEX 10K devices serially or in parallel, and synchronously or asynchronously. The interface also enables microprocessors to treat a FLEX 10K device as memory and configure the device by writing to a virtual memory location, making it very easy for the designer to reconfigure the device.

For more information, see the following documents:

- *Configuration Devices for APEX & FLEX Devices Data Sheet*
- *BitBlaster Serial Download Cable Data Sheet*
- *ByteBlasterMV Parallel Port Download Cable Data Sheet*
- *Application Note 116 (Configuring APEX 20K, FLEX 10K & FLEX 6000 Devices)*

FLEX 10K devices are supported by Altera development systems; single, integrated packages that offer schematic, text (including AHDL), and waveform design entry, compilation and logic synthesis, full simulation and worst-case timing analysis, and device configuration. The Altera software provides EDIF 2 0 0 and 3 0 0, LPM, VHDL, Verilog HDL, and other interfaces for additional design entry and simulation support from other industry-standard PC- and UNIX workstation-based EDA tools.

The Altera software works easily with common gate array EDA tools for synthesis and simulation. For example, the Altera software can generate Verilog HDL files for simulation with tools such as Cadence Verilog-XL. Additionally, the Altera software contains EDA libraries that use device-specific features such as carry chains which are used for fast counter and arithmetic functions. For instance, the Synopsys Design Compiler library supplied with the Altera development systems include DesignWare functions that are optimized for the FLEX 10K architecture.

The Altera development systems run on Windows-based PCs and Sun SPARCstation, and HP 9000 Series 700/800 workstations.

See the *MAX+PLUS II Programmable Logic Development System & Software Data Sheet* for more information.

Functional Description

Each FLEX 10K device contains an embedded array to implement memory and specialized logic functions, and a logic array to implement general logic.

The embedded array consists of a series of EABs. When implementing memory functions, each EAB provides 2,048 bits, which can be used to create RAM, ROM, dual-port RAM, or first-in first-out (FIFO) functions. When implementing logic, each EAB can contribute 100 to 600 gates towards complex logic functions, such as multipliers, microcontrollers, state machines, and DSP functions. EABs can be used independently, or multiple EABs can be combined to implement larger functions.

ประวัติผู้เขียน

นายคมศักดิ์ หาดขุนทด เกิดเมื่อวันที่ 3 พฤศจิกายน พ.ศ. 2509 ที่จังหวัดนครราชสีมา สำเร็จการศึกษาวิทยาศาสตรบัณฑิต สาขาคอมพิวเตอร์และอิเล็กทรอนิกส์ จากมหาวิทยาลัยรามคำแหง ปีการศึกษา 2533

ปัจจุบันรับราชการตำแหน่งอาจารย์1 ระดับ 5 ภาควิชาเทคโนโลยีอุตสาหกรรม(อิเล็กทรอนิกส์) คณะวิทยาศาสตร์และเทคโนโลยี สถาบันราชภัฏพระนครศรีอยุธยา จังหวัดพระนครศรีอยุธยา