การบีบอัดภาพนิ่งด้วยวิธี JPEG บน FPGA

JPEG-Based Still Image Compression on FPGA

โดย

นางสาวนภาพรรณ พรมภักดี

นายนิสัย สืบจากจุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

การบีบอัดภาพนิ่งด้วยวิธี JPEG บน FPGA

JPEG-Based Still Image Compression on FPGA

โดย

นางสาวนภาพรรณ พรมภักดี 46015056

นายนิสัย สืบจากจุล 46015058

อาจารย์ที่ปรึกษา

รศ.ดร. กอบชัย เดชหาญ

อ. ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การบีบอัดภาพนิ่งด้วยวิธี JPEG บน FPGA**

JPEG-Based Still Image Compression on FPGA

ผู้จัดทำ

1. นางสาวนภาพรรณ พรมภักดี 46015056

2. นายนิสัย สืบจากจุล 46015058

อาจารย์ที่ปรึกษา

(รศ.ดร.กอบชัย เดชหาญ)

อาจารย์ที่ปรึกษา

(อ.ศรวัฒน์ ชิวปรีชา)

การบีบอัดภาพนิ่งด้วยวิธี JPEG บน FPGA

JPEG-Based Still Image Compression on FPGA

โดย นางสาวนภาพรรณ พรมภักดี 46015056
นายนิสัย สืบจากจุล 46015058

อาจารย์ที่ปรึกษา รศ.ดร.กอบชัย เดชหาญ
อ.ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

การบีบอัดข้อมูลถือว่าเป็นส่วนสำคัญในการออกแบบวงจรที่เกี่ยวข้องทางการจัดเก็บข้อมูลและการสื่อสารในปัจจุบัน โครงงานนี้เป็นการศึกษาการบีบอัดภาพนิ่งโดยวิธีการของ JPEG(Joint Photographic Expert Group) แบ่งภาพออกเป็นบล็อกย่อยๆ แล้วอาศัยหลักการของ DCT (Discrete Cosine Transform) ในการบีบอัด แล้วนำผลลัพธ์ไปทำการควอนไตซ์เพื่อลดจำนวนบิต จากนั้นจะทำการเข้ารหัสโดยใช้วิธีการของ Huffman encoding ในการออกแบบได้ใช้ภาษา VHDL จำลองการทำงาน, สังเคราะห์วงจรและทดสอบ โดยใช้ร่วมกับอุปกรณ์ FPGA.

ABSTRACT

The data compression is important to design the circuit concerning about data mining of the present communication system. This project is a study of still image compression based on JPEG (Joint Photographic Expert Group) by dividing into sub-block and using the method of DCT (Discrete Cosine Transform).After the compression the data are quantized to reduce the number of bits, then the data are encoded based on Huffman encoding. This design is carried out based on VHDL language, simulation, circuit synthesis and testing with FPGA.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดีโดยได้รับคำแนะนำและคำปรึกษา ตลอดจนความช่วยเหลือต่างๆ จากหลายท่าน ผู้จัดทำขอขอบพระคุณ รศ.ดร.กอบชัย เดชหาญ และอ.ศรวัฒน์ ชิวปรีชา อาจารย์ที่ปรึกษา ที่ให้ความอนุเคราะห์ด้านคำปรึกษาในทุกๆ เรื่องตลอดจนความช่วยเหลือในด้านข้อมูลและอุปกรณ์ที่ใช้ทำปริญญานิพนธ์

ขอขอบพระคุณบิดามารดาที่ได้คอยให้การสนับสนุนและให้กำลังใจมาโดยตลอด และขอขอบคุณเพื่อนๆ ทั้งหลายที่ให้ความช่วยเหลือมาโดยตลอด ผู้เขียนพึงระลึกอยู่เสมอว่าปริญญานิพนธ์ฉบับนี้จะไม่สามารถสำเร็จลุล่วงได้เลย หากขาดความช่วยเหลือจากทุกท่านจึงขอขอบพระคุณมาอย่างสูง

นางสาวนภาพรรณ พรมภักดี

นายนิสัย สืบจากจุล

สารบัญ

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

สารบัญรูปภาพ (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

การบีบอัดข้อมูลถือว่าเป็นส่วนสำคัญในการออกแบบวงจรที่เกี่ยวข้องทางการจัดเก็บข้อมูลและการสื่อสารในปัจจุบัน ซึ่งเทคนิคการบีบอัดข้อมูลแบ่งออกเป็น 2 ประเภทหลักคือ

1. การบีบอัดข้อมูลที่ไม่มีการสูญเสีย (Lossless compression) เป็นวิธีการที่ข้อมูลข่าวสารเมื่อผ่านกระบวนการนี้แล้วผลลัพธ์ของข้อมูลที่ได้จะเหมือนข้อมูลเริ่มต้น

2. การบีบอัดข้อมูลแบบยอมให้มีการสูญเสีย (Lossy compression) วิธีนี้ข้อมูลผลลัพธ์ที่ได้หลังการขยายข้อมูลกลับจะมีความแตกต่างกับข้อมูลเดิม

สำหรับระบบที่เราศึกษาคือระบบการบีบอัดข้อมูลแบบที่ยอมให้มีการสูญเสียข้อมูลบางส่วนโดยนำภาพนิ่งจากรูปขาวดำแบบ Windows Bitmap files (.bmp) เนื่องจากข้อมูลแบบ Bitmap fileนั้นไม่มีการบีบอัดข้อมูลทำให้ข้อมูลมีขนาดใหญ่ต้องใช้พื้นที่ในการจัดเก็บมากกว่ารูปแบบอื่น ที่ใช้การบีบอัดข้อมูลภาพด้วยวิธีการของ JPEG ทำการลดขนาดของภาพ เพื่อให้ลดเนื้อที่ในการจัดเก็บ ทำให้อุปกรณ์ขนาดเท่าเดิมแต่เก็บข้อมูลได้มากขึ้น หรือทำการส่งข้อมูลได้เร็วขึ้น โดยทำการออกแบบวงจรให้มีขนาดเล็กเพื่อสามารถนำไปประยุกต์ใช้กับระบบสื่อสารในปัจจุบันได้

การบีบอัดข้อมูลภาพแบบ JPEG โดยใช้อัลกอริทึมของการแปลงแบบดิสครีตโคซายน์(Discrete Cosine Transform) และ Huffman Coding เพื่อทำการบีบอัดข้อมูลภาพ โดย การแปลงแบบดิสครีตโคซายน์จะทำการแปลงข้อมูลภาพจากโดเมนเวลา (Time Domain) เป็นโดเมนความถี่ (Frequency Domain) แล้วนำผลลัพธ์มาประมวลผลเพื่อลดขนาด ด้วยวิธีการควอนไตเซชั่น(Quantization) เพื่อลดจำนวนบิตและนำข้อมูลที่ได้ไปทำการเรียงข้อมูลแบบซิกแซก (Zigzag) จากนั้นจะทำการเข้ารหัสโดยใช้วิธีการของ Huffman encoding โดยเราได้นำเทคนิคการบีบอัดภาพนิ่งแบบ JPEG มาออกแบบเป็นวงจรในตัว FPGA

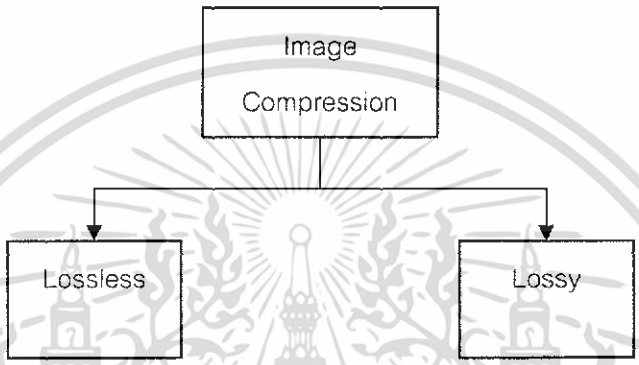
ทำการออกแบบวงจร, สังเคราะห์วงจรและทดสอบ ด้วยภาษา VHDL หลังจากนั้นจึงนำมาสร้างเป็น Hardware โดยใช้ร่วมกับอุปกรณ์ FPGA ซึ่งอุปกรณ์ FPGA นั้นจะมีขีดความสามารถในการทำหน้าที่ในส่วนของการประมวลผลสัญญาณ

บทที่ 2
ทฤษฎีและหลักการ

2.1 การบีบอัดข้อมูลภาพ

2.1.1 ประเภทของการบีบอัดข้อมูลภาพ

โดยทั่วไป รูปแบบการบีบอัดข้อมูลภาพสามารถแบ่งเป็น 2 ประเภทดังนี้



รูปที่ 2.1 ประเภทของการบีบอัดข้อมูลภาพ

1. การบีบอัดข้อมูลแบบไม่มีการสูญเสีย (Lossless compression) เป็นวิธีการที่ข้อมูลภาพเมื่อผ่านขั้นตอนการบีบอัดแล้วผลลัพธ์ของข้อมูลที่ได้จะเหมือนกับภาพต้นแบบ (Original image) ทุกประการนิยมใช้กับข้อมูลที่มีความสำคัญมากและสูญเสียไม่ได้ เช่น ข้อมูลภาพที่ใช้วินิจฉัยทางการแพทย์ เป็นต้น

2. การบีบอัดข้อมูลแบบมีการสูญเสีย (Lossy compression) เป็นวิธีการที่ผลลัพธ์ของข้อมูลภาพที่ได้จากการขยายข้อมูลกลับมามีความแตกต่างจากต้นแบบ โดยยอมให้มีการสูญเสียข้อมูลได้บ้าง นิยมใช้ในการประมวลผลภาพทั่วไปเป็นอย่างมาก เนื่องจากตาของคนเราไม่สามารถตรวจจับข้อมูลในส่วนที่มีการสูญเสียได้ อย่างไรก็ตาม คุณภาพของภาพที่ได้มีความสัมพันธ์กับอัตราการบีบอัดข้อมูลด้วย (compression ratio) นั่นคือ ข้อมูลภาพที่มีอัตราการลดขนาดที่สูง จะได้ขนาดของข้อมูลภาพที่มีขนาดเล็ก แต่ทำให้คุณภาพของภาพแย่ลงไปด้วย

2.1.2 มาตรฐานการบีบอัดข้อมูลภาพนิ่ง

JPEG เป็นมาตรฐานการลดขนาดและขยายภาพนิ่งที่กำหนดขึ้นโดย Joint Photographic Expert Group ซึ่งเป็นความร่วมมือกันระหว่าง ISO (International Organization for Standardization) และ ITU (International Telecommunication Union) โดยมีวัตถุประสงค์ที่จะพัฒนามาตรฐานการบีบอัดข้อมูลภาพดังนี้

1. ผู้ใช้สามารถปรับเปลี่ยนอัตราส่วนของการบีบอัดข้อมูลภาพได้
2. สามารถนำไปใช้งานกับภาพดิจิตอลต่างๆได้
3. มีขั้นตอนการคำนวณที่ไม่ยุ่งยากซับซ้อนเพื่อนำไปใช้งานต่างๆได้อย่างกว้างขวาง
4. JPEG ประกอบด้วยการทำงาน 4 โหมดดังนี้

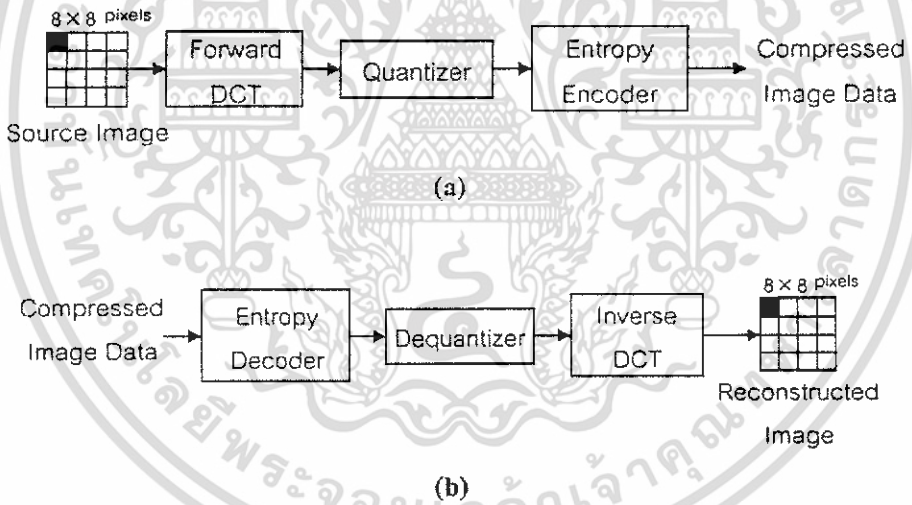
    4.1.1 การเข้ารหัสแบบ Sequential

    4.1.2 การเข้ารหัสแบบ Progressive

    4.1.3 การเข้ารหัสแบบ Lossless

    4.1.4 การเข้ารหัสแบบ Hierarchical

โดยจะแบ่งเป็นแบบมีการสูญเสีย 3 โหมด และแบบไม่มีการสูญเสีย 1 โหมดในหัวข้อนี้จะเน้นการบีบอัดข้อมูลภาพชนิด Sequential Baseline JPEG ซึ่งสามารถกำหนดระดับคุณของภาพด้วยค่า Q (Quality factor) โดยค่า Q ที่มีค่าสูงจะทำให้คุณภาพของภาพที่ได้สูงแต่จะมีขนาดของไฟล์ที่ใหญ่กว่าเมื่อเปรียบเทียบกับค่า Q ที่มีค่าต่ำกว่า ซึ่งจะให้มีขนาดของไฟล์ที่เล็กกว่าแต่คุณภาพของภาพที่ได้แย่กว่า เป็นความสัมพันธ์พื้นฐานของคุณภาพและอัตราการบีบอัดข้อมูล ในการบีบอัดข้อมูลแบบมีการสูญเสีย JPEG มีการบีบอัดข้อมูลที่สมมาตร (Symmetrical algorithm) เนื่องจากมีขั้นตอนลดขนาดและขั้นตอนการขยายข้อมูลที่เท่ากันดังรูปที่ 2.2 ดังนั้นเวลาที่ใช้ย่อมเท่ากันด้วย



รูปที่ 2.2 JPEG ชนิด Sequential Baseline (a) การเข้ารหัส (b) การถอดรหัส

2.1.3 กระบวนการทำงานของ JPEG

ลำดับของการทำงานแบบ JPEG เป็นไปตามรูปที่ 2.2 โดยการแปลงค่าสัมประสิทธิ์การแปลงแบบดิสครีตโคซายน์ ถ้าจำนวนข้อมูลยิ่งมาก (n ยิ่งมาก) จะทำให้ใช้เวลาและกำลังในการคำนวณมาก ดังนั้นภาพต้นฉบับจะถูกแบ่งออกเป็นบล็อก (Block) เล็กๆ ขนาด 8×8 พิกเซล (pixel) ถ้าเป็นภาพที่แสดงผลด้วยจำนวน n บิทต่อพิกเซลแล้ว ค่าข้อมูลพิกเซลจะมีค่าตั้งแต่ 0 ถึง 2^n -1 ก่อนที่จะผ่านการแปลงแบบดิสครีตโคซายน์ (Forward Discrete Cosine Transform) หรือ Forward DCT จะมีการสร้างค่าของพิกเซลให้เป็นจำนวนเต็มแบบมีเครื่องหมาย ซึ่งมีค่าตั้งแต่ -2^n ถึง 2^n-1 ตัวอย่างเช่น

ภาพแบบเกรย์สเกล(Gray scale) ที่มี 8 บิทต่อพิกเซล (ความเข้ม $2^8 = 256$ ระดับ) มีค่าตั้งแต่ 0 – 255 เมื่อจะผ่านการแปลงต้องทำข้อมูลให้เป็น -128 ถึง 127 ก่อน

เมื่อผ่านการแปลงแบบดิสครีตโคซายน์แล้ว เอาท์พุทที่ได้จะเป็นค่าสัมประสิทธิ์ทางความถี่ โดยมีข้อมูลตัวแรกของบล็อกเรียกว่าค่าสัมประสิทธิ์ DC (DC Coefficient) เป็นค่าเฉลี่ยของข้อมูลภายในบล็อกนั้นๆ และข้อมูลที่เหลือจำนวน 63 ค่า เรียกว่าสัมประสิทธิ์ค่า AC (AC Coefficient) ต่อมาจะเป็นการควอนไตซ์ค่าสัมประสิทธิ์ DCT (DCT coefficient) โดยทำการควอนไตซ์บล็อกย่อยๆด้วยค่าสัมประสิทธิ์ของการควอนไตซ์และทำการปัดเศษค่าให้เป็นเลขจำนวนเต็ม ทำให้ข้อมูลเป็นศูนย์มากเพื่อสะดวกในการเข้ารหัส ในขั้นตอนนี้เองที่จะมีการลดขนาดของข้อมูล และเกิดมีความผิดพลาดจากการควอนไตซ์ขึ้น ในการควอนไตซ์นี้จะเป็นแบบไม่เป็นยูนิฟอร์ม (Non - Uniform) กล่าวคือ ค่าสัมประสิทธิ์แต่ละค่าจะถูกควอนไตซ์ด้วยระดับการควอนไตซ์ (Step Size) ที่มีเท่ากันเสมอ ขนาดของระดับขั้นในการควอนไตซ์จะถูกกำหนดโดยค่าในตารางเรียกว่าเป็นตารางค่าควอนตัมนั่นเอง ซึ่งค่าในตารางนี้จะเป็นค่าที่ใช้เฉพาะกับสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์แต่ละค่าเท่านั้น

ค่าสัมประสิทธิ์ที่ถูกควอนไตซ์และจะถูกนำมาจัดตามลำดับซิกแซก เพื่อความสะดวกในการเข้ารหัสซึ่งอาจใช้วิธีของ Huffman Encoding หรือวิธีการเข้ารหัสแบบความยาวของรหัสไม่คงที่ (Variable Length Encoding) เอาท์พุทจากกระบวนการนี้จะเป็นข้อมูลภาพที่ถูกลดขนาดเรียบร้อยแล้ว

2.1.4 การแปลงแบบดิสครีตโคซายน์

การแปลงแบบดิสครีตโคซายน์นั้น เป็นวิธีที่นิยมมากในเรื่องของการบีบอัดของข้อมูลที่ต้องการความรวดเร็วและมีประสิทธิภาพ และวิธีการนี้ก็เป็นกระบวนการของการบีบอัดข้อมูลที่มีการสูญเสีย (Lossy compression) โดยวิธีที่ใช้เป็นมาตรฐานอยู่ขณะนี้ก็คือ JPEG (Joint Photographics Expert Group) โดยมีพื้นฐานของการแปลงแบบดิสครีตโคซายน์เป็นหลัก

การแปลงแบบดิสครีตโคซายน์เป็นการประมวลผลทางคณิตศาสตร์ ที่เป็นการคำนวณส่วนประกอบทางความถี่ของแซมเปิล (sample) ของสัญญาณที่อัตราการแซมปลิง(sampling)ใดๆ ซึ่งการการแปลงแบบดิสครีตโคซายน์นี้จะต้องประยุกต์ใช้กับจำนวนของแซมเปิลที่รู้ค่าแน่นอน ในทางคณิตศาสตร์สามารถพิจารณาได้ง่ายๆ คือ ถ้าจำนวนที่ใช้เป็นกำลังคู่ของ 2 การแปลงแบบดิสครีตโคซายน์แบบทิศทางเดียวใช้ในการเปลี่ยนอาร์เรย์ (Array) หนึ่งของจำนวนที่แสดงค่าแอมปลิจูด (Amplitude)ของสัญญาณที่จุดที่ทราบค่าใดๆ บนโดเมนของเวลาไปเป็นอีกอาร์เรย์หนึ่ง ซึ่งใช้แสดงค่าแอมปลิจูดที่เป็นองค์ประกอบ (Component) ทางความถี่ ของสัญญาณที่เป็นอินพุทอาร์เรย์ผลลัพธ์จะมีอีเลเมนท์ (Element) แรกเป็นค่าเฉลี่ยของทุกๆ แซมเปิลของอาร์เรย์อินพุทเรียกว่าเป็นสัมประสิทธิ์ DC และอีเลเมนท์ที่เหลือก็เริ่มมีความถี่เข้าเกี่ยวข้องจึงเรียกว่าสัมประสิทธิ์ AC ที่เป็นค่าเฉพาะของแต่ละอีเลเมนท์ (ค่าสัมประสิทธิ์ (Coefficient) คือ แอมปลิจูดขององค์ประกอบทางความถี่)

สมการที่ใช้ในการแปลงแบบดิสครีตโคซายน์แบบทิศทางเดียวนั้นเป็นดังสมการที่ 2.1

$$y(n) = e(n)\sum_{k=0}^{N-1} x(k)\cos\left[\frac{(2k+1)n\pi}{2N}\right] \quad :n = 0,1,\ldots,N-1 \qquad (2.1)$$

เมื่อ $e(n) = 1/\sqrt{2}$ เมื่อ $n = 0$

$= 1$ เมื่อ $n \neq 0$

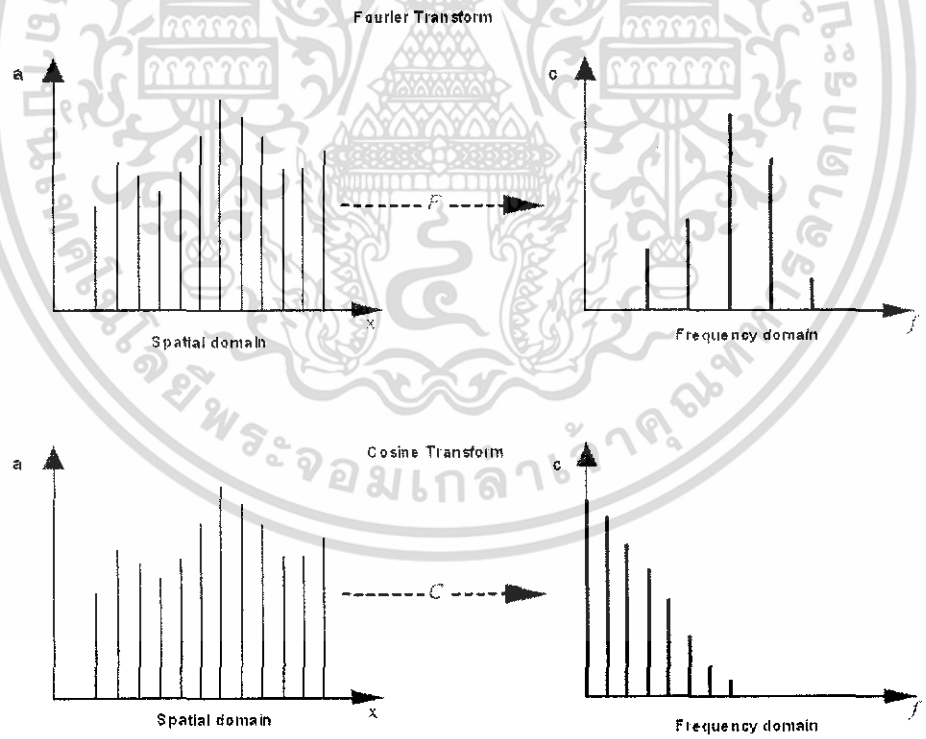
(n = ดัชนีของอาร์เรย์ผลลัพธ์)

(k = ดัชนีของแซมเปิลอาร์เรย์)

(N = ขนาดของแซมเปิลอาร์เรย์)

พิจารณาจากรูปที่ 2.3 เป็นการเปรียบเทียบระหว่างการแปลงโดเมนทางเวลาของสัญญาณใดๆ เป็นโดเมนของความถี่โดยใช้การแปลงแบบฟูเรียร์ (Fourier Transform) และการแปลงแบบดิสครีตโคซายน์จะเห็นได้อย่างชัดเจนว่าทั้งสองวิธีมีการรวบรวมส่วนประกอบที่สำคัญของข้อมูลไว้ได้เหมือนกัน แต่ที่ความถี่ต่างกันกล่าวคือ การแปลงแบบดิสครีตโคซายน์จะสามารถเก็บส่วนประกอบส่วนใหญ่ของภาพไว้ที่ความถี่เท่ากับศูนย์ ส่วนการแปลงฟูเรียร์ ความถี่ที่รวบรวมข้อมูลส่วนใหญ่ที่สำคัญของภาพจะไม่ได้อยู่ที่ความถี่เท่ากับศูนย์ ซึ่งจากรูปจะเห็นว่าข้อมูลในโดเมน ความถี่จะมีลักษณะเด่นอยู่ 3 ประการที่เห็นได้ชัดคือ

1. ค่าความถี่ที่ศูนย์จะเป็นค่าของความเข้มเฉลี่ยของข้อมูล
2. ค่าความถี่สูงเป็นค่าที่บอกถึงข้อมูลที่มีการเปลี่ยนแปลง
3. ค่าความถี่ต่ำเป็นค่าที่บอกรายละเอียดโดยรวมของข้อมูล



รูปที่ 2.3 แสดงการแปลงแบบฟูเรียร์และการแปลงแบบดิสครีตโคซายน์ตามลำดับ

การที่การแปลงแบบดิสครีตโคซายน์สามารถรวบรวมส่วนประกอบส่วนใหญ่ของภาพไว้ที่ความถี่ต่ำๆ ได้ ซึ่งสิ่งนี้มีความสำคัญมากเพราะภาพส่วนใหญ่นั้นข้อมูลของภาพจะรวมกันอยู่ในช่วงของความถี่ต่ำและส่วนประกอบที่พบในจุดที่แถวและคอลัมน์เป็นศูนย์ (ส่วนประกอบ DC) จะเก็บข้อมูลที่มีประโยชน์เกี่ยวกับภาพมากกว่าส่วนประกอบที่ความถี่ที่สูงกว่านี้ ส่วนประกอบตัวที่อยู่ห่างจากส่วนประกอบ DC นั้นจะพบว่าไม่เพียงแต่ค่าสัมประสิทธิ์มีแนวโน้มที่จะต่ำลงแต่ยังมีความสำคัญต่อการอธิบายภาพน้อยลงอีกด้วย ดังนั้นการแปลงแบบดิสครีตโคซายน์จะทำให้สามารถตัดข้อมูลบางส่วนออกไปได้โดยไม่มีผลต่อคุณภาพของภาพมากนักซึ่งถ้าภาพไม่ได้ถูกทำการแปลงแต่ยังคงอยู่ในรูปของจุดพิกเซลจะไม่สามารถตัดข้อมูลบางส่วนของภาพออกไปได้เช่นนี้เพราะแต่ละพิกเซลมีผลต่อการมองเห็นของภาพโดยรวม

2.1.5 การแปลงโคซายน์แบบไม่ต่อเนื่อง 2 ทิศทาง

เนื่องจากภาพโดยทั่วๆ ไปประกอบขึ้นจากฟังก์ชัน 2 ฟังก์ชันดังนั้นในการกระทำกระบวนใดๆ กับภาพจำเป็นต้องทำในลักษณะ 2 แนวพร้อมๆ กัน ในการทำการแปลงรูปสัญญาณก็เช่นเดียวกันจำเป็นต้องทำใน 2 ทิศทางดังนั้นจึงต้องทำการแปลงโคซายน์แบบไม่ต่อเนื่องที่ได้กล่าวถึงไปแล้วนั้นสามารถอธิบายการแปลงโคซายน์แบบไม่ต่อเนื่อง 2 ทิศทางได้โดยนำภาพเริ่มต้นมาแบ่งเป็นบล็อกของพิกเซลขนาด 8×8 พิกเซล ซึ่งเป็นลักษณะของอาร์เรย์ของพิกเซล 2 มิติ โดยในแต่ละบล็อกของ 8×8 พิกเซล นั้นจะมีด้วยกันทั้งหมด 64 ค่า และแต่ละค่าจะแสดงค่าแอมปลิจูดของแต่ละค่าที่เข้ามาขนาดของสัญญาณนี้จะเป็นฟังก์ชันของจุดพิกัด 2 จุด โดยสมมติให้ $a = f(x,y)$ โดย x,y จะเป็นมิติของสปาเชียลโดเมน เมื่อพิจารณาอาร์เรย์ของบล็อกขนาด 8×8 พิกเซล การแปลงโคซายน์แบบไม่แบบทิศทางเดียวจะถูกนำมาใช้แยกกันสำหรับแต่ละแถวและแต่ละหลัก การแปลงแบบดิสครีตโคซายน์จะทำการแปลงข้อมูลในลักษณะของพิกเซลไปเป็นสัมประสิทธิ์ทางความถี่ โดยจะรวมข้อมูลของภาพทั้งหมดแล้วแทนด้วยค่าสัมประสิทธิ์ทางความถี่ ซึ่งผลจากการแปลงจะมีลักษณะเป็นจุดๆ เช่นเดียวกับจุดพิกเซลก่อนที่จะทำการแปลงแต่จุดต่างๆ นี้จะแทนค่าสัมประสิทธิ์ทางความถี่แต่ละค่าและสมการที่ใช้ในการแปลงแบบดิสครีตโคซายน์ 2 มิตินั้นมีสมการดังนี้

สมการการแปลงแบบดิสครีตโคซายน์(Forward DCT)

$$y(m,n) = \frac{4}{N^2} e(m)e(n) \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} x(k,l) \cos\left[\frac{(2k+1)n\pi}{2N}\right] \cos\left[\frac{(2l+1)n\pi}{2N}\right] \qquad (2.2)$$

สมการการแปลงกลับแบบดิสครีตโคซายน์(Inverse DCT)

$$x(k,l) = \frac{2}{N} \sum_{n=0}^{N-1} \sum_{n=0}^{N-1} e(m)e(n)y(m,n) \cos\left[\frac{(2k-1)m\pi}{2N}\right] \cos\left[\frac{(2l-1)n\pi}{2N}\right] \qquad (2.3)$$

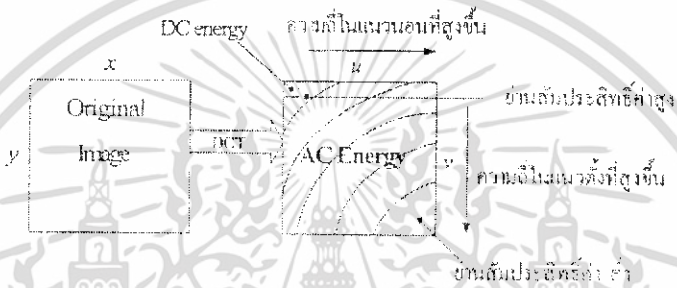
โดย

$$e(m,n) = \begin{cases} \frac{1}{\sqrt{2}} & ;n = 0 \quad ;u,v = 0 \\ 1 & ;etc \quad ;other \end{cases}$$

และ $y(m,n)$ เป็นข้อมูลภาพเริ่มต้นและภาพผลลัพธ์

$x(k,l)$ เป็นสัมประสิทธิ์ของการแปลง

ผลลัพธ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ 2 มิติ ของข้อมูลภาพ จะประกอบด้วย 2 ส่วน คือส่วนพลังงาน DC (DC component) หรือระดับความสว่าง และส่วนพลังงาน AC (AC Component) หรือระดับความคมชัด โดยพลังงาน DC จะเป็นค่าเฉลี่ยของจุดภาพทั้งหมดจะปรากฏที่มุมซ้ายบนของภาพผลลัพธ์หรือที่จุด 0,0 ส่วนพลังงานของ AC จะกระจายแบ่งแยกเป็นชั้นๆ โดยมีค่าสัมประสิทธิ์จากมากไปหาน้อย ส่วนที่อยู่ใกล้จุดพลังงาน DC จะมีค่าสัมประสิทธิ์สูงสุด ส่วนที่อยู่ไกลจากจุดพลังงาน DC จะมีค่าสัมประสิทธิ์ต่ำสุด ดังแสดงได้ดังรูปที่ 2.4 และ ตารางที่ 2.1 แสดงตัวอย่างผลของการแปลงแบบดิสครีตโคซายน์



รูปที่ 2.4 แสดงส่วนประกอบของผลลัพธ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ 2 มิติ

ตารางที่ 2.1 (a) แสดงข้อมูลขนาด 8×8 (b) แสดงผลลัพธ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์

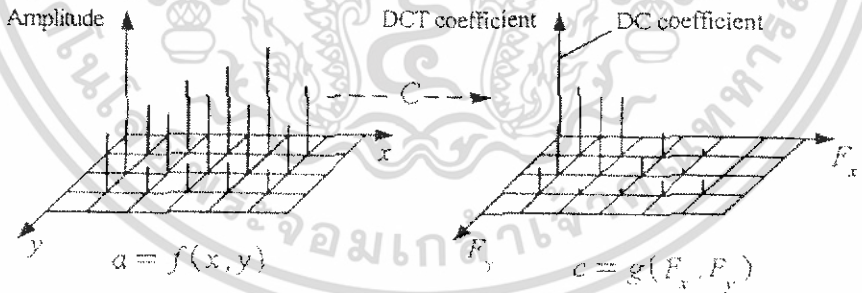
140	144	147	140	140	155	179	175
144	152	140	147	140	148	167	179
125	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	132
147	167	140	155	155	140	136	132
136	156	123	162	162	144	140	147
148	155	136	155	152	147	147	175

(a)

151	18	15	-9	23	-9	-14	-19
21	-34	26	-9	-11	11	14	7
-10	-24	-2	6	-18	3	-20	-1
-8	-5	14	-15	-8	-3	-3	8
-3	10	8	1	-11	18	18	15
4	-2	-18	8	8	-4	1	-7
9	1	-3	4	-1	-7	-1	-2
0	-8	-2	2	1	4	-6	0

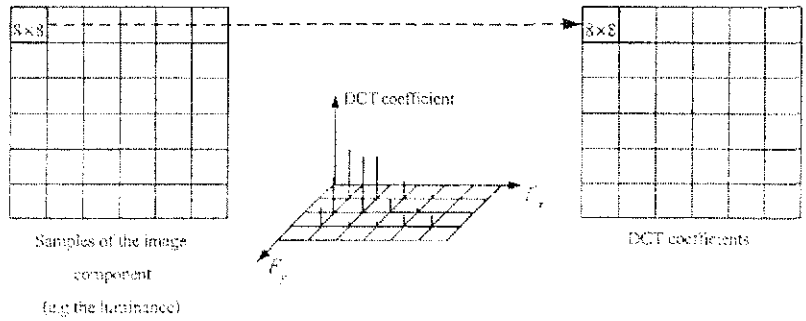
(b)

จากคุณลักษณะที่ได้จากการแปลงแบบดิสครีตโคซายน์จะเห็นได้ว่า เมื่อความถี่ยิ่งสูงขึ้นค่าสัมประสิทธิ์ในส่วนของพลังงาน AC จะมีค่าน้อยลงหรือเข้าใกล้ศูนย์ ดังนั้นเมื่อต้องการบีบอัดข้อมูลภาพก็ทำการส่งเฉพาะส่วนที่มีค่าสัมประสิทธิ์สูงออกไป และตัดส่วนสัมประสิทธิ์ต่ำที่ไม่มีผลกระทบต่อข้อมูลภาพมากนักทิ้งไปก็จะสามารถลดทอนข้อมูลในการส่งภาพได้ นอกจากนี้สามารถพิจารณาลักษณะการกระจายของสัมประสิทธิ์การแปลงแบบดิสครีตโคซายน์เทียบกับ สัมประสิทธิ์จากการแปลงแบบฟูเรียร์ ดังแสดงในรูปที่ 2.5 แสดงภาพในลักษณะ 3 มิติ ของการแปลงแบบดิสครีตโคซายน์ของบล็อก 8 × 8 โดย a เป็นขนาดของข้อมูลในสปาเชียลโดเมน และ C เป็นค่าขนาดของกลุ่มพลังงานในโดเมนความถี่



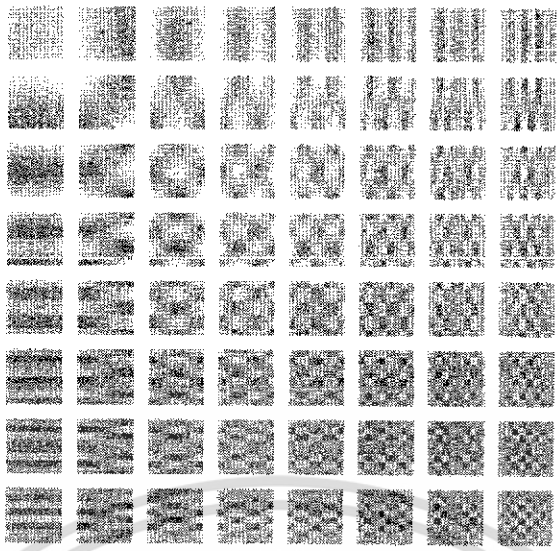
รูปที่ 2.5 แสดงการแปลงแบบดิสครีตโคซายน์แบบ 2 มิติ

จากในภาพที่ 2.5 จะเห็นว่าค่าสัมประสิทธิ์การแปลงแบบฟูเรียร์นั้นมีลักษณะการกระจายที่มากกว่าการแปลงแบบดิสครีตโคซายน์ และเมื่อพิจารณาสัมประสิทธิ์ที่ได้ที่ความถี่เดียวกันในแต่ละบล็อกจะมีค่าอยู่ในช่วงที่ใกล้เคียงกัน และตัวอย่างในภาพลักษณะ 3 มิติในบางส่วนของบล็อก 8×8 ดังแสดงในรูปที่ 2.6

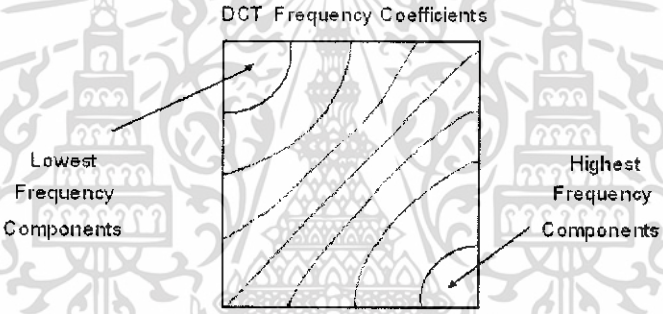


รูปที่ 2.6 แสดงการแปลงแบบดิสครีตโคซายน์ที่ทำในบล็อกขนาด 8×8

หลังจากผ่านกระบวนการแปลงด้วยการแปลงแบบดิสครีตโคซายน์จะได้รูปแบบของฟังก์ชันที่อยู่ในโดเมนของความถี่ ซึ่งจะได้สัมประสิทธิ์ทางความถี่ที่อยู่ในรูปของบล็อกขนาด 8×8 ค่า (8 แถวและ 8 หลัก) โดยสมมติให้มีฟังก์ชันเป็น $c = g(F_x, F_y)$ โดย c จะเป็นค่าของสัมประสิทธิ์ ส่วนค่าของ F_x และ F_y จะแสดงถึงค่าที่เกิดขึ้นในแกนของความถี่ในแต่ละทิศทาง สัมประสิทธิ์ของ g(0,0) นั้นจะเป็นสัมประสิทธิ์ที่ค่าความถี่ศูนย์ซึ่งจะเรียกว่าสัมประสิทธิ์ DC ซึ่งจะเป็นค่าเฉลี่ยของค่าในแต่ละบล็อก ค่าต่างๆ ในแถวที่ศูนย์จะมีส่วนประกอบทางความถี่เป็นศูนย์ในทิศทางหนึ่งและค่าทั้งหมดในคอลัมน์ที่ศูนย์ที่จะมีส่วนประกอบทางความถี่เป็นศูนย์ในอีกทิศทางหนึ่ง เมื่อแถวและคอลัมน์เพิ่มขึ้นในทิศทางที่ห่างออกจากจุดเริ่มต้นสัมประสิทธิ์ในเมตริกซ์ที่ได้ จากการแปลงแบบดิสครีตโคซายน์จะแทนความถี่ที่สูงขึ้น และความถี่ที่สูงที่สุดคือที่ตำแหน่ง *N-1* ของบล็อกข้อมูล ซึ่งโดยปกติพิกเซลจะมีการเปลี่ยนแปลงน้อยลงจากจุดถึงจุด ดังนั้นขนาดของความถี่ที่ต่ำที่สุดจะสูงที่สุดส่วนค่าของความถี่กลางและสูงจะมีค่าที่น้อยหรืออาจจะเป็นศูนย์ซึ่งในส่วนนี้จะไม่นำมาใช้อาจจะตัดทอนลงไปดังจะเห็นจากรูปที่ 2.6 ซึ่งจะเป็นประโยชน์ต่อการลดขนาดของภาพลงได้ในขั้นตอนของการควอนไตเซชั่น และเมื่อนำแต่ละสัมประสิทธิ์ทางความถี่มาผ่านการแปลงกลับแบบดิสครีตโคซายน์โคซายน์แบบ 2 มิติแล้ว จะได้ค่าแอมปลิจูดเป็นพื้นที่ที่มีความขาวดำด้วยอัตราความคมชัด (Contrast Ratio) ซึ่งกำหนดโดยแอมปลิจูดซึ่งในรูปที่ 2.7 แสดงถึงแพทเทิร์น(pattern)ที่สร้างจากค่าสัมประสิทธิ์ทางความถี่ของการแปลงแบบดิสครีตโคซายน์ของ 8×8 พิกเซล สมมุติว่าแต่ละค่าเป็นค่าสูงที่สุดแล้ว จะเห็นได้ว่าบล็อกบนซ้ายมือจะเป็นสีเทาทึบแสดงถึงค่าเฉลี่ยของความเข้มทั้งบล็อกขนาด 8 × 8 พิกเซล ส่วนบล็อกที่อยู่มุมล่างขวามือจะมีลักษณะเป็นตารางหมากรุก เพราะมีองค์ประกอบทางความถี่ที่สูงสุดตามแนวตั้งและแนวนอน บล็อกที่มุมขวามือด้านบนและซ้ายมือด้านล่างแสดงแถบตามแนวนอนและแนวตั้งตามลำดับ เพราะมีความถี่สูงสุดทางแนวนอนแต่ต่ำสุดตามแนวตั้งและความถี่สูงสุดทางแนวตั้งแต่ต่ำสุดทางแนวนอนตามลำดับ ดังรูปที่ 2.7



รูปที่ 2.7 แสดงแพทเทิร์นของความเข้มของสัมประสิทธิ์การถ่วงน้ำหนักการแปลงแบบดิสครีตโคซายน์



รูปที่ 2.8 การกระจายของส่วนประกอบทางความถี่ (ประมาณ) การแปลงแบบดิสครีตโคซายน์ 2 มิติ

2.1.6 การควอนไตซ์ (Quantization)

การควอนไตซ์เป็นการปรับลดค่าสัมประสิทธิ์ที่ได้จากการทำการแปลงแบบดิสครีตโคซายน์ เพื่อลดจำนวนบิทที่ใช้ในการเก็บซึ่งมีผลทำให้ความแม่นยำ (precision) ของข้อมูลลดลง กล่าวคือ ทำให้เกิดการสูญเสียข้อมูลบางส่วนไป โดยสัมประสิทธิ์จากการแปลงแบบดิสครีตโคซายน์ทั้ง 64 ค่าจะถูกทำการควอนไตซ์โดยใช้ตารางการควอนไตซ์ขนาด 64 อีเลเมนท์ การควอนไตซ์จะลดค่าแอมปลิจูดของสัมประสิทธิ์ที่มีผลกับคุณภาพของภาพน้อยที่สุด เพื่อให้เป็นไปตามหลักการเพิ่มค่าสัมประสิทธิ์ที่มีค่าศูนย์ให้มากขึ้น และลดค่าสัมประสิทธิ์ที่ไม่มีความสำคัญลง การควอนไตซ์กระทำได้ดังสมการต่อไปนี้

$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Q(u,v)}\right] \tag{2.4}$$

และมีสมการดิควอนไตเซชั่น (Dequantization) ดังนี้

$$\dot{T}(u,v) = T(u,v)Q(u,v) \tag{2.5}$$

เมื่อ $T(u,v)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์

$\hat{T}(u,v)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ เมื่อถูกควอนไตซ์

$\dot{T}(u,v)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ เมื่อถูกดิควอนไตซ์

$Q(u,v)$ คือ ค่าการควอนไตซ์

และ *round* คือ การหาจำนวนเต็มที่มีค่าใกล้เคียงที่สุดโดย $0 \le u \le 7, 0 \le v \le 7$

หลังจากผ่านการควอนไตซ์แล้ว ขั้นตอนต่อไปคือการเรียงข้อมูลแบบซิกแซก โดยเป็นการเรียงค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ให้เป็นในบล็อกข้อมูล 8×8 ค่าให้เป็นแถวข้อมูล 64 ค่าโดยเรียงลำดับจากค่าสัมประสิทธิ์ของความถี่ต่ำไปหาค่าสัมประสิทธิ์ของความถี่สูงซึ่งทำให้ค่าสัมประสิทธิ์ที่มีค่าศูนย์ส่วนใหญ่ในช่วงความถี่สูงอันเนื่องมาจากการควอนไตซ์มาอยู่ติดกันซึ่งจะเป็นประโยชน์ในการเข้ารหัสในขั้นตอน

2.1.7 การเข้ารหัสข้อมูล

กระบวนการสุดท้ายของ JPEG คือการเข้ารหัสข้อมูลที่ผ่านการควอนไตซ์แล้ว ซึ่งมาตรฐาน JPEG มีการเข้ารหัสเอนโทรปี(Entropy encoding) 2 แบบ คือ การเข้ารหัสฮัฟแมน (Huffman encoding) และการเข้ารหัสเชิงอริธเมติก (Arithmetic Encoding) ซึ่งในระบบ Sequential Baseline จะใช้การเข้ารหัสแบบฮัฟแมนเท่านั้น โดยวิธีการเข้ารหัสฮัฟแมน นั้นเป็นวิธีการเข้ารหัส (encode) ข้อมูลด้วยการสร้างรหัส ขนาดต่างๆ ขึ้นจากสถิติของจำนวนข้อมูลที่เกิดขึ้น เพื่อนำรหัสที่สร้างขึ้นนั้นไปใช้แทนตัวข้อมูลเดิม โดยพยายามสร้างรหัส ที่มีขนาดสั้นสำหรับใช้แทนตัวข้อมูลที่เกิดขึ้นจำนวนมากและรหัส ที่มีขนาดยาวขึ้นเพื่อแทนตัวข้อมูลที่เกิดขึ้นน้อยลงลดหลั่นกันไป สำหรับวิธีการเข้ารหัสฮัฟแมน ที่ใช้ใน JPEG นั้นได้มีการอาศัยเทคนิคการการเข้ารหัส วิธีอื่นมาช่วยด้วยคือ การเข้ารหัสแบบแวริเอเบิลเลนจ์ (variable-length encoding) และ การเข้ารหัสแบบรันเลนจ์ (runlength encoding) ซึ่งจะแบ่งการเข้ารหัส เป็น 2 ส่วนคือ การเข้ารหัส ค่า DC และค่า AC ตามลำดับ โดยในการเข้ารหัสข้อมูลจะประกอบด้วย 3 ขั้นตอนคือ

1. การเก็บค่าสัมประสิทธิ์กระแสตรงของบล็อกพิกเซล (8×8) บล็อกแรกก่อน แล้วทำการเปลี่ยนค่าสัมประสิทธิ์กระแสตรงที่สุด (0,0) ของบล็อกต่อมาเป็นค่าที่สัมพัทธ์กับค่าแรกหรือเป็นค่าผลต่างนั่นเอง แล้วจึงทำการเข้ารหัสข้อมูลผลต่างนี้แทน เพื่อประโยชน์คือสามารถเข้ารหัสโดยใช้ค่าผลต่างที่ถือว่าน้อยด้วยจำนวนบิตที่น้อยกว่าการเข้ารหัสค่าจริง (เพราะถือว่าบล็อกของพิกเซลที่อยู่ใกล้กันจะมีส่วนที่แตกต่างกันน้อยมากและสัมประสิทธิ์กระแสตรงก็ได้รวมข้อมูลที่สำคัญของภาพไว้แล้ว)

2. การจัดลำดับสัมประสิทธิ์กระแสสลับเป็นลำดับแบบซิกแซกเพื่อให้ 0 ที่เกิดขึ้นจากกระบวนการควอนไตซ์มีความต่อเนื่องกันยาวๆ ทำให้สามารถเข้ารหัสได้ง่าย

3. เลือกทำการเข้ารหัสสัมประสิทธิ์กระแสสลับที่จัดลำดับแล้ว

3.1 ด้วยวิธีความต่อเนื่องของข้อมูล RLE (Run Length Encoding) ซึ่งข้อดีของวิธีนี้คือถ้า 0 ที่เรียงติดกันมีความต่อเนื่องกันยาวๆ มาก จะทำให้สามารถเข้ารหัสได้ง่าย สั้นลง

3.2 การเข้ารหัสแบบเอนโทรปี ซึ่งอาจจะทำได้โดยวิธีฮัฟแมน หรือการประมวลผลทางคณิตศาสตร์ (Arithmetic) ก็ได้ ซึ่งในที่นี้จะใช้การเข้ารหัสแบบฮัฟแมน

ซึ่งในวิธีการของ JPEG จะใช้วิธีการเข้ารหัสข้อมูลแบบฮัฟแมน ซึ่งเป็นการเข้ารหัสข้อมูลด้วยการสร้างรหัสขนาดต่างๆ ขึ้นจากสถิติของจำนวนข้อมูลที่เกิดขึ้น เพื่อนำรหัสที่สร้างนั้นไปใช้แทนตัวข้อมูลเดิม โดยพยายามสร้างรหัสที่มีขนาดสำหรับใช้แทน ตัวข้อมูลที่เกิดขึ้นจำนวนมาก และรหัสที่มีขนาดยาวขึ้นเพื่อแทน ตัวข้อมูลที่เกิดขึ้นน้อยลดหลั่นกันไปตามลำดับ

2.1.8 การเข้ารหัสข้อมูลแบบฮัฟแมน

การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นเป็นการเข้ารหัสเฉพาะข้อมูลที่มีค่าไม่เป็นศูนย์ ดังนั้นจากการพยายามลดค่าของข้อมูลให้เป็นศูนย์มากๆ ในขั้นตอนของการควอนไตซ์ และการอ่านข้อมูลแบบซิกแซกที่พยายามทำให้ค่าที่เป็นศูนย์มากอยู่เรียงกันนั้นจึงมีประโยชน์ต่อการเข้ารหัสข้อมูลแบบฮัฟแมนมาก เพราะจะทำให้ข้อมูลที่ต้องทำการเข้ารหัสมีจำนวนน้อยลง การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นจะแบ่งการเข้ารหัสข้อมูลเป็นสองส่วนคือการเข้ารหัสข้อมูลของค่า DC และการเข้ารหัสข้อมูลของค่า AC

2.1.8.1 การเข้ารหัสค่า DC ของแถวข้อมูล

ในการเข้ารหัสค่า DC ของแถวข้อมูลนั้น (ข้อมูลตัวแรกในแถวซึ่งคิดค่าสเปกตรัมสัมประสิทธิ์กระแสตรงที่จุด (0,0) ของบล็อกพิกเซลมาก 8×8) จะถูกเข้ารหัสเฉพาะค่าความแตกต่างระหว่างค่า DC ของแถวข้อมูลปัจจุบันกับค่า DC ของแถวข้อมูลซึ่งเป็นสีชนิดเดียวกัน ที่ถูกเข้ารหัสแถวล่าสุดที่ผ่านไป เช่น เมื่อกำลังทำการเข้ารหัส ค่า DC ของแถวข้อมูล C_b อยู่ซึ่งมีค่าเท่ากับ 40 จะทำการหาค่าความแตกต่างกันค่า DC ของแถวข้อมูล C_b ที่ถูกเข้ารหัสผ่านไปแล้วแถวล่าสุด (สมมติว่ามีค่าเท่ากับ 15) ดังนั้นค่าที่จะถูกนำมาเข้ารหัสคิด 25 (40-15) ซึ่งการเข้ารหัสข้อมูลค่า DC จะมีการเข้ารหัสสองส่วนคือ

1. SIZE หมายถึง จำนวนบิตที่จะต้องใช้ในการใส่ค่าข้อมูล (ผลต่าง) ซึ่งจะหาจากตารางฮัฟแมน
2. AMPLITUDE หมายถึง ค่าขนาดของผลต่าง

ดังนั้นรหัสที่เข้า คือ

(SIZE) (AMPILTUDE)

ในตารางฮัฟแมนจะเก็บความยาวต่าง ที่จะใช้ในการแทนข้อมูลตามค่า Category ของข้อมูลนั้น ซึ่งการหาค่า Category ของข้อมูลสามารถหารหัสข้อมูลได้จากตารางค่า JPEG Coefficient Coding Categories ซึ่งแสดงไว้ในตารางที่ 2.2 เมื่อได้ค่า Category แล้วก็สามารถหารหัสข้อมูลได้จากตารางค่า DC ของฮัฟแมน (ตารางที่ 2.3) โดยการเทียบหาตามค่า Category ที่ได้จากตารางที่ 2.2

ตารางที่ 2.2 ตารางแสดงค่า Categories ของข้อมูล

Range	DC Difference Category	AC Category
0	0	N/A
-1,1	1	1
-3,-2,2,3	2	2
-7,…,-4,4,…,7	3	3
-15,…,-8,8,…,15	4	4
-31,…,-16,16,…,31	5	5
-63,…,-32,32,…,63	6	6
-127,…,-64,64,…,127	7	7
-255,…,-128,128,…,255	8	8
-511,…,-256,256,…,511	9	9
-1023,…,-512,512,…,1023	A	A
-2047,…,-1024,1024,…,2047	B	B
-4095,…,-2048,2048,…,4095	C	C
-8191,…,-4096,4096,…,8191	D	D
-16383,…,-8192,8192,…,16383	E	E
-32767,…,-16384,16384,…,32767	F	F

ตารางที่ 2.3 แสดงตารางค่า DC ของการเข้ารหัสแบบฮัฟแมน

Category	Base Code	Length	Category	Base Code	Length
0	010	3	6	1110	10
1	011	4	7	11110	12
2	100	5	8	111110	14
3	001	5	9	1111110	16
4	101	7	A	11111110	18
5	110	8	B	111111110	20

จะเห็นว่ารหัสที่ได้จากตารางที่ 2.3 นั้นใช้เป็นตัวบอกช่วยของข้อมูลที่ถูกเข้ารหัสเท่านั้น ซึ่งจะได้รหัสตัวที่หนึ่งของข้อมูลออกมา ส่วนการระบุค่าของข้อมูลนั้น จำเป็นจะใช้รหัสตัวที่สองเป็นตัวระบุ โดยรหัสตัวที่สองนี้จะหามาจากค่าของข้อมูลโดยตรง โดยจะมีจำนวนบิทเท่ากับค่า Category ของข้อมูลที่นำมาเข้ารหัสนั่นเอง และจะเอาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

2.1.8.2 การเข้ารหัสค่า AC ของแถวข้อมูล

เนื่องจากการจัดเรียงข้อมูลแบบซิกแซก นั้น จะทำให้ค่าศูนย์ที่เกิดขึ้นจากการควอนไตซ์อยู่เรียงกันอย่างต่อเนื่องยาวๆ ทำให้สามารถเข้ารหัสได้ง่าย และยิ่งมีศูนย์ที่ต่อเนื่องกันมากก็จะสามารถเข้ารหัสด้วยจำนวนที่น้อยลงได้มาก และโดยมากค่าศูนย์จะต่อเนื่องกันอยู่ในส่วนท้ายๆ ของแถวข้อมูล

การเข้ารหัสค่า AC (ข้อมูลตั้งแต่ตัวที่ 2 ถึงตัวที่ 64 ในแถวข้อมูล) จะต่างจากในการเข้ารหัสค่า DC คือจะทำการเข้ารหัสค่าของข้อมูลโดยตรง (ไม่เข้ารหัสเฉพาะค่าความแตกต่างเหมือนใน DC) และการเข้ารหัสค่า AC นั้นจะทำการเข้ารหัสเฉพาะค่า AC ที่มีค่าไม่เป็นศูนย์เท่านั้น ส่วนค่า AC ที่เป็นศูนย์นั้น JPEG จะอาศัยการเข้ารหัสแบบ run-length มาช่วยในการเข้ารหัสดังนี้คือในการเข้ารหัสค่า AC ที่ไม่เป็นศูนย์แต่ละตัวจะเน้นจำนวนข้อมูล AC ที่เป็นศูนย์ซึ่งอยู่ติดกันด้านหน้าของข้อมูลที่จะเข้ารหัสนั้นโดยถือเป็นค่า Run ของข้อมูลในการเข้ารหัส เช่น ค่า AC ภายในแถวเป็น 5 8 0 0 9 7 0 0 0 0 4 ..... จะทำการเข้ารหัสเฉพาะข้อมูลที่ไม่เป็นศูนย์และมีค่า Run ของข้อมูลแต่ละตัวดังนี้คือ 5 (Run =0), 8(Run = 0), 9(Run =2) , 7(Run = 0), 4(Run = 4), ...

โดยมีการเข้ารหัสข้อมูลดังกล่าวคล้ายใน DC แต่จะใช้รหัส 3 ตัว แทนข้อมูลแต่ละตัวดังนี้

1. RUNLENGTH คือ จำนวนค่า 0 ที่ต่อเนื่องกันก่อนหน้าข้อมูลตัวที่จะเข้ารหัส
2. SIZE คือ จำนวนบิตที่จะต้องใช้ในการใส่ค่าข้อมูลที่ไม่เท่ากับ 0 ซึ่งหาจากตารางฮัฟแมน
3. AMPLITUDE คือ ค่าแอมปลิจูดหรือค่าของข้อมูลที่ไม่เท่ากับ 0 นั่นเอง

ดังนั้นรหัสคือ

(RUNLENGTH, SIZE) (AMPLITUDE)

หมายเหตุ

มีเงื่อนไขพิเศษ คือ ถ้ามี 0 ต่อเนื่องกัน มากกว่า 16 ตัว สมมุติว่าเป็น 22 ตัว จะได้รหัสเป็น (15, 0) (6, 4) (13)

(15,0) ความหมายคือ มี 0 ต่อเนื่องกัน 16 ตัว (เป็นรูปแบบที่กำหนดตายตัว)

(6,4)(13) ความหมายคือ มี 0 อีก 6 ตัวใช้ 4 บิตในการแทนค่าข้อมูลที่มีค่าเท่ากับ 13

ซึ่งตารางที่ใช้ในการดูค่าจำนวนบิตที่ใช้ต่างๆ กัน เรียกว่าเป็นรหัสแบบความยาวของรหัสไม่คงที่หรือ (Variable Length Code) เนื่องจากข้อมูลที่มาเข้ารหัสจะมีค่า Run เข้ามาเกี่ยวข้องดังนั้นในตารางค่า AC ของฮัฟแมน จะต่างจากในตารางค่า DC ของฮัฟแมน คือจะมีการเปลี่ยนแปลงจาก Category ใน DC เป็น Run/Category นั่นคือการหารรหัสข้อมูลจากตารางค่า AC ของฮัฟแมน (ในภาคผนวก) จะต้องทราบค่า Run และ Category ของข้อมูลที่นำมาเข้ารหัส ซึ่งการหาค่า Category ของข้อมูล AC จะเทียบหาจากตาราง JPEG Coefficient Coding Categories ตามตารางที่ 2.2

เช่นเดียวกับใน DC เมื่อทราบค่า Run และ Category ของข้อมูลแล้วก็จะสามารถหารหัสข้อมูลจากตารางค่า AC ของฮัฟแมนได้ โดยเทียบตามค่า Run/Category

ภายในตารางค่า AC ของฮัฟแมนจะมีรหัสพิเศษ 2 ตัว คือ รหัสเมื่อค่า Run/Category เท่ากับ 0/0 ใช้ในกรณีเมื่อทำการเข้ารหัสข้อมูลภายในแถวจนกระทั่งเหลือแต่ข้อมูลที่เป็นศูนย์เพียงอย่างเดียว ก็จะใช้รหัสนี้เป็นตัวบอกตัวถอดรหัสว่าข้อมูลที่เหลือทั้งหมดภายในแถวมีค่าเป็นศูนย์ และรหัสอีกตัวหนึ่งเมื่อ Run/Category เท่ากับ F/0 ใช้เมื่อมีข้อมูลที่มีค่าศูนย์อยู่ติดกัน 16 ตัวภายในแถว โดยยังมีข้อมูลที่ไม่เป็นศูนย์ที่ยังไม่ถูกเข้ารหัสเหลืออยู่ในแถวอีก ก็จะใช้รหัสตัวนี้แทน ข้อมูลที่มีค่าศูนย์ 16 ตัวดังกล่าว

จะเห็นได้ว่ารหัสข้อมูลที่ได้จากตารางค่า AC ของฮัฟแมนนั้นใช้ในการบอกจำนวนศูนย์ (ค่า Run) ที่อยู่ด้านหน้าของข้อมูลนั้น และช่วงของค่าข้อมูล (ค่า Category) ที่นำมาเข้ารหัส ดังนั้นจะต้องมีรหัสตัวที่สองสำหรับใช้ในการระบุค่าของข้อมูล ซึ่งการหารหัสตัวที่สองนี้ใช้วิธีเดียวกันกับใน DC คือเอามาจาก LSB จำนวน Category บิตของข้อมูลที่นำมาเข้ารหัสและจะเอามาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

2.2 ภาษาวีเอชดีแอล (VHDL)

วีเอชดีแอล (VHDL) VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Language) เป็นภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ทั้งในระดับเกทจนถึงระดับดิจิตอลที่ซับซ้อน ทั้งนี้ เนื่องจาก VHDL เป็นภาษาที่เหมาะในการนำมาใช้ในการเลียนแบบการทำงานของอุปกรณ์ อีกทั้งยังมีความยืดหยุ่นและไม่ถูกจำกัดโดยความสามารถทางเทคโนโลยีใดๆ ทำให้ประหยัดเวลาและค่าใช้จ่ายในการออกแบบ จึงได้มีการนำมาใช้กันอย่างกว้างขวางในวงการอุตสาหกรรม รูปแบบของภาษา VHDL ประกอบด้วย 2 ส่วนใหญ่ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล (Sequential Language) และภาษาคอนเคอร์เรนท์ (Concurrent Language) การเขียนโปรแกรมด้วยภาษา VHDL สามารถเขียนได้ทั้งสองรูปแบบรวมกัน นอกจากนี้ ตัวภาษายังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยเข้าด้วยกัน เพื่อให้ความเป็นระบบใหญ่ได้และสามารถกำหนดรูปแบบไวยากรณ์ (Syntex) อีกทั้งยังมีการตรวจสอบความหมายของภาษาว่าจะ Simulate ได้หรือไม่ เพราะโปรแกรมที่เขียนโดย VHDL ต้องผ่านการ Simulate เพื่อตรวจสอบการทำงาน ดังนั้น ในการ Compile จะมีการตรวจสอบทั้งวงจรและ Simulation Semantic อย่างไรก็ตาม แม้ตัวภาษาจะมีความซับซ้อนในรูปแบบและกฎเกณฑ์ของภาษา แต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำไปใช้งานได้โดยไม่ต้องศึกษารายละเอียดทั้งหมด

2.2.1 การใช้งานภาษา VHDL

VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง ความสามารถในการเลียนแบบ (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนี้ VHDL ยังถูกกำหนดไว้สำหรับการ

บรรยายฮาร์ดแวร์คือ ระบบจนถึงระดับเกทอีกด้วย เนื่องจากในการทำงานของระบบดิจิตอลจริงๆ ทุกๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อมๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ถือว่าเป็นข้อกำหนดที่สำคัญอย่างหนึ่งใน VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้ว ความพร้อมเพรียงจะหมายถึงทุกๆ คำสั่งองค์ประกอบเกท หรือวงจรต่างๆ จะถูกนำมาปฏิบัติทั้งหมด ดังนั้น ในตอนท้ายแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไปพร้อมๆ กัน) ด้วยความสามารถของ VHDL ในด้านของการกำหนดพฤติกรรมการทำงานของวงจร ทำให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งวงจรของดิจิตอลทั่วๆ ไป และในระบบที่แตกต่างกันออกไป เช่น พฤติกรรมการทำงานของระบบเรดาร์ หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในสมองมนุษย์ได้ ข้อดีหลักที่สำคัญของ VHDL ก็คือ ภาษานี้จะสามารถถูกใช้ได้ตลอดในทุกๆ ระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือ ในกระบวนการออกแบบตั้งแต่ระดับสูง (System Level) จนถึงระดับที่ต่ำกว่า (Lower hardware level) สามารถใช้ภาษาเดียวกันได้โดยตลอด ทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

2.2.1.1 ข้อกำหนด (VHDL Requirement)

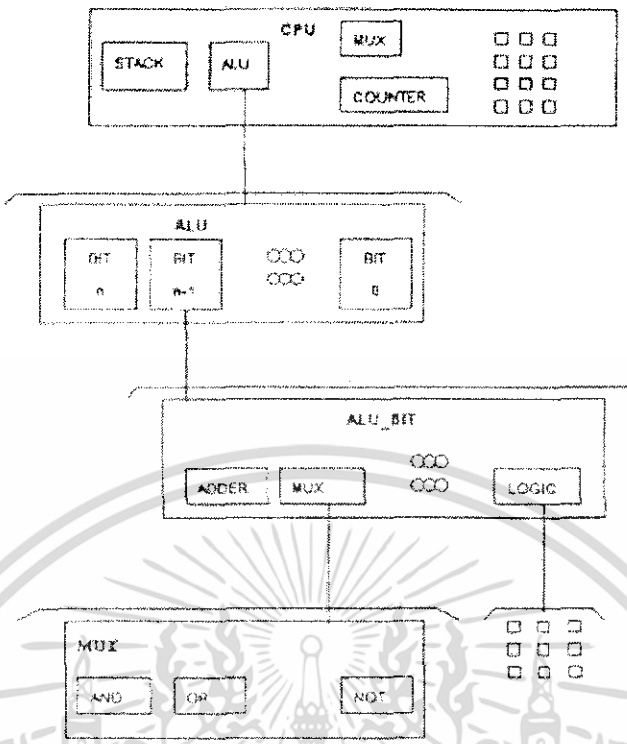
ในเอกสารของ DoD (Department of Defense Requirement for Hardware Description Language) ซึ่งออกมาในเดือนมกราคมปี 1983 ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ไว้ดังนี้

1). ลักษณะทั่วไป (Generation Features)

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

2). สนับสนุนการออกแบบแบบลำดับขั้น (Support for Design Hierarchy)

การออกแบบลำดับขั้นเป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบที่มีหลายๆ ระดับ ในการออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน ซึ่งหน้าที่การทำงานของระบบก็สามารถกำหนดได้ด้วยตนเองหรือถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อยๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้ ดังรูปที่ 2.9



รูปที่ 2.9 แสดงตัวอย่างการออกแบบแบบลำดับขั้น

3). ไลบรารี่ (Library Support)

VHDL ได้สนับสนุนการมีไลบรารี่เพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารี่ที่ระบบได้จัดเตรียมไว้แล้วก็ได้โมเดล และการบรรยายที่ถูกต้องควรจะเก็บไว้ในไลบรารี่หลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้ว เพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไปใช้ได้ด้วย

4). ลำดับคำสั่ง (Sequential Statement)

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงาน ซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case if-then-else และ loop ทั่วๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานพร้อมเพรียงกันเช่นเดิม

5). การกำหนดคุณสมบัติ (Generic Design)

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่นๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน สิ่งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้นๆ ภาษาสำหรับการออกแบบที่ดีควรจะสามารถให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาด ลักษณะทางกายภาพ เวลาโหลด และเงื่อนไขทางสภาพแวดล้อมอื่นๆ ความสามารถในการกำหนดคุณสมบัติก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

6). ชนิดของข้อมูล (Type Declaration and usage)

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

7). โปรแกรมย่อย (Use of Subprogram)

ความสามารถในการใช้ฟังก์ชั่นและโพรซีเดอร์ (Procedure) เป็นข้อกำหนดอีกอย่างหนึ่งใน VHDL เราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก (Logic) การกำหนดตัวกระทำต่างๆ ทั้งเก่าและใหม่หรืออะไรก็ตามได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

8). การควบคุมเวลา (Timing Control)

VHDL อนุญาตให้ผู้ออกแบบ สามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการการตรวจสอบ การออกแบบเกท หรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ (Event) นอกจากนี้ ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

9). การกำหนดแบบโครงสร้าง (Structural specification)

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุกๆ ระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวมที่เกิดจากองค์ประกอบย่อยที่ต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่งเช่นกัน

2.2.2 ความสามารถของภาษาวีเอชดีแอล (Capability)

- ตัวภาษา VHDL สามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิพกับผู้ออกแบบ (CAD Tools)

- ใช้เป็นสื่อกลางในการแลกเปลี่ยนสื่อสารระหว่างซีเออี (CAE) และซีเอดีทูล (CAD Tools) เช่น ตัวภาษาซ้อสโค้ด (SOURCE CODE) ของ VHDL สามารถคอมไพล์โดยใช้คอมไพล์เลอร์ (Compiler) และซิมมูเลเตอร์ (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 IEEE และ ANSI ทำให้โมเดลที่ออกแบบโดยภาษา VHDL สามารถเคลื่อนย้ายไปยังระบบใดๆ ก็ได้ และสามารถนำกลับมาใช้ใหม่ได้

- สามารถเขียนโมเดลได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของโมเดล (ขึ้นอยู่กับซอฟต์แวร์)

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

- สนับสนุนการออกแบบขนาดใหญ่ โดยใช้ความสามารถของส่วนประกอบ (Component) ฟังก์ชั่นโพรซีเดอร์ (Function Procedure) และแพ็คเกจ (Package)

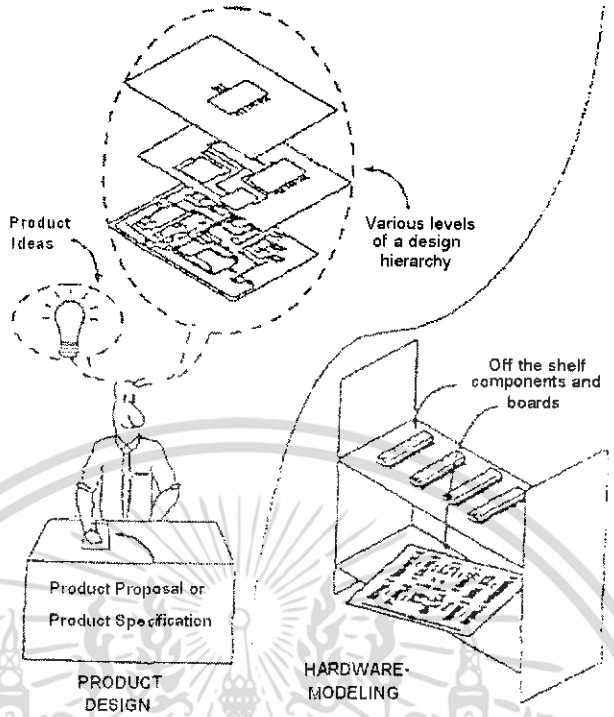
- สามารถอธิบายตัวแปรที่เกี่ยวกับฟังก์ชั่นทางด้านเวลา เช่น Propagation delay, Min – Max Delay, Setup, Holding Time สามารถอธิบายได้โดยตัวภาษา

- ภาษา VHDL เป็นมาตรฐานที่ใช้ได้บริษัทและผู้ออกแบบหลายๆ แห่ง ฉะนั้น จึงง่ายที่จะทำความเข้าใจถึงแม้ว่าจะมาจากแหล่งต่างๆ

- โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะตัวแปรภาษาได้ตรวจสอบตัวแปรทางซิมมูเลชั่นซีแมนติกไว้ด้วย

2.2.3 หลักการสร้างโมเดลโดยใช้ภาษาวีเอชดีแอล (General VHDL Modelling Principles)

วีเอชดีแอลเป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้ ซึ่งจะช่วยในการสร้างและออกแบบวงจรรวมดิจิตอลและส่วนประกอบต่างๆ อาจใช้อธิบายระบบทั้งระบบหรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของ (Component Block) จากนั้นก็ทำการจำลองการทำงาน (Simulate) โดยที่รูปแบบนั้นยังไม่ได้สร้างขึ้นจริงหรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังจากจำลองการทำงานจนได้ตามที่ต้องการจึงนำไปทำการ Synthesis เพื่อให้ได้วงจรเกทระดับต่อไป ประโยชน์จริงของการใช้วงจรวีเอชดีแอลเป็น Design Tools แทนการสร้างต้นแบบ (Prototype) ขึ้นมาจริงคือ เราสามารถอธิบาย Product Idea, Product Proposal, Product Specification



รูปที่ 2.10 สิ่งต่างๆ ที่สามารถอธิบายได้ด้วย VHDL

ในรูปของ Text จากนั้นก็นำมาคอมไพล์เพื่อดู Timing การทำงานแล้วแก้ไข (Refine) จนกว่าจะได้ Specification ตามต้องการ เมื่อ product ได้ผลตามที่ต้องการแล้วจึงนำไปสู่การสังเคราะห์ (Synthesis) เพื่อให้ได้เกทระดับ Schematic เพื่อนำไปสร้างเป็นต้นแบบจริงต่อไป ซึ่งต้นแบบที่สร้างนั้นทำงานได้จริง เพราะได้ทำการ Simulate เรียบร้อยแล้ว ทำให้การลดเวลาและค่าใช้จ่ายในการสร้างต้นแบบได้มาก

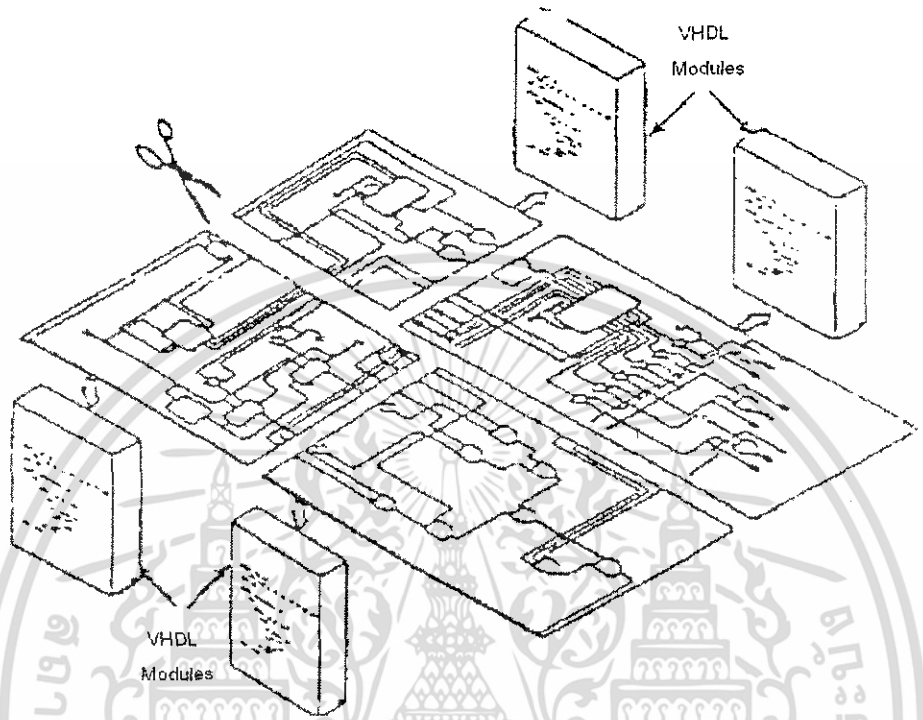
นอกจากนี้ ตัวภาษา VHDL สนับสนุนหลักการต่างๆ ให้เขียน แก้ไข และบำรุงรักษาวงจรดิจิตอลที่มีความซับซ้อนให้เป็นไปอย่างรวดเร็วและมีประสิทธิภาพ โดยมีหลักการดังนี้

2.2.3.1 Top down Design

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน เช่น ASIC (Application Specific Integrated Circuit) วิศวกรหรือผู้ออกแบบมักจะมองรูปแบบให้อยู่ในรูปของ Block Diagram ก่อนที่จะย่อยรูปแบบให้ถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายการทำงานของแต่ละ Block วิเคราะห์การทำงาน จัดการ แก้ไข และปรับปรุงการทำงานจากการวิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการก่อนที่จะทำการออกแบบให้ละเอียดลึกลงไปในขั้นตอนต่อไป การแก้ไขในขั้นตอนนี้จะทำให้ลดค่าใช้จ่ายกว่าการแก้ไขในช่วงของการพัฒนาในระดับสร้างซิลิกอนชิพ

2.2.3.2 Modularity

Modularity คือ หลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ ออกเป็นส่วนย่อยเล็กลงไป ซึ่งปกติการทำงานฮาร์ดแวร์ใหญ่ๆ ต้องประกอบด้วยฮาร์ดแวร์ย่อยๆ ลงไปดังรูปที่ 2.11

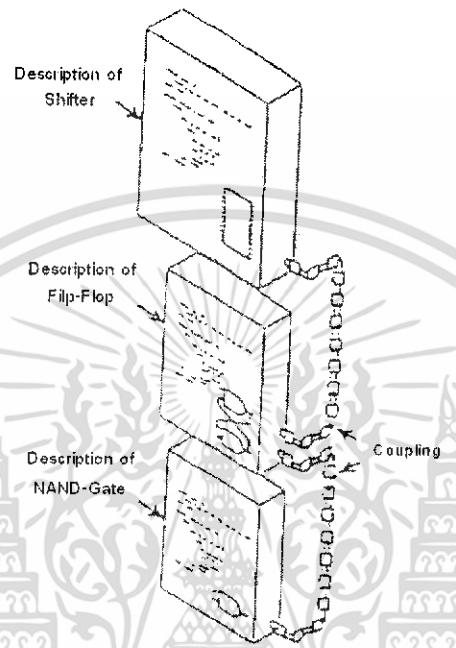


รูปที่ 2.11 การแบ่งย่อยในระดับราบของการออกแบบฮาร์ดแวร์

จากรูปได้แสดงวงจรทั้งหมดในรูปเดียว (Flatten Design) หลังจากนั้น ตัดเป็นส่วนย่อยๆ เล็กลงมา เมื่อเราออกแบบโดยใช้ภาษา VHDL หน้าที่การทำงานของแต่ละส่วนต้องสามารถอธิบายได้โดยโมดูลของโค้ด (คล้ายฟังก์ชั่นหรือโพรซีเดอร์) ซึ่งแสดงการทำงานของส่วนย่อยนั้นอย่างชัดเจน นอกจากนี้ การแยกรูปใหญ่ๆ ออกเป็นส่วนย่อยๆ ทำให้ง่ายต่อการจัดการและการทำความเข้าใจ

รูปที่ 2.12 แสดง Hierarchy Method โดยการแยกส่วนรูปแบบออกเป็นส่วนย่อยๆ ส่วนบนสุดอธิบายการทำงานของ Shifter ส่วนล่างๆ ลงมาคือ การแยกส่วนของ Shifter ออกเป็นฟลิปฟลอป จากฟลิปฟลอปแยกเป็น NAND เกท ภายใน Shifter ได้อธิบายการทำงานโดยใช้การต่อกันของฟลิปฟลอป ในระดับที่ต่ำลงมา ฟลิปฟลอปก็เกิดจาก NAND เกทต่อกัน 2 ตัวในระดับที่ต่ำลงมาอีกก็เป็น NAND เกท ซึ่งมีการอธิบายการทำงานอยู่ภายใน โดยแต่ละโมดูลจะมีคำอธิบายการทำงานในตัวของมันเองอยู่แล้วคำอธิบายในแต่ละโมดูลมีไว้เพื่อให้สามารถใช้ฟลิปฟลอปโมดูลได้ ส่วนฟลิบฟลอบโมดูลก็อธิบายการเชื่อมต่อไว้อย่างดีทำให้สามารถเชื่อมต่อกับ NAND เกท ในระดับล่างสุด ได้ประโยชน์อย่างหนึ่งของการแยกส่วนฟลิปฟลอปและ NAND เกทออกจากกัน เนื่องจากทำให้ง่ายใน

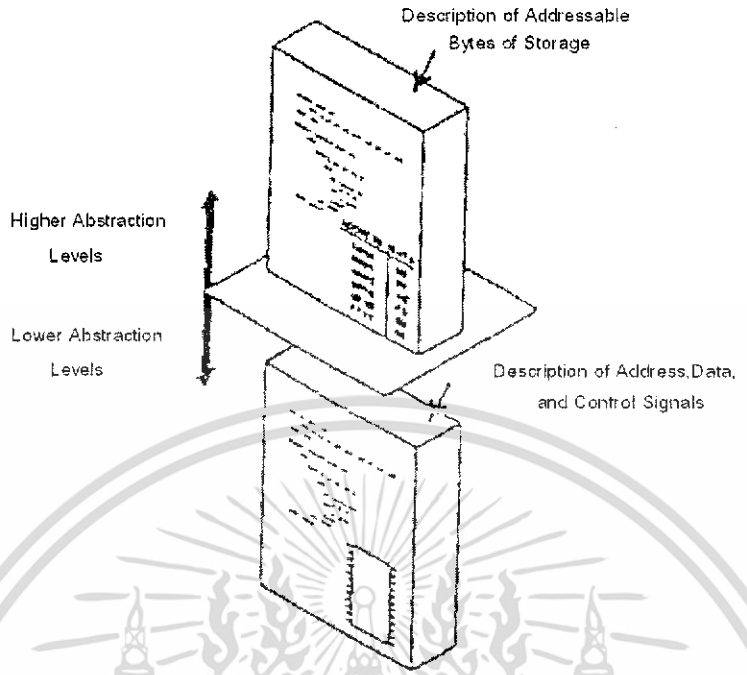
การใช้ NAND เกท ตัวนี้ในรูปแบบระดับสูงตัวอื่น ๆ ทำให้นำออกไปใช้งานได้อีกและลดความซับซ้อนในการใช้อุปกรณ์เพื่อแก้ไขการทำงานของ Shifter ง่ายขึ้นโดยปราศจากการแก้ไขฟลิปฟลอปและ NAND เกท ประโยชน์ที่ได้จากการทำ Modularity นี้ทำให้รูปแบบที่ออกแบบง่ายต่อการเข้าใจและแก้ไขได้เสมอ



รูปที่ 2.12 การแบ่งแบบ Hierarchy ของ VHDL Shifter Description

2.2.3.3 Abstraction

คำนิยามของรูปแบบ จะอธิบายการทำงานของตัวรูปแบบมากกว่าจะอธิบายว่าพัฒนาตัวรูปแบบนั้นได้อย่างไร หลักการนี้มีความสำคัญอย่างใกล้ชิดกับหลักการของ Modularity ในรูปที่ 2.12 ฟลิปฟลอป เป็นการนิยามในการใช้ NAND เกท และ Shifter เป็นนิยามในการใช้ฟลิปฟลอป รูปที่ 2.13 แสดงถึงการอธิบายการทำงานของรูปแบบโดยใช้ VHDL ในหลายๆ ระดับของการนิยาม ROM (Read Only Memory) อธิบายโดยใช้ภาษาระดับสูง แสดงถึงตำแหน่งต่างๆ ซึ่งเก็บข้อมูลไว้ ในตำแหน่งนั้นๆ ที่ระดับนี้ไม่ต้องสนใจถึง Address Line, Data Line และ Control Line เราสามารถเพ่งจุดสนใจไปที่ขนาดของข้อมูลโดยไม่ต้องคำนึงถึงสัญญาณควบคุมต่างๆ ภายในเพราะว่าส่วนนั้นจะถูกจัดการเองในระดับที่ต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ในการจัดการสัญญาณภายในทุกเส้นภายใน การที่จะอ่านข้อมูลหรือโปรแกรมข้อมูลใน ROM ถ้าต้องการเปลี่ยนค่าข้อมูลภายใน ROM ควรแก้ไขในระดับที่สูงขึ้นมาจะง่ายกว่าการควบคุมสัญญาณภายใน จะเห็นว่าแต่ละระดับมีความเหมาะสมแตกต่างกันออกไปทำให้รูปแบบที่เราออกแบบไปง่ายต่อการแก้ไขโดยการใช้ประโยชน์ของ Abstraction

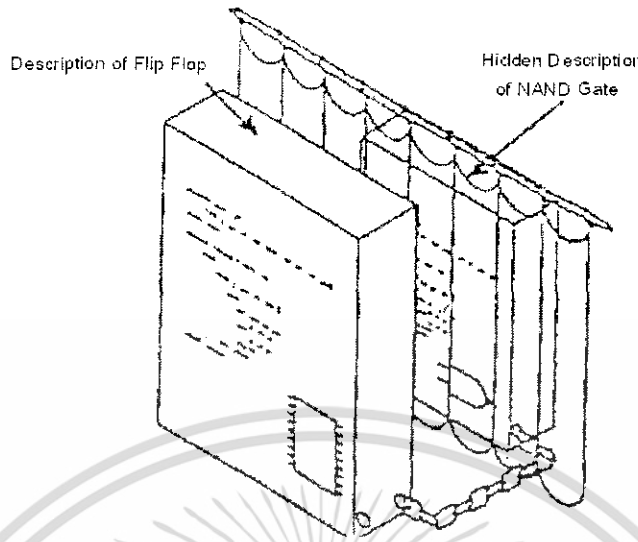


รูปที่ 2.13 Applying Abstraction to a ROM Description

2.2.3.4 Information Hiding

เมื่อทำการเขียน VHDL Code ขึ้นมาเพื่ออธิบายการทำงานของฮาร์ดแวร์ตัวหนึ่ง บางครั้งอาจต้องการที่จะซ่อนรายละเอียดการพัฒนาโมดูลนั้นโดยไม่ต้องการให้ส่วนโมดูลอื่นๆ รู้การทำงานภายใน Information Hiding มีประโยชน์คือ ทำให้รูปแบบภาษา VHDL สามารถจัดการและอ่านเข้าใจได้ง่าย หลักการนี้จะสนับสนุนหลักการ Abstraction คือ สนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่ารูปแบบนั้นจะถูกสร้างขึ้นมาอย่างไร เป็นต้น การซ่อนรายละเอียดภายในโมดูล ทำให้ความสนใจของผู้ออกแบบนั้นสนใจในส่วนที่สำคัญมากกว่าในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ ดังรูปที่ 2.14

การอธิบายการทำงานของฟลิปฟลอปไม่ต้องสนใจว่า NAND เกท จะทำงานอย่างไร หรือจะต่อกันภายในอย่างไร เพราะ NAND เกท สามารถเขียนขึ้นมาแล้วคอมไพล์เก็บไว้ในไลบรารี ผู้ที่ออกแบบฟลิปฟลอประดับสูงขึ้นมา เพียงแต่ต้องรู้ว่าจะเชื่อมต่ออินพุท/เอาท์พุทของ NAND เกท มาใช้งานได้อย่างไร โดยไม่ต้องสนใจว่า NAND เกท จะถูกสร้างและพัฒนามาอย่างไร ประโยชน์อีกอย่างหนึ่งคือ ป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย VHDL โมดูลไปยังที่อื่นทำให้เราป้องกันทรัพย์สินทางปัญญาได้อีกในระดับหนึ่ง



รูปที่ 2.14 การซ่อนรายละเอียดที่ไม่จำเป็นของระดับ NAND เกท

2.2.3.5 Uniforminity

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

2.2.4 องค์ประกอบพื้นฐานในวีเอชดีแอล (Basic concept in VHDL)

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบใน VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 2.15 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำ ENTITY ตามด้วยชื่อขององค์ประกอบและคำ IS ภายในบรรยายถึงพอร์ตการติดต่อ อินพุท/เอาท์พุท พอร์ตขององค์ประกอบ ส่วนลักษณะกายภาพภายนอกอื่นๆ เช่น เวลา อุณหภูมิ ก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ ซึ่งจะขึ้นอยู่กับสัญญาณ อินพุท/เอาท์พุท และพารามิเตอร์อื่นๆ ที่กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 2.15 การบรรยายหน้าที่ขององค์ประกอบจะเริ่มต้นหลังคำว่า BEGIN เป็นต้นไป

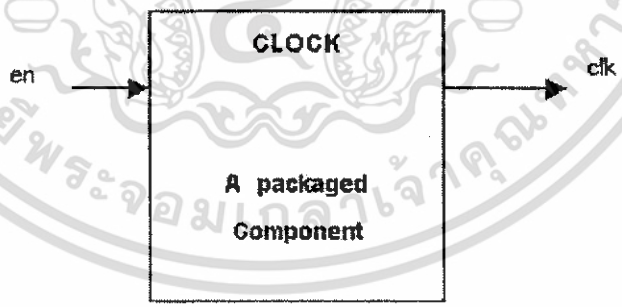
```
ENTITY component_name IS
        Input and output ports.
        Physical and other parameters.
END component_name;
```

```
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 2.15 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม

2.2.4.1 การกำหนดการเชื่อมต่อ (Interface Description)

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ต สำหรับการติดต่อกับองค์ประกอบภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 2.16 บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็นชื่อ Clock component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ในวงเล็บ IN และ OUT กำหนดโหมดของสัญญาณเป็นอินพุทหรือเอาท์พุท BIT แสดงชนิดของข้อมูล




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

รูปที่ 2.16 แสดงบล็อกไดอะแกรมและการบรรยายการเชื่อมต่อของ Clock component

2.2.4.2 การกำหนดรูปแบบการบรรยาย (Architecture Description)

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ การบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในเทอมของ Clock component ในรูปที่ 2.17 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมมี en เป็นอินพุทและมี clk เป็นเอาท์พุท PROCESS เป็นค่าเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม ภายในโพรเซสกำหนดให้ Periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น "0" ถ้าสัญญาณ en มีค่าเป็น "1" ค่าของ periodic จะถูกคอมพลีเมนต์และส่งค่าให้กับ clk ซึ่งเป็นสัญญาณเอาท์พุทคำสั่ง WAIT กำหนดให้สัญญาณมีคาบเป็นเวลา 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 2.17 แสดงการบรรยายเชิงพฤติกรรมของ Clock component

2.2.4.3 โปรแกรมย่อย (Subprogram)

การใช้ฟังก์ชั่นและโพรซีเดอร์ใน VHDL เปรียบเทียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆ ไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่น ถ้าเราให้ฟังก์ชั่นแทนการกระทำในสมการบูลีน ก็จะมีผลต่อวงจรตรรกะจริงๆ ในขณะที่เราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าหน่วงเวลา แล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

รูปที่ 2.18 แสดงการใช้ฟังก์ชั่นโพรซีเดอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิท เป็นค่าจำนวนเต็มรูปที่ 2.19 แสดงการใช้ฟังก์ชั่นโดยกำหนดให้ X เป็นตัวแปรชนิดบิทแทนการกระทำในสมการบูลีน

```
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;
```

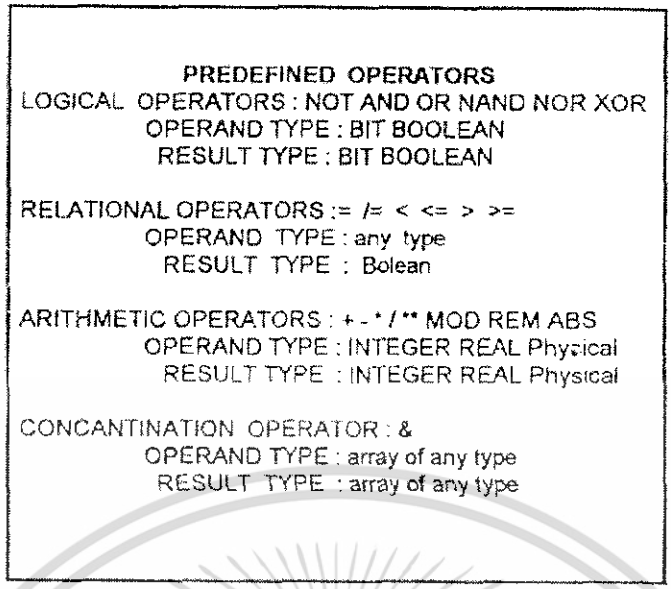
รูปที่ 2.18 แสดงการใช้โพรซีเดอร์

```
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 2.19 แสดงการใช้ฟังก์ชั่น

2.2.4.4 โอเปอร์เรเตอร์ (VHDL OPERATORS)

การบรรยายเชิงพฤติกรรมใน VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกันกับภาษาซอฟแวร์ทั่วไป ดังรูปที่ 2.20



รูปที่ 2.20 แสดงตัวกระทำใน VHDL

2.2.4.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)

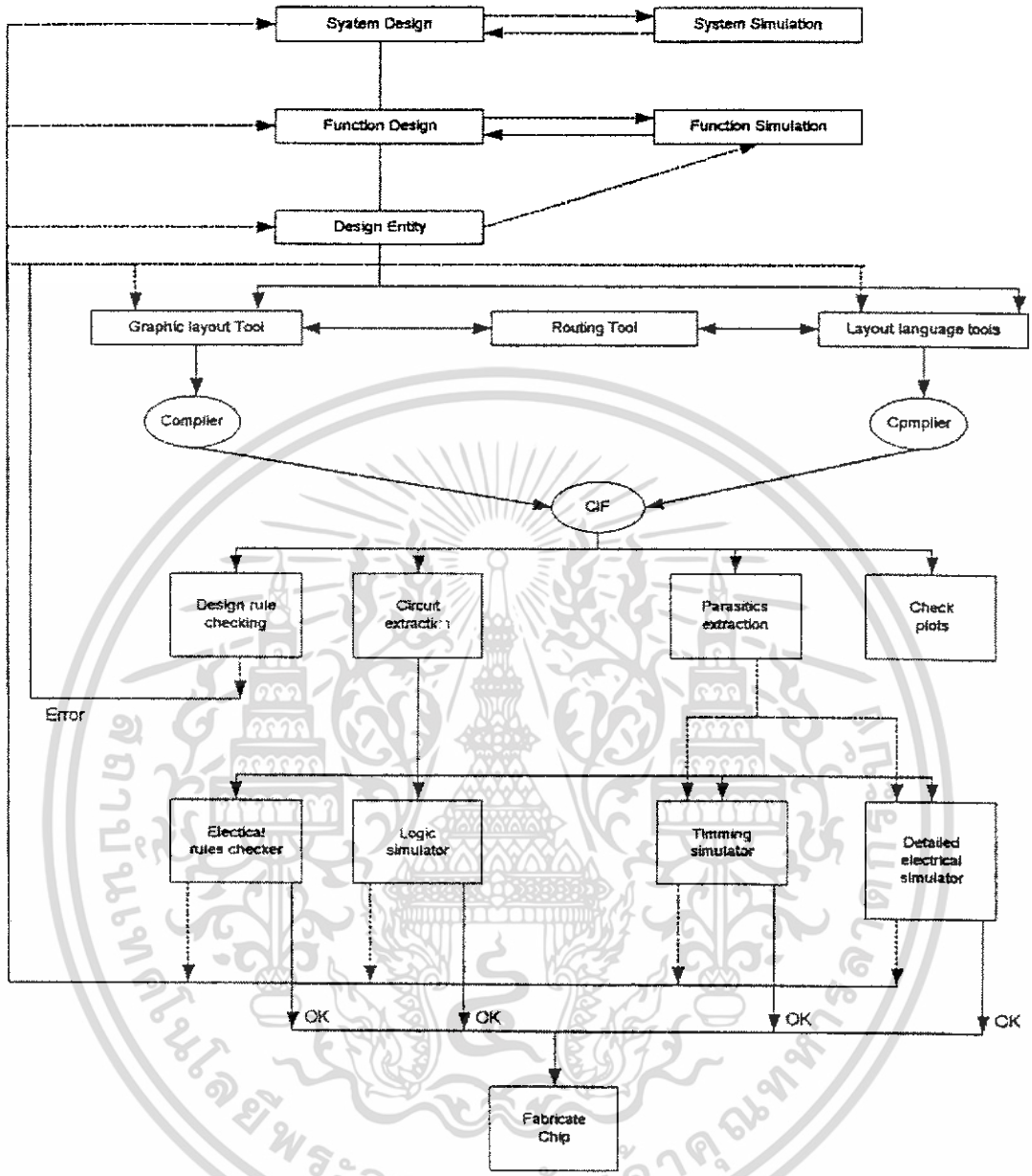
ในวงจรอิเล็กทรอนิกส์ อุปกรณ์ทุกตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (always active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วยเสมอในทุกๆ เหตุการณ์ที่เกิดขึ้น VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของสถาปัตยกรรมการบรรยายจะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โพรเซสที่มีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายๆ โพรเซสอยู่ภายในโครงการสร้างเดียวกันทุกๆ โพรเซสก็จะทำงานไปพร้อมๆ กันด้วย

2.2.4.6 สัญญาณและตัวแปร (Signals and Variable)

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

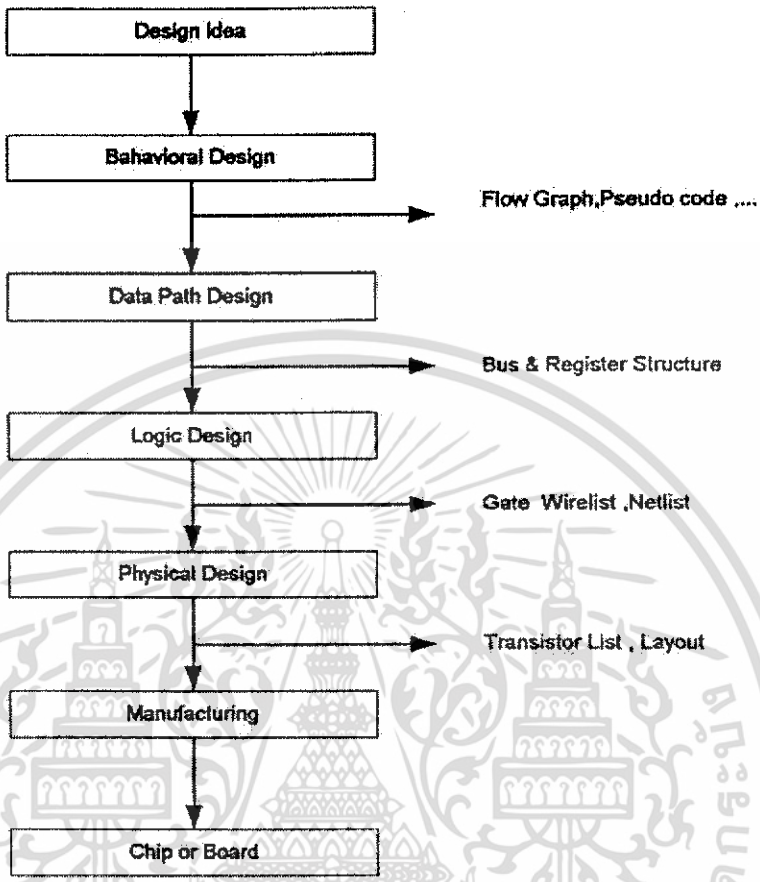
2.2.5 โครงสร้างของวีเอชดีแอล

ในการออกแบบดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้ จะต้องผ่านขั้นตอนต่างๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้น ทำการเพิ่มเติมตามความจำเป็นและเข้ากระบวนการออกแบบในขั้นต่อไป โครงสร้างการออกแบบ VHDL สามารถเขียนเป็นแผนผังได้ ดังรูปที่ 2.21



รูปที่ 2.21 รูปแสดงโครงสร้างการออกแบบ VHDL

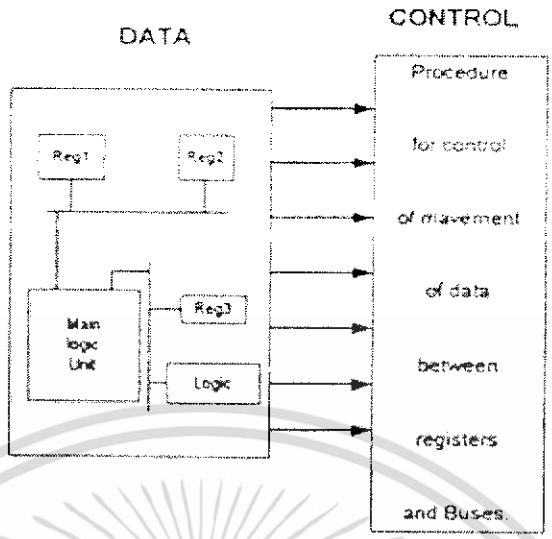
รูปที่ 2.22 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วๆ ไป ซึ่งขั้นแรกผู้ออกแบบกำหนดแนวความคิดในการออกแบบเสียก่อนและทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นในขั้นตอนนี้จึงมีความจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจเป็นผังงาน ผังแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo Code)



รูปที่ 2.22 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะต้องกำหนดส่วนประกอบของรีจีสเตอร์ ผู้ออกแบบจะกำหนดส่วนของรีจีสเตอร์ (Register) และวงจรตรรกะ (Logic) ที่จำเป็นทั้งหมดที่ประกอบกันเป็นระบบที่สมบูรณ์ แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) กระบวนการควบคุมในการเคลื่อนย้ายข้อมูลระหว่างรีจีสเตอร์และวงจรตรรกะจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ ดังรูปที่ 2.23

การออกแบบวงจรตรรกะจะเป็นขั้นตอนต่อไป การออกแบบในขั้นนี้เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิปฟลอปเป็นส่วนของอุปกรณ์แยกต่างๆ ได้แก่ รีจีสเตอร์เก็บข้อมูลวงจรตรรกะและส่วนควบคุมฮาร์ดแวร์ ซึ่งในขั้นสุดท้ายจะได้ออกมาเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง



รูปที่ 2.23 แสดงการออกแบบระบบเส้นทางของข้อมูล

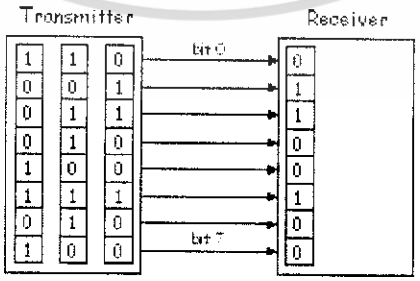
ต่อมาเป็นขั้นตอนของการเปลี่ยนเครือข่ายการโยงใยในขั้นตอนที่แล้วให้เป็นทรานซิสเตอร์และเลย์เอาท์ (Transistor list and layout) ในขั้นตอนนี้เกี่ยวข้องกับการจัดวางเกทและฟลิปฟลอปแทนด้วยทรานซิสเตอร์หรือไลบรารีเซล ขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

2.3 การสื่อสารข้อมูล

ในการสื่อสารหรือการส่งข้อมูลในระบบดิจิตอลนั้นมีรูปแบบในการสื่อสารที่สำคัญอยู่ 2 แบบ

2.3.1 การสื่อสารแบบขนาน (Parallel Communication)

การสื่อสารในรูปแบบที่ส่งข้อมูลออกไปทีเดียวพร้อมๆกันทั้ง 8 bits ผ่านสายสัญญาณทั้ง 8 เส้น การสื่อสารและการส่งข้อมูลแบบขนานจะแสดงให้เห็นดังรูปที่ 2.24

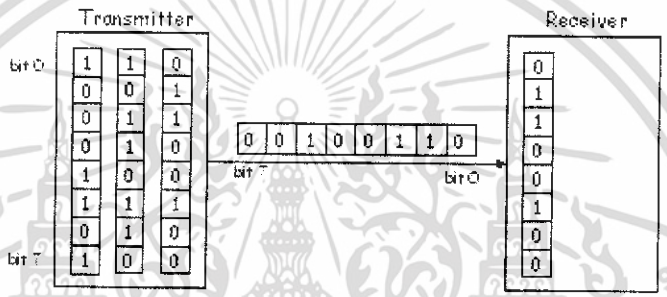


รูปที่ 2.24 ลักษณะการสื่อสารแบบขนาน

จะเห็นได้ว่าการสื่อสารแบบขนานมีข้อดีคือ สามารถส่งข้อมูลทีละมากๆ และรวดเร็วกว่าการส่งแบบอนุกรม แต่การสื่อสารแบบขนานมีข้อจำกัดคือ ไม่สามารถส่งข้อมูลในระยะไกลๆได้และยังต้องใช้สายสัญญาณหลายเส้นในการส่งข้อมูล ทำให้สิ้นเปลืองกว่าการสื่อสารแบบอนุกรม ทำให้ไม่สะดวกในการใช้งาน ตัวอย่างของการสื่อสารแบบขนาน เช่น เครื่องพิมพ์ (Printer) และการสื่อสารทางพอร์ตขนาน (ECP Printer Port) เป็นต้น

2.3.2 การสื่อสารแบบอนุกรม (Serial Communication)

เป็นการสื่อสารโดยการส่งข้อมูลทีละบิท ผ่านสายสัญญาณเส้นเดียว จนครบทั้ง 8 bits หรือไบท์ โดยจะส่งบิทต่ำ (LSB) ออกไปก่อน สามารถแสดงให้เห็นหลักการส่งข้อมูล แบบอนุกรมได้ ดังรูปที่ 2.25



รูปที่ 2.25 การส่งข้อมูลแบบอนุกรม

2.3.3 การอินเตอร์เฟสตามมาตรฐาน RS-232

มาตรฐาน RS-232 เป็นมาตรฐานที่ได้รับการพัฒนามานานและถูกใช้งานกันอย่างแพร่หลาย เราใช้ RS-232 เชื่อมต่อ DTE (Data Terminal Equipment) เข้ากับ DCE (Data Communication Equipment) เช่น การต่อเทอร์มินัลเข้ากับโมเด็ม มาตรฐาน RS-232 กล่าวถึงลักษณะทางกล, ลักษณะของสัญญาณไฟฟ้า และลักษณะการทำงานที่ใช้การอินเตอร์เฟส ตัวอย่างของอุปกรณ์ที่ใช้ในการอินเตอร์เฟสตามมาตรฐาน RS-232 ได้แก่ เทอร์มินัล, พล็อตเตอร์, ลอจิก อนาไลเซอร์ (Logic Analyzer) และเครื่องพิมพ์ ถ้าการประยุกต์ใช้งานของเราต้องการทำอินเตอร์เฟสอุปกรณ์เข้ากับอินเตอร์เฟสมาตรฐาน RS-232 เราจำเป็นต้องแปลงระดับ

2.3.4 ลักษณะสัญญาณที่ใช้ในการอินเตอร์เฟส

มาตรฐาน RS-232 ใช้สัญญาณเพียงเส้นเดียวในการส่งสัญญาณ โดยสัญญาณจะส่งไปในทิศทางเดียวกัน ในกรณีที่อัตราเร็วในการส่งข้อมูลมีค่าเท่ากับ 20 kbps (กิโลบิทต่อวินาที) ซึ่งค่านี้เป็นค่าสูงสุดที่ใช้ในการสื่อสารข้อมูล (ในปัจจุบันพัฒนาให้สามารถส่งข้อมูลได้มากกว่านี้) ระยะทางในการส่งข้อมูลไม่ควรเกิน 50 ฟุต (ตามข้อกำหนดในมาตรฐาน) สำหรับการแทนแรงดันของระดับสัญญาณจะ

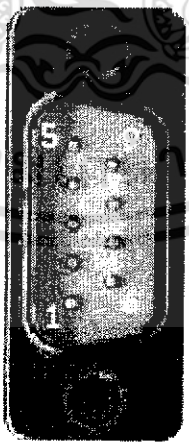
แทนระดับสัญญาณของลอจิก "0" ด้วยค่าแรงดัน +3 โวลท์ ถึง +12 โวลท์ ส่วนลอจิก "1" จะแทนระดับสัญญาณด้วยค่าแรงดันระหว่าง –3 โวลท์ ถึง –12 โวลท์

2.3.5 การออกแบบตัวแปลงสัญญาณ

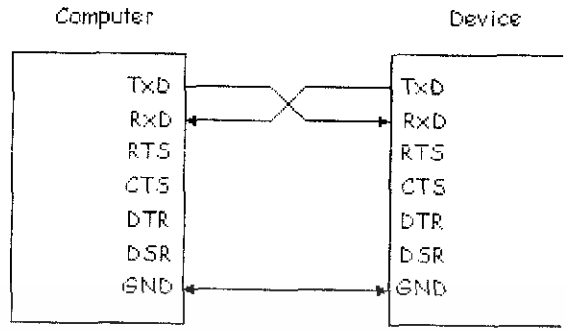
การเชื่อมต่อกับพอร์ตสื่อสารของคอมพิวเตอร์ส่วนบุคคลจะเลือกใช้พอร์ตสื่อสารแบบอนุกรม 9 ขา (DB-9) ซึ่งสามารถทำการรับส่งข้อมูลได้แบบอนุกรม โดยลักษณะของสัญญาณจะเป็นไปตามมาตรฐาน RS-232 และลักษณะของการเชื่อมต่อของพอร์ตสื่อสารสำหรับคอนเน็คเตอร์แบบ DB-9 สามารถแสดงให้เห็นได้ ดังรูปที่ 2.26 และรูปที่ 2.27

ตารางที่ 2.4 แสดงขาสัญญาณของพอร์ตสื่อสารอนุกรมแบบ DB-9

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



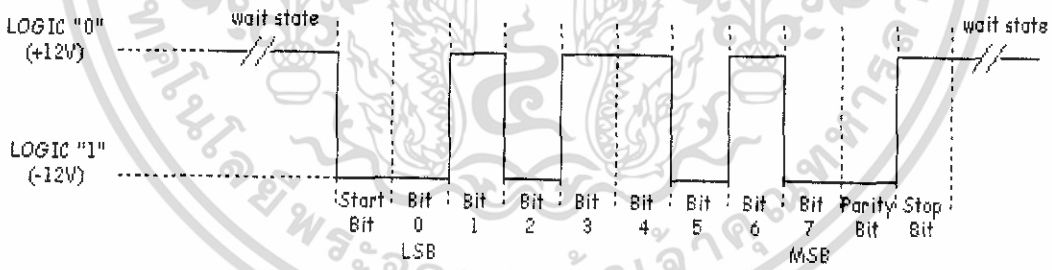
รูปที่ 2.26 การจัดขาของคอนเน็คเตอร์พอร์ตอนุกรมแบบ DB-9



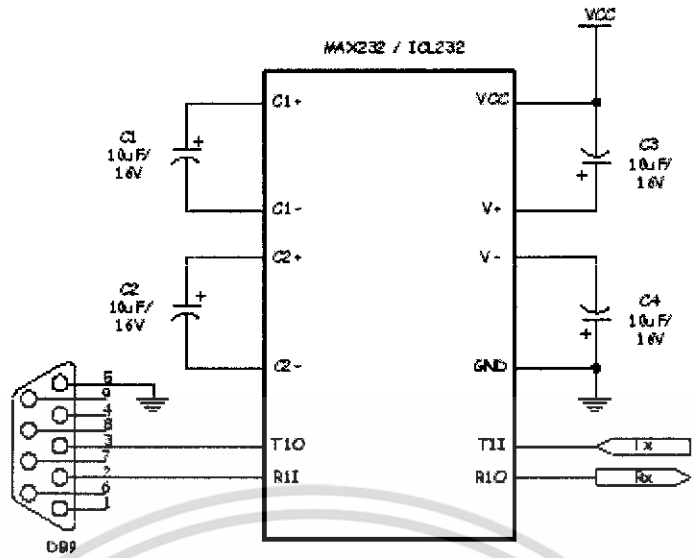
รูปที่ 2.27 การต่ออุปกรณ์ภายนอกกับคอมพิวเตอร์โดยใช้สัญญาณเพียง 3 เส้น

2.3.6 วงจรเชื่อมต่อระหว่าง DB-9 กับ FPGA

จากหัวข้อที่ผ่านมาตอนนี้เราทราบแล้วว่าในการสื่อสารแบบอนุกรมนั้น ในระดับของลอจิก "0" จะแทนระดับลอจิกด้วยระดับแรงดันระหว่าง +3 โวลท์ ถึง +12 โวลท์และในระดับของลอจิก "1" จะแทนระดับลอจิกด้วยระดับแรงดันระหว่าง –3 โวลท์ ถึง –12 โวลท์ เพราะฉะนั้นต้องทำการแปลงระดับของลอจิก "1" และ "0" ให้เป็นระดับแรงดันดังกล่าว ซึ่งจะต้องใช้วงจรในการแปลงระดับแรงดัน สามารถแสดงลักษณะการส่งข้อมูลแบบอนุกรมที่ผ่านตัวแปลงแรงดันได้ดังรูปที่ 2.28 ในส่วนของวงจรแปลงระดับแรงดันสำหรับการสื่อสาร RS-232 นั้นจะใช้ไอซีเบอร์ MAX232 หรือ ICL232 ดังรูปที่ 2.29



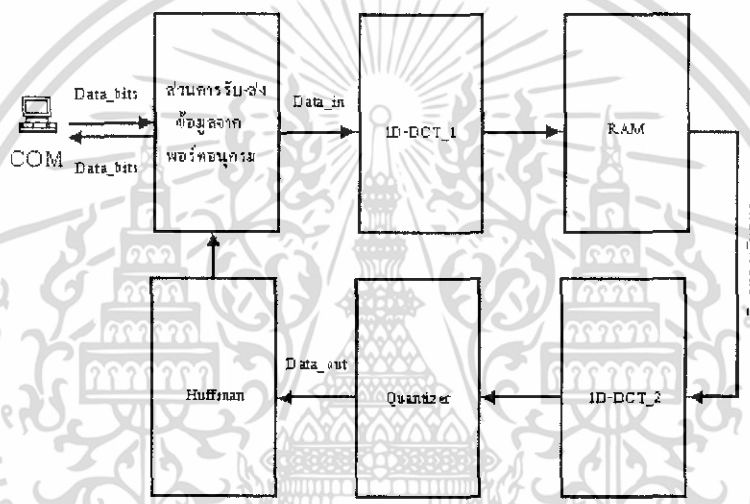
รูปที่ 2.28 รูปแบบการสื่อสารแบบอนุกรม



รูปที่ 2.29 วงจรแปลงระดับแรงดันของการสื่อสารแบบอนุกรม

บทที่ 3
การคำนวณและการสร้าง

ปริญญานิพนธ์นี้จะทำการลดขนาดของข้อมูลรูปภาพด้วยวิธี JPEG เพื่อใช้ทดลองในโครงงานนี้ โดยใช้การแปลงดิสครีตโคซายน์แบบ 2 มิติด้วยวิธีการแยกเป็นการแปลงดิสครีตโคซายน์ 1 มิติ จำนวน 2 ครั้ง จากนั้นนำผลลัพธ์มาประมวลผลเพื่อลดขนาดด้วยวิธีการควอนไตเซชั่น (Quantization) เพื่อลดจำนวนบิท และนำข้อมูลที่ได้ไปทำการเรียงข้อมูลแบบซิกแซก (Zigzag) จากนั้นจะทำการเข้ารหัสโดยใช้วิธีการของ Huffman encoding สามารถแสดงขั้นตอนของการลดขนาดข้อมูลภาพเป็นบล็อกไดอะแกรมได้ดังรูปที่ 3.1



รูปที่ 3.1 แสดงลำดับขั้นตอนการทำงาน

3.1 การออกแบบส่วนการแปลงดิสครีตโคซายน์

3.1.1 สมการของการทรานสฟอร์ม

การทรานสฟอร์มทั้งหมดที่กล่าวถึงนี้ จะกล่าวถึงการทรานสฟอร์มที่เป็นเชิงเส้นซึ่งสามารถเขียนสมการของการทรานสฟอร์มแบบ 1 มิติ (1 D Forward Transform) ได้ดังนี้

$$\theta_n = \sum_{i=0}^{N-1} X_i a_{n,i} \tag{3.1}$$

เมื่อ θ_n เป็นลำดับการทรานสฟอร์ม (Transform Sequence)

X_i เป็นลำดับของอินพุท (Original Sequence)

$a_{n,i}$ เป็นสัมประสิทธิ์การทรานสฟอร์ม

สามารถเขียนสมการของการอินเวอร์สทรานสฟอร์มแบบ 1 มิติ (1D Inverse Transform) ได้ดังนี้

$$X_n = \sum_{i=0}^{N-1} \theta_i b_{n,i} \tag{3.2}$$

เมื่อพิจารณาสมการที่ 3.1 สามารถเขียนสมการการทรานสฟอร์ม 1 มิติ ในรูปเมตริกซ์ ได้ดังนี้

$$\theta = CX \tag{3.3}$$

เมื่อพิจารณาสมการที่ 3.2 สามารถเขียนสมการอินเวอร์สทรานสฟอร์ม 1 มิติ ในรูปเมตริกซ์ ได้ดังนี้

$$X = D\theta \tag{3.4}$$

โดย เมตริกซ์ θ, X มีขนาด $N \times 1$

เมตริกซ์ C, D มีขนาด $N \times N$

เมื่อพิจารณาข้อมูลแบบ 2 มิติ สามารถเขียนสมการของการทรานสฟอร์มแบบ 2 มิติ (2D Forward Transform) ได้ดังนี้

$$\Theta_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{i,j} a_{k,j} a_{k,i} \tag{3.5}$$

เมื่อพิจารณาสมการที่ 3.5 สามารถเขียนสมการทรานสฟอร์ม 2 มิติ ในรูปเมตริกซ์ ได้ดังนี้

$$\Theta = CXC^T \tag{3.6}$$

หรือสามารถเขียนสมการการอินเวอร์สทรานสฟอร์ม 2 มิติ ในรูปเมตริกซ์ ได้ดังนี้

$$X = C^T \Theta C \tag{3.7}$$

โดย Θ เป็นเอาท์พุทของการทรานสฟอร์ม เป็นเมตริกซ์ขนาด $N \times N$

C,D เป็นสัมประสิทธิ์ของการทรานสฟอร์ม เป็นเมตริกซ์ขนาด $N \times N$

X เป็นอินพุทของการทรานสฟอร์ม เป็นเมตริกซ์ขนาด $N \times N$

3.1. 2 การแปลงแบบดิสครีตโคซายน์ (Discrete Cosine Transform)

สมการที่ใช้สำหรับการแปลงดิสครีตโคซายน์ (Forward Discrete Cosine Transform) สามารถเขียนสมการให้อยู่ในรูปของผลคูณของอินพุทกับค่าสัมประสิทธิ์ได้ดังนี้

หนึ่งมิติ

$$y(n) = e(n)\sum_{k=0}^{N-1} x(k)\cos\left[\frac{(2k+1)n\pi}{2N}\right] \quad ;n = 0,1,\ldots,N-1 \tag{3.8}$$

สองมิติ

$$y(m,n) = \frac{4}{N^2}e(m)e(n)\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} x(k,l)\cos\left[\frac{(2k+1)n\pi}{2N}\right]\cos\left[\frac{(2l+1)n\pi}{2N}\right] \tag{3.9}$$

m = $0,1,\ldots,N\text{-}1$

n = $0,1,\ldots,N\text{-}1$

$$e(n) = \begin{cases} \dfrac{1}{\sqrt{2}} & ;n = 0 \\ 1 & ;etc \end{cases}$$

$$e(m) = \begin{cases} \dfrac{1}{\sqrt{2}} & ;m = 0 \\ 1 & ;etc \end{cases}$$

ดิสครีตโคซายน์ทรานสฟอร์มคือการแปลงข้อมูลที่อยู่ในสปาเชียล (Spatial Domain) ให้อยู่ในรูปแบบของโดเมนความถี่ เพื่อทำให้ค่าที่ได้ออกมาเป็นอิสระต่อกัน และลดขนาดของข้อมูลลงให้ค่าพลังงานส่วนใหญ่ของข้อมูลอยู่ในช่วงความถี่ต่ำ โดยดิสครีตโคซายน์ทรานสฟอร์ม จะเป็นการแปลงแบบเชิงเส้นที่มีค่าสัมประสิทธิ์ในการแปลงคงที่ และค่าที่ได้ในโดเมนความถี่จะมีเฉพาะจำนวนจริงเท่านั้น ซึ่งข้อมูลส่วนใหญ่ที่จะใช้การแปลงแบบดิสครีตโคซายน์จะเป็นข้อมูลจากภาพนิ่ง หรือเป็นภาพเคลื่อนไหว อินพุทที่เข้ามาจะถูกแยกเป็นบล็อกเล็กๆ โดยส่วนมากแล้วขนาดของบล็อกจะมีค่าเป็นเลขยกกำลังสอง เช่น 4, 8, 16

ในกรณีของการแปลงดิสครีตโคซายน์ ค่าสัมประสิทธิ์ของการทรานสฟอร์มสามารถหาได้ดังนี้

$$[C]_{i,j} = \begin{cases} \sqrt{\dfrac{1}{N}}\cos\dfrac{(2j+1)i\pi}{2N} & i = 0, j = 0,1,\ldots,N-1 \\ \sqrt{\dfrac{2}{N}}\cos\dfrac{(2j+1)i\pi}{2N} & i = 1,2,\ldots,N-1, j = 0,1,\ldots,N-1 \end{cases} \tag{3.10}$$

เมื่อ N คือ ขนาดของบล็อก ($N = 2^n$, $n = 2,3,4,\ldots$)

จากสมการที่ 3.10 สามารถเขียนค่าสัมประสิทธิ์ของการแปลงดิสครีตโคซายน์ให้อยู่ในรูปของเมตริกซ์ ขนาด $N \times N$ เมื่อ $N = 8$ ได้ดังนี้

$$C = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos\left(\frac{\pi}{16}\right) & \cos\left(\frac{3\pi}{16}\right) & \cdots & \cos\left(\pi - \frac{\pi}{16}\right) \\ \cdots & \cdots & \cdots & \cdots \\ \cos\left(\frac{7\pi}{16}\right) & \cos\left(\pi + \frac{5\pi}{16}\right) & \cdots & \cos\left(7\pi - \frac{7\pi}{16}\right) \end{bmatrix} \quad (3.11)$$

จากค่าสัมประสิทธิ์ของการทรานสฟอร์มในสมการที่ 3.11 สามารถเขียนแทนค่าฟังก์ชั่นตรีโกณมิติดังกล่าวด้วยตัวแปรได้ดังตารางที่ 3.1

ตารางที่ 3.1 แสดงค่าสัมประสิทธิ์ของการทรานสฟอร์มในรูปแบบของฟังก์ชั่นตรีโกณมิติ

ค่าสัมประสิทธิ์	ฟังก์ชั่นโคไซน์ (cosine)
C_1	$\cos\left(\frac{\pi}{16}\right)$
C_2	$\cos\left(\frac{2\pi}{16}\right)$
C_3	$\cos\left(\frac{3\pi}{16}\right)$
C_4	$\cos\left(\frac{4\pi}{16}\right)$
C_5	$\cos\left(\frac{5\pi}{16}\right)$
C_6	$\cos\left(\frac{6\pi}{16}\right)$
C_7	$\cos\left(\frac{7\pi}{16}\right)$

จากสมการที่ 3.3 เมื่อแทนค่าสัมประสิทธิ์การแปลงดิสครีตโคซายน์ จากสมการที่ 3.11 แล้ว สามารถเขียนแสดงในรูปเมตริกซ์ได้ ดังนี้

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \end{bmatrix} = \begin{bmatrix} c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2 \\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4 \\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6 \\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \quad (3.12)$$

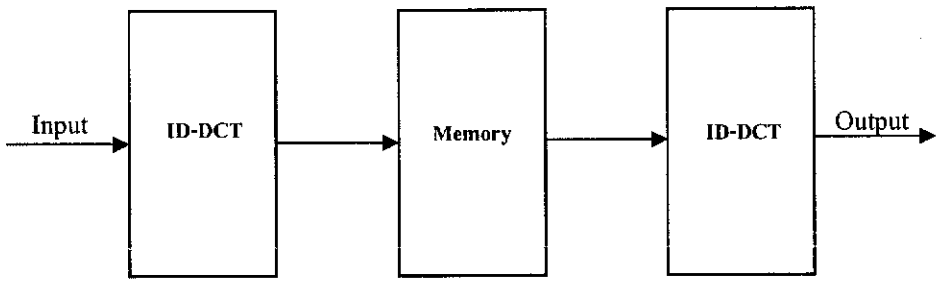
จากสมการที่ 3.3 จะได้ค่าสัมประสิทธิ์ที่ใช้ในการถ่วงน้ำหนักสำหรับแต่ละคอมโพเนนท์ทางความถี่ตามสมการที่ 3.11 คือ

$$C = \begin{bmatrix} c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2 \\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4 \\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6 \\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \end{bmatrix} \quad (3.13)$$

เมื่อแทนค่า C ตามตารางที่ 3.1 ลงไปจะได้

$$C = \begin{bmatrix} +0.707 & +0.707 & +0.707 & +0.707 & +0.707 & +0.707 & +0.707 & +0.707 \\ +0.981 & +0.831 & +0.556 & +0.195 & -0.195 & -0.556 & -0.831 & -0.981 \\ +0.924 & +0.383 & -0.383 & -0.924 & -0.924 & -0.383 & +0.383 & +0.924 \\ +0.831 & -0.195 & -0.981 & -0.556 & +0.556 & +0.981 & +0.195 & -0.831 \\ +0.707 & -0.707 & -0.707 & +0.707 & +0.707 & -0.707 & -0.707 & +0.707 \\ +0.556 & -0.981 & +0.195 & +0.831 & -0.831 & -0.195 & +0.981 & -0.556 \\ +0.383 & -0.924 & +0.924 & -0.383 & -0.383 & +0.924 & -0.924 & +0.383 \\ +0.195 & -0.556 & +0.831 & -0.981 & +0.981 & -0.831 & +0.556 & -0.195 \end{bmatrix} \quad (3.14)$$

การแยกการแปลงดิสครีตโคซายน์แบบ 2 มิติ ให้มาเป็นการแปลงดิสครีตโคซายน์แบบ 1 มิติ จำนวน 2 ครั้ง อธิบายโดยใช้บล็อกไดอะแกรมดังแสดงในรูปที่ 3.2



รูปที่ 3.2 แสดงบล็อกไดอะแกรมของการแปลงดิสครีตโคซายน์แบบ 2 มิติ

เนื่องจากเราแยกการแปลงดิสครีตโคซายน์แบบ 2 มิติ ให้มาเป็นการแปลงดิสครีตโคซายน์แบบ 1 มิติ จำนวน 2 ครั้ง

ดังนั้นค่าสัมประสิทธิ์ที่ใช้ในการถ่วงน้ำหนักสำหรับแต่ละคอมโพเนนท์ทางความถี่ หรือค่า C จะถูกหารด้วย 2 ก่อน จะได้ค่าสัมประสิทธิ์ที่ใช้ในการถ่วงน้ำหนักสำหรับแต่ละคอมโพเนนท์ทางความถี่ที่นำไปใช้ดังนี้คือ

$$C = \begin{bmatrix} +0.354 & +0.354 & +0.354 & +0.354 & +0.354 & +0.354 & +0.354 & +0.354 \\ +0.491 & +0.416 & +0.278 & +0.098 & -0.098 & -0.278 & -0.416 & -0.491 \\ +0.462 & +0.192 & -0.192 & -0.462 & -0.462 & -0.192 & +0.192 & +0.462 \\ +0.416 & -0.098 & -0.491 & -0.278 & +0.278 & +0.491 & +0.098 & -0.416 \\ +0.354 & -0.354 & -0.354 & +0.354 & +0.354 & -0.354 & -0.354 & +0.354 \\ +0.278 & -0.491 & +0.098 & +0.416 & -0.416 & -0.098 & +0.491 & -0.278 \\ +0.192 & -0.462 & +0.462 & -0.192 & -0.192 & +0.462 & -0.462 & +0.192 \\ +0.098 & -0.278 & +0.416 & -0.491 & +0.491 & -0.416 & +0.278 & -0.098 \end{bmatrix} \quad (3.15)$$

แปลงสมการที่ 3.15 ให้เป็นเลขฐาน 2 ที่มีเครื่องหมายได้ดังนี้

$$C = \begin{bmatrix} 01011010 & 01011010 & 01011010 & 01011010 & 01011010 & 01011010 & 01011010 & 01011010 \\ 01111101 & 01101010 & 01000111 & 00011000 & 10011000 & 11000111 & 11101010 & 11111101 \\ 01110110 & 00110000 & 10110000 & 11110110 & 11110110 & 10110000 & 00110000 & 01110110 \\ 01101010 & 10011000 & 11111101 & 11000111 & 01000111 & 01111101 & 00011000 & 11101010 \\ 01011010 & 11011010 & 11011010 & 01011010 & 01011010 & 11011010 & 11011010 & 01011010 \\ 01000111 & 11111101 & 00011000 & 01101010 & 11101010 & 10011000 & 01111101 & 11000111 \\ 00110000 & 11110110 & 01110110 & 10110000 & 10110000 & 01110110 & 11110110 & 00110000 \\ 00011000 & 11000111 & 01101010 & 11111101 & 01111101 & 11101010 & 01000111 & 10011000 \end{bmatrix} \quad (3.16)$$

ทำการ Transpose ค่าที่ได้เพื่อนำไปเก็บไว้ในรอมของ FPGA ได้ดังนี้

$$C^T = \begin{bmatrix} 01011010 & 01111101 & 01110110 & 01101010 & 01011010 & 01000111 & 00110000 & 00011000 \\ 01011010 & 01101010 & 00110000 & 10011000 & 11011010 & 11111101 & 11110110 & 11000111 \\ 01011010 & 01000111 & 10110000 & 11111101 & 11011010 & 00011000 & 01110110 & 01101010 \\ 01011010 & 00011000 & 11110110 & 11000111 & 01011010 & 01101010 & 10110000 & 11111101 \\ 01011010 & 10011000 & 11110110 & 01000111 & 01011010 & 11101010 & 10110000 & 01111101 \\ 01011010 & 11000111 & 10110000 & 01111101 & 11011010 & 10011000 & 01110110 & 11101010 \\ 01011010 & 11101010 & 00110000 & 00011000 & 11011010 & 01111101 & 11110110 & 01000111 \\ 01011010 & 11111101 & 01110110 & 11101010 & 01011010 & 11000111 & 00110000 & 10011000 \end{bmatrix} \quad (3.17)$$

การแยกการแปลงดิสครีตโคซายน์แบบ 2 มิติ ให้มาเป็นการแปลงดิสครีตโคซายน์แบบ 1 มิติ จำนวน 2 ครั้ง สามารถอธิบายการทำงาน ได้ดังนี้

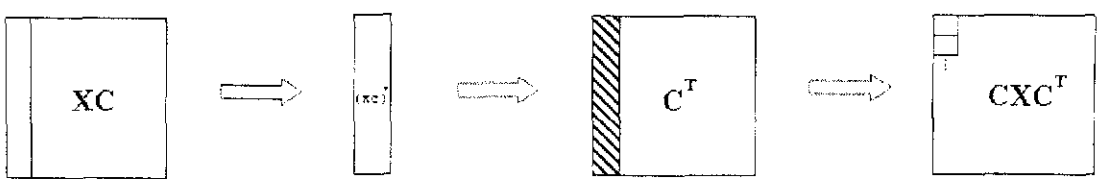
1. การแปลงดิสครีตโคซายน์แบบ 1 มิติ โดยคูณข้อมูลภาพในแนวนอนกับสัมประสิทธิ์ในแนวนอนของเมตริกซ์ C

โดยนำข้อมูลตามแนวนอน 1 มิติ ขนาด 1 x N จากข้อมูลเริ่มต้น 2 มิติ ขนาด N x N มาทำการแปลงดิสครีตโคซายน์แบบ 1 มิติ ทีละแถว จนครบทั้งหมด N แถว จากนั้นพักข้อมูลไว้ในแรม สามารถแสดงบล็อกไดอะแกรมของการแปลงดิสครีตโคซายน์แบบ 1 มิติ ตามแนวนอนในรูปของเมตริกซ์ได้ดังรูปที่ 3.3

รูปที่ 3.3 แสดงบล็อกไดอะแกรมการแปลงดิสครีตโคซายน์แบบ 1 มิติ ตามแนวนอน

2. การแปลงดิสครีตโคซายน์แบบ 1 มิติ โดยคูณข้อมูลภาพในแนวนอนกับสัมประสิทธิ์ในแนวตั้งของเมตริกซ์ C

โดยนำข้อมูลตามแนวตั้ง 1 มิติ ขนาด 1 x N จากข้อมูลที่ได้จากการแปลงดิสครีตโคซายน์แบบ 1 มิติ ในส่วนที่ 1 ที่เป็น 2 มิติ ขนาด N x N มาทำการแปลงดิสครีตโคซายน์ 1 มิติ ทีละหลัก จนครบทั้งหมด N หลัก สามารถแสดงบล็อกไดอะแกรมของการแปลงดิสครีตโคซายน์ 1 มิติ ตามแนวตั้งในรูปของเมตริกซ์ได้ในรูปของเมตริกซ์ได้ดังรูปที่ 3.4



รูปที่ 3.4 แสดงบล็อกไดอะแกรมการแปลงดิสครีตโคซายน์แบบ 1 มิติ ตามแนวตั้ง

ผลลัพธ์ที่ได้จากการแปลงดิสครีตโคซายน์แบบ 2 มิติ โดยใช้การแปลงดิสครีตโคซายน์แบบ 1 มิติ จำนวน 2 ครั้ง คือ

$$Y = CXC^T \tag{3.18}$$

จะเห็นได้ว่าผลลัพธ์ที่ได้จากการแปลงดิสครีตโคซายน์แบบ 2 มิติ โดยใช้การแปลงดิสครีตโคซายน์แบบ 1 มิติ จำนวน 2 ครั้ง ดังแสดงในสมการที่ 3.18 ได้ผลลัพธ์เท่ากับการแปลงดิสครีตโคซายน์แบบ 2 มิติ ในสมการที่ 3.6

3.2 การออกแบบส่วนการควอนไตซ์

การควอนไตซ์ในที่นี้เป็นเวกเตอร์ควอนไตซ์ลดค่าของข้อมูลภายในบล็อกเพื่อจะลดจำนวนบิตที่ใช้ในการเก็บข้อมูลเหล่านี้ในขั้นตอนเข้ารหัสต่อไป และจากการที่ภาพกระจายตัวของข้อมูลสีอยู่ในช่วงความถี่ต่ำๆ JPEG จึงให้ความสำคัญต่อข้อมูลในช่วงความถี่ต่ำในช่วงความถี่สูง ดังนั้นการควอนไตซ์ คือ ความพยายามลดความสำคัญของข้อมูลในช่วงความถี่สูงโดยพยายามทำให้กลายเป็นค่าศูนย์ให้มากที่สุดนั่นเองโดยใช้บล็อกข้อมูลของควอนไตเซอร์ 8 × 8 ค่า หารบล็อกข้อมูลจากการแปลงแบบดิสครีตโคซายน์ 8×8 ค่าสมการต่อไปนี้

$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Q(u,v)}\right] \tag{3.19}$$

และมีสมการดิควอนไตเซชั่น ดังนี้

$$\dot{T}(u,v) = \hat{T}(u,v)Q(u,v) \tag{3.20}$$

เมื่อ $T(u,v)$ คือ ค่าสัมประสิทธิ์ DCT

$\hat{T}(u,v)$ คือ ค่าสัมประสิทธิ์ DCT เมื่อถูกควอนไตซ์

$\dot{T}(u,v)$ คือ ค่าสัมประสิทธิ์ DCT เมื่อดิควอนไตซ์

$Q(u,v)$ คือ ค่าควอนไตเซอร์

และ *round* คือ การหาจำนวนเต็มที่มีค่าใกล้เคียงที่สุด

โดย $0 \le u \le 7, 0 \le v \le 7$

ซึ่งผู้เขียนได้ออกแบบบล็อกข้อมูลควอนไตเซอร์ ที่ใช้ในการทำงานโดยใช้สมการคือ

$$Q(u,v) = 1 + (u + v + 1) \times quality \tag{3.21}$$

และได้ออกแบบให้สามารถเปลี่ยนค่า $Q(u,v)$ ภายในบล็อกได้หลายระดับด้วยการนำค่าของควอนไตเซอร์แฟคเตอร์ (quantizer factor) (ตัวแปรที่ใช้ในโปรแกรมมีค่าระหว่าง 0 ถึง 15 มาเพิ่ม

และได้ออกแบบให้สามารถเปลี่ยนค่า $Q(u,v)$ ภายในบล็อกได้หลายระดับด้วยการนำค่าของควอนไตเซอร์ แฟคเตอร์ (quantizer factor) ตัวแปรที่ใช้ในโปรแกรมมีค่าระหว่าง 0 ถึง 15 มาเพิ่มให้กับควอนไตเซอร์ทั้ง 64 ตัวภายในบล็อก ซึ่งค่าของ Q.F. (Quantizer Factor) นี้สามารถกำหนดได้โดยปรับที่ DIP Switch ที่บอร์ด FPGA

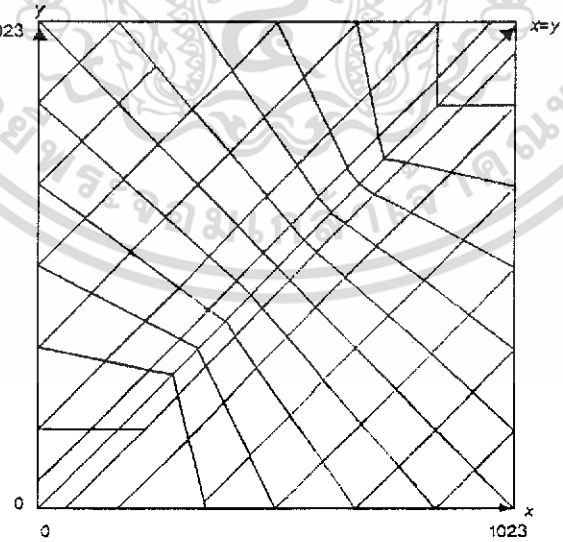
จากสมการ เนื่องจากจะต้องหารด้วยบล็อกข้อมูลควอนไตเซอร์ ซึ่งไม่สามารถทำกระบวนการหารบน FPGA ได้จึงต้องทำการสร้างข้อมูลในส่วนของ $1/Q(U,V)$ แปลงเป็นไบนารี่เก็บไว้ในรอมก่อน จากนั้นค่อยเรียกข้อมูลส่วนนี้มาทำการคูณกับข้อมูลที่ได้จากส่วน DCT

ข้อมูลในส่วนของ $1/Q(U,V)$ จะเป็นค่าทศนิยมจึงต้องคูณด้วย 1024 เพื่อให้เป็นจำนวนเต็ม จากนั้นจึงแปลงเป็นไบนารี่ที่เก็บไว้ในรอมมีดังนี้

"10000000000","10000000000","01000000000","00101010101","00100000000","00110011001",
"00010101011","00010010010","00010000000","00001110010","00001100110","00001011101",
"00001010101","00001001111","00001001001","00001000100","00001000000","00000111100",
"00000111001","00000110110","00000110011","00000110001","00000101111","00000101101",
"00000101011","00000101001","00000100111","00000100110","00000100101","00000100011",
"00000100010","00000100001","00000100000","00000011111","00000011110","00000011101",
"00000011100","00000011100","00000011011","00000011010","00000011010","00000011001",
"00000011000","00000011000","00000010111","00000010111","00000010110","00000010110",
"00000010101","00000010101","00000010100","00000010100","00000010100","00000010011",
"00000010011","00000010011","00000010010","00000010010","00000010010","00000010001",
"00000010001","00000010001","00000010001","00000010000","00000010000","00000010000",
"00000010000","00000001111","00000001111","00000001111","00000001111","00000001110",
"00000001110","00000001110","00000001110","00000001110","00000001101","00000001101",
"00000001101","00000001101","00000001101","00000001101","00000001100","00000001100",
"00000001100","00000001100","00000001100","00000001100","00000001100","00000001100",
"00000001011","00000001011","00000001011","00000001011","00000001011","00000001011",
"00000001011","00000001011","00000001010","00000001010","00000001010","00000001010",
"00000001010","00000001010","00000001010","00000001010","00000001010","00000001010",
"00000001001","00000001001","00000001001","00000001001","00000001001","00000001001",
"00000001001","00000001001","00000001001","00000001001","00000001001","00000001001",
"00000001001","00000001000","00000001000","00000001000","00000001000","00000001000",
"00000001000","00000001000","00000001000","00000001000","00000001000","00000001000",
"00000001000","00000001000","00000001000","00000001000","00000001000","00000000111",
"00000000111","00000000111","00000000111","00000000111","00000000111","00000000111",
"00000000111","00000000111","00000000111","00000000111","00000000111","00000000111",

"00000000110","00000000110","00000000110","00000000110","00000000110","00000000110",
"00000000110","00000000110","00000000110","00000000110","00000000110","00000000110",
"00000000110","00000000110","00000000110","00000000110","00000000110","00000000110",
"00000000110","00000000110","00000000110","00000000110","00000000110","00000000110",
"00000000110","00000000101","00000000101","00000000101","00000000101","00000000101",
"00000000101","00000000101","00000000101","00000000101","00000000101","00000000101",
"00000000101","00000000101","00000000101","00000000101","00000000101","00000000101",
"00000000101","00000000101","00000000101","00000000101","00000000101","00000000101",
"00000000101","00000000101","00000000101","00000000101","00000000101","00000000101",
"00000000101","00000000101","00000000101","00000000101","00000000101","00000000101",
"00000000100","00000000100","00000000100","00000000100","00000000100","00000000100",
"00000000100","00000000100","00000000100","00000000100","00000000100","00000000100",
"00000000100","00000000100","00000000100","00000000100","00000000100","00000000100",
"00000000100","00000000100","00000000100","00000000100","00000000100","00000000100");

และสามารถเขียนเป็นกราฟความสัมพันธ์ระหว่างค่า $T(u,v)$ กับ $\hat{T}(u,v)$ เมื่อเปลี่ยนค่า Q.F. ได้อย่างคร่าวๆ ดังรูปที่ 3.5



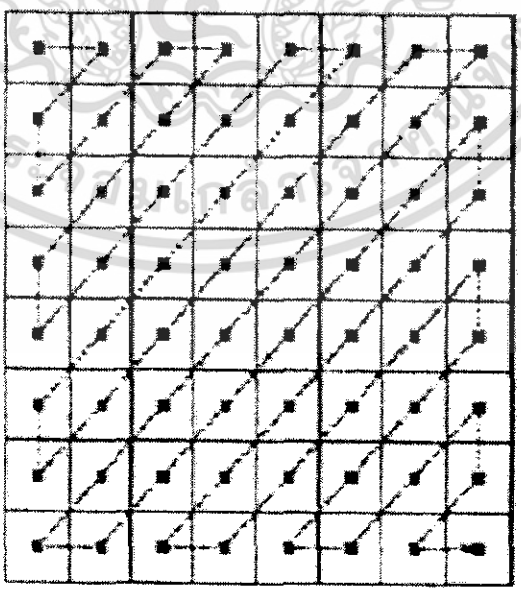
รูปที่ 3.5 กราฟแสดงความสัมพันธ์ระหว่าง $T(u,v)$ กับ $\hat{T}(u,v)$ ที่มีค่า Q.F. ต่างๆ

จากกราฟจะเห็นว่าการเพิ่มค่า Q.F. จะมีผลทำให้มีจำนวนค่า $\hat{T}(u,v)$ ที่เป็นศูนย์เพิ่มมากขึ้น ซึ่งจำนวนของค่า $\hat{T}(u,v)$ ที่เป็นศูนย์ที่เพิ่มขึ้นนี้จะเป็นประโยชน์ต่อการบีบอัดข้อมูล เนื่องจากในขั้นตอนการเข้ารหัสค่า AC แบบฮัฟแมน (Huffman) นั้นจะทำการเข้ารหัสข้อมูลเฉพาะค่าที่ไม่เป็นศูนย์ โดยข้อมูลที่เป็นศูนย์นั้นจะมีเทคนิคการเข้ารหัสรันเลนจ์ (runlength encoding) มาช่วยในการเข้ารหัส ดังนั้นถ้าจำนวนข้อมูลที่เป็นศูนย์มีมากขึ้นจะทำให้จำนวนข้อมูลที่ถูกเข้ารหัส (คือข้อมูลที่ไม่เป็นศูนย์) น้อยลง ซึ่งจะส่งผลให้จำนวนรหัส ที่ได้มีจำนวนน้อยลงด้วย และทำให้ได้อัตราการบีบอัดข้อมูลที่สูงขึ้น แต่ข้อมูล $\hat{T}(u,v)$ ที่ถูกควอนไตซ์เป็นศูนย์จะมีผลเสีย คือ เมื่อนำมาดิควอนไตซ์ตามสมการที่ 3.19 จะได้ค่าที่เป็นศูนย์คืนกลับมา ซึ่งมีความเป็นไปได้สูงที่มีค่าผิดพลาดจากค่าเดิมเป็นอย่างมาก อันจะมีผลทำให้คุณภาพของภาพที่ได้คืนมาแย่ลงมาก

3.3 การออกแบบส่วนการเข้ารหัสข้อมูล

3.3.1 ส่วนการอ่านข้อมูลแบบซิกแซก

การอ่านข้อมูลแบบซิกแซก เป็นการอ่านข้อมูลที่ได้จากการควอนไตซ์ซึ่งอยู่ในรูปแบบของบล็อกขนาด 8 แถว 8 หลัก ให้อยู่ในรูปของชุดข้อมูลที่เรียงต่อกันไป โดยลักษณะของการอ่านข้อมูลแบบซิกแซกนี้มีลักษณะดังรูปที่ 3.6 ซึ่งการอ่านข้อมูลในลักษณะนี้ก็เพื่อให้สอดคล้องกับข้อมูลที่ได้จากการแปลงแบบดิสครีตโคซายน์และการควอนไตซ์เพราะในการแปลงแบบดิสครีตโคซายน์นั้นผลที่ได้จากการแปลงจะมีการเรียงค่าส่วนประกอบทางความถี่เพิ่มขึ้นตามแนวเส้นทแยงมุมของบล็อกขนาด 8 แถว 8 หลัก ดังได้กล่าวไว้แล้วในเรื่องการแปลงแบบดิสครีตโคซายน์ และผลที่ได้จากการแปลงนี้เมื่อนำมาทำการควอนไตซ์ค่าของข้อมูลก็จะลดลงตามแนวเส้นทแยงมุมของบล็อกเช่นเดียวกันและในช่วงความถี่สูงๆ ค่าจะถูกลดลงเป็นศูนย์มาก ดังนั้นการอ่านข้อมูลแบบซิกแซกก็มีความพยายามที่จะนำค่าเป็นค่าศูนย์มาเรียงให้ติดกันเพื่อประโยชน์ในการบีบอัดข้อมูลในขั้นตอนการเข้ารหัสต่อไป



รูปที่ 3.6 แสดงลำดับการอ่านข้อมูลแบบซิกแซก

3.3.2 ส่วนการเข้ารหัสข้อมูล

กระบวนการสุดท้ายของ JPEG คือ การเข้ารหัสข้อมูลที่ผ่านการควอนไตซ์แล้ว ด้วยกระบวนการ 3 ขั้นตอน

1. การเก็บค่าสัมประสิทธิ์กระแสตรงของบล็อกพิกเซล (8×8) บล็อกแรกก่อน แล้วทำการเปลี่ยนค่าสัมประสิทธิ์กระแสตรงที่จุด (0, 0) ของบล็อกต่อมาเป็นค่าที่สัมพัทธ์กับค่าแรกหรือเป็นค่าผลต่างนั่นเอง แล้วจึงทำการเข้ารหัสข้อมูลผลต่างนี้แทน เพื่อประโยชน์คือ สามารถเข้ารหัสโดยใช้ค่าผลต่างที่ถือว่าน้อยด้วยจำนวนบิทที่น้อยกว่าการเข้ารหัสค่าจริง เพราะถือว่าบล็อกของพิกเซลที่อยู่ใกล้กันจะมีส่วนที่แตกต่างกันน้อยมากและสัมประสิทธิ์กระแสตรงก็ได้รวมข้อมูลที่สำคัญของภาพไว้แล้ว

2. การจัดลำดับสัมประสิทธิ์กระแสสลับเป็นลำดับแบบซิกแซกเพื่อให้ค่าศูนย์ที่เกิดขึ้นจากกระบวนการควอนไตซ์มีความต่อเนื่องกันยาวๆ ทำให้สามารถเข้ารหัสได้ง่าย

3. เลือกทำการเข้ารหัสสัมประสิทธิ์กระแสสลับที่จัดลำดับแล้ว

3.1 ด้วยวิธีความต่อเนื่องของข้อมูล RLE (Run Length Encoding) ซึ่งข้อดีของวิธีนี้คือ ถ้าค่าศูนย์ที่เรียงติดกันมีความต่อเนื่องกันยาวมากๆ จะทำให้สามารถเข้ารหัสได้ง่าย สั้นลง

3.2 การเข้ารหัสแบบเอนโทรปี (Entropy Encoding) ซึ่งอาจจะทำได้โดยวิธีฮัฟแมน หรือการประมวลผลทางคณิตศาสตร์ (Arithmetic) ก็ได้ ซึ่งในที่นี้จะใช้การเข้ารหัสแบบฮัฟแมน

3.3.3 การเข้ารหัสข้อมูลแบบฮัฟแมน

การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นเป็นการเข้ารหัสเฉพาะข้อมูลที่มีค่าไม่เป็นศูนย์ ดังนั้น จากการพยายามลดค่าของข้อมูลให้เป็นศูนย์มากๆ ในขั้นตอนของการควอนไตซ์และการอ่านข้อมูลแบบซิกแซกที่พยายามทำให้ค่าที่เป็นศูนย์มาอยู่เรียงกันนั้น จึงมีประโยชน์ต่อการเข้ารหัสข้อมูลแบบฮัฟแมนมาก เพราะจะทำให้ข้อมูลที่ต้องทำการเข้ารหัสมีจำนวนน้อยลง การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นจะแบ่งการเข้ารหัสข้อมูลเป็นสองส่วนคือ การเข้ารหัสข้อมูลของค่า DC และการเข้ารหัสข้อมูลของค่า AC

การเข้ารหัสค่า DC ของแถวข้อมูล

ในการเข้ารหัสค่า DC ของแถวข้อมูลนั้น (ข้อมูลตัวแรกในแถวซึ่งคิดค่าสเปกตรัมสัมประสิทธิ์กระแสตรงที่จุด (0, 0) ของบล็อกพิกเซลขนาด 8 × 8) จะถูกเข้ารหัสเฉพาะค่าความแตกต่างระหว่างค่า DC ของแถวข้อมูลปัจจุบัน กับค่า DC ของแถวข้อมูลซึ่งเป็นสีชนิดเดียวกันที่ถูกเข้ารหัสแถวล่าสุดที่ผ่านไป เช่น เมื่อกำลังทำการเข้ารหัส ค่า DC ของแถวข้อมูล C_b อยู่ซึ่งมีค่าเท่ากับ 40 จะทำการหาค่าความแตกต่างกัน ค่า DC ของแถวข้อมูล C_b ที่ถูกเข้ารหัสผ่านไปแล้วแถวล่าสุด (สมมติว่ามีค่าเท่ากับ 15) ดังนั้น ค่าที่จะถูกนำมาเข้ารหัสคิด 25 (40 - 15) ซึ่งการเข้ารหัสข้อมูลค่า DC จะมีการเข้ารหัสสองส่วนคือ

1. SIZE หมายถึง จำนวนบิทที่จะต้องใช้ในการใส่ค่าข้อมูล (ผลต่าง) ซึ่งจะหาจากตารางฮัฟแมน
2. AMPLITUDE หมายถึง ค่าขนาดของผลต่าง

ดังนั้น รหัสที่เข้าคือ (SIZE) (AMPILTUDE)

ในตารางฮัฟแมน จะเก็บรหัสความยาวต่างๆ ที่ใช้ในการแทนข้อมูลตามค่า Category ของข้อมูลนั้น ซึ่งการหาค่า Category ของข้อมูลสามารถหาได้จากตาราง JPEG Coefficient Coding Categories ซึ่งแสดงไว้ในตารางที่ 3.2 เมื่อได้ค่า Category แล้วก็สามารถหารหัสข้อมูลได้จากตารางค่า DC ของฮัฟแมน (ในภาคผนวก) โดยการเทียบหาตามค่า Category ที่ได้จากตารางที่ 3.2

ตารางที่ 3.2 ตารางแสดงค่า Categories ของข้อมูล

Range	DC Difference Category	AC Category
0	0	N/A
-1,1	1	1
-3,-2,2,3	2	2
-7,...,-4,4,...,7	3	3
-15,...,-8,8,...,15	4	4
-31,...,-16,16,...,31	5	5
-63,....-32,32,...,63	6	6
-127....,-64,64,...,127	7	7
-255,...,-128,128,...,255	8	8
-511,...,-256,256,...,511	9	9
-1023,...,-512,512,...,1023	A	A
-2047,...,-1024,1024,...,2047	B	B
-4095,...,-2048,2048,...,4095	C	C
-8191,...,-4096,4096,...,8191	D	D
-16383,...,-8192,8192,...,16383	E	E
-32767,...,-16384,16384,...,32767	F	F

จะเห็นว่ารหัสที่ได้จากตารางที่ 3.2 นั้น ใช้เป็นตัวบอกช่วงของมูลที่ถูกเข้ารหัสเท่านั้นซึ่งจะได้รหัสตัวที่หนึ่งของข้อมูลออกมา ส่วนการระบุค่าของข้อมูลนั้นจำเป็นจะใช้รหัสตัวที่สองเป็นตัวระบุ โดยรหัสตัวที่สองนี้จะหามาจากค่าของข้อมูลโดยตรง โดยจะมีจำนวนบิทเท่ากับค่า Category ของข้อมูลที่นำมาเข้ารหัสนั่นเอง และจะเอาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

การเข้ารหัสค่า AC ของแถวข้อมูล

เนื่องจากการจัดเรียงข้อมูลแบบซิกแซกนั้น จะทำให้ค่าศูนย์ที่เกิดขึ้นจากการควอนไตซ์อยู่เรียงกันอย่างต่อเนื่องยาวๆ ทำให้สามารถเข้ารหัสได้ง่าย และยิ่งมีศูนย์ที่ต่อเนื่องกันมากก็จะสามารถเข้ารหัสด้วยจำนวนที่น้อยลงได้มาก และโดยมากค่าศูนย์จะต่อเนื่องกันอยู่ในส่วนท้ายๆ ของแถวข้อมูล

การเข้ารหัสค่า AC (ข้อมูลตั้งแต่ตัวที่ 2 ถึงตัวที่ 64 ในแถวข้อมูล) จะต่างจากในการเข้ารหัสค่า DC คือ จะทำการเข้ารหัสค่าของข้อมูลโดยตรง (ไม่เข้ารหัสเฉพาะค่าความแตกต่างเหมือนใน DC) และการเข้ารหัสค่า AC นั้นจะทำการเข้ารหัสเฉพาะค่า AC ที่มีค่าไม่เป็นศูนย์เท่านั้น ส่วนค่า AC ที่เป็นศูนย์นั้น JPEG จะอาศัยการเข้ารหัสแบบ run-length มาช่วยในการเข้ารหัสดังนี้คือ ในการเข้ารหัสค่า AC ที่ไม่เป็นศูนย์แต่ละตัว จะเน้นจำนวนข้อมูล AC ที่เป็นศูนย์ซึ่งอยู่ติดกันด้านหน้าของข้อมูลที่จะเข้ารหัสนั้น โดยถือเป็นค่า Run ของข้อมูลในการเข้ารหัส เช่น ค่า AC ภายในแถวเป็น 5 8 0 0 9 7 0 0 0 0 4.... จะทำการเข้ารหัสเฉพาะข้อมูลที่ไม่เป็นศูนย์และมีค่า Run ของข้อมูลแต่ละตัวคือ 5 (Run = 0), 8(Rum = 0), 9(Run = 2), 7(Run=0), 4(Run = 4), ... โดยมีการเข้ารหัสข้อมูลดังกล่าวคล้ายใน DC แต่จะใช้รหัส 3 ตัว แทนข้อมูลแต่ละตัว ดังนี้

1. RUNLENGTH คือ จำนวนค่า 0 ที่ต่อเนื่องกันก่อนหน้าข้อมูลตัวที่จะเข้ารหัส
2. SIZE คือ จำนวนบิทที่จะต้องใช้ในการใส่ค่าข้อมูลที่ไม่เท่ากับ 0 ซึ่งหาจากตารางฮัฟแมน
3. AMPLITUDE คือ ค่าแอมปลิจูดหรือค่าของข้อมูลที่ไม่เท่ากับ 0 นั่นเอง

ดังนั้น รหัสคือ (RUNLENGTH, SIZE) (AMPLITUDE)

หมายเหตุ

มีเงื่อนไขพิเศษ คือ ถ้ามี 0 ต่อเนื่องกัน มากกว่า 16 ตัว สมมติว่าเป็น 22 ตัว จะได้รหัสเป็น (15, 0) (6, 4) (13)

(15, 0) ความหมายคือ มี 0 ต่อเนื่องกัน 16 ตัว (เป็นรูปแบบที่กำหนดตายตัว)

(6, 4) (13) ความหมายคือ มี 0 อีก 6 ตัว ใช้ 4 บิทในการแทนค่าข้อมูลที่มีค่าเท่ากับ 13

ซึ่งตารางที่ใช้ในการดูค่าจำนวนบิทที่ใช้ต่างๆ กัน เรียกว่าเป็นรหัสแบบความยาวของรหัสไม่คงที่หรือแวริเอเบิลเลนจ์โค้ด (Variable Length Code) เนื่องจากข้อมูลที่นำมาเข้ารหัสจะมีค่า Run เข้ามาเกี่ยวข้อง ดังนั้น ในตารางค่า AC ของฮัฟแมนจะต่างจากในตารางค่า DC ของฮัฟแมน คือจะมีการเปลี่ยนแปลงจาก Category ใน DC เป็น Run/Category นั่นคือ การหารหัสข้อมูลจากตารางค่า AC ของฮัฟแมน จะต้องทราบค่า Run และ Category ของข้อมูลที่นำมาเข้ารหัส ซึ่งการหาค่า Category ของข้อมูล AC จะเทียบหาจากตาราง JPEG Coefficient Coding Categories ตามตารางที่ 3.2 เช่นเดียวกับใน DC เมื่อทราบค่า Run และ Category ของข้อมูลแล้วก็จะสามารถหารหัสข้อมูลจากตารางค่า AC ของฮัฟแมนได้โดยเทียบตามค่า Run/Category

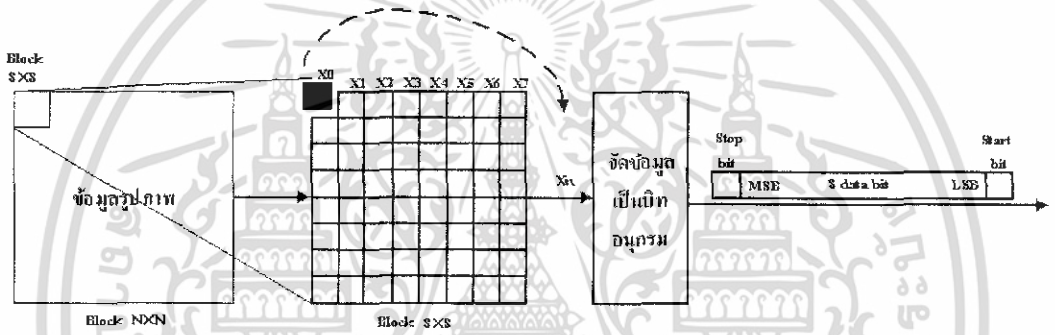
ภายในตารางค่า AC ของฮัฟแมนจะมีรหัสพิเศษ 2 ตัว คือ รหัสเมื่อค่า Run/Category เท่ากับ 0/0 ใช้ในกรณีเมื่อทำการเข้ารหัสข้อมูลภายในแถวจนกระทั่งเหลือแต่ข้อมูลที่เป็นศูนย์เพียงอย่างเดียว ก็จะใช้รหัสนี้เป็นตัวบอกตัวถอดรหัสว่าข้อมูลที่เหลือทั้งหมดภายในแถวมีค่าเป็นศูนย์ และรหัสอีกตัวหนึ่งเมื่อ Run/Category เท่ากับ F/0 ใช้เมื่อมีข้อมูลที่มีค่าศูนย์อยู่ติดกัน 16 ตัวภายในแถว โดยยังมีข้อมูลที่ไม่เป็นศูนย์ที่ยังไม่ถูกเข้ารหัสเหลืออยู่ในแถวอีก ก็จะใช้รหัสตัวนี้แทนข้อมูลที่มีค่าศูนย์ 16 ตัวดังกล่าว

จะเห็นได้ว่ารหัสข้อมูลที่ได้จากตารางค่า AC ของฮัฟแมนนั้นใช้ในการบอกจำนวนศูนย์ (ค่า Run) ที่อยู่ด้านหน้าของข้อมูลนั้น และช่วงของค่าข้อมูล (ค่า Category) ที่นำมาเข้ารหัส ดังนั้น

จะต้องมีรหัสตัวที่สองสำหรับใช้ในการระบุค่าของข้อมูล ซึ่งการหารหัสตัวที่สองนี้ใช้วิธีเดียวกันกับใน DC คือ เอามาจาก LSB จำนวน Category บิทของข้อมูลที่นำมาเข้ารหัสและจะเอามาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

3.4 ส่วนการรับส่งข้อมูลจากพอร์ตอนุกรม

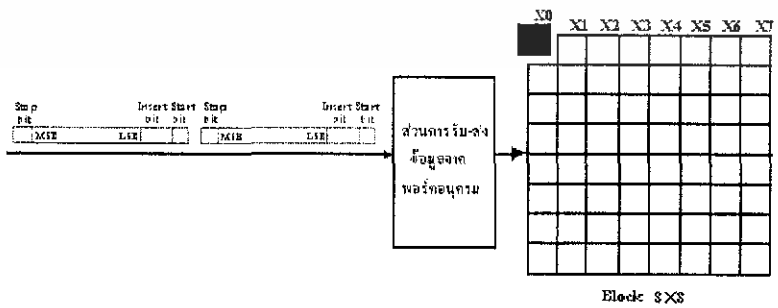
เมื่อแบ่งข้อมูลรูปภาพขนาด N×N ออกเป็นบล็อกเล็กๆ ขนาด 8 × 8 แล้วนำค่าข้อมูลตามแถวแต่ละพิกเซลของบล็อก 8 × 8 มาทำการแปลงให้เป็นข้อมูลไบนารี่แบบอนุกรม โดยกำหนดเฟรมข้อมูลที่ประกอบด้วย สตาร์ทบิท (Start bit) 1 บิท สตอบบิท (Stop bit) 2 บิท ข้อมูล (Data bit) 8 บิท แล้วส่งเฟรมข้อมูลดังกล่าวออกพอร์ตอนุกรม (Serial port) ไปยังบอร์ด FPGA ด้วยความถี่บอดเรต (Baud Rate) ที่กำหนดไว้ โดยทำการส่งข้อมูลดังกล่าวจนครบทั้ง 8× 8 พิกเซล สามารถแสดงบล็อกไดอะแกรมการแปลงบิทข้อมูลส่งออกพอร์ตอนุกรมไปยังบอร์ด FPGA ได้ดังรูปที่ 3.7



รูปที่ 3.7 แสดงบล็อกไดอะแกรมการแปลงบิทข้อมูลส่งออกพอร์ตอนุกรมไปยังบอร์ด FPGA

3.4.1 การรับข้อมูลจากพอร์ตอนุกรม

ส่วนของการรับข้อมูลจากพอร์ตอนุกรมนั้นทำหน้าที่รับเฟรมบิทของข้อมูลจากพอร์ตอนุกรมตามความถี่บอดเรต (Baud Rate) ที่กำหนดไว้มาทำการตัดสตาร์ทบิท (Start bit) และสตอบบิท (Stop bit) แล้วทำการเก็บข้อมูลไว้ในวงจรเก็บค่าข้อมูลอินพุทหรือแรม เมื่อรับข้อมูลครบ 64 ค่า ก็จะส่งต่อไปให้ส่วนที่ทำงานตามสัญญาณนาฬิกาของระบบ สามารถแสดงบล็อกไดอะแกรมการรับข้อมูลจากพอร์ตอนุกรมแล้วแปลงข้อมูลเป็นบิทขนานได้ ดังรูปที่ 3.8

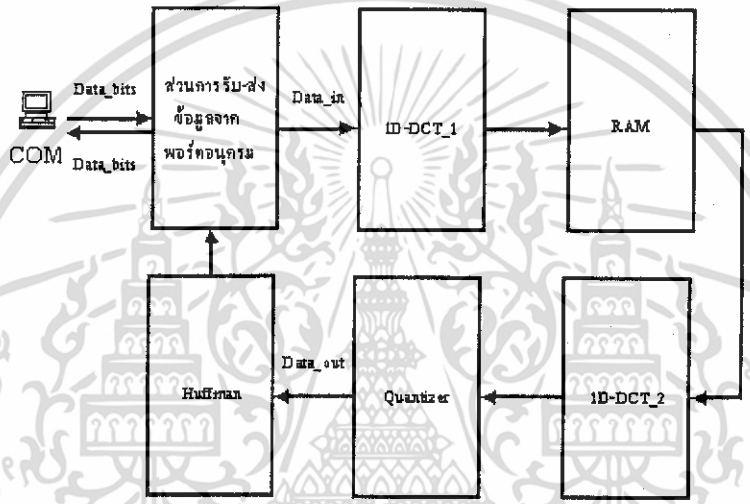


รูปที่ 3.8 แสดงการรับบิทข้อมูลมาเก็บข้อมูลไว้ในวงจรเก็บค่าข้อมูล

บทที่ 4

การทดลองและผลการทดลอง

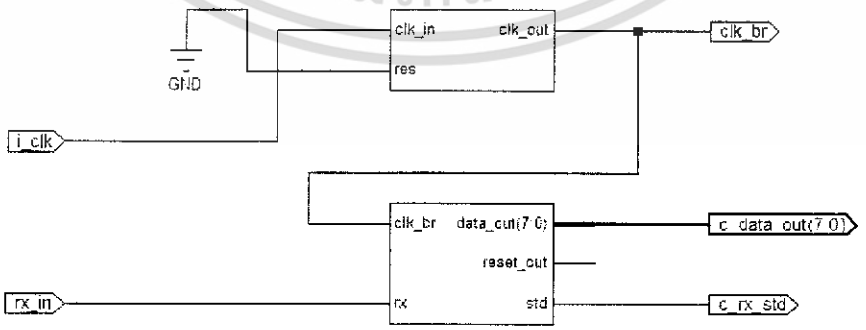
การออกแบบส่วนต่างๆ ของอุปกรณ์บีบอัดข้อมูลภาพนิ่งแบบ JPEG โดยทำการแยกการแปลงดิสครีตโคซายน์แบบ 1 มิติ จำนวน 2 ครั้ง จากนั้นทำการควอนไตซ์ระดับข้อมูลที่ได้แล้วทำการเข้ารหัสโดยใช้วิธีเข้ารหัสแบบ Huffman Coding สามารถทำการเขียนโปรแกรมการทำงานโดยให้แต่ละส่วนทำงานตามที่ได้ออกแบบโดยใช้ภาษา VHDL ทำการคอมไพล์ (Compile) แล้วเขียนโปรแกรมการทำงานจริงลงบนบอร์ด FPGA ส่วนของการสื่อสารข้อมูลทางพอร์ตอนุกรมอาศัยการทำงานและออกแบบโดยใช้โปรแกรม Delphi การออกแบบและการทดลองได้แบ่งออกเป็นส่วนๆ ดังรูปที่ 4.1



รูปที่ 4.1 แสดงลำดับขั้นตอนการทำงาน

4.1 ส่วนของการรับข้อมูลจากคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม

ส่วนประกอบของวงจรรับข้อมูลจากคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม



รูปที่ 4.2 แสดงส่วนประกอบของวงจรรับข้อมูลจากคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม

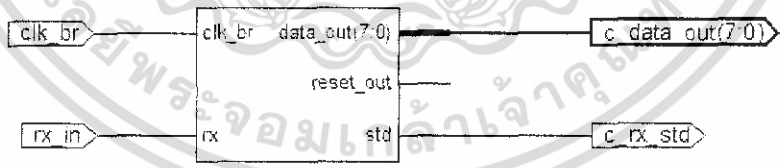
4.1.1 ส่วนของวงจรหารความถี่บอดเรต

ส่วนของวงจรหารความถี่บอดเรต ทำหน้าที่หารความถี่สัญญาณนาฬิกา Oscillator ให้ได้เอาท์พุทเป็นความถี่บอดเรต (Baud Rate) ที่ใช้ในการรับ - ส่งบิทข้อมูลผ่านทางพอร์ตอนุกรมปริญญานิพนธ์นี้นำสัญญาณอินพุทจาก Oscillator ความถี่ 24 MHz หารด้วยสองเท่าของความถี่บอดเรต 9600 Hz สามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรหารความถี่ได้สัญลักษณ์ (Symbol) ดังรูปที่ 4.3

รูปที่ 4.3 แสดงสัญลักษณ์ของวงจรหารความถี่บอดเรต

4.1.2 ส่วนของการรับบิทข้อมูลจากพอร์ตออนุกรม (SERIAL_RX)

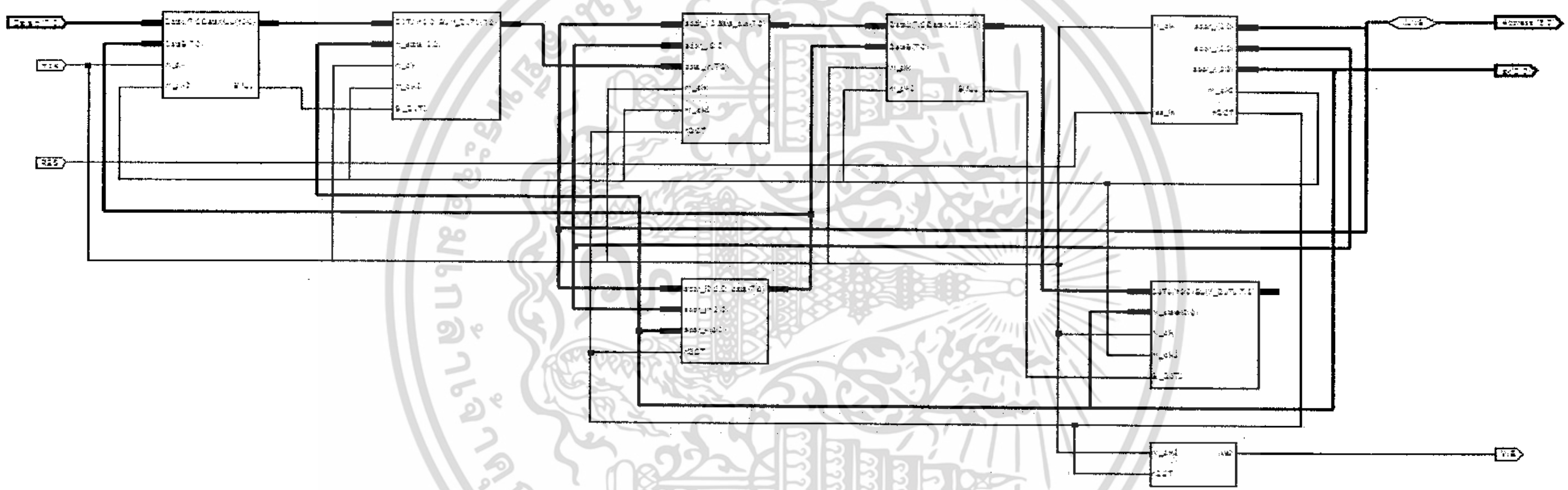
ส่วนของการรับข้อมูลจากพอร์ตอนุกรม ทำหน้าที่รับเฟรมบิทของข้อมูลจากพอร์ตอนุกรมมาทำการแปลงข้อมูลจากบิทอนุกรมเป็นบิทขนาน โดยจะทำการแซมปลิ้ง (Sampling) ค่าอินพุทที่รับเข้ามาจากพอร์ตอนุกรมโดยบิทหนึ่งทำการแซมปลิ้ง 10 ค่า จากนั้นนำค่าที่ได้จากการแซมปลิ้งมาเก็บไว้เพื่อหาค่าความเป็นไปได้ของแต่ละบิทหนึ่งทำการแซมปลิ้ง 10 ค่า จากนั้นนำค่าที่ได้จากการแซมปลิ้งมาเก็บไว้เพื่อหาค่าความเป็นไปได้ของแต่ละบิทอินพุท เพื่อป้องกันการรับข้อมูลจากพอร์ตอนุกรมผิดพลาด เมื่อรับข้อมูลครบ 8 ค่า ก็จะส่งต่อไปยังวงจรเก็บค่าข้อมูลและเรียกค่าข้อมูล (RAM) ของส่วนการแปลงดิสครีตโคซายน์ สามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรการรับบิทข้อมูลจากพอร์ตอนุกรมได้สัญลักษณ์ (Symbol) ดังรูปที่ 4.4



รูปที่ 4.4 แสดงสัญลักษณ์ของวงจรการรับบิทข้อมูลจากพอร์ตอนุกรม

4.2 ส่วนของการแปลงดิสครีตโคซายน์ทรานสฟอร์ม

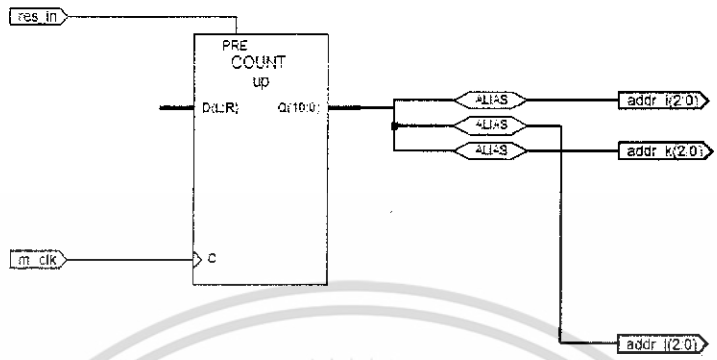
การแยกการแปลงดิสครีตโคซายน์แบบ 2 มิติ ให้มาเป็นการแปลงดิสครีตโคซายน์แบบ 1 มิติ จำนวน 2 ครั้ง ในการคำนวณค่าการแปลงดีสครีตโคซายน์แบบ 1 มิติของข้อมูลที่รับข้อมูลอินพุทมาจากส่วนของการรับบิทข้อมูลจากพอร์ตออนุกรม (SERIAL_RX) โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรการแปลงดิสครีตโคซายน์ทรานสฟอร์มทั้งหมดได้ ดังรูปที่ 4.5



รูปที่ 4.5 แสดงสัญลักษณ์ของวงจรการแปลงดิสครีตโคซายน์ทรานสฟอร์ม

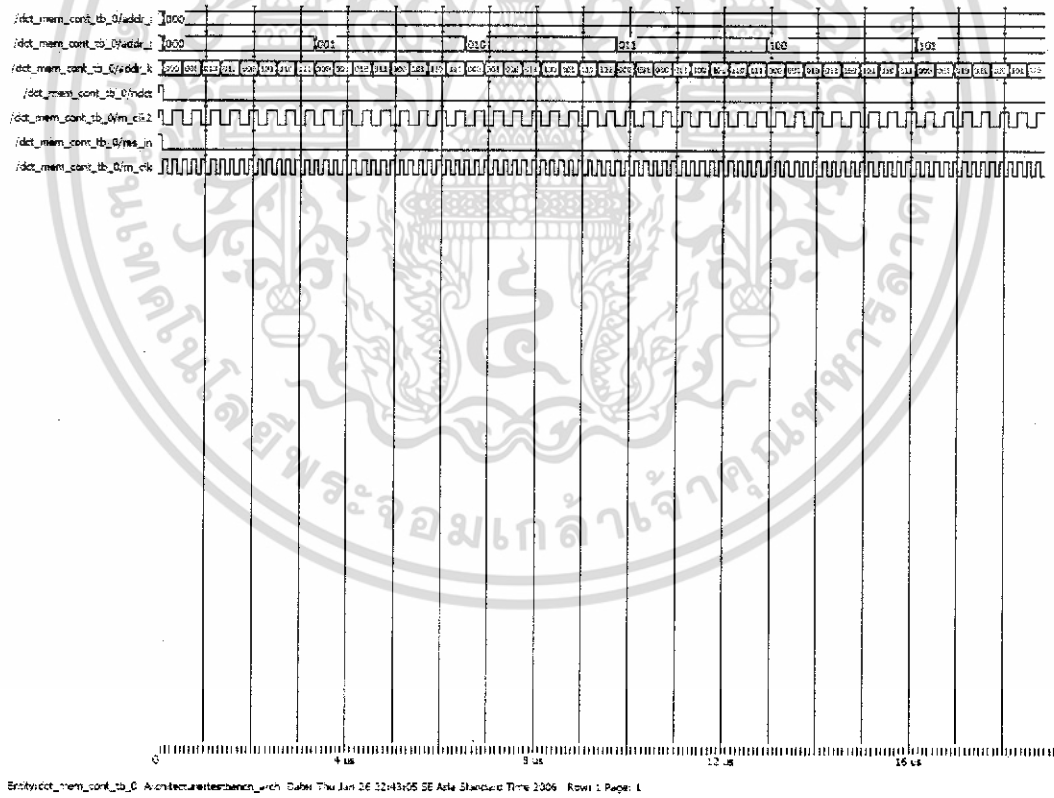
4.2.1 ส่วนของวงจรควบคุมการเก็บค่าข้อมูลและเรียกค่าข้อมูล

ทำหน้าที่สร้างสัญญาณควบคุมการการเก็บค่าข้อมูลและเรียกค่าข้อมูล ให้มีความถูกต้อง โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรควบคุมเก็บค่าข้อมูลและเรียกค่าข้อมูลได้ สัญลักษณ์ ดังรูปที่ 4.6



รูปที่ 4.6 แสดงสัญลักษณ์ของวงจรควบคุมการเก็บค่าข้อมูลและเรียกค่าข้อมูล

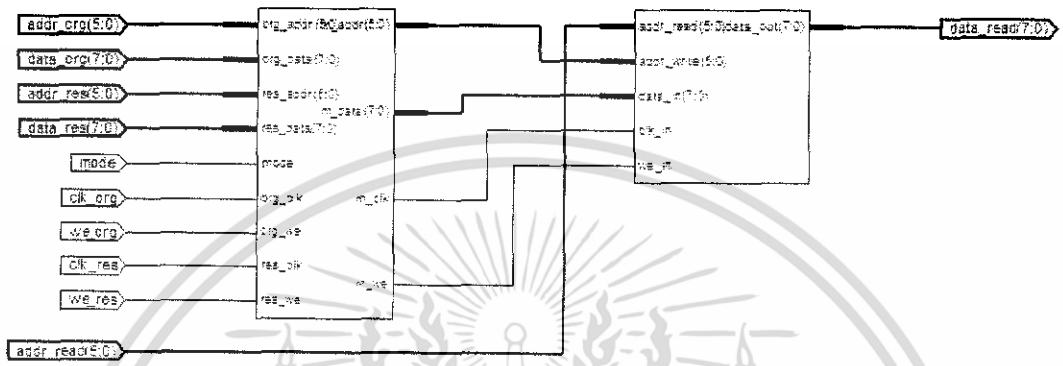
จากวงจรควบคุมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังรูปที่ 4.7



รูปที่ 4.7 แสดงผลการจำลองการทำงานของวงจรควบคุมการเก็บค่าข้อมูลและเรียกค่าข้อมูล

4.2.2 ส่วนของวงจรเก็บค่าข้อมูลและเรียกค่าข้อมูล (RAM)

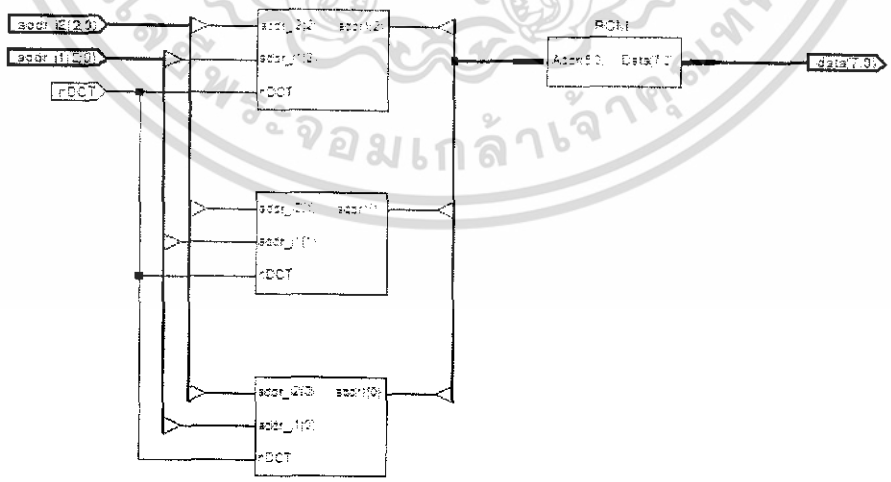
ของวงจรเก็บค่าสัมประสิทธิ์การแปลงดิสครีตโคซายน์เป็นส่วนที่ทำหน้าที่ในการเก็บข้อมูลอินพุททั้ง 64 ค่า ที่รับมาจากพอร์ตอนุกรม มาเก็บในแรมแล้วจะทำการเรียกข้อมูลไปยังส่วนถัดไป โดยอาศัยการควบคุมการทำงานจากวงจรควบคุม โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรเก็บค่าข้อมูลและเรียกค่าข้อมูลได้สัญลักษณ์ ดังรูปที่ 4.8



รูปที่ 4.8 แสดงสัญลักษณ์ของวงจรเก็บค่าข้อมูลและเรียกค่าข้อมูล (RAM)

4.2.3 ส่วนของวงจรเก็บค่าสัมประสิทธิ์การแปลงดิสครีตโคซายน์

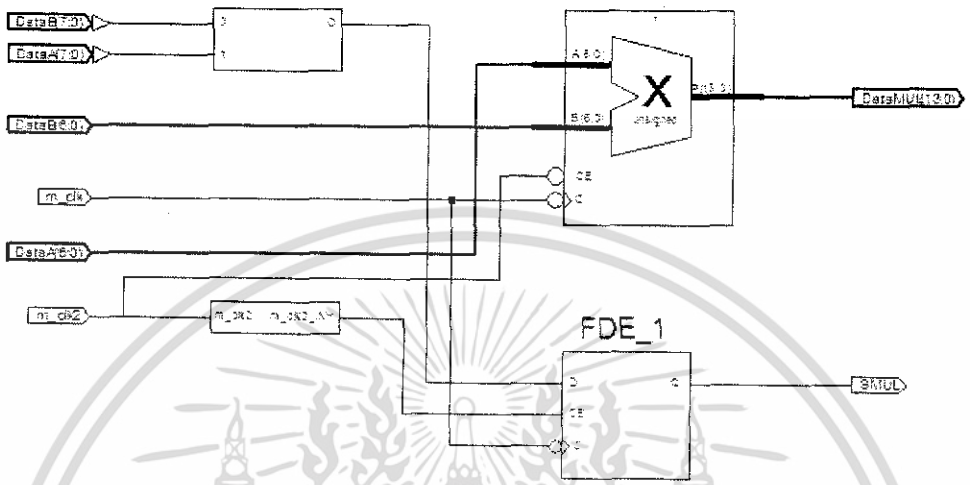
เป็นวงจรที่ ทำการเก็บค่าสัมประสิทธิ์การแปลงดิสครีตโคซายน์ และส่งข้อมูลออกตามอินพุทที่ชี้ตำแหน่งข้อมูลที่ต้องการ สามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรเก็บค่าสัมประสิทธิ์การแปลงดิสครีตโคซายน์ได้สัญลักษณ์ (Symbol) ดังรูปที่ 4.9



รูปที่ 4.9 แสดงสัญลักษณ์ของวงจรเก็บค่าสัมประสิทธิ์การแปลงดิสครีตโคซายน์

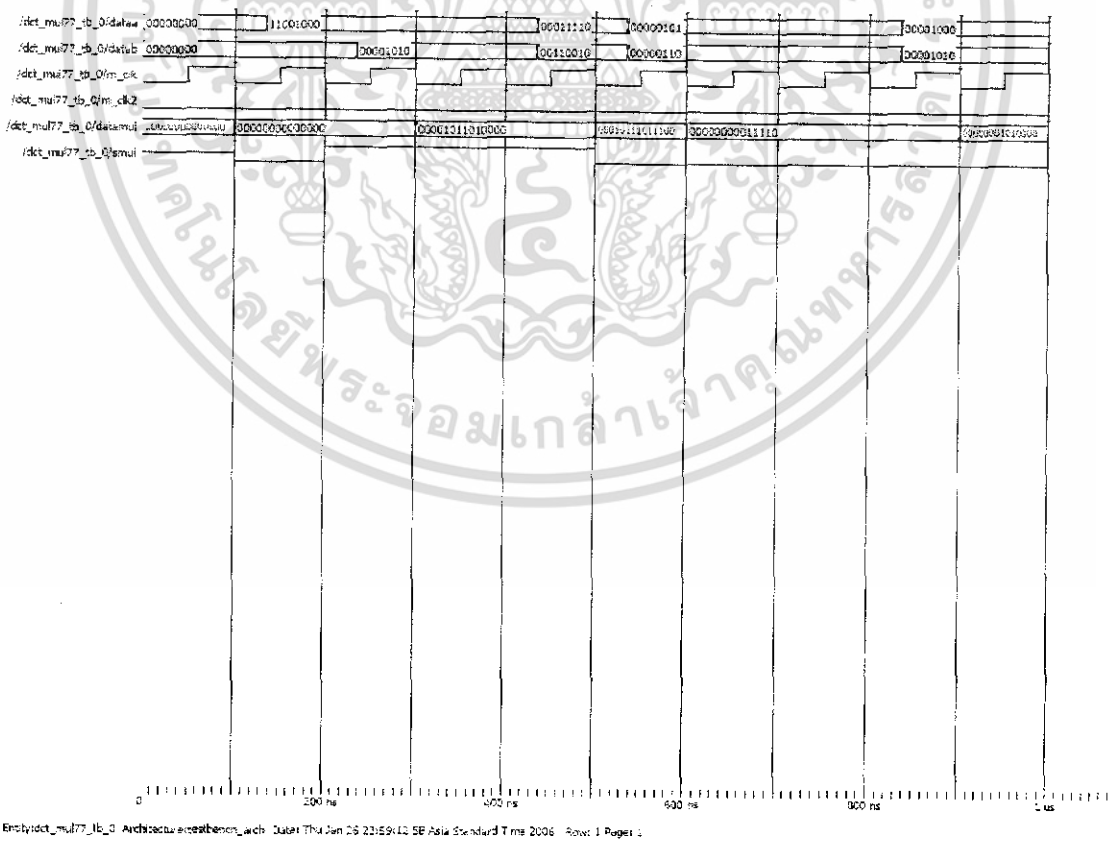
4.2.4 ส่วนของ MULTIPLIER 7 bits

เป็นวงจรที่อยู่ในส่วนของการแปลงดิสครีตโคซายน์แบบ 1 มิติ ทำหน้าที่ในการคูณข้อมูลอินพุตขนาด 7 บิท กับค่าคงที่สัมประสิทธิ์การแปลงดิสครีตโคซายน์ 7 บิท จะได้ผลลัพธ์เป็นข้อมูลขนาด 14 บิท สามารถเขียนโปรแกรมที่สังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ดังรูปที่ 4.10



รูปที่ 4.10 แสดงสัญลักษณ์ของ MULTIPLIER 7 bits

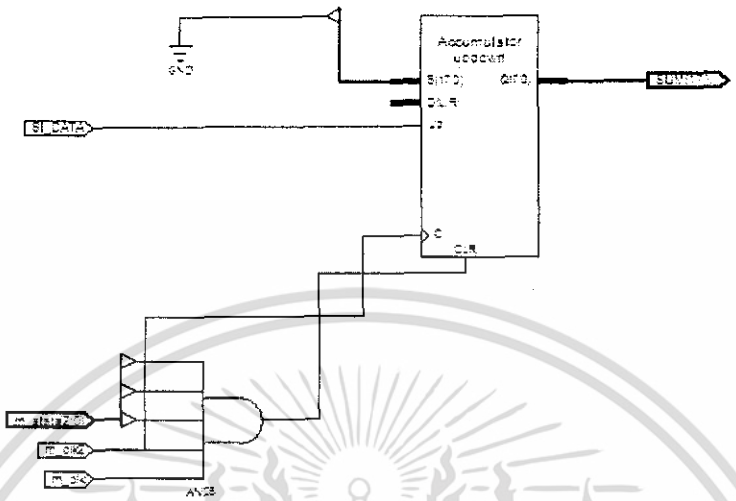
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังรูปที่ 4.11



รูปที่ 4.11 แสดงผลการจำลองการทำงานของวงจร MULTIPLIER 7 bits

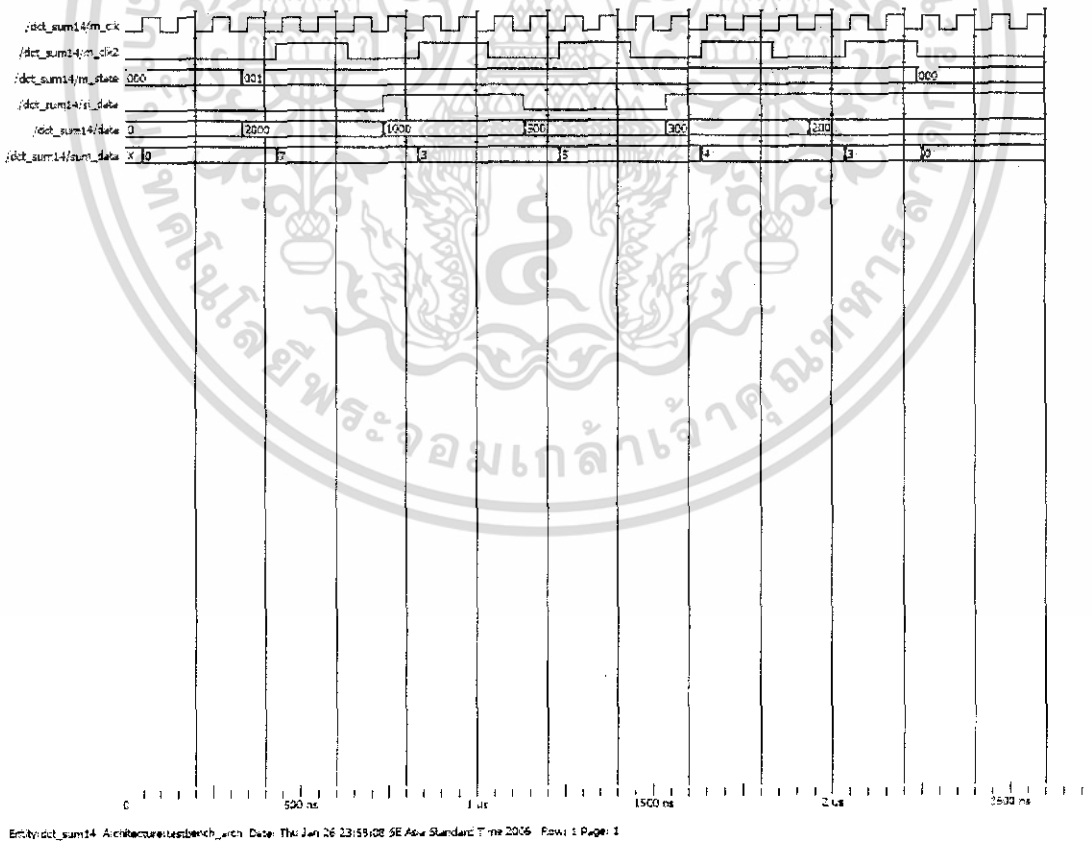
4.2.5 ส่วนของวงจรบวกและลบขนาด 14 บิท

เป็นส่วนที่อยู่ภายในวงจรการแปลงดิสครีตโคซายน์แบบ 1 มิติ ทำหน้าที่บวกและลบข้อมูลขนาด 14 บิท ซึ่งผลจากการบวกและลบข้อมูลนี้จะได้ผลลัพธ์เป็นข้อมูลขนาด 14 บิท สามารถเขียนโปรแกรมที่สังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ดังรูปที่ 4.12



รูปที่ 4.12 แสดงสัญลักษณ์ของวงจรบวกและลบขนาด 14 บิท

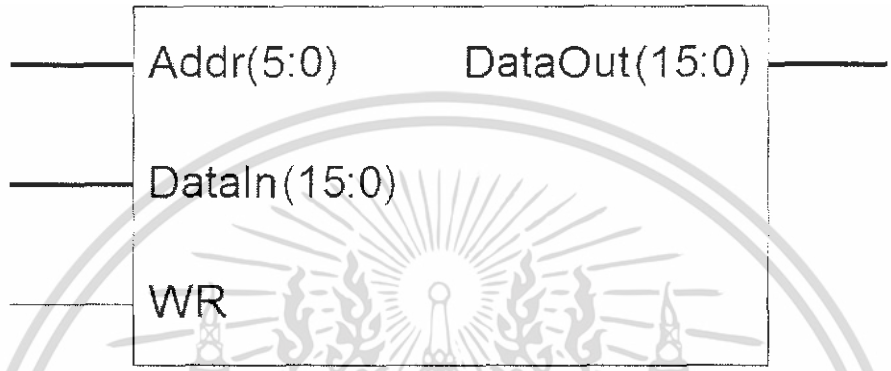
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังรูปที่ 4.13



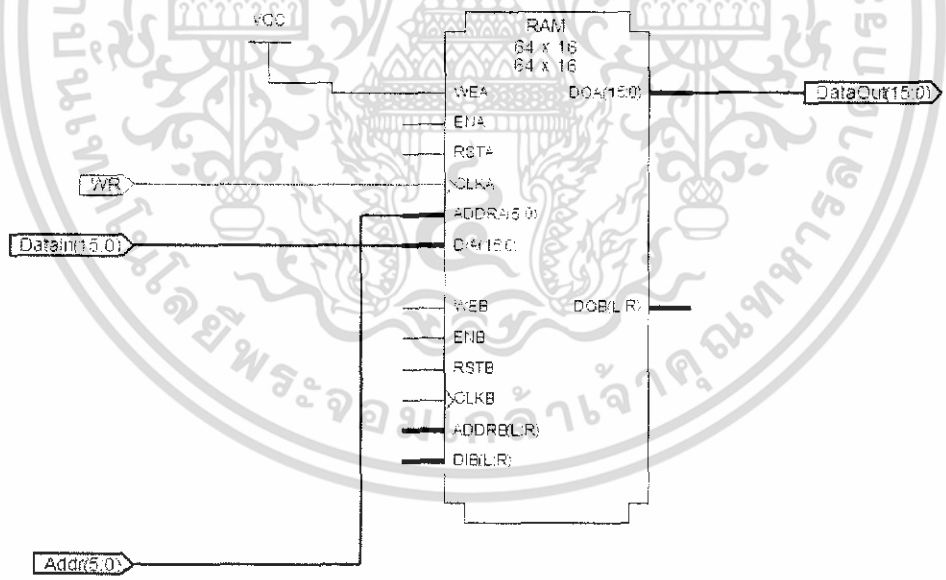
รูปที่ 4.13 แสดงผลการจำลองการทำงานของวงจรบวกและลบขนาด 14 บิท

4.2.6 ส่วนของวงจรเก็บพักข้อมูล (RAM)

เป็นส่วนที่ทำหน้าที่ในการเก็บข้อมูลอินพุททั้ง 64 ค่า ซึ่งในการทดลองนี้จะใช้ 2 ตัวด้วยกัน ตัวแรกจะใช้สำหรับเก็บข้อมูลที่รับมาจากส่วนอินพุทพอร์ต และส่วนการแปลงดิสครีตโคซายน์ทรานสฟอร์มเสร็จแล้ว ส่วนที่สองจะใช้สำหรับพักค่าข้อมูลที่ทำการแปลงดิสครีตโคซายน์ทรานสฟอร์มครั้งแรก (Buffer) มาเก็บในแรมแล้วจะทำการเรียกข้อมูลไปยังส่วนถัดไป โดยอาศัยการควบคุมการทำงานจากวงจรควบคุม โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมของวงจรเก็บพักข้อมูลได้สัญลักษณ์ ดังรูปที่ 4.14



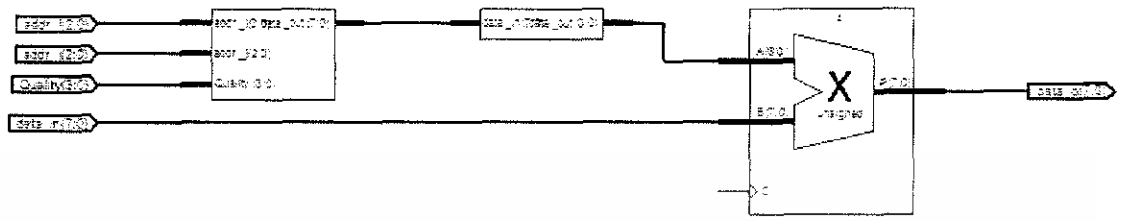
รูปที่ 4.14 แสดงสัญลักษณ์ของวงจรพักค่าข้อมูล



รูปที่ 4.15 แสดงลักษณะของวงจรพักค่าข้อมูล

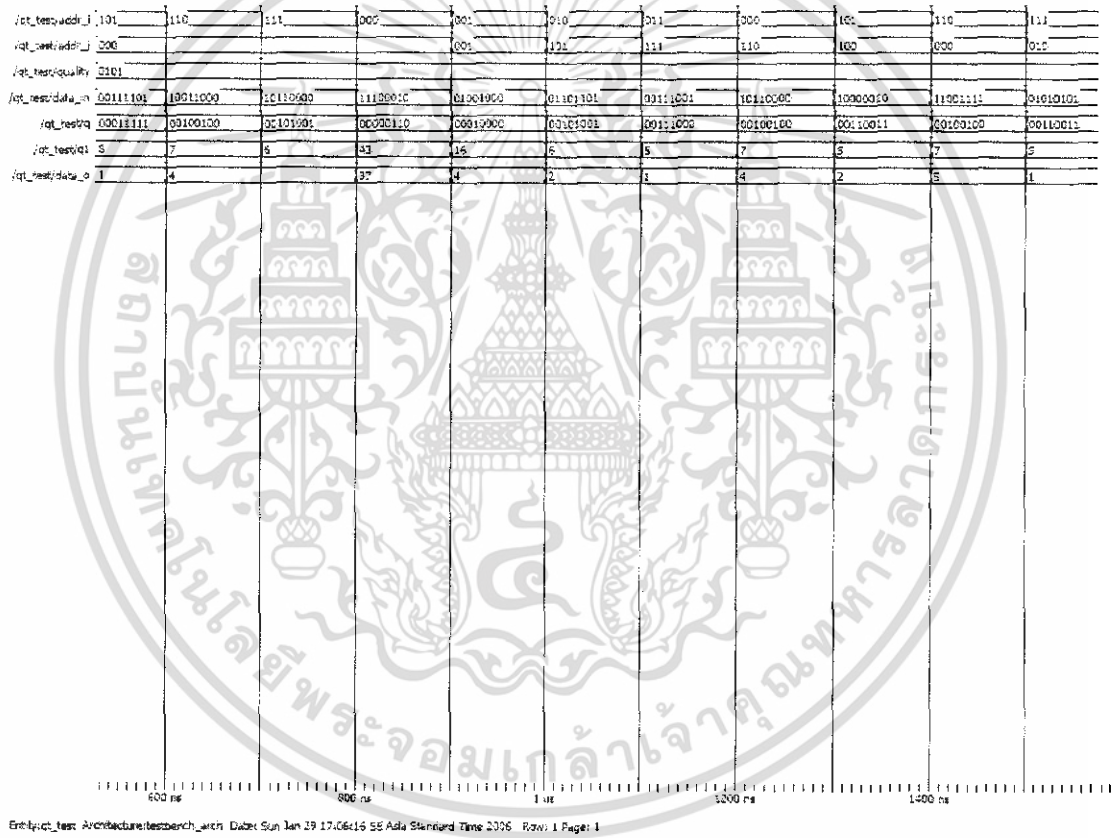
4.3 ส่วนของการควอนไตซ์

โครงสร้างจะแบ่งออกเป็น 2 วงจร ย่อยทำหน้าที่ปรับระดับสัญญาณเพื่อลดค่าของข้อมูลภายในบล็อกเพื่อจะลดจำนวนบิทที่ใช้ในการเก็บข้อมูลเหล่านี้ในขั้นตอนเข้ารหัสต่อไป โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมได้สัญลักษณ์ ดังรูปที่ 4.16



รูปที่ 4.16 แสดงสัญลักษณ์ของวงจรการควอนไตซ์

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังรูปที่ 4.17

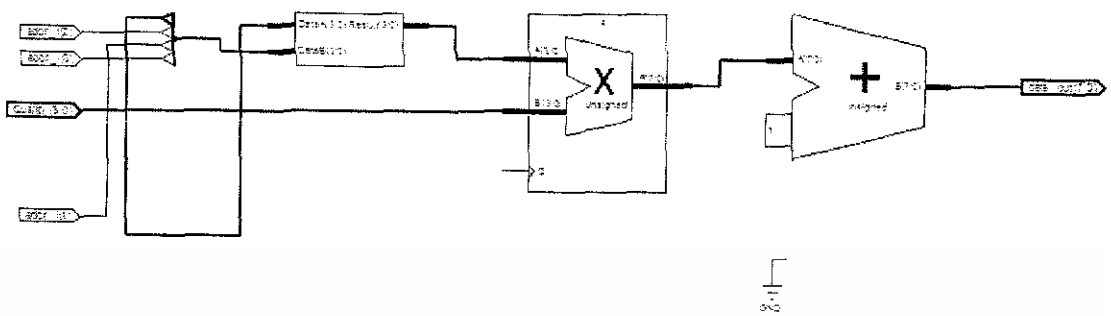


รูปที่ 4.17แสดงผลการจำลองการทำงานของวงจรการควอนไตซ์

4.3.1 ส่วน QT_FACTOR

เป็นส่วนที่ทำหน้าที่สร้างตาราง Quality Factor ซึ่งสามารถเลือกค่า Q ได้ 16 ค่าโดย

การปรับเลือกที่ dip Switch ส่งให้กับส่วน math_div เพื่อเก็บค่าต่อไป โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมได้สัญลักษณ์ ดังรูปที่ 4.18



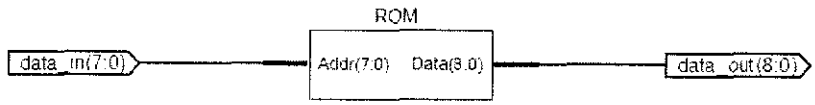
รูปที่ 4.18 แสดงสัญลักษณ์ของวงจรสร้างตาราง Quality Factor

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังรูปที่ 4.19

รูปที่ 4.19 แสดงผลการจำลองการทำงานของวงจรสร้างตาราง Quality Factor

4.3.2ส่วนของวงจรเก็บค่าสำหรับเทียบ (main_div)

ทำหน้าที่เปรียบเสมือนรอม เก็บค่าไว้เพื่อนำไปเทียบกับค่าอินพุท เพื่อหลีกเลี่ยงการหาร โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมได้สัญลักษณ์ ดังรูปที่ 4.19

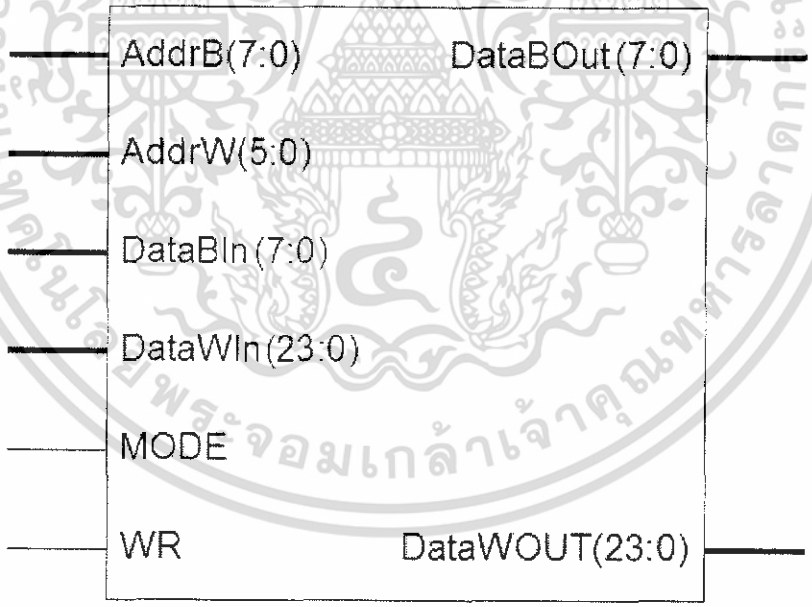


รูปที่ **4.20** แสดงสัญลักษณ์ของวงจรเก็บค่าสำหรับเทียบ

4.4 ส่วนของการเข้ารหัสโดยใช้ Huffman Code

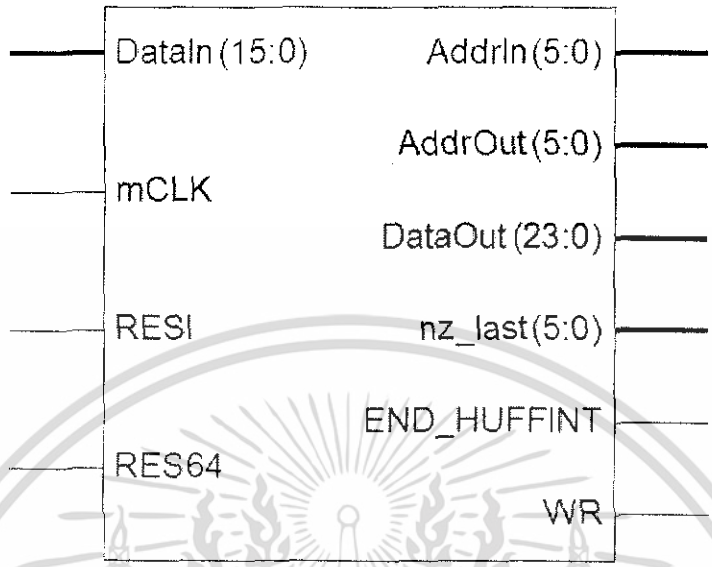
4.4.1 ส่วนของวงจรสร้างแอดเดรสและเก็บค่าข้อมูลแบบซิกแซกเพื่อเตรียมเข้ารหัส

ส่วนนี้จะประกอบด้วยวงจรสร้างแอดเดรสให้กับข้อมูล โดยกำหนดค่าแอดเดรสให้เป็นแบบซิกแซก แล้วนำค่าไปพักในแรมเพื่อเตรียมเข้ารหัส



รูปที่ **4.21** แสดงสัญลักษณ์ของวงจรพักค่าข้อมูล

นำข้อมูลที่ได้มาเทียบแยกส่วนที่เป็น DC และ AC ออกจากกันแล้วนำไปเก็บในเพื่อนำไปเข้ารหัส โดยสามารถสังเคราะห์อุปกรณ์จากโปรแกรมได้สัญลักษณ์ ดังรูปที่ 4.22



รูปที่ 4.22 แสดงสัญลักษณ์ของวงจรเข้ารหัส Huffman Code

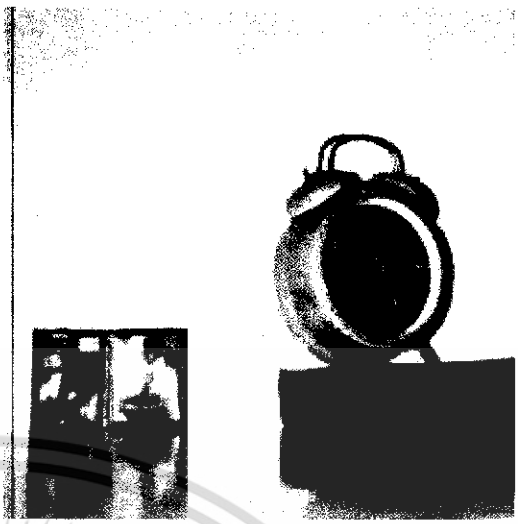
4.5 ผลการทดสอบการบีบอัดข้อมูลภาพแบบ JPEG

การทดลองจะใช้โปรแกรมบีบอัดข้อมูลภาพบิทแมพชนิด 8 บิทต่อพิกเซล จะได้ผลการทดสอบดังตารางที่ 4.1

ตารางที่ 4.1 แสดงการทดสอบการบีบอัดข้อมูลภาพแบบ BMP ด้วยวิธี JPEG

In-File	Input (byte)	Output (byte)	Quality factor	Compression ratio
Clock.bmp	512,000	12,899	5	39.69%
	512,000	11,548	10	44.33%
	512,000	10,877	15	47.07%
Lena.bmp	2,000,000	46,435	5	43.07%
	2,000,000	41,358	10	48.35%
	2,000,000	39,478	15	50.66%

a). ภาพต้นแบบ Clock ขนาด256 ×256



b). ภาพที่ได้จากการแปลงกลับหลังถูกบีบอัดโดยใช้ค่า Q.F. เท่ากับ 5

c). ภาพที่ได้จากการแปลงกลับหลังถูกบีบอัดโดยใช้ค่า Q.F. เท่ากับ 10

d). ภาพที่ได้จากการแปลงกลับหลังถูกบีบอัดโดยใช้ค่า Q.F. เท่ากับ 15

รูปที่ 4.23 แสดงผลการทดลองบีบอัดข้อมูลภาพต้นแบบ Clock ด้วยค่า Q.F. ต่างๆ

a). ภาพต้นแบบ Lena ขนาด 512×512

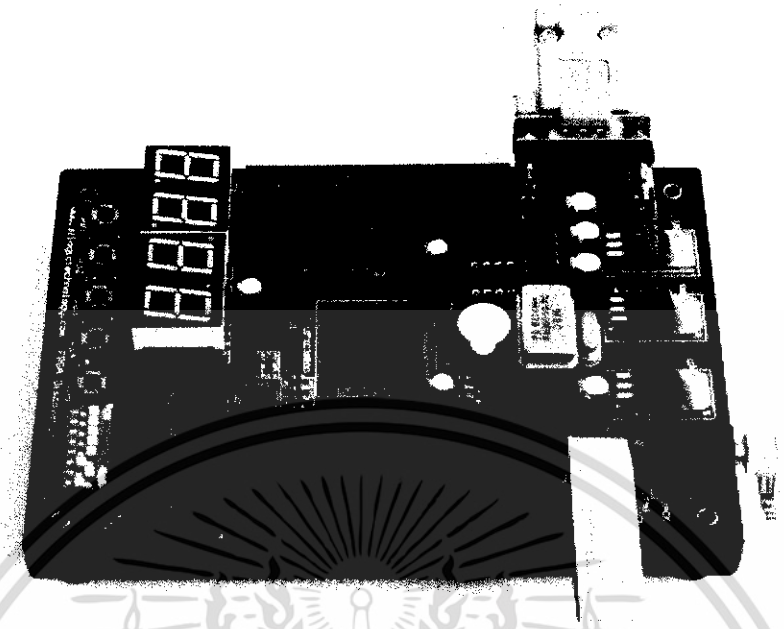
b). ภาพที่ได้จากการแปลงกลับหลังถูก บีบอัดโดยใช้ค่า Q.F. เท่ากับ 5

c). ภาพที่ได้จากการแปลงกลับหลังถูกบีบอัดโดยใช้ค่า Q.F. เท่ากับ 10

d). ภาพที่ได้จากการแปลงกลับหลังถูกบีบอัดโดยใช้ค่า Q.F. เท่ากับ 15

รูปที่ 4.24 แสดงผลการทดลองบีบอัดข้อมูลภาพต้นแบบ Lena ด้วยค่า Q.F. ต่างๆ

4.6 ภาพอุปกรณ์ที่ใช้ในการทดสอบบีบอัดข้อมูลภาพแบบ JPEG



รูปที่ 4.25 แสดงอุปกรณ์ที่ใช้ในการทดสอบการบีบอัดภาพด้วยวิธีการของ JPEG

รูปที่ 4.26 แสดงบอร์ด FPGA ที่ทำการเชื่อมต่อกับพอร์ตอนุกรมขณะทำการทดสอบ

บทที่ 5
บทวิจารณ์และบทสรุป

จากการทดลองศึกษาวิธีการบีบอัดภาพแบบ JPEG บน FPGA โดยใช้ภาพเกรย์สเกลขนาด 8 บิท ทำการออกแบบให้ขนาดของวงจรมีขนาดเล็ก เพื่อนำไปประยุกต์ใช้งานร่วมกับวงจรการจัดเก็บข้อมูลหรือวงจรสื่อสารอื่นๆ ที่ต้องใช้การบีบอัดขนาดของข้อมูลก่อนทำการจัดเก็บข้อมูล หรือวงจรสื่อสารอื่นๆ ที่ต้องใช้การบีบอัดขนาดของข้อมูลก่อนทำการจัดเก็บข้อมูลหรือการส่ง ในลักษณะงานที่ไม่ต้องการความเร็วในการประมวลผลสูงมาก ซึ่งในการทดลองจะใช้ภาพแบบ Bitmap file ซึ่งไม่มีการบีบอัดข้อมูลมาทำการบีบอัด

ผลการทดลองในบทที่ 4 จะแบ่งเป็น 2 ส่วน ส่วนแรกเป็นผลการออกแบบจำลองผลการทำงานและสังเคราะห์วงจร ซึ่งแบ่งออกเป็นส่วนของวงจรการแปลงดิสครีตโคซายน์ เพื่อแยกส่วนของค่า DC (DC Component) ซึ่งเป็นส่วนที่บอกระดับความสว่าง และมีค่าสัมประสิทธิ์สูงสุด และส่วนพลังงาน AC (AC Component) ซึ่งเป็นส่วนที่บอกระดับความคมชัด จากนั้นจึงส่งให้กับวงจรส่วนควอนไตซ์ เพื่อปัดระดับสัมประสิทธิ์ส่วน AC Component ให้มีค่าเป็นศูนย์มากขึ้น เพื่อสะดวกในการเข้ารหัส จากนั้นจึงส่งต่อให้กับส่วนของวงจร Huffman Coding เพื่อทำการเข้ารหัสส่วน DC Component แบบ DPCM และเข้ารหัสส่วน AC Component แบบ Run Length Coding

ส่วนที่ 2 เป็นผลของภาพที่ได้จากการบีบอัดภาพแบบ JPEG บน FPGA ซึ่งพิจารณาจากอัตราส่วน (Compression Ratio) และคุณภาพของภาพที่ได้กับค่า Q.F. จะเห็นได้ว่า ค่า Q.F.ที่สูงขึ้นในการบีบอัดจะทำให้อัตราส่วนการบีบอัดข้อมูลภาพที่ได้สูงขึ้นด้วย แต่คุณภาพของภาพที่ได้จะแย่ลงตามลำดับ

หนังสืออ้างอิง

[1] Khalid Sayood, "Introduction to Data Compression", Morgan Kaufmann Publishers, Inc., San Francisco, Californai, 1996

[2] Rafael C. Gonzales and Richard E. Woods and Steven L.Eddins, "Digital Image Processing USING MATLAB", Pearson Prentice Hall, Upper Saddle River, New Jersey, 2004

[3] Yun Q. Shi and Huifang Sun, "Image and Video Compression for Multimedia Engineering", CRC Press LIC, Boca Raton, 2000

[4] กฤช เรืองฤทธิ์ และกฤติกานต์ ยมภักดี และจักร แซ่เอา,"อุปกรณ์การแปลงข้อมูลแบบดิสครีตโคซายน์ทรานสฟอร์ม 2 มิติ ด้วยอุปกรณ์ FPGA", ปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมโทรคมนาคม, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2547

[5] คมศักดิ์ หาดขุนทด, "การออกแบบและการสร้างวงจรบีบอัดข้อมูลตัวอักษรและภาพโดยใช้ FPGA", วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2546

[6] วรลักษณ์ คงเด่นฟ้า และวาสนา กล้าการนา, "การบีบอัดภาพนิ่งด้วยวิธี JPEG ชนิด Sequential Baseline System", วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต, สาขาวิชาวิศวกรรมโทรคมนาคม, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2540

[7] สายน้ำฝน หอมจันทร์, "การบีบอัดข้อมูลแบบมีการสูญเสียต่ำชนิดความซับซ้อนต่ำ", วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2545

ภาคผนวก

ภาคผนวก ก.

##

โปรแกรมที่ใช้ในการทดสอบ

##

โปรแกรมประกอบด้วยโปรแกรมหลัก 2 โปรแกรม คือ โปรแกรมการแปลง DCT กับ Quantization และโปรแกรมการเข้ารหัส Huffman

##

1. โปรแกรมการแปลง DCT กับ Quantization

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DCT1 is
    Port ( AddrIn       : out  STD_LOGIC_VECTOR (5 downto 0);
-- Address For Data In
           AddrOut      : out  STD_LOGIC_VECTOR (5 downto 0);
-- Address For Data Out
           DataIn       : in  STD_LOGIC_VECTOR (15 downto 0);
-- Data In
           DataOut      : out  STD_LOGIC_VECTOR (15 downto 0);
-- Data Out
           WR           : out  STD_LOGIC;
-- Write For Data Out
           EN           : in  STD_LOGIC;
-- Enable Module
           END_DCT      : out STD_LOGIC;
-- Module End is '1'
           MODE         : in std_logic;
-- MODE '0' IS 1DCT , '1' IS 2DCT

           Quality      : in STD_LOGIC_VECTOR (3 downto 0);
-- input for Quantitize control

           mCLK         : in  STD_LOGIC);
-- Main Clock Process
end DCT1;

architecture RTL of DCT1 is
  type rom8x8 is array(0 to 63) of std_logic_vector(7 downto 0);
  type rom1divx is array(0 to 255) of std_logic_vector(10 downto 0);

-- CT Constant bit7 is sign , bit0 - bit6 is data is complmemt
  constant cT_rom : rom8x8 := (
"01011010","01111101","01110110","01101010","01011010","01000111","00
110000","00011000",
"01011010","01101010","00110000","10011000","11011010","11111101","11
110110","11000111",
"01011010","01000111","10110000","11111101","11011010","00011000","01
110110","01101010",
"01011010","00011000","11110110","11000111","01011010","01101010","10
110000","11111101",
```

```
"01011010","10011000","11110110","01000111","01011010","11101010","10
110000","01111101",
"01011010","11000111","10110000","01111101","11011010","10011000","01
110110","11101010",
"01011010","11101010","00110000","00011000","11011010","01111101","11
110110","01000111",
"01011010","11111101","01110110","11101010","01011010","11000111","00
110000","10011000"
  );

-- (1 / x) *1024
  constant divx : romldivx := (
"10000000000","10000000000","01000000000","00101010101","00100000000"
,"00110011001",
"00010101011","00010010010","00010000000","00001110010","00001100110"
,"00001011101",
"00001010101","00001001111","00001001001","00001000100","00001000000"
,"00000111100",
"00000111001","00000110110","00000110011","00000110001","00000101111"
,"00000101101",
"00000101011","00000101001","00000100111","00000100110","00000100101"
,"00000100011",
"00000100010","00000100001","00000100000","00000011111","00000011110"
,"00000011101",
"00000011100","00000011100","00000011011","00000011010","00000011010"
,"00000011001",
"00000011000","00000011000","00000010111","00000010111","00000010110"
,"00000010110",
"00000010101","00000010101","00000010100","00000010100","00000010100"
,"00000010011",
"00000010011","00000010011","00000010010","00000010010","00000010010"
,"00000010001",
"00000010001","00000010001","00000010001","00000010000","00000010000"
,"00000010000",
"00000010000","00000001111","00000001111","00000001111","00000001111"
,"00000001110",
"00000001110","00000001110","00000001110","00000001110","00000001101"
,"00000001101",
"00000001101","00000001101","00000001101","00000001101","00000001100"
,"00000001100",
"00000001100","00000001100","00000001100","00000001100","00000001100"
,"00000001100",
"00000001011","00000001011","00000001011","00000001011","00000001011"
,"00000001011",
"00000001011","00000001011","00000001010","00000001010","00000001010"
,"00000001010",
"00000001010","00000001010","00000001010","00000001010","00000001010"
,"00000001010",
"00000001001","00000001001","00000001001","00000001001","00000001001"
,"00000001001",
"00000001001","00000001001","00000001001","00000001001","00000001001"
,"00000001001",
"00000001001","00000001000","00000001000","00000001000","00000001000"
,"00000001000",
"00000001000","00000001000","00000001000","00000001000","00000001000"
,"00000001000",
"00000001000","00000001000","00000001000","00000001000","00000001000"
,"00000000111",
"00000000111","00000000111","00000000111","00000000111","00000000111"
,"00000000111",
```

```
"000000000111","000000000111","000000000111","000000000111","000000000111"
,"000000000111",
"000000000111","000000000111","000000000111","000000000111","000000000111"
,"000000000111",
"000000000111","000000000111","000000000110","000000000110","000000000110"
,"000000000110",
"000000000110","000000000110","000000000110","000000000110","000000000110"
,"000000000110",
"000000000110","000000000110","000000000110","000000000110","000000000110"
,"000000000110",
"000000000110","000000000110","000000000110","000000000110","000000000110"
,"000000000110",
"000000000110","000000000110","000000000110","000000000110","000000000110"
,"000000000110",
"000000000110","000000000101","000000000101","000000000101","000000000101"
,"000000000101",
"000000000101","000000000101","000000000101","000000000101","000000000101"
,"000000000101",
"000000000101","000000000101","000000000101","000000000101","000000000101"
,"000000000101",
"000000000101","000000000101","000000000101","000000000101","000000000101"
,"000000000101",
"000000000101","000000000101","000000000101","000000000101","000000000101"
,"000000000101",
"000000000101","000000000101","000000000101","000000000101","000000000101"
,"000000000101",
"000000000100","000000000100","000000000100","000000000100","000000000100"
,"000000000100",
"000000000100","000000000100","000000000100","000000000100","000000000100"
,"000000000100",
"000000000100","000000000100","000000000100","000000000100","000000000100"
,"000000000100",
"000000000100","000000000100","000000000100","000000000100");
  signal ijk            : std_logic_vector(8 downto 0) := "000000000";
--for address loop

  signal DataCT8bit     : std_logic_vector(7 downto 0) := "00000000";
-- Constant table data

  -- signal for MUL+SUM
  signal MUL_DATA24bit  : STD_LOGIC_VECTOR (23 downto 0);
--after MUL adj. bit
  signal SUM_DATA24bit  : STD_LOGIC_VECTOR (23 downto 0);

  signal DataDCT16bit   : std_logic_vector(15 downto 0);
  signal DataQ16bit     : std_logic_vector(15 downto 0);
begin
  --------- MAIN DCT GENARATE ADDRESS LOOP COUNT ijk ---------------
  --------- mCLK '0' TO '1' FOR WRITE TO MEMORY --------------------
  process(EN,mCLK)
  begin
    if EN = '0' then     -- PROCESS DISABLE
      ijk       <= "000000000";
      END_DCT   <= '0';
    else
      if mCLK'event and mCLK = '1' then
        if ijk < "111111111" then
```

```
          ijk <= ijk + '1';
          END_DCT   <= '0';
        else
          ijk <= "111111111";
          END_DCT   <= '1';
        end if;
      end if;
    end if;
  end process;

  process(EN,mCLK,ijk)
  begin
    if EN = '0' then     -- PROCESS DISABLE
      WR    <= '0';
    else
      if mCLK'event and mCLK = '1' then       --write to memory at
xxx xxx 111
        WR <= ijk(0) and ijk(1) and ijk(2);
      end if;
    end if;
  end process;

  --------- PROCESS SUM AND LATCH ADDRESS FOR WRITE OUTPUT ----------
  process(EN,mCLK)
    variable AddrO : std_logic_vector(5 downto 0) := "000000";
  begin
    if EN = '0' then
      AddrO := "000000";
    else
      if mCLK'event and mCLK = '0' then
        AddrO(5 downto 3) := ijk(8 downto 6);   -- i
        AddrO(2 downto 0) := ijk(5 downto 3);   -- j
      end if;
    end if;
    AddrOut   <= AddrO;
  end process;

  ------------ PROCESS GEN ORG.DATA AND GET cT from Table "cT_rom"
  process(mode,ijk)
    variable AddrCT     : STD_LOGIC_VECTOR (5 downto 0):= "000000";
-- Address for C Table
  begin
    if (mode = '0') then          --mode 0 is 1DCT
      AddrIn(5 downto 3) <= ijk(8 downto 6);    -- i
      AddrIn(2 downto 0) <= ijk(2 downto 0);    -- k
      AddrCT(5 downto 3) := ijk(2 downto 0);    -- k
      AddrCT(2 downto 0) := ijk(5 downto 3);    -- j
    else                          -- mode 1 is 2DCT
      AddrIn(5 downto 3) <= ijk(2 downto 0);    -- k
      AddrIn(2 downto 0) <= ijk(5 downto 3);    -- j
      AddrCT(5 downto 3) := ijk(2 downto 0);    -- k
      AddrCT(2 downto 0) := ijk(8 downto 6);    -- i
    end if;
    DataCT8bit <= cT_rom(Conv_integer(AddrCT));
  end process;


  ------- PROCESS CAL. MUL betwin DataIn and cT
  process(DataIn,DataCT8bit)
    variable DataA : std_logic_vector(6 downto 0);
--Data From Contant Table [cT]
```

```
variable DataB : std_logic_vector(14 downto 0);
--ABS(DataIn)
    variable MULDATA22  : STD_LOGIC_VECTOR (21 downto 0);
--Data After MUL [15bit*7bit
  begin
    --Check - is 2'comprement Data Input
    if DataIn(15) = '1' then
      DataB := not(DataIn(14 Downto 0)) + '1';
    else
      DataB := DataIn(14 Downto 0);
    end if;

    --For Cal
    DataA := DataCT8bit(6 Downto 0);
    MULDATA22 := DataA * DataB;

    MUL_DATA24bit(21 downto 0) <= MULDATA22;
    MUL_DATA24bit(23 downto 22) <= "00";
  end process;

  process(mCLK,ijk,MUL_DATA24bit,DataIn)
    variable MS_RESET   : std_logic;
    variable smul       : std_logic;
  begin
    MS_RESET := not(ijk(0) or ijk(1) or ijk(2));
    smul     := DataCT8bit(7) xor DataIn(15);
-- sign calculator

    if (mCLK'event and mCLK = '0') then    -- Active From '1' to '0'
      if MS_RESET = '1' then       -- NEW SUM
        if smul = '1' then
          SUM_DATA24bit <= not(MUL_DATA24bit(23 Downto 0)) + '1';
--'0' - MUL_DATA24bit;
        else
          SUM_DATA24bit <= MUL_DATA24bit;
        end if;
      else
        if smul = '1' then
          SUM_DATA24bit <= SUM_DATA24bit - MUL_DATA24bit;
        else
          SUM_DATA24bit <= SUM_DATA24bit + MUL_DATA24bit;
        end if;
      end if;
    end if;

    DataDCT16bit <= SUM_DATA24bit(23 downto 8);
  end process;

  process(mode,Quality,DataDCT16bit,DataQ16bit)
  begin
    if (MODE = '0')or(Quality = "0000") then
      DataOut <= DataDCT16bit;                --Quantitize by pass
    else
      DataOut <= DataQ16bit;
    end if;
  end process;
```

```
------------------------------------------------------------------------
---------------------------- Quantitize Process ------------------------
------------------------------------------------------------------------
--Genarate Quantitize Factor
  process (ijk,SUM_DATA24bit,Quality)
    variable ai     : std_logic_vector(3 downto 0);
    variable aj     : std_logic_vector(3 downto 0);
    variable aa     : std_logic_vector(3 downto 0);
    variable factor : std_logic_vector(7 downto 0);
    variable plfactor   : std_logic_vector(10 downto 0);
    variable DCTfQ  : std_logic_vector(14 downto 0);
    variable DQ     : std_logic_vector(25 downto 0);
    variable DQO    : std_logic_vector(15 downto 0);
  begin
    ai(2 downto 0)     := ijk(8 downto 6);
    ai(3)              := '0';
    aj(2 downto 0)     := ijk(5 downto 3);
    aj(3)              := '0';
    aa                 := (ai + aj + '1');
    factor             := aa * Quality + '1';

    --2'complete SUM_DATA if "-"
    if SUM_DATA24bit(23) = '1' then
      DCTfQ := not(SUM_DATA24bit(22 Downto 8)) + '1';
    else
      DCTfQ := SUM_DATA24bit(22 Downto 8);
    end if;

    plfactor := divx(conv_integer(factor));

    --DataQ16bit(7 downto 0) <= plfactor(10 downto 3);
    --DataQ16bit(15 downto 8)  <= "00000000";

    DQ := plfactor * DCTfQ;

    DQO(14 downto 0) := DQ(24 downto 10);
    DQO(15)          := '0';

    if SUM_DATA24bit(23) = '1' then
      DataQ16bit <= not(DQO) + '1';
    else
      DataQ16bit <= DQO;
    end if;
  end process;
end RTL;
```

2. โปรแกรมการเข้ารหัส Huffman

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity HuffmanInt is
    Port ( AddrIn       : out  STD_LOGIC_VECTOR (5 downto 0);
           DataIn       : in  STD_LOGIC_VECTOR (15 downto 0); --Data
DCT and Quat.
           mCLK         : in  STD_LOGIC;
           RES64        : in STD_LOGIC;
           RESI         : in std_logic;
           AddrOut      : out  STD_LOGIC_VECTOR (5 downto 0);
           DataOut      : out std_logic_vector (23 downto 0);     --
4 bit : RL , 4 bit : Category , 16 bit : Amplitude
           WR           : out  STD_LOGIC;
           END_HUFFINT  : out std_logic;
           nz_last      : out std_logic_vector(5 downto 0)
           );
end HuffmanInt;

architecture RTL of HuffmanInt is
-- for zigzag Address Map
type type_addr6 is array(0 to 63) of std_logic_vector(5 downto 0);
constant zza_rom : type_addr6 := (
"000000","000001","001000","010000","001001","000010","000011","00101
0",
"010001","011000","100000","011001","010010","001011","000100","00010
1",
"001100","010011","011010","100001","101000","110000","101001","10001
0",
"011011","010100","001101","000110","000111","001110","010101","01110
0",
"100011","101010","110001","111000","111001","110010","101011","10010
0",
"011101","010110","001111","010111","011110","100101","101100","11001
1",
"111010","111011","110100","101101","100110","011111","100111","10111
0",
"110101","111100","111101","110110","101111","110111","111110","11111
1"
  );

  signal AddrCount : std_logic_vector (5 downto 0);     -- Address
count for Read Data
  signal last00 : std_logic_vector(15 downto 0);         -- last
@address 0
  signal RunLength : STD_LOGIC_VECTOR (3 downto 0);     -- ZERO COUNT
  signal Category : STD_LOGIC_VECTOR (3 downto 0);      -- Category
of Data
  signal DataIn2com : std_logic_vector(15 downto 0);
  signal WRD : std_logic;
  signal iend :std_logic;
  signal DCOut : std_logic_vector(15 downto 0);
  signal ao : std_logic_vector(5 downto 0);
  signal zero_count : std_logic_vector(3 downto 0);
  signal wr_en : std_logic;
  signal mCLK2 : std_logic;
```

```
  signal mCLK2p :std_logic;
  signal nzl : std_logic_vector(5 downto 0);
begin
  process(mCLK,RES64,RESI,iend)
  begin
    if (RES64 = '1' or RESI = '1' or iend = '1') then
      mCLK2 <= '1';
    else
      if mCLK'event and mCLK = '1' then
        mCLK2 <= not mCLK2;
      end if;
    end if;
  end process;
  ---------------- process for AddrIn and END_HUFFINT output --------
  process(mCLK2,RES64,RESI)
  begin
    if RES64 = '1' or RESI = '1' then
      AddrCount <= "000000";
      iend <= '0';
    else
      if mCLK2'event and mCLK2 = '1' then
        if AddrCount = "111111" then
          AddrCount <= AddrCount;
          iend <= '1';
        else
          AddrCount <= AddrCount + '1';
          iend <= '0';
        end if;
      end if;
    end if;
  end process;
  AddrIn <= zza_rom(Conv_integer(AddrCount));
  END_HUFFINT <= iend;

  ------------------- process for WR output -----------------------
  process(mCLK,mCLK2,RES64,RESI,AddrCount,wr_en,iend)
  begin
    if (RES64 = '1' or RESI = '1' or iend = '1') then
      WRD <= '0';
    else
      if (mCLK'event and mCLK = '0') then
        if mCLK2 = '1' then
          WRD <= '0';
        else
          if (AddrCount = "000000")or(AddrCount = "111111") then
            WRD <= '1';
          else
            WRD <= wr_en;
          end if;
        end if;
      end if;
    end if;
  end process;
  WR <= WRD;

  --------------------- process for DataOut -------------------------
  DataOut(7 downto 4)   <= RunLength;
  DataOut(3 downto 0)   <= Category;
```

```
----------------------------process check not zero last ------------
process(mCLK2p,ao,DataIn,AddrCount)
begin
    if (mCLK2p'event and mCLK2p = '1') then
        if (DataIn /= "0000000000000000") then
          nzl <= ao;
        else
          nzl <= nzl;
        end if;
    end if;
end process;
nz_last <= nzl;

------------------------- process for Addr Out --------------------
process(RES64,WRD,ao,AddrCount)
begin
  if (RES64 = '1') then
    ao <= "000000";
  else
    if (AddrCount = "000000") then
      ao <= ao;
    else
      if WRD'event and WRD = '0' then
        ao <= ao + '1';
      end if;
    end if;
  end if;
end process;
AddrOut <= ao;

------------------------ process for DC ----------------------------
process (mCLK2,RES64,RESI,AddrCount,DataIn)
begin
  if RESI = '1' then            -- reset all
    last00 <= "0000000000000000";
  else
    if RES64 = '1' then         -- reset with out last00 register
      last00 <= last00;
    else
      if mCLK2'event and mCLK2 = '0' then
          if AddrCount = "000000" then
            last00 <= DataIn;
          else
            last00 <= "0000000000000000";
          end if;
      end if;
    end if;
  end if;
end process;
DCOut <= DataIn - last00;

-------------------- mCLK/2 + tmCLK/2 -----------------------------
process(mCLK,mCLK2)
begin
  if mCLK'event and mCLK = '0' then
    if mCLK2 = '0' then
      mCLK2p  <= '1';
    else
      mCLK2p  <= '0';
    end if;
  end if;
```

```
end process;

------------------------ Run Length Generator --------------------
process(mCLK2,RES64,RESI,iend)
begin
  if (RES64 = '1' or RESI = '1' or iend = '1') then
    zero_count <= "0000";
  else
    if mCLK2'event and mCLK2 = '0' then
        if DataIn = "0000000000000000" then
          zero_count <= zero_count + '1';
        else
          zero_count <= "0000";
        end if;
    end if;
  end if;
end process;

process (mCLK2p,AddrCount,zero_count)
begin
  if AddrCount = "1111111" then
    RunLength <= zero_count;
    wr_en <= '1';
  else
    if mCLK2p'event and mCLK2p = '0' then
      RunLength <= zero_count;
      if zero_count = "1111" or DataIn /= "0000000000000000" then
        wr_en <= '1';
      else
        wr_en <= '0';
      end if;
    end if;
  end if;
end process;

------------------- 2'completment with out sign --------------
process(DataIn)
begin
  if DataIn(15)= '1' then
    DataIn2com <= not(DataIn) + 1;
  else
    DataIn2com <= DataIn;
  end if;
end process;

------------------- Category Table ----------------------
process(DataIn2com)
begin
  if DataIn2com = "0000000000000000" then Category <= "0000";
  else if DataIn2com < "0000000000000010" then Category <= "0001";
  else if DataIn2com < "0000000000000100" then Category <= "0010";
  else if DataIn2Com < "0000000000001000" then Category <= "0011";
  else if DataIn2com < "0000000000010000" then Category <= "0100";
  else if DataIn2com < "0000000000100000" then Category <= "0101";
  else if DataIn2com < "0000000001000000" then Category <= "0110";
  else if DataIn2com < "0000000010000000" then Category <= "0111";
  else if DataIn2com < "0000000100000000" then Category <= "1000";
  else if DataIn2com < "0000001000000000" then Category <= "1001";
  else if DataIn2com < "0000010000000000" then Category <= "1010";
  else if DataIn2com < "0000100000000000" then Category <= "1011";
  else if DataIn2com < "0001000000000000" then Category <= "1100";
```

```
    else if DataIn2com < "0010000000000000" then Category <= "1101";
    else if DataIn2com < "0100000000000000" then Category <= "1110";
    else if DataIn2com < "1000000000000000" then Category <= "1111";
    else Category <= "1000";
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
    end if;
  end process;

  ------------------ Amplitude -----------------------------
  process(DataIn,Category)
       variable base_and      : std_logic_vector(15 downto 0);
   variable data_and  : std_logic_vector(15 downto 0);
   variable Amplitude : STD_LOGIC_VECTOR (15 downto 0);     --
Amplitude of Category
  begin
    case Category is
          when "0001"    => base_and := "0000000000000001";
      when "0010" => base_and := "0000000000000011";
      when "0011" => base_and := "0000000000000111";
      when "0100" => base_and := "0000000000001111";
      when "0101" => base_and := "0000000000011111";
      when "0110" => base_and := "0000000000111111";
      when "0111" => base_and := "0000000001111111";
      when "1000" => base_and := "0000000011111111";
      when "1001" => base_and := "0000000111111111";
      when "1010" => base_and := "0000001111111111";
      when "1011" => base_and := "0000011111111111";
      when "1100" => base_and := "0000111111111111";
      when "1101" => base_and := "0001111111111111";
      when "1110" => base_and := "0011111111111111";
      when "1111" => base_and := "0111111111111111";
              when others => base_and := "0000000000000000";
        end case;
    data_and := DataIn and base_and;

    if DataIn(15) = '1' then          --sign
      Amplitude := Data_and - '1';
    else
          Amplitude := data_and;
    end if;

    --Amplitude := DataIn;

    DataOut(23 downto 8)       <= Amplitude;
  end process;

end RTL;
```

ภาคผนวก ข.

Table 6.16 JPEG Default AC Code (Luminance)

Run/ Category	Base Code	Length	Run/ Category	Base Code	Length
0/0	1010 (= EOB)	4			
0/1	00	3	8/1	11111010	9
0/2	01	4	8/2	1111111111000000	17
0/3	100	6	8/3	1111111110110111	19
0/4	1011	8	8/4	1111111110111000	20
0/5	11010	10	8/5	1111111110111001	21
0/6	111000	12	8/6	1111111110111010	22
0/7	1111000	14	8/7	1111111110111011	23
0/8	1111110110	18	8/8	1111111110111100	24
0/9	1111111110000010	25	8/9	1111111110111101	25
0/A	1111111110000011	26	8/A	1111111110111110	26
1/1	1100	5	9/1	111111000	10
1/2	111001	8	9/2	1111111110111111	18
1/3	1111001	10	9/3	1111111111000000	19
1/4	1111110110	13	9/4	1111111111000001	20
1/5	1111111110110	16	9/5	1111111111000010	21
1/6	1111111110000100	22	9/6	1111111111000011	22
1/7	1111111110000101	23	9/7	1111111111000100	23
1/8	1111111110000110	24	9/8	1111111111000101	24
1/9	1111111110000111	25	9/9	1111111111000110	25
1/A	1111111110001000	26	9/A	1111111111000111	26
2/1	11011	6	A/1	111111001	10
2/2	11111000	10	A/2	1111111111001000	18
2/3	1111110111	13	A/3	1111111111001001	19
2/4	1111111110001001	20	A/4	1111111111001010	20
2/5	1111111110001010	21	A/5	1111111111001011	21
2/6	1111111110001011	22	A/6	1111111111001100	22
2/7	1111111110001100	23	A/7	1111111111001101	23
2/8	1111111110001101	24	A/8	1111111111001110	24
2/9	1111111110001110	25	A/9	1111111111001111	25
2/A	1111111110001111	26	A/A	1111111111010000	26
3/1	111010	7	B/1	111111010	10
3/2	111110111	11	B/2	1111111111010001	18
3/3	1111111110111	14	B/3	1111111111010010	19
3/4	1111111110010000	20	B/4	1111111111010011	20
3/5	1111111110010001	21	B/5	1111111111010100	21
3/6	1111111110010010	22	B/6	1111111111010101	22
3/7	1111111110010011	23	B/7	1111111111010110	23
3/8	1111111110010100	24	B/8	1111111111010111	24
3/9	1111111110010101	25	B/9	1111111111011000	25
3/A	1111111110010110	26	B/A	1111111111011001	26

Table 6.16 Continued

Run/ Category	Base Code	Length	Run/ Category	Base Code	Length
4/1	111011	6	C/1	1111111010	11
4/2	1111111000	10	C/2	1111111111011010	16
4/3	1111111100010111	19	C/3	1111111111011011	16
4/4	1111111100011000	20	C/4	1111111111011100	16
4/5	1111111100011001	21	C/5	1111111111011101	16
4/6	1111111100011010	22	C/6	1111111111011110	16
4/7	1111111100011011	23	C/7	1111111111011111	16
4/8	1111111100011100	24	C/8	1111111111100000	16
4/9	1111111100011101	25	C/9	1111111111100001	16
4/A	1111111100011110	26	C/A	1111111111100010	16
5/1	1111010	8	D/1	1111111010	13
5/2	1111111001	12	D/2	1111111111100011	16
5/3	1111111100011111	19	D/3	1111111111100100	16
5/4	1111111101000000	28	D/4	1111111111100101	16
5/5	1111111101000001	21	D/5	1111111111100110	16
5/6	1111111101000010	22	D/6	1111111111100111	16
5/7	1111111101000011	23	D/7	1111111111101000	16
5/8	1111111101000100	24	D/8	1111111111101001	16
5/9	1111111101000101	25	D/9	1111111111101010	16
5/A	1111111101000110	26	D/A	1111111111101011	16
6/1	1111011	8	E/1	1111111110110	15
6/2	1111111000	13	E/2	1111111111101100	16
6/3	1111011101000111	19	E/3	1111111111101101	16
6/4	1111011101010000	20	E/4	1111111111101110	16
6/5	1111011101010001	21	E/5	1111111111101111	16
6/6	1111011101010010	22	E/6	1111111111110000	16
6/7	1111011101010011	23	E/7	1111111111110001	16
6/8	1111011101010100	24	E/8	1111111111110010	16
6/9	1111011101010101	25	E/9	1111111111110011	16
6/A	1111011101010110	26	E/A	1111111111110100	16
7/1	1111001	9	F/0	11111111110111	14
7/2	1111011001	15	F/1	1111111111110101	16
7/3	1111011101010111	19	F/2	1111111111110110	16
7/4	1111011101011000	20	F/3	1111111111110111	16
7/5	1111011101011001	21	F/4	1111111111111000	16
7/6	1111011101011010	22	F/5	1111111111111001	16
7/7	1111011101011011	23	F/6	1111111111111010	16
7/8	1111011101011100	24	F/7	1111111111111011	16
7/9	1111011101011101	25	F/8	1111111111111100	16
7/A	1111011101011110	26	F/9	1111111111111101	16
			F/A	1111111111111110	16