สถาปัตยกรรมการเข้ารหัสเลขคณิตสำหรับระบบการบีบอัด

ARITHMETIC CODING ARCHITECTURE FOR COMPRESSION SYSTEM

โดย

นางสาวปรารถนา ติวะวงศ์

นายเปรมินทร์ กมลประเสริฐสุข

นายสุภชัย จินต์จันทรวงศ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

สถาปัตยกรรมการเข้ารหัสเลขคณิตสำหรับระบบการบีบอัด

ARITHMETIC CODING ARCHITECTURE FOR COMPRESSION SYSTEM

โดย

นางสาวปรารถนา ติวะวงศ์ 46010424

นายเปรมินทร์ กมลประเสริฐสุข 46010468

นายสุภชัย จินต์จันทรวงศ์ 46010858

อาจารย์ที่ปรึกษา

ผศ. อัครพล ตรีรัตน์

อ. ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **สถาปัตยกรรมการเข้ารหัสเลขคณิตสำหรับระบบการบีบอัด**

ARITHMETIC CODING ARCHITECTURE FOR COMPRESSION SYSTEM

ผู้จัดทำ

1. นางสาวปรารถนา ติวะวงศ์ 46010424
2. นายเปรมินทร์ กมลประเสริฐสุข 46010468
3. นายสุภชัย จินต์จันทรวงศ์ 46010858

..... อาจารย์ที่ปรึกษา

(ผศ. อัครพล ตรีรัตน์)

..... อาจารย์ที่ปรึกษา

(อ.ศรวัฒน์ ชิวปรีชา)

สถาปัตยกรรมการเข้ารหัสเลขคณิตสำหรับระบบการบีบอัด
ARITHMETIC CODING ARCHITECTURE FOR COMPRESSION SYSTEM

โดย นางสาวปรารถนา ติวะวงศ์ 46010424
นายเปรมินทร์ กมลประเสริฐสุข 46010468
นายสุภชัย จินต์จันทรวงศ์ 46010858

อาจารย์ที่ปรึกษา ผศ.อัครพล ตรีรัตน์
อ.ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

ปริญญานิพนธ์นี้ได้ศึกษาเกี่ยวกับการเข้ารหัสเลขคณิต (Arithmetic coding) ซึ่งนำไปใช้ในการบีบอัดข้อมูล นอกจากนี้ยังได้สร้างเป็นฮาร์ดแวร์เพื่อให้สามารถนำไปใช้งานได้จริงในระบบการบีบอัดวีดีโออีกด้วย

ABSTRACT

In this thesis, we propose an implementation of binary arithmetic coder.Arithmetic coding allows a significant enhancement in compression.In this thesis, we also show the need for a hardware implementation of alithmetic coding in current video compression systems.

กิตติกรรมประกาศ

รายงานฉบับนี้สำเร็จลุล่วงได้ด้วยโดยได้รับความช่วยเหลือและชี้แนะจากหลายท่าน ผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษา ผศ.อัครพล ตรีรัตน์ และ อ.ศรวัฒน์ ชิวปรีชา ที่ให้คำปรึกษาและให้ความช่วยเหลือด้านข้อมูล , และอุปกรณ์ในการทำปริญญานิพนธ์เป็นอย่างดี ขอขอบพระคุณ ผศ.ดร.อรฉัตร จิตต์ไสภักตร์ ที่ได้บรรยายวิธีการบีบอัดข้อมูล (DATA COMPRESSTION) แบบต่างๆ ขอขอบคุณนายอัศนัย นิธิไรจนานนท์, นายอรรถพล วีระนพนันท์, นายสุรชัย แก้วศรีนาค ที่ช่วยตรวจสอบ CODE ในส่วนที่มีปัญหา

ผู้เขียนระลึกอยู่เสมอว่าหากไม่ได้รับความช่วงเหลือจากบุคคลที่กล่าวมานั้น รายงานฉบับนี้ก็ไม่สามารถที่จะสำเร็จลุล่วงไปด้วยดี จึงขอขอบพระคุณมา ณ ที่นี้

นางสาวปรารถนา ติวะวงศ์
นายเปรมินทร์ กมลประเสริฐสุข
นายสุภชัย จินต์จันทรวงศ์

สารบัญ

สารบัญรูป

หน้า

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การบีบอัดข้อมูลมีความสำคัญอย่างยิ่ง ไม่ว่าจะเป็นการบีบอัดข้อมูลเพื่อจะลดพื้นที่ในการจัดเก็บข้อมูล หรือในการส่งข้อมูลที่จะทำได้รวดเร็วมากขึ้น ซึ่งวิธีการที่นำมาใช้คือ การเข้ารหัสฮัฟแมนและการเข้ารหัสเลขคณิตซึ่งจะทำลายข้อจำกัดที่ว่าสัญลักษณ์ที่ถูกเข้ารหัสจะต้องแทนด้วยทุกจำนวนบิตนั้นจะนำไปสู่การเข้ารหัสที่มีประสิทธิภาพและอัตราการบีบอัดข้อมูลที่ดีขึ้น และสามารถใช้ได้กับข้อมูลตัวอักษร รูปภาพ หรือแม้แต่ข้อมูลมัลติมีเดีย

การเข้ารหัสเลขคณิตไม่ได้มีความซับซ้อนมากนักแต่ก็ไม่ได้เป็นที่รู้จักในก่อนปลายยุค 70's ในรูปแบบที่ใช้กันในปัจจุบัน และได้รับความสนใจมากขึ้นในยุค 80's เนื่องจากประสิทธิภาพที่สูงและวิธีการแปลงเป็นฮาร์ดแวร์นั้นทำได้โดยตรง ผู้แรกที่ได้พูดถึงเรื่องนี้คือ Abramson และ Elias ในปี 1960 แต่ก็ยังไม่มีวิธีการแก้ปัญหาที่ถูกต้อง เพราะความแม่นยำในการเข้ารหัสเลขคณิตจะเพิ่มขึ้นเมื่อความยาวของข้อมูลเพิ่มขึ้น ในปี 1976 Pasco และ Rissanen ได้พิสูจน์ว่าความยาวที่จำกัดก็เพียงพอสำหรับการเข้ารหัสโดยไม่สูญเสียความแม่นยำในการเข้ารหัส แต่วิธีนี้ก็ยังมีประสิทธิภาพทางหน่วยความจำที่ยังไม่ดีนัก จากนั้นในปี 1979 และ 1980 Rubin,Guazzo,Rissanen และLangdon ได้ตีพิมพ์วิธีการเข้ารหัสพื้นฐานเหมือนกับที่ใช้อยู่ในปัจจุบัน ที่มีความแม่นยำจำกัดโดยนำกลไกFIFOมาใช้ การนำมาใช้โดย Rissanen และ Langdon ก็ใกล้เคียงกับการแปลงมาใช้กับฮาร์ดแวร์ในภายหลัง

วิธีการบีบอัดข้อมูลที่ใช้คือ การเข้ารหัสเลขคณิต ซึ่งรับข้อมูลเข้ามาทั้งชุดและแปลงเป็นสัญลักษณ์ได้อย่างชาญฉลาด แม้ว่าจะรับข้อมูลเข้ามาหมดทั้งชุดก็สามารถสร้างออกมาได้เป็นโค้ดเดียวซึ่งกระทำในคุณสมบัติแบบเป็นลำดับคือทำเป็นสัญลักษณ์ต่อสัญลักษณ์ การเข้ารหัสเลขคณิตนี้ทำงานเวลาแบบเชิงเส้นโดยใช้หน่วยความจำที่คงที่ และยังมีคุณสมบัติที่สามารถปรับให้เหมาะสมกับการแก้ไขปัญหาทางฮาร์ดแวร์ด้วย นอกจากนี้จากการถอดรหัสยังมีแนวทางที่คล้ายคลึงกับวิธีการเข้ารหัสด้วย

การออกแบบทำได้โดยใช้ภาษาวีเอชดีแอล (VHDL) จากนั้นสร้างเป็นฮาร์แวร์ด้วยวงจรลอจิกรวมที่สามารถโปรแกรมได้ (FPGA) และทดสอบความสามารถในการบีบอัดข้อมูลว่ามีประสิทธิภาพเพียงใด

1.2 วัตถุประสงค์

1. เพื่อศึกษาประสิทธิภาพของการบีบอัดข้อมูลตัวอักษร ด้วยวิธีการเข้ารหัสเลขคณิตว่ามีอัตราการลดขนาดข้อมูลได้ดีเพียงใด

2. เพื่อศึกษาการออกแบบ แบบจำลองการเข้ารหัสเลขคณิตด้วยภาษาแมทแลบ (MATLAB) เพื่อนำผลที่ได้ไปเปรียบเทียบความถูกต้องกับการออกแบบด้วยภาษาวีเอชดีแอล และใช้เป็นแนวคิดในการออกแบบด้วยภาษาวีเอชดีแอลด้วย

3. เพื่อศึกษาการออกแบบฮาร์ดแวร์ด้วยภาษาวีเอชดีแอล และศึกษาถึงวิธีการตรวจสอบสัญญาณและทำการสังเคราะห์เป็นวงจรระดับเกต

4. เพื่อให้สามารถนำไปประยุกต์ใช้ร่วมกับฮาร์ดแวร์และทำงานได้อย่างมีประสิทธิภาพ

1.3 ขอบเขตของการพัฒนา

ปริญญานิพนธ์นี้จะทำการบีบอัดข้อมูลโดยใช้วิธีการเข้ารหัสเลขคณิต ที่ประยุกต์ให้พัฒนาเป็นฮาร์ดแวร์ได้เป็นการเข้ารหัสเลขคณิตแบบใช้เลขจำนวนเต็ม และจำลองการทำงานผ่านโปรแกรม Xillinx-Project Navigator ซึ่งออกแบบกระบวนการการทำงานโดยใช้ภาษาวีเอชดีแอล เพื่อจำลองผลการทดลองและทำลงบอร์ด FPGA เพื่อทำเป็นฮาร์ดแวร์ และทดสอบความถูกต้องโดยเปรียบเทียบผลจากการจำลองการทำงานจากโปรแกรมภาษาแมทแลบ

1.4 เนื้อหาขั้นตอนการดำเนินงาน

บทที่ 2 อธิบายทฤษฎีเทคนิคที่ใช้ในการบีบอัดข้อมูล,วิธีการวัดประสิทธิภาพของการบีบอัดข้อมูล,การเข้ารหัสฮัฟแมน,การเข้ารหัสเลขคณิตแบบต่างๆ,ทฤษฎีภาษาวีเอชดีแอล,ทฤษฎีของการนำไปประยุกต์ใช้งานบนอุปกรณ์เอฟพีจีเอ

บทที่ 3 เป็นการคำนวณและการสร้าง อธิบายการทำงานการจำลองผลจากโปรแกรมแมทแลบและการออกแบบโดยใช้ภาษาวีเอชดีแอล ของแต่ละส่วนประกอบของการเข้ารหัสและถอดรหัส

บทที่ 4 ผลการทดลอง จะประกอบด้วยผลจากการจำลองผลการเข้ารหัสเลขคณิต จากโปรแกรมแมทแลบ และผลจากการจำลองผลการทำงานด้วยโปรแกรม Xillinx-Project Navigator โดยใช้ภาษาวีเอชดีแอล

บทที่ 5 บทวิจารณ์และบทสรุป

กิตติกรรมประกาศ

หนังสืออ้างอิง

บทที่ 2
ทฤษฎีและหลักการเทคนิคที่ใช้ในการบีบอัดข้อมูล

การบีบอัดข้อมูลเป็นวิธีลดขนาดของข้อมูล โดยข้อมูลอาจอยู่ในรูปของตัวอักษร , ข้อมูลเสียง , ข้อมูลภาพ การบีบอัดข้อมูลมีความสำคัญต่อหลายๆสาขา เช่น การส่งสัญญาณโทรทัศน์แบบดิจิตอล ถ้าต้องการส่งสัญญาณ HDTV (High Definition Television) โดยไม่ทำการบีบอัดข้อมูล จะต้องทำการส่งข้อมูลด้วยความเร็ว 884 เมกะบิท/วินาที แต่ถ้ามีการบีบอัดข้อมูลก่อนส่งสามารถส่งได้น้อยกว่า 20 เมกะบิท/วินาที ข้อมูลเสียงก็มีความจำเป็นที่ต้องบีบอัดข้อมูลเช่นเดียวกัน ในระบบสื่อสารข้อมูลทางคอมพิวเตอร์หลายแบบเช่น โมเด็ม การเก็บข้อมูลในหน่วยความจำสำรองก็จำเป็นต้องใช้การบีบอัดข้อมูลมาช่วยลดขนาดของข้อมูลก่อนทำการส่ง หรือจัดเก็บข้อมูลเช่นเดียวกัน บทนี้จะแสดงให้เห็นรายละเอียดในการบีบอัดข้อมูล เช่น เทคนิคที่ใช้ วิธีการหรืออัลกอริทึมที่ใช้ในการบีบอัดข้อมูล การค้นหาเทคนิคการบีบอัดขนาดที่เหมาะสมมากที่สุดสำหรับข้อมูลทางกายภาพเป็นต้น

2.1 การประยุกต์ใช้งานในการบีบอัดข้อมูล

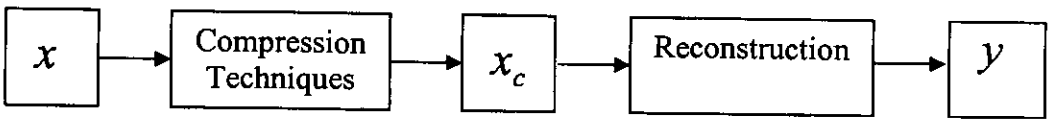
ตัวอย่างแรกของการบีบอัดขนาดข้อมูล คือ รหัสมอส(Morse) ที่พัฒนาโดย Samuel Morse ในกลางศตวรรษที่ 19 จดหมายที่ส่งทางโทรเลขถูกเข้ารหัสด้วย จุด(.)และขีด(-) Morseสังเกตว่าจดหมายจะมีการใช้ตัวอักษรซ้ำๆบ่อย ให้ใช้สัญลักษณ์สั้นเพื่อที่จะลดเวลาเพื่อที่จะลดเวลาเฉลี่ยในการส่งข้อความ เช่น a(. -) และe(.) และถ้าตัวอักษรที่ไม่ค่อยได้ใช้ เช่น q(- - . -) หรือ j(. - - -) จะใช้รหัสที่ยาวกว่าเป็นต้น จะเห็นได้ว่าสิ่งที่ใช้เป็นเครื่องมือในการบีบอัดข้อมูลในรหัสมอสคือ โครงสร้างสถิติของข้อความนั่นเอง ในปัจจุบันมีการใช้การบีบอัดขนาดข้อมูลในการสื่อสารและการจัดเก็บมากมาย เช่น การส่งข้อมูลภาพในระบบอินเตอร์เน็ท การส่งข้อมูลภาพถ่ายในระยะทางไกลในระบบดาวเทียม รวมทั้งงานประยุกต์ต่างๆ เกี่ยวกับการเก็บข้อมูล หรือค่าใช้จ่ายในการสื่อสารจำเป็นต้องลดความซ้ำซ้อน และการจัดเก็บข้อมูลภาพหรือเสียงเพลงในซีดีรอม

นอกจากนี้การบีบอัดข้อมูลจะทำให้ความลับของข้อมูลมีความปลอดภัยมากขึ้น เพราะข้อมูลที่ถูกบีบอัดจะไม่สามารถอ่านรู้เรื่อง

2.2 เทคนิคในการบีบอัดข้อมูล

เทคนิคในการบีบอัดข้อมูลหรืออัลกอริทึมที่ใช้ในการบีบอัดข้อมูล จะประกอบด้วย 2 ขั้นตอน ขั้นตอนแรกได้แก่การนำอินพุท x มาทำการบีบอัดข้อมูลและแทนด้วย x_c ซึ่งจะมีจำนวนบิทที่น้อยกว่าอินพุท x ขั้นตอนที่สองจะนำค่าที่ได้มาทำสร้างใหม่ให้ได้ข้อมูลเดิมหรือใกล้เคียงข้อมูลเดิม y ดังแสดงในรูปที่ 2.1 โดยหลักการในการบีบอัดข้อมูลแบ่งออกเป็น 2 แบบได้แก่

- การบีบอัดข้อมูลแบบไม่สูญเสีย(lossless) ในกรณีนี้ y จะมีค่าเหมือน x ทุกประการ
- การบีบอัดข้อมูลที่มีการสูญเสีย(lossy) ซึ่งวิธีนี้จะมีเทคนิคในการบีบอัดข้อมูลสูงกว่าแบบแรก แต่ค่า y ที่ได้จะมีความแตกต่างจากค่าของ x



รูปที่2.1 กระบวนการบีบอัดข้อมูล

2.2.1 การบีบอัดข้อมูลแบบไม่สูญเสีย

เทคนิคในการบีบอัดข้อมูลในวิธีนี้จะไม่มีการสูญเสียของข้อมูลที่เกิดขึ้น ถ้ามีการบีบอัดข้อมูลขึ้น ข้อมูลต้นฉบับจะสามารถได้คืนมาเหมือนกับต้นฉบับร้อยเปอร์เซนต์ วิธีนี้จะใช้ในการบีบอัดข้อมูลหรือสัญญาณที่ไม่ต่อเนื่อง เช่น ข้อความ และข้อมูลที่คอมพิวเตอร์สร้างขึ้น

2.2.2 การบีบอัดข้อมูลที่มีการสูญเสีย

เทคนิคการบีบอัดข้อมูลแบบนี้ ข้อมูลบางส่วนจะมีการสูญเสียดังนั้นในการคืนสภาพข้อมูลจะไม่สามารถได้ข้อมูลครบถ้วนอย่างต้นฉบับอย่างแน่นอน และวิธีนี้จะมีอัตราการบีบอัดสูงกว่าการบีบอัดแบบที่ไม่มีการสูญเสีย วิธีนี้จะใช้ในการบีบอัดข้อมูลหรือสัญญาณที่ต่อเนื่อง เช่น ข้อมูลเสียง ข้อมูลรูปภาพ

2.2.3 วิธีการวัดประสิทธิภาพ

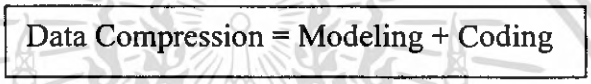
อัลกอริทึมในการบีบอัดข้อมูลมีแตกต่างกันหลายวิธี แต่ละวิธีก็มีข้อดีและข้อเสียแตกต่างกันไป สิ่งที่จะบอกว่าดีหรือไม่นั้น อาจจะดูจากประสิทธิภาพของอัลกอริทึมที่พอจะสามารถวัดได้ เช่น ความสามารถในการนำไปใช้งานจริง ความเร็วของอัลกอริทึมในการประมวลผลเมื่อนำไปสร้างฮาร์ดแวร์หรือซอฟท์แวร์ ความสามารถในการคืนสภาพข้อมูลให้เหมือนต้นฉบับเดิมเป็นต้น มีหลายๆวิธีการที่จะวัดความสามารถของอัลกอริทึมที่ใช้ในการบีบอัดข้อมูล เช่น อัตราส่วนจำนวนบิทก่อนทำการบีบอัดเมื่อเทียบกับจำนวนบิทหลังการบีบอัดที่เรียกว่า " อัตราส่วนในการบีบอัดข้อมูล (Compression ratio) " ตัวอย่างเช่น การที่จัดเก็บข้อมูลภาพที่อยู่ในรูปของอาร์เรย์ ขนาด 256×256 8 บิทต่อพิกเซล(pixel) ซึ่งต้องใช้พื้นที่ในการจัดเก็บ 64 กิโลไบต์ ถ้าทำการบีบอัดข้อมูลภาพนี้และมีขนาด 16 กิโลไบต์แสดงว่าประหยัดพื้นที่ในการจัดเก็บถึง 48 กิโลไบต์ และอัตราส่วนในการบีบอัดข้อมูลเท่ากับ 4

วิธีการอื่นที่จะวัดประสิทธิภาพในการบีบอัดข้อมูล ก็คือค่าเฉลี่ยของจำนวนบิทที่ใช้ต่อหนึ่งตัวอย่าง ซึ่งโดยทั่วไปก็หมายถึงอัตราส่วน ตัวอย่างเช่น ข้อมูลภาพที่กล่าวมาแล้วข้างต้น ค่าเฉลี่ยของจำนวนบิทต่อพิกเซลในการบีบอัดข้อมูลแทนด้วย 2 ดังนั้น อัตราส่วนคือ 2 บิทต่อพิกเซล

ในการบีบอัดข้อมูลแบบสูญเสีย การคืนสภาพข้อมูลจะแตกต่างจากข้อมูลต้นฉบับเช่นนี้ สิ่งที่จะชี้วัดประสิทธิภาพของการบีบอัดข้อมูลก็คือ ปริมาณความแตกต่างของข้อมูลต้นฉบับและข้อมูลที่ทำการขยายกลับ ซึ่งนิยมเรียกว่า การสูญเสีย จะสามารถคำนวณเป็นตัวเลขหรือเปอร์เซนต์ความแตกต่างระหว่างข้อมูลก่อนการบีบอัดและหลังการบีบอัดได้

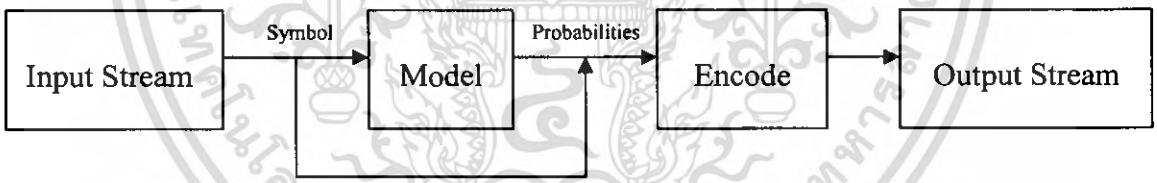
2.2.4 โมเดลและการเข้ารหัส

โดยทั่วไปการบีบอัดข้อมูลประกอบด้วย การอ่านลำดับข้อมูลเข้ามาแล้วทำการแปลงข้อมูลเหล่านี้ให้อยู่ในรูปแบบโค้ดต่างๆ ถ้าหากการบีบอัดข้อมูลมีประสิทธิผล โค้ดที่ได้ต้องมีขนาดสั้นกว่าข้อมูลเดิม โดยโค้ดที่ได้จะเป็นอย่างไรนั้นขึ้นอยู่กับโมเดลที่ใช้ ซึ่งโมเดลคือ กลุ่มของข้อมูลที่เลือกไว้และกฎที่ใช้ในการประมวลผลข้อมูลเหล่านั้นเพื่อที่จะหาโค้ดที่สามารถแทนข้อมูลเหล่านั้นได้ โปรแกรมที่จะใช้โมเดลในการคำนวณหาความน่าจะเป็นของการซ้ำซ้อนของข้อมูลแต่ละตัว เพื่อทำการสร้างโค้ดสำหรับข้อมูลนั้นๆ ในด้านการบีบอัดข้อมูลโดยส่วนใหญ่เรามักจะใช้คำว่าการเข้ารหัสเพื่อหมายถึงขบวนการบีบอัดข้อมูลทั้งขบวนการ แทนที่จะหมายถึงส่วนหนึ่งของขบวนการ เช่น เราอาจจะได้ยินคำว่าการเข้ารหัสฮัฟแมนที่จะถูกใช้ในการอธิบายเทคนิคการบีบอัดข้อมูล แต่ในความเป็นจริงมันเป็นเพียงวิธีการเข้ารหัสข้อมูลที่ถูกนำมาใช้ร่วมกับโมเดลในการบีบอัดข้อมูลเท่านั้น ดังนั้นจึงสามารถนิยามการบีบอัดข้อมูลได้ดังรูปที่ 2.2



รูปที่ 2.2 นิยามการบีบอัดข้อมูล

ถ้าเราดูตัวอย่างจากอัลกอริทึมของการเข้ารหัสฮัฟแมน ขบวนการบีบอัดข้อมูลสามารถแบ่งออกได้เป็นขั้นตอนย่อยๆได้ดังรูปที่ 2.3 ต่อไปนี้



รูปที่ 2.3 ตัวอย่างอัลกอริทึม

2.2.4.1 การเข้ารหัส

ในการบีบอัดข้อมูลเราต้องการสร้างโค้ดสำหรับแต่ละตัวอักษร โดยใช้จำนวนบิทตามความเป็นจริง เช่น ตัวอักษร e ใช้ 4 บิท หรือตัวอักษร a ใช้ 6 บิทเป็นต้น แต่ในความเป็นจริงถ้าหากเราเข้ารหัสตัวอักษรโดยใช้แอสกี(ASCII) เราก็ไม่สามารถที่จะเข้ารหัสทุกๆตัวอักษรได้อย่างกะทัดรัดที่สุด ซึ่งในการแก้ไขปัญหาเหล่านี้จึงเป็นที่นิยมในการใช้เทคนิคการเข้ารหัสแชนนอน และการเข้ารหัสฮัฟแมนในการสร้างโค้ดที่มีความยาวเปลี่ยนแปลงได้ โดยที่จำนวนบิทของโค้ดแต่ละตัวก็ขึ้นกับความน่าจะเป็นของการเกิดซ้ำของข้อมูลในกลุ่มนั้น

2.2.4.2 การสร้างโมเดล

ถ้าหากมองการบีบอัดข้อมูลเป็นรถจักรยานยนต์ การเข้ารหัสก็จะเป็นส่วนวงล้อและโมเดลก็จะเป็นส่วนเครื่องยนต์ ดังนั้นถ้าหากเราเลือกโมเดลที่ไม่ดีพอ ในการป้อนข้อมูลให้กับส่วนเข้ารหัสอาจจะทำให้การบีบอัดข้อมูลมีประสิทธิภาพต่ำก็ได้ เช่น การบีบอัดข้อมูลแบบไม่มีการสูญเสีย โดยทั่วไปจะมีการเลือกใช้เพียง 2 โมเดลคือ โมเดลทางสถิติ(statistical) และ โมเดลที่อ้างอิงดิกชันารี(dictionary-based) โดยโมเดลทางสถิติจะอ่านข้อมูลและทำการเข้ารหัสแต่ละตัวอักษร ส่วนโมเดลที่อ้างอิงดิกชันารีจะใช้โค้ดตัวหนึ่งที่มีในดิกชันารีในการแทนที่อักขระชุดหนึ่งๆ โดยถ้าหากพบอักขระดังกล่าว ผลลัพธ์ที่ได้ก็จะเป็นค่าดัชนีหรือค่าชี้ตำแหน่งสำหรับตำแหน่งที่พบในคำศัพท์แทนที่จะเป็นโค้ดสำหรับอักขระนั้นๆ

2.3 เทคนิคการบีบอัดข้อมูลแบบไม่สูญเสีย

ในหัวข้อนี้จะอธิบายรายละเอียดของเทคนิคการบีบอัดข้อมูลแบบสูญเสีย ซึ่งอัลกอริทึมที่นิยมนำมาใช้ในการบีบอัดได้แก่ การเข้ารหัสฮัฟแมน(Huffman Coding),การเข้ารหัสตามความยาวที่เคลื่อนที่ (run length encoding),เทคนิคดิกชันนารี โดยในหัวข้อนี้จะอธิบายเฉพาะส่วนของการเข้ารหัสฮัฟแมนและการเข้ารหัสเลขคณิต

2.3.1 คำอธิบายตัวแปรต่างๆที่ใช้ในการเข้ารหัส

<u>คำอธิบายที่ 1</u> :อักขระและสัญลักษณ์

อักขระ คือ เซทที่จำกัดและไม่ใช่เซทว่าง ความยาว หรือขนาดของอักขระ A จะหาจาก $|A|$ ลำดับ $\{a_1,....,a_m\}$ ของอักขระเรียกว่าสัญลักษณ์

<u>คำอธิบายที่ 2</u> :ลำดับ

อนุกรม $S=(a_1,...,a_m)$ ของสัญลักษณ์ s_i จากอักขระ A เรียกว่าลำดับ

<u>คำอธิบายที่ 3</u> :ความน่าจะเป็น

ให้ $S=(s_1,...,s_n)$ เป็นอนุกรมความยาวจำกัด ด้วย $|S|=n$ บน $A=\{a_1,...,a_m\}$ แล้วให้ $|S|_{a_i}$ เป็นความถี่ของ a_i ใน S ต่อจากนั้นระบุ $P(a_i):=\frac{|S|_{a_j}}{n}$ เป็นความน่าจะเป็นของ a_i ใน S

จากการอธิบายเราสามารถสรุปได้ว่า $P(a_i)$ จะบรรจุอยู่ในช่วง [0,1) เสมอ สำหรับทุกสัญลักษณ์ทุกที่ผลรวมความน่าจะเป็นทั้งหมดจะคือ $\sum_{i=1}^{m}P(a_i)=1$ เสมอ กรุณาจำว่าช่วงมีทั้งเปิดและปิดเพราะว่ามันไม่มีเหตุผลที่จะเข้ารหัสอนุกรมที่มีโอกาสที่จะเกิดแน่นอนคือค่าความน่าจะเป็นเท่ากับ 1 มันหมายความว่าอย่างนั้นเราก็รู้อยู่แล้วว่ามันจะมีค่าเท่าไหร่เพราะว่ามันมีแค่อย่างเดียว เราจะใช้คุณสมบัตินี้ในการสรุปเรื่องที่เป็นเรื่องแน่นอนในคราวต่อไป

กลับมาเรื่องเดิมอีกครั้ง ยกตัวอย่าง e/z ไม่ว่าเราจะขยายความน่าจะเป็นของสัญลักษณ์ซึ่งจะมากหรือน้อยขึ้นอยู่กับจำนวนคำที่ใช้ ถ้าเราพิจารณา e และ z เป็นสัญญาลักษณ์ขนาดไบต์ ในการประมวลผล

แบบไบนารี ยกตัวอย่างเช่น มันมีโอกาสที่จะเกิดเลข 1 ในตัวหนังสือตำราทางสายวิทยาศาสตร์มากกว่าบทความบนหนังสือพิมพ์

บางข้อมูลเป็นหลักสำคัญในการอธิบาย ยกตัวอย่างเช่น พิจารณาอนุกรม 1 1 1 1 1 3 1 1 3 1 1 มันสามารถเป็นการอธิบายเหมือนเป็นอนุกรมสัญลักษณ์ 1,3 หรือ 11,13 อย่างน้อยที่สุดตัวอย่างนี้สามารถพิสูจน์ว่าเราต้องการกฎบางอย่างเพื่อที่จะทำให้มันไม่สับสนว่าความน่าจะเป็นจะสัมพันธ์กับสัญลักษณ์อย่างไร ความสัมพันธ์ระหว่างสัญญาลักษณ์นี้ของตัวหนังสือและความน่าจะเป็นปกติรู้จักกันในรูปแบบของโมเดลในรูปแบบของการบีบอัดข้อมูล

<u>คำอธิบายที่ 4</u> : โมเดล

ให้ A เป็นตัวหนังสือ โมเดล M คือฟังก์ชั่น

$$M : A \to [0,1) : a_i \to P_M(a_i)$$

ความน่าจะเป็น $P_M(a_i)$ จะถูกเปลี่ยนไปเป็นสัญลักษณ์ a_i

ความน่าจะเป็นนี้บางทีอาจถูกประมาณหรือคำนวณ และไม่สำคัญว่าจะต้องเข้ากับความน่าจะเป็นจริงๆของสัญลักษณ์ $P(a_i)$ แทนที่ส่วนมากทั้งที่ปกติไม่เป็นอย่างนั้น กรุณาจำไว้ด้วยว่าตัวหนังสือไม่ถูกจำกัดเพียงแค่สัญลักษณ์ความยาวเท่ากับ 1 ในตัวอย่างข้างบนได้ค่า 11 และ 13 เหมือนเป็นสัญลักษณ์ เราได้เห็นภาพนั้นไปแล้ว ถ้ามีหนึ่งตัวประมาณความน่าจะเป็นของสัญลักษณ์ที่ให้มาไม่ใช่แค่มองที่ตัวสัญลักษณ์ของตัวมันเองแต่ดูที่ตัวหนังสือที่ให้มาเป็นจำนวนสัญลักษณ์ n ตัว ที่มองผ่านมา มีคนพูดว่ามันคือลำดับ n โมเดล ยกตัวอย่างเช่นค่าประมาณความน่าจะเป็นของตัวหนังสือ u ที่เกิดขึ้นในหนังสือของเยอรมันมีเพียงแค่ประมาณ 0.435 ถ้ามีใครพิจารณาความน่าจะเป็นที่เกิดขึ้นหลังจากตัวอักษร q อย่างไรก็ตามค่านี้เพิ่มขึ้นเข้าใกล้ 1 เหมือนที่เราสามารถเห็นได้จากตอนนี้ ค่าที่เพิ่มขึ้นของค่า n บางทีจะเพิ่มการทำนายที่แม่นยำขึ้นของความน่าจะเป็น

เหมือนที่ได้กล่าวย่อไว้แล้วข้างต้น จะได้ความว่าการกระจายของความน่าจะเป็นจริงๆที่ดีที่สุดนั้นจะมีเกิดขึ้นโดยบังเอิญ (การกระจายของความน่าจะเป็นที่ถูกให้นิยามว่าเป็นลำดับของความน่าจะเป็นภายใต้รูปแบบโมเดลที่แน่นอน) ซึ่งโดยปกติแล้วกรณีนี้จะไม่เกิดขึ้น ยกตัวอย่างเช่น ไม่มีหนังสือเยอรมันเล่มไหนจะมีการกระจายที่เหมือนกับตารางที่ 2.1 อย่างแน่นอนมันเพียงเป็นแค่ค่าประมาณ เพื่อที่จะจำแนกความแตกต่างความน่าจะเป็นที่ถูกกำหนดโดยโมเดลของค่าความน่าจะเป็น 0 ถึง 1 เราให้เป็น $P_M(a_i)$ ถ้าลำดับที่จะขยายการขึ้นอยู่กับโมเดล และนอกจากนั้นให้เป็น $P(a_i)$

ดังนั้นเราสามารถสรุปได้ว่าสามารถถูกพบในการอธิบายของกลุ่มข้อมูลที่เป็นตัวกำหนด โมเดลอย่างง่ายสามารถยกตัวอย่างได้โดยการให้การกระจายของข้อมูลในตารางที่ 2.1 ตารางนี้แสดงความน่าจะเป็นของตัวหนังสือเยอรมันที่เกิดขึ้นในหนังสือเยอรมันส่วนใหญ่ เป็นไปได้ว่าผู้อ่านที่ฉลาดสามารถคาดเดาได้แล้ว ว่าอัตราการบีบอัดจะมากขึ้นขึ้นอยู่กับโมเดลเหมาะสมกับความเป็นจริงแค่ไหน

ทำให้เราสามารถที่จะเปรียบเทียบประสิทธิภาพในการบีบอัดแต่ละแบบ นี้เป็นการการวัดว่าข้อมูลถูกบรรจุอยู่แค่ไหนและถูกเรียกว่าเอ็นโทรปี

คำอธิบายที่ 5 :เอ็นโทรปี

ให้ s เป็นอนุกรมบนตัวหนังสือ $A=\{a_1,...,a_m\}$ เอ็นโทรปี $H_M(s)$ ของอนุกรม s ภายใต้โมเดล M ถูกระบุเป็น

$$H_M(s)=\sum_{i=1}^{m}P(a_i)ld\frac{1}{P_M(a_i)} \tag{2.1}$$

a	0.0651	h	0.0476	o	0.0251	v	0.0067
b	0.0189	i	0.0755	p	0.0079	w	0.0189
c	0.0306	j	0.0027	q	0.0002	x	0.0003
d	0.0508	k	0.0121	r	0.0700	y	0.0004
e	0.1740	l	0.0344	s	0.0727	z	0.0113
f	0.0166	m	0.0253	t	0.0615		
g	0.0301	n	0.0978	u	0.0435		

ตารางที่ 2.1 ความน่าจะเป็นของตัวหนังสือเฉลี่ยในหนังสือเยอรมัน

หน่วยของเอ็นโทรปีคือ1บิทต่อสัญลักษณ์ เพราะว่าสูตรเพียงแค่อ้างอิงความน่าจะเป็นเหมือนกับความถี่ของข้อมูลที่มีค่าเข้าใกล้ 1 โดยสมการที่หนึ่ง เอ็นโทรปีของลำดับขึ้นอยู่กับโมเดล M โดย $P_M(a_i)$ เป็นความน่าจะเป็นภายใต้โมเดลนี้และ $ld\frac{1}{P_M(a_i)}$ สามารถเป็นความยาวสัญลักษณ์ไบนารีที่น้อยที่สุดสำหรับ a_i ขณะที่แฟกเตอร์ $P(a_i)$ (เป็นความน่าจะเป็นจริงๆของ a_i) เป็นความน่าจะเป็นของการร้องขอเข้ารหัสไปเป็นการเข้ารหัสแบบไบนารีของสัญลักษณ์นี้

การพิจารณาโมเดลที่ดีขึ้นอยู่กับการรับค่าความน่าจะเป็นที่ถูกต้องเพื่อที่จะนำไปคำนวณค่าของเอ็นโทรปี

$$H_M(s)=\sum_{i=1}^{m}P(a_i)ld\frac{1}{P_M(a_i)}$$

เอ็นโทรปีชนิดนี้ขึ้นอยู่กับข้อมูลที่ป้อนเข้าและไม่มีหลักการแปลงที่แน่นอน อย่างไรก็ตามค่าของเอ็นโทรปีของการเข้ารหัสเลขคณิตแต่ละชนิดก็ไม่แตกต่างกัน

ตัวอย่างของความน่าจะเป็น

ให้เราดูอนุกรม S = abaabcda ตัวหนังสือ $\{a,b,c,d\}$ เราต้องเข้ารหัสไบนารีของอนุกรมนี้ แต่ตอนแรกเราไม่รู้เลยว่าความน่าจะเป็นจะกระจายอย่างไร เราเลือกใช้รูปแบบง่ายๆ ซึ่งจะ นำไปสู่ความ

น่าจะเป็นที่ถูกต้อง ค่า $P_{M_1}(a)$=0.5 $P_{M_1}(b)$=0.25 $P_{M_1}(c)$=0.125 $P_{M_1}(d)$=0.125 เราสามารถเห็นได้โดยง่ายว่ารูปแบบนี้ เป็นการประมาณความน่าจะเป็น $P_{M_1}(s)$ ที่เหมาะสมกับ $P(s)$

$$P_{M_1}(s) = P(s) \forall s \in A := \{a,b,c,d\}$$

เมื่อเข้ารหัสอนุกรมนี้เราสามารถทำด้วยวิธีอย่างตรงๆโดยใช้ 2 บิทต่อสัญลักษณ์ $\{00,01,10,11\}$ ซึ่งทำให้ได้ทั้งหมด 16 บิท แล้วเอ็นโทรปีของ $H_{M_1}(s)$ จะคือ

$$H_{M_1} = \sum_{S\in\{a,b,c,d\}} P(s) ld \frac{1}{P_{M_1}(s)}$$

$$= (0.5 \cdot ld \frac{1}{0.5}) + (0.25 \cdot ld \frac{1}{0.25}) + (0.125 \cdot ld \frac{1}{0.125}) + (0.125 \cdot ld \frac{1}{0.125})$$

$$= 0.5 \cdot ld2 + 0.25 \cdot ld4 + 0.125 \cdot ld8 + 0.125 \cdot ld8$$

$$= 0.5 + 0.5 + 0.375 + 0.375$$

$$= 1.75 \text{ bits/symbol}$$

จำไว้ว่าใน1บิทต่อสัญลักษณ์ ซึ่งหมายความว่าต้องการอย่างต่ำ $8 \times 1.75 = 14$ บิท เพื่อจะเข้ารหัสอนุกรมอินพุททั้งหมด เราไม่สามารถทำได้ดีกว่านี้ได้ นี้ทำให้ประหยัดไป 16-14 = 2 บิท

อย่างไรก็ตามอะไรจะเกิดขึ้นถ้าเราโชคไม่ดีที่จะเดาความน่าจะเป็นที่ถูกต้องของการกระจายครั้งหน้า ดูตามโมเดล M_2 กับ $P_{M_2}(a) = 0.125$ $P_{M_2}(b) = 0.125$ $P_{M_2}(c) = 0.5$ $P_{M_2}(d) = 0.25$ เอ็นโทรปีจากการคำนวณ M_2 จะได้

$$H_{M_2} = \sum_{S\in\{a,b,c,d\}} P(s) ld \frac{1}{P_{M_2}(s)}$$

$$= (0.5 \cdot ld \frac{1}{0.125}) + (0.25 \cdot ld \frac{1}{0.125}) + (0.125 \cdot ld \frac{1}{0.5}) + (0,125 \cdot ld \frac{1}{0.25})$$

$$= 0.5 \cdot ld8 + 0.25 \cdot ld8 + 0.125 \cdot ld2 + 0.125 \cdot ld4$$

$$= 1.5 + 0.75 + 0.125 + 0.25$$

= 2.625 bits/symbol

เราควรจะมองตัวอย่างนี้เพื่อเป็นการเตือนว่าอย่ารวมแนวคิดของการเข้ารหัสเข้ากับการบีบอัด เหตุผลคือเราสามารถเห็นภายใต้โมเดล M_2 ว่าเราไม่ต้องการที่จะใช้ $2.625 \times 8 = 21$ บิท เพื่อจะเข้ารหัสอนุกรมอินพุท อย่างไรก็ตามนี่จะไม่ถูกบีบอัดเลย ถ้ามีใครจำได้ว่าการเข้ารหัสตรงๆ 2 บิทต่อสัญลักษณ์ ใช้ 16 บิทด้วยกัน เราสามารถสรุปได้ว่าอัตราการบีบอัดสามารถดีสำหรับบางโมเดลเท่านั้น ถ้าโมเดลเหมาะกับความจริงก็ทำให้การบีบอัดดี

จากการพิสูจน์จะเห็นว่าโมเดลของการเข้ารหัสเลขคณิตมีอัตราการบีบอัดที่ดี

คำอธิบายที่ 6 :ความสามารถในการถอดรหัสได้เป็นลักษณะเฉพาะ

เราจะเรียกโค้ดที่สามารถถอดรหัสได้ออกมามีความเป็นลักษณะเฉพาะคือ เมื่อลำดับได้ระบุค่าโค้ดไปในแนวทางเดียวกัน กรณีเช่นนี้ก็จะสามารถหาค่าสัญลักษณ์อินพุตที่ทำให้เกิดโค้ดนี้ได้

ชนิดพิเศษชนิดหนึ่งของรหัสที่สามารถถอดออกมาได้มีความเป็นลักษณะเฉพาะนั้นถูกเรียกว่ารหัสที่มีคำนำหน้า(prefix code) จะมีคุณสมบัติคือไม่มีรหัสที่แทนคำใด เป็นรหัสนำหน้าของรหัสที่แทนคำอื่น

คำอธิบายที่ 7 :รหัสที่มีคำนำหน้า

ให้โค้ด C คือรหัสที่มีคำนำหน้า หากไม่มีคู่สัญลักษณ์ (x, y) ของตัวอักษร $C(x)$ จะเป็นคำนำหน้าของ $C(y)$

รหัสที่มีคำนำหน้านี้จะมีประโยชน์อย่างมากเมื่อตัวถอดรหัสได้อ่านโค้ด $C(x)$ สำหรับ x เมื่อถอดรหัสแล้วจะได้สัญลักษณ์ x ออกมา ในกรณีที่เป็นรหัสแบบที่ไม่มีกฎเกณฑ์อาจเกิดการอ่านเพียง $C(x)$ แม้ว่ามันจะเป็นเพียงคำนำหน้าของโค้ดอื่น $C(y)$

2.3.2 การเข้ารหัสฮัฟแมน

เป็นวิธีการเข้ารหัสข้อมูลที่มีประสิทธิภาพวิธีหนึ่ง ซึ่งถูกนำเสนอโดย D.A. Huffman และถูกตีพิมพ์ครั้งแรกในปี ค.ศ. 1952 ในบทความ "A Method for the Construction of Minimum Redundancy Codes" ซึ่งคุณสมบัติหลายอย่างที่เหมือนกับการเข้ารหัสแชนนอน คือ การสร้างโค้ดที่มีความยาว(จำนวนบิท)ที่แตกต่างกันได้ ข้อมูลที่มีความน่าเป็นสูงกว่าก็จะถูกเข้ารหัสด้วยโค้ดที่มีจำนวนบิทน้อยกว่า และที่สำคัญข้อมูลที่ได้จะมีคุณสมบัติที่ทำหน้าที่ไม่เหมือนกัน ดังนั้นโค้ดที่ได้จึงสามารถถูกถอดรหัสได้อย่างถูกต้อง ขั้นตอนการถอดรหัสสายโค้ดโดยวิธีนี้ส่วนใหญ่มักกระทำโดยใช้แผนภูมิต้นไม้(Binary Decoder Tree)

การสร้างแผนภูมิต้นไม้โดยวิธีฮัฟแมนนั้นจะสร้างจากล่างขึ้นบน ดังนั้นโค้ดทุกตัวจะเริ่มถูกสร้างจากกิ่งก้าน(left node) และไปสิ้นสุดที่ราก(root node) โดยมีขั้นตอนในการสร้างโดยสังเขปดังนี้

1. ทำการนับค่าความน่าจะเป็นของข้อมูลทุกตัว และเรียงลำดับข้อมูลตามค่าความน่าจะเป็นจากน้อยมามาก

2. โหนด(node)จะถูกสร้างทีละคู่ จากคู่ความน่าจะเป็นน้อยไปยังคู่ความน่าจะเป็นมาก แต่หากข้อมูลตัวสุดท้ายไม่สามารถจัดเข้าคู่ได้ก็จะแยกเป็นอิสระที่มีกิ่งเดียว
3. สร้างโหนดแม่(Parent Node)ของโหนดคู่ที่ได้จากขั้นตอนแรกและจะมีค่าน้ำหนักเท่ากับผลรวมของค่าความน่าจะเป็นของโหนดลูก(Child Node)ทั้งสอง
4. โหนดแม่ที่ได้ก็จะถูกเพิ่มเข้ามาเป็นโหนดอิสระ และโหนดลูกก็จะถูกยกเลิก
5. โหนดลูกโหนดหนึ่งจะถูกกำหนดให้เป็นไบนารี่ '0' ส่วนอีกโหนดจะถูกกำหนดเป็น '1'
6. ให้ทำซ้ำขั้นตอนทั้งหมดจนเหลือโหนดอิสระเพียงโหนดเดียวซึ่งก็คือโหนดราก

<u>ตัวอย่าง</u>

สมมติมีข้อมูล 5 ข้อมูล ได้แก่ a_1, a_2, a_3, a_4 และ a_5 และมีค่าความน่าจะเป็นดังนี้ $P(a_1) = 0.2$, $P(a_2) = 0.4, P(a_3) = 0.2, P(a_4) = 0.1, P(a_5) = 0.1$ สิ่งแรกที่ต้องทำคือการเข้ารหัส $c(a_i)$ ของข้อมูล a_i โดยเรียงลำดับจากค่าความน่าจะเป็นจากมากไปหาน้อย

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

ตารางที่ 2.2 ตัวอย่างและอัตราการเกิดข้อมูล

ค่าความน่าจะเป็นน้อยที่สุดสองค่า จะถูกสร้างรหัสก่อนดังนี้

$$c(a_4) = \alpha_1 * 0$$

$$c(a_5) = \alpha_1 * 1$$

เมื่อ α_1 เป็นอักขระเลขฐานสอง และ * แทนการเชื่อมเข้าด้วยกันระหว่างสองข้อมูล

ให้ a_4' เป็นข้อมูลที่เกิดจากความสัมพันธ์ $P(a_4') = P(a_4) + P(a_5) = 0.2$ ดังแสดงข้อมูลในตารางที่ 2.3

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

ตารางที่ 2.3 สร้างข้อมูล a_4'

ในตารางนี้ค่าความน่าจะเป็นต่ำสองค่าคือ a_3 และ a_4' จึงสามารถสร้างรหัสได้เป็น

$$c(a_3) = \alpha_2 * 0$$

$$c(a_4') = \alpha_2 * 1$$

กำหนดให้ $c(a_4') = \alpha_1$ ดังนั้น

$$c(a_4) = \alpha_2 * 10$$

$$c(a_5) = \alpha_2 * 11$$

จากตารางที่ 2.3 ค่าที่จะเข้ามาสร้างรหัสต่อไปคือ a_3 และ a_4' เพราะเป็นคู่ที่มีอัตราการเกิดน้อยที่สุด กำหนดให้ a_3' เป็นข้อมูลที่เกิดจากความสัมพันธ์ $P(a_3') = P(a_3) + P(a_4') = 0.4$ ดังแสดงข้อมูลในตารางที่ 2.4

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

ตารางที่ 2.4 สร้างข้อมูล a_3'

จากตารางที่ 2.4 ค่าที่จะเข้ามาสร้างรหัสต่อไปคือ a_3' และ a_1 จึงสามารถสร้างรหัสได้เป็น

$$c(a_3') = \alpha_3 * 00$$

$$c(a_4) = \alpha_3 * 010$$

$$c(a_5) = \alpha_3 * 011$$

กำหนดให้ a_3'' เป็นข้อมูลที่เกิดจากความสัมพันธ์ $P(a_3'') = P(a_3') + P(a_1) = 0.6$ ดังแสดงข้อมูลในตารางที่ 2.5

Data	*Probability*	*Code*
a_3''	0.6	α_3
a_2	0.4	$c(a_2)$

ตารางที่ 2.5 สร้างข้อมูล a_3''

ในขั้นตอนสุดท้ายจะเหลือข้อมูลให้สร้างรหัส 2 ตัว และจะได้

$$c(a_3'') = 0$$

$$c(a_2) = 1$$

ดังนั้นจะได้รหัสฮัฟแมนดังตารางที่ 2.6

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

ตารางที่ 2.6 กำหนดรหัสฮัฟแมน

2.3.3 การเข้ารหัสเลขคณิต

การเข้ารหัสเลขคณิตเป็นการเข้าสร้างรหัสที่มีความยาวเปลี่ยนแปลงไม่คงที่ ซึ่งจะใช้ได้ดีกับแหล่งข้อมูลที่มีอักขระจำนวนน้อยเช่น ข้อมูลเลขฐานสอง และอักขระที่มีความน่าจะเป็นแตกต่างกันมากๆ

อัตราการเข้ารหัส คือ ค่าเฉลี่ยของจำนวนบิทที่ใช้แทนสัญลักษณ์จากแหล่งข้อมูลและจากความน่าจะเป็นที่ให้มา ค่าความถี่ของสัญลักษณ์ที่เกิดขึ้นใน เอนโทรปีคืออัตราต่ำสุดที่ต้นทางสามารถเข้ารหัส

ได้ อย่างไรก็ตามเราสามารถเข้ารหัสได้สูงกว่าอัตรานี้จากวิธีฮัฟแมนซึ่งจะสร้างรหัสในช่วง $p_{\max} + 0.086$ ของอัตราการเข้ารหัสต่ำสุด เมื่อ $p_{\max}$ คือ ความน่าจะเป็นของสัญลักษณ์ที่มีความถี่ในการเกิดสูงสุดใน ข้อมูลที่อักขระมีขนาดใหญ่ $p_{\max}$ มักจะมีค่าน้อย และมีค่าใกล้เคียงกับอัตราการเข้ารหัสต่ำสุดด้วยเมื่อ เทียบเป็นเปอร์เซ็นต์ แต่ในกรณีที่อักขระมีขนาดเล็ก มีความน่าจะเป็นแตกต่างกันมากๆ ค่า $p_{\max}$ จะ ค่อนข้างสูงทำให้การเข้ารหัสฮัฟแมนจะไม่มีประสิทธิภาพเมื่อเทียบกับอัตราการเข้ารหัสต่ำสุดวิธีหนึ่งใน การหลีกเลี่ยงปัญหาเหล่านี้คือ การรวมสัญลักษณ์มากกว่าหนึ่งสัญลักษณ์เข้าด้วยกัน และสร้างการ เข้ารหัสฮัฟแมนที่ขยายขึ้น (Extended Huffman Code) อย่างไรก็ตามวิธีนี้ก็ไม่ค่อยได้ผลมากนัก

2.3.3.1 การเข้ารหัสของลำดับ

การที่จะแยกกลุ่มของสัญลักษณ์หนึ่งออกจากกลุ่มสัญลักษณ์หนึ่งนั้นเราจำเป็นต้องระบุค่า ด้วยสิ่งที่ชี้บ่งที่มีลักษณะเฉพาะ ทางหนึ่งในการแทนลำดับของสัญลักษณ์คือการระบุค่าด้วยตัวเลขในช่วง [0,1) เพราะจำนวนของตัวเลขในช่วงนี้มีไม่จำกัดดังนั้นเราจึงสามารถที่จะระบุค่าที่มีลักษณะเฉพาะ สำหรับแต่ละกลุ่มของสัญลักษณ์ที่เพิ่มขึ้น ฟังก์ชันที่นำค่าเหล่านี้ให้อยู่ในช่วง [0,1) คือ ฟังก์ชันการแจก แจงสะสม (cumulative distribution function : cdf) ของตัวแปรสุ่มที่เกี่ยวข้องกับแหล่งข้อมูล

จากค่าตัวแปรสุ่มระบุตำแหน่งความน่าจะเป็นที่จะเกิดขึ้นไปเป็นค่าบนเส้นจำนวนจริง ตัวอย่างเช่น การทดลองโยนเหรียญ ค่าตัวแปรสุ่มที่ได้สามารถระบุตำแหน่งต้นเป็น 0 และท้ายเป็น 1 การ ใช้เทคนิคนี้ เราจำเป็นต้องระบุตำแหน่งของสัญลักษณ์หรือตัวอักษร ไปจนถึงตัวเลขเพื่อความสะดวก การ อธิบายในบทนี้จะใช้การระบุตำแหน่ง

$$X(a_i) = i \qquad a_i \in A \tag{2.2}$$

เมื่อ $A = \{a_1, a_2, ..., a_m\}$ เป็นอักขระ หรือตัวอักษรสำหรับแหล่งข้อมูลที่ไม่ต่อเนื่อง และ X คือค่าตัว แปรสุ่ม การระบุตำแหน่งนี้หมายถึงจากโมเดลความน่าจะเป็น P สำหรับแหล่งข้อมูล เราจะได้ฟังก์ชัน ความหนาแน่นของความน่าจะเป็นสำหรับตัวแปรสุ่ม

$$P(X = i) = P(a_i) \tag{2.3}$$

และฟังก์ชันความหนาแน่นสะสมจะถูกแสดงเป็น

$$F_X(i) = \sum_{k=1}^{i} P(X = k) \tag{2.4}$$

2.3.3.2 การสร้างแท็ก (tag)

กระบวนการสร้างแท็กทำโดยการลดขนาดของช่วงที่แท็กอยู่ ถ้าลำดับที่เข้ามามากการลดช่วงก็จะทำหลายครั้งขึ้น เราเริ่มโดยการแบ่งช่วงยูนิตเป็นช่วงยูนิตย่อยๆก่อนในรูปแบบ $[F_x(i-1),F_x(i)),i=1,...,m$ เพราะค่าที่ต่ำสุดของ cdf เป็น 0 และค่ามากที่สุดเป็น 1 ส่วนเริ่มต้นนี้ช่วงยูนิตของเราจะเกี่ยวกับช่วงย่อย $[F_x(i-1),F_x(i))$ กับสัญลักษณ์ a_i สัญลักษณ์ตัวแรกปรากฏในลำดับจำกัดจะบอกช่วงไว้ ค่าแท็กที่จะชี้ค่าช่วงย่อยๆ สมมติสัญลักษณ์ตัวแรกเป็น a_k ต่อจากนั้นช่วงที่บรรจุค่าแท็กจะแบ่งออกเป็นช่วงย่อยๆ $[F_x(k-1),F_x(k))$ ช่วงที่แบ่งย่อยนี้จะถูกแบ่งด้วยอัตราส่วนที่เท่ากันกับช่วงดั้งเดิม จะเป็นช่วงที่ j มีค่าตามสัญลักษณ์ a_j ที่ได้จาก

$[F_X(K-1)+F_X(j-1))/(F_X(K)-F_X(K-1)),F_X(K-1)+F_X(j)/F_X(K)-F_X(K-1)))$

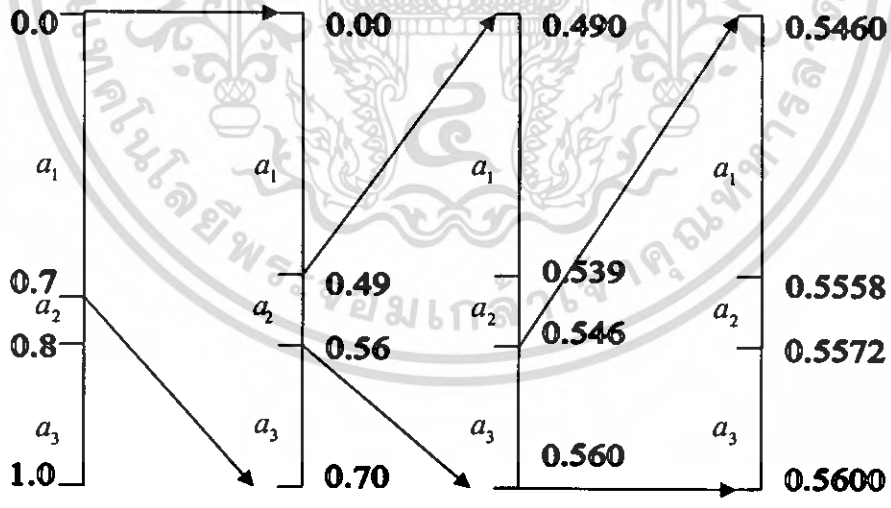
ดังนั้นถ้าสัญลักษณ์ที่สองในอนุกรมคือ a_j ดังนั้นช่วงที่มีค่าแท็กอยู่จะกลายเป็น

$[F_X(K-1)+F_X(j-1))/(F_X(K)-F_X(K-1)),F_X(K-1)+F_X(j)/F_X(K)-F_X(K-1)))$

สัญลักษณ์ทุกตัวที่ได้นำไปแทนในแต่ละช่วงแล้วทำให้เกิดแท็ก โดยช่วงย่อยนั้นจะถูกแบ่งต่อไปในสัดส่วนที่เท่าๆกัน ขบวนการนี้สามารถที่จะเข้าใจมากกว่านี้ผ่านตัวอย่างข้างล่าง

<u>ตัวอย่าง</u>

พิจารณาตัวอักษรสามตัว $A=\{a_1,a_2,a_3\}$ ด้วย $P(a_1)=0.7$ และ $P(a_2)=0.1$ และ $P(a_3)=0.2$ โดยใช้ตาราง (4.1) $F_X(1)=0.7,F_X(2)=0.8,F_X(3)=1$ การแบ่งเป็นช่วงๆนี้แสดงในรูป 2.4



รูปที่ 2.4 จำกัดช่วงที่มีค่าแท็กอยู่สำหรับลำดับ $\{a_1,a_2,a_3,...\}$ ที่รับเข้ามา

การแบ่งเป็นออกไปเป็นส่วนๆนั้น ขึ้นอยู่กับสัญลักษณ์โดยสัญลักษณ์ตัวแรกคือ a_1 วางอยู่ในช่วง $[0.0, 0.7)$ ถ้าสัญลักษณ์ตัวแรกคือ a_2 จะวางอยู่ในช่วง $[0.7, 0.8)$ ถ้าสัญลักษณ์ตัวแรกคือ a_3 จะวางอยู่ในช่วง $[0.8, 1.0)$ ถ้าช่วงที่บรรจุแท็กนี้ได้ถูกนำมาคิดแล้วเราก็จะทิ้งมันไปและช่วงก็ถูกแบ่งออกไปอีกในอัตราส่วนเดียวกับช่วงอันเดิม สมมติสัญลักษณ์ตัวแรกคือ a_1 จะถูกบรรจุอยู่ในช่วงย่อย $[0.0, 0.7)$ ช่วงย่อยจะถูกแบ่งในสัดส่วนที่แน่นอนเหมือนช่วงเดิม ให้ช่วงย่อย $[0.0, 0.49), [0.49, 0.56)$ และ $[0.56, 0.7)$ ส่วนแบ่งแรกตรงกับสัญลักษณ์ a_1 ส่วนแบ่งที่สองตรงกับสัญลักษณ์ a_2 และส่วนแบ่งที่สาม $[0.56, 0.7)$ ตรงกับสัญลักษณ์ a_3 สมมติสัญลักษณ์ที่สองในอนุกรมคือ a_2 ค่ าของแท็ก i เป็นค่าจำกัดบนช่วง เราก็จะแบ่งช่วงนี้ในสัดส่วนที่เท่ากันกับช่วงเดิมเพื่อที่จะได้รับช่วงย่อย $[0.49, 0.539)$ ตรงกับสัญลักษณ์ a_1

$[0.539, 0.546)$ ตรงกับสัญลักษณ์ a_2 และ $[0.546, 0.56)$ ตรงกับสัญลักษณ์ a_3 ถ้าสัญลักษณ์ตัวที่ 3 คือ a_3 แท็กจะถูกจำกัดอยู่ในช่วง $[0.546, 0.56)$ ซึ่งสามารถแบ่งย่อยอีกได้ต่อไป ขบวนการนี้ถูกอธิบายเป็นรูปที่ 2.4 จงจำไว้ว่าการเกิดของสัญลักษณ์จะทำให้เกิดช่วงย่อยและทำให้เกิดแท็ก

ใช้ขบวนการนี้ สำหรับอนุกรมที่เริ่มต้นด้วย $\{a_1, a_2, a_3, ...\}$ จนกระทั่งสัญลักษณ์ที่3 a_3 และจะได้รับค่าแท็กถูกจำกัดอยู่ในช่วง $[0.546, 0.56)$ ถ้าสัญลักษณ์ที่ 3 เป็น a_1 แทนที่จะเป็น a_3 จะอยู่ในช่วง $[0.49, 0.539)$ ซึ่งจะไม่ลงอยู่ในช่วงย่อย $[0.546, 0.56)$ แม้ถ้าสองตัวจะเป็นตัวเดียวกันจากตอนนี้(อันแรกเริ่มด้วย a_1, a_2, a_3 และอีกอันเริ่มด้วย a_1, a_2, a_1)ช่วงแท็กสำหรับอนุกรมจะไม่ตรงกันเสมอ

เท่าที่เราสามารถเห็นได้ ช่วงในแท็ก วิธีหนึ่งซึ่งเป็นที่นิยมคือใช้ จำกัดขอบล่างของช่วงอันอื่นที่เป็นไปได้ คือจุดกึ่งกลางของช่วง สำหรับตอนนี้ให้ใช้จุดกึ่งกลางเป็นแท็ก

ในรูปแบบที่เราจะเห็นว่าแท็ก มีกระบวนการสร้างทำงานเป็นคณิตศาสตร์ได้อย่างไร เราเริ่มด้วยอนุกรมที่มีความยาวเท่ากับ 1 สมมติเรามีแหล่งข้อมูลซึ่งใส่ด้วยสัญลักษณ์จาก
ตัวหนังสือ $A = \{a_1, a_2, a_1, ..., a_m\}$ เราสามารถจับคู่สัญลักษณ์ $\{a_i\}$ ไปเป็นจำนวนจริง $\{i\}$ ระบุ $\overline{T}_X(a_i)$ คือ

$$\overline{T}_X(a_i) = \sum_{k=1}^{i-1} P(X = K) + \frac{1}{2} P(X = i) \tag{2.5}$$

$$= F_x(i-1) + \frac{1}{2} P_x(X = i) \tag{2.6}$$

เท่าที่เราเห็นจากตัวอย่างข้างบนได้ให้แท็กที่ไม่เหมือนกันของอนุกรมความยาวเท่ากับ 1 เป็นเรื่องที่ง่าย การเข้าใกล้นี้สามารถถูกต่อยาวเป็นอนุกรมได้อีกโดยการกำหนดให้มีลำดับบนอนุกรม เราต้องจัดลำดับอนุกรมเพราะว่าเราจะให้แท็กเป็นอนุกรมเฉพาะ x_i คือ

$$\bar{T}_X^{(m)}(a_i) = \sum_{y<x_i} P(y) + \frac{1}{2}P(x_i) \qquad (2.7)$$

เมื่อ y<x หมายความว่า y มาก่อน x ในการจัดลำดับบอกถึงความยาวของอนุกรม

วิธีจัดลำดับที่จะใช้คือจัดลำดับแบบเล็กซิโกกราฟฟิก (lexicographic) คือการนำตัวอักษรที่อยู่ในหนังสือมาจัดลำดับ ลำดับของคำในพจนานุกรมนั้นดีเพราะว่ามันเป็นรูปแบบที่มีมานานแล้ว

ตัวอย่าง

เราสามารถต่อจากตัวอย่างข้างต้นอนุกรมประกอบด้วย 2 แถว ใช้ลำดับแบบแผน อธิบายตามด้านบน ผลลัพธ์จะเป็น 11 12 13 ... 66 สามารถถูกสร้างโดยใช้สมการ 2.6 ยกตัวอย่างเช่น แท็กสำหรับอนุกรม 13 จะเป็น

$$\begin{aligned}\bar{T}_X(13) &= P(x=11) + P(x=12) + \frac{1}{2}P(x=13) \\ &= 1/36 + 1/36 + \frac{1}{2}(1/36) \\ &= 5/72\end{aligned}$$

จำไว้ว่าการสร้างแท็กสำหรับ 13 เราไม่ต้องสร้างแท็กสำหรับทุกข้อความที่เป็นไปได้ ตามสมการ 2.6 ที่ถูกสร้างขึ้นตามความต้องการของเรา ซึ่งความน่าจะเป็นของทุกๆอนุกรมนั้นมีค่าต่ำ อนุกรมสำหรับแท็กไหนที่ถูกสร้างตามความต้องการของเราความน่าจะเป็นของทุกอนุกรมที่ให้ความยาวมาจะคำนวณได้อย่างแม่นยำ เรามีรหัสที่ใช้แทนคำ(codeword) สำหรับทุกอนุกรมของความยาวที่ให้มา โชคดีเราจะเห็นว่าการคำนวณแท็กที่ให้มาเป็นอนุกรมสัญลักษณ์ทั้งหมดที่เราต้องการคือความน่าจะเป็นของแต่ละสัญลักษณ์หรือความน่าจะเป็นของแต่ละโมเดล

จากที่กล่าวมาแล้วว่าช่วงบรรจุค่าแท็กสำหรับอนุกรมที่ให้มาไม่ตรงกันจากช่วงที่บรรจุค่าแท็กของทุกอนุกรม หมายความว่าทุกค่าในช่วงจะเป็นค่าที่ระบุที่ไม่เหมือนกันสำหรับ x_i ก่อนหน้านี้ เพื่อที่จะเติมค่าจุดประสงค์แรกของอนุกรมที่ไม่ซ้ำกันให้เต็ม แต่มันเพียงพอที่จะคำนวณขอบเขตบนและล่างของช่วงที่บรรจุอยู่ในแท็ก และเลือกค่าใดๆในช่วงนั้นจากขอบเขตบนและล่างเพื่อที่จะสามารคำนวณย้อนกลับดังที่แสดงในตัวอย่าง

ตัวอย่าง

เราจะใช้ตัวหนังสือของตัวอย่างเดิม และหาขอบเขตบนและล่างของช่วงที่บรรจุแท็กสำหรับอนุกรมจากตัวอย่างเดิม ในแบบอนุกรมนั้นคือ เราเห็น 3 ก่อนต่อจากนั้นก็เห็น 2 และ 2 อีกทีหลังจากการ

สังเกตแต่ละครั้งเราจะคำนวณขอบเขตบนและล่างของช่วงที่บรรจุแท็กอยู่ของอนุกรมที่ถูกสังเกตที่จุดนั้น เราจะให้ $l^{(n)}$ คือขอบเขตล่าง และ $u^{(n)}$ คือ ขอบเขตบน ซึ่ง n เป็นความยาวของอนุกรม

พิจารณาที่ 3 ก่อน

$$u^{(n)} = F_X(3), \quad l^{(n)} = F_X(2)$$

ต่อจากนั้นเราดูที่ 2 จะได้อนุกรม x = 32

$$u^{(2)} = F^{(2)}{}_X(32), \quad l^{(2)} = F^{(2)}{}_X(31)$$

เราคำนวณได้ค่า

$$F^{(2)}{}_X(32) = P(x=11) + P(x=12) + .. + P(x=16)$$

$$+P(x=21) + P(x=22) + .. + P(x=26)$$

$$+P(x=31) + P(x=32)$$

แต่

$$\sum_{i=1}^{i=b} P(x=ki) = \sum_{i=1}^{i=b} P(x_1=k, x_2=i) = P(x_1=k)$$

ซึ่ง $x = x_1x_2$ เพราะฉะนั้น

$$F^{(2)}{}_X(32) = P(x=11) + P(x=12) + P(x=31) + P(x=32)$$

$$= F_x(2) + P(x=31) + P(x=32)$$

อย่างไรก็ตาม สมมติว่าแต่ละแถวไม่ขึ้นต่อกัน

$$P(x=31) = P(x=3)P(x=1)$$

$$P(x=32)=P(x=3)P(x=2)$$

ดังนั้น

$$P(x=31)+P(x=32)=P(x_1=3)(P(x_2=1)+P(x_2=2))$$

$$=P(x_1=3)F_x(2)$$

หาจาก

$$P(x_1=3)=F_x(3)-F_x(2)$$

เราสามารถเขียนได้ว่า

$$P(x=31)+P(x=32)=(F_x(3)-F_x(2))F_x(2)$$

และ

$$F_x^{(2)}(32)=F_x(2)+(F_x(3)-F_x(2))F_x(2)$$

เราสามารถเขียนเป็น

$$u^{(2)}=l^{(l)}+(u^{(l)}-l^{(l)})F_x(2)$$

เราสามารถแสดงว่า

$$F_x^{(2)}(31)=F_x(2)+(F_x(3)-F_x(2))F_x(1)$$

หรือ

$$l^{(2)}=l^{(l)}+(u^{(l)}-l^{(l)})F_x(1)$$

ส่วนประกอบที่สามของอนุกรมคือ 2 และอนุกรมคือ x = 322 ขอบเขตบนและล่างของช่วงที่บรรจุแท็กสำหรับอนุกรม คือ

$$l^{(3)} = F_x^{(3)}(321) \quad u^{(3)} = F_x^{(3)}(322)$$

เราพบว่า

$$F_x^{(3)}(322) = F_x^{(2)}(31) + (F_x^{(2)}(32) - F_x^{(2)}(31))F_x(2)$$

$$F_x^{(3)}(321) = F_x^{(2)}(31) + (F_x^{(2)}(32) - F_x^{(2)}(31))F_x(1)$$

หรือ

$$u^{(3)} = l^{(2)} + (u^{(2)} - l^{(2)})F_x(2)$$

$$l^{(3)} = l^{(2)} + (u^{(2)} - l^{(2)})F_x(1)$$

ปกติเราแสดงอนุกรมเป็น

$$l^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_x(x_n - 1) \tag{2.8}$$

$$u^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_x(x_n) \tag{2.9}$$

จำไว้ว่าตลอดขบวนการนี้เราไม่จำเป็นต้องคำนวณความน่าจะเป็น โดยเราจะใช้จุดกึ่งกลางของช่วงเป็นแท็ก ดังนั้น

$$\bar{T}_X(x) = \frac{u^{(n)} + l^{(n)}}{2}$$

ก่อนหน้านี้แท็กและข้อมูลที่เราต้องการของอนุกรมใดๆสามารถคำนวณได้จากอุนกรมทั่วไปโดยขบวนการสร้างแท็ก คือ cdf ของแหล่งกำเนิด ซึ่งสามารถถูกได้รับโดยตรงมาจากโมเดลความน่าจะเป็น

ตัวอย่าง การสร้างแท็ก

พิจารณาแหล่งกำเนิดในตัวอย่างข้างต้น ระบุตัวแปรสุ่ม $x(a_i) = i$ สมมติเราต้องการที่จะเข้ารหัส **1 3 2 1** จากโมเดลความน่าจะเป็นเรารู้ว่า $F_x(k) = 0, k \le 0$ $F_x(1) = 0.8$, $F_x(2) = 0.82$, $F_x(k) = 1, k < 3$ เราสามารถใช้ 2.8 และ 2.9 อนุกรมถูกระบุขอบบนและล่างของช่วงที่บรรจุแท็กเริ่มด้วย $u^{(0)}$ สู่ 1 และ $l^{(0)}$ สู่ 0

ส่วนประกอบตัวแรกของอนุกรม **1** ได้ผลเป็น

$$l^{(1)} = 0 + (1-0)0 = 0$$

$$u^{(1)} = 0 + (1-0)(0.8) = 0.8$$

นั้นคือแท็กสำหรับอนุกรม **1 3** คือ [0.8) รากที่ **2** คือ 3

$$l^{(2)} = 0 + (0.8-0)F_x(2) = 0.8 \times 0.82 = 0.656$$

$$u^{(2)} = 0 + (0.8-0)F_x(3) = 0.8 \times 1.0 = 0.8$$

ก่อนหน้านี้ช่วงที่บรรจุแท็กสำหรับอนุกรม **1 3** คือ $[0.656, 0.8)$ รากที่ 3 คือ 2

$$l^{(3)} = 0.656 + (0.8-0.656)F_x(1) = 0.7712$$

$$u^{(3)} = 0.656 + (0.8-0.656)F_x(2) = 0.77408$$

และช่วงสำหรับแท็กคือ

$$l^{(4)} = 0.7712 + (0.77408-0.7712)F_x(0) = 0.7712$$

$$u^{(4)} = 0.7712 + (0.77408-0.1152)F_x(1) = 0.773504$$

แท็กสามารถถูกสร้างโดย

$$\bar{T}_x(x) = \frac{0.7712 + 0.773504}{2} = 0.772352$$

จำไว้ว่าช่วงที่เสร็จแล้วในแต่ละช่วง จะถูกบรรจุอยู่ในช่วงก่อนหน้านี้ ถ้าเรายกตัวอย่างสมการที่ถูกใช้เพื่อที่จะสร้างช่วง เราจะเห็นว่ามันจะเป็นกรณีเสมอ คุณสมบัตินี้จะถูกใช้ถอดอักขระของแท็กได้ ผลลัพธ์ที่ไม่สามารถกำหนดได้ของขบวนการนี้ คือ ถ้าช่วงเล็กลงเรื่อยๆ เราก็ต้องการความแม่นยำสูงขึ้นเพื่อที่จะแก้ปัญหานี้ สูตรการรีสเกลต้องถูกนำมาช่วย ในข้อ 2.3.3.6 เราจะอธิบายการรีสเกล เพื่อที่จะสามารถช่วยปัญหานี้ได้

2.3.3.3 การถอดรหัสแท็ก

เราใช้เวลาในการพิจารณามากเพื่อแสดงว่าอนุกรมสามารถไม่เหมือนกันได้อย่างไรในการให้ข้อมูลจำนวนน้อย อย่างไรก็ตามแท็กจะไม่มีค่าอะไร ถ้าไม่สามารถถอดรหัสได้ด้วยการคำนวณน้อยๆ โชคดีที่การถอดรหัสแท็กนั้นเป็นเรื่องง่ายเหมือนการสร้างมัน เราสามารถเห็นทั้งหมดนี้ผ่านตัวอย่าง

<u>ตัวอย่าง</u> การถอดรหัสแท็ก

นำแท็กที่ได้รับมาจากตัวอย่างการสร้างแท็ก ลองพยายามที่จะรับอนุกรมที่ได้โดยแสดงแท็กที่เราจะลองพยายามจำลองการเข้ารหัสในแต่ละลำดับ เพื่อที่จะทำการถอดรหัส เมื่อค่าแท็กเป็น 0.772352 ช่วงที่บรรจุค่าแท็กนี้เป็นสับเซ็ตของทุกช่วงที่ได้รับในการเข้ารหัส สูตรการถอดรหัสของเราจะเป็นการถอดรหัสส่วนที่เล็กสุดในอนุกรม ในแต่ละวิธีซึ่งขอบเขตบนและล่างจะบรรจุค่าแท็กอยู่เสมอ สำหรับแต่ละ k เราเริ่มด้วย $l^{(0)} = 0$ และ $u^{(0)} = 1$ หลังจากการถอดรหัสค่าตัวแรกของอนุกรม x_1 และ ขอบเขตบนและล่าง จะกลายเป็น

$$l^{(1)} = 0 + (1-0)F_x(x_1 - 1) = F_x(x_1 - 1)$$

$$u^{(1)} = 0 + (1-0)F_x(x_1) = F_x(x_1)$$

อีกคำพูดหนึ่งคือช่วงบรรจุค่าแท็กคือ $\left[F_x(x_1-1), F_x(x_1)\right)$ เราต้องหาค่าของ x_1 ซึ่ง 0.772352 วางอยู่ในช่วง $\left[F_x(x_1-1), F_x(x_1)\right)$ ถ้าเราเลือก $x_1 = 1$ ช่วงคือ $[0, 0.8)$ ถ้าเราเลือก $x_1 = 2$ ช่วงคือ $[0.8, 0.82)$ ถ้าเราเลือก $x_1 = 3$ ช่วงคือ $[0.82, 1.0)$ เราเลือก $x_1 = 1$ ตอนนี้เราทำซ้ำขบวนการเดิมสำหรับการหาค่ารากตัวที่สอง และใช้ x_2 ในการอัพเดท ค่าของ $l^{(1)}$ และ $u^{(1)}$

$$l^{(2)} = 0 + (0.8-0)F_x(x_2 - 1) = 0.8F_x(x_2 - 1)$$

$$u^{(2)} = 0 + (0.8-0)F_x(x_2) = 0.8F_x(x_2)$$

ถ้าเราเลือก $x_2 = 1$ ช่วงที่ได้ คือ $[0, 0.64)$ ซึ่งไม่ได้บรรจุค่าแท็ก ก่อนหน้านี้ x_2 ไม่สามารถเป็น 1 ถ้าเราเลือก $x_2 = 2$ ช่วงที่ได้ คือ $[0.64, 0.656)$ ซึ่งไม่บรรจุค่าแท็ก ถ้าเราเลือก $x_2 = 3$ ช่วงที่ได้ คือ $[0.656, 0.8)$ ซึ่งระบุค่าแท็ก 0.772352 ก่อนหน้านี้ รากตัวที่สองคือ 3 เราสามารถอัพเดท ค่าของ $l^{(2)}$ และ $u^{(2)}$ และหารากx$_3$ ซึ่งจะให้ช่วงที่บรรจุอยู่ในแท็ก

$$l^{(3)} = 0.656 + (0.8 - 0.656)F_x(x_3 - 1) = 0.656 + 0.144 \times F_x(x_3 - 1)$$

$$u^{(3)} = 0.656 + (0.8 - 0.656)F_x(x_3) = 0.656 + 0.144 \times F_x(x_3)$$

อย่างไรก็ตามการแสดง $u^{(3)}$ และ $l^{(3)}$ ทำให้รูปแบบยุ่งยาก เพื่อให้การเปรียบเทียบง่ายขึ้น เราสามารถลบค่าของ $l^{(2)}$ จาก ขอบเขตบนและขอบเขตล่างทั้งสองได้และเราสามารถหาค่าของx_3 สำหรับช่วง $[0.144 \times F_x(x_3 - 1), 0.144 \times F_x(x_3))$ ซึ่งบรรจุค่า 0.772352-0.656 = 0.116352 หรือเราสามารถทำง่ายๆโดยแบ่งค่าแท็กที่อยู่ที่ 0.116352 โดยใช้ 0.14 เพื่อให้ค่าเป็น 0.808 และหาค่าของ x_3 ซึ่ง 0.808 อยู่ในช่วง $[F_x(x_3 - 1), F_x(x_3))$ เราสามารถพบว่าค่าเดียวของ x_3 ซึ่งนี้เป็น 2 ได้และแทนที่ 2 สำหรับ x_3 และ อัพเดทสมการ เราสามารถอัพเดทค่าสำหรับ $u^{(3)}$ และ $l^{(3)}$ เราสามารถหาค่าx_4 โดยการคำนวณขอบเขตบนและขอบเขตล่าง

$$l^{(4)} = 0.7712 + (0.77408 - 0.7712)F_x(x_4 - 1) = 0.7712 + 0.0288 \times F_x(x_4 - 1)$$

$$u^{(4)} = 0.7712 + (0.77408 - 0.7712)F_x(x_4) = 0.7712 + 0.0288 \times F_x(x_4)$$

เราสามารถลบ $l^{(3)}$ จากแท็กเพื่อที่จะได้ค่า 0.772352 – 0.7712 = 0.001152 และหาค่าของ x_4 ซึ่งช่วง $\left[0.00288 \times F_X(x_4 - 1), 0.00288 \times F_X(x_4)\right)$ที่บรรจุค่า 0.001152 เพื่อจะทำให้การเปรียบเทียบนั้นง่ายขึ้นเราสามารถแบ่งค่าที่อยู่ใน $[0.00288 \times F_x(x_4 - 1), 0.144 \times F_x(x_4))$ โดยใช้ 0.00288 เพื่อที่จะได้ค่า 0.4 และหาค่าของx_4 สำหรับ 0.4 ที่อยู่ในช่วง $[F_x(x_4 - 1), F_x(x_4))$ เราสามารถพบว่าค่าที่ได้คือ $x_4 = 1$ และเราต้องถอดรหัสอนุกรมทั้งหมดออกมา จำไว้ว่าเราต้องรู้ความยาวของอนุกรมก่อนเพื่อที่เราจะได้รู้ว่าเมื่อไหร่เราจะหยุด

จากตัวอย่างข้างบนเราสามารถสรุปการถอดรหัสเป็น

1. เริ่มด้วยให้ $l^{(0)} = 0$ $u^{(0)} = 1$
2. สำหรับแต่ละค่า k หา $t^* = (tag - l^{(k-1)} / u^{(k-1)} - l^{(k-1)})$
3. หาค่า x_k สำหรับแต่ละ $F_x(x_k - 1) \le t^* < F_x(x_k)$
4. อัพเดท $l^{(k)}$ $u^{(k)}$
5. ทำต่อไปเรื่อยๆจนถอดรหัสหมด

มี 2 วิธี ที่จะรู้ว่าเมื่อไหร่ทั้งอนุกรมได้ถูกถอดรหัสหมดแล้ว ตัวถอดรหัสต้องรู้ความยาวของอนุกรม ในแต่ละกรณีขบวนการถอดรหัสต้องหยุดเมื่อสัญลักษณ์ทุกตัวได้รับครบแล้ว ซึ่งเป็นอีกวิธีหนึ่งเพื่อที่จะรู้ว่าอนุกรมทั้งหมดได้ถูกถอดรหัสหมดแล้ว

2.3.3.4 การสร้างโค้ดไบนารี

การใช้อัลกอริธึม (algorithm) นี้ได้อธิบายไว้ในหัวข้อก่อนหน้านี้ เราสามารถได้รับค่าแท็กสำหรับชุดข้อมูล x ด้วยวิธีการคำนวณที่มีประสิทธิภาพ อย่างไรก็ตามโค้ดสำหรับลำดับนั้นคือสิ่งที่เราต้องการที่จะรู้ค่าของมันจริงๆ เราต้องหาไบนารีโค้ดซึ่งเป็นตัวแทนของชุดข้อมูล x ที่มีเอกลักษณ์เฉพาะและด้วยวิธีที่มีประสิทธิภาพ

เราพูดได้ว่าค่าแท็กนั้นจะอยู่ในรูปแบบที่เป็นเอกลักษณ์เฉพาะสำหรับชุดข้อมูล หมายความว่าตัวแทนที่เป็นไบนารีของค่าแท็กนั้นมีรูปแบบเป็นไบนารีโค้ดที่มีเอกลักษณ์เฉพาะของชุดข้อมูล อย่างไรก็ตามเราจะจัดวางได้อย่างไม่มีข้อจำกัดบนค่าในหน่วยของช่วงที่ค่าแท็กสามารถมีค่าอยู่ได้ ตัวแทนที่เป็นไบนารีของบางค่าจะยาวได้อย่างไม่จำกัด ซึ่งเป็นเหตุให้แม้ว่าโค้ดที่ได้จะเป็นเอกลักษณ์เฉพาะ แต่บางทีมันก็จะไม่มีประสิทธิภาพ การจะทำให้โค้ดมีประสิทธิภาพนั้น ตัวแทนที่เป็นไบนารีนั้นจะต้องถูกตัดให้เล็กลง แต่ถ้าเราตัดให้เล็กลงแล้วตัวแทนนั้นยังคงให้ผลลัพธ์ของโค้ดเป็นเอกลักษณ์เฉพาะอีกหรือไม่ และสุดท้ายผลลัพธ์ของโค้ดยังมีประสิทธิภาพหรือไม่ และจำนวนเฉลี่ยของบิทต่อสัญลักษณ์นั้นจะไกลหรือใกล้จากค่าเอนโทรปีขนาดไหน โดยเราจะทดสอบคำถามเหล่านี้ทั้งหมดในหัวข้อถัดไป

แม้ว่าถ้าเราแสดงโค้ดที่มีเอกลักษณ์เฉพาะและประสิทธิภาพได้ แต่วิธีที่อธิบายถึงจุดนี้จะไม่สามารถปฏิบัติได้จริง ในหัวข้อ 2.4.2.2 เราจะอธิบายอัลกอริธึมที่ปฏิบัติได้จริงมากกว่าเดิมเพื่อการสร้างโค้ดเลขคณิตสำหรับชุดข้อมูล

2.3.3.5 ความมีเอกลักษณ์เฉพาะและประสิทธิภาพของโค้ดเลขคณิต

$\overline{T}_X(x)$ คือค่าที่อยู่ในช่วง [0,1) ไบนารีโค้ดสำหรับ $\overline{T}_X(x)$ สามารถได้รับโดยการนำตัวแทนที่เป็นไบนารีของค่านี้และการตัดค่าให้เท่ากับ $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ บิท

<u>ตัวอย่าง</u>

พิจารณาแหล่ง A ซึ่งสร้างอักษรจากขนาดพยัญชนะ 4 ตัวอักษร

$$A = \{a_1, a_2, a_3, a_4\}$$

ด้วยค่าความน่าจะเป็น

$$P(a_1) = \frac{1}{2}, P(a_2) = \frac{1}{4}, P(a_3) = \frac{1}{8}, P(a_4) = \frac{1}{8}$$

ไบนารีโค้ดสำหรับแหล่งข้อมูลที่สามารถสร้างได้ตามตารางที่ 2.7 โดยปริมาณ $[\lfloor \bar{T}_X(x) \rfloor_{l(x)}$, $\lfloor \bar{T}_X(x) \rfloor_{l(x)} + \frac{1}{2^{l(x)}})$ สามารถหาได้โดยการใช้สมการที่ 2.6 ตัวแทนที่เป็นไบนารีของ $\bar{T}_X$ ถูกตัดให้เหลือเป็น $\left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ บิท เพื่อที่จะได้รับไบนารีโค้ด

สัญลักษณ์	F_X	$\bar{T}_X$	*ค่าไบนารี*	$\left\lceil \log \frac{1}{P(x)} \right\rceil + 1$	*โค้ด*
1	.5	.25	.010	2	01
2	.75	.625	.101	3	101
3	.875	.8125	.1101	4	1101
4	1.0	.9375	.1111	4	1111

ตารางที่ 2.7 โค้ดไบนารีสำหรับพยัญชนะ 4 ตัว

เราจะแสดงว่าโค้ดที่ได้รับในวิธีการนี้คือโค้ดที่เป็นเอกลักษณ์เฉพาะและสามารถถอดรหัสได้ อย่างแรกเราจะแสดงว่าโค้ดนี้เป็นเอกลักษณ์เฉพาะและมันจะแสดงว่าการถอดรหัสนั้นก็จะได้เป็นเอกลักษณ์เฉพาะด้วย

จำไว้ว่าขณะที่เราใช้ $\bar{T}_X(x)$ ในฐานะที่ใช้เป็นค่าแท็กสำหรับชุดข้อมูล x ค่าต่างๆในช่วง $[F_X(x-1), F_X(x))$ จะเป็นค่าซึ่งชี้เฉพาะซึ่งเป็นเอกลักษณ์ ดังนั้นเพื่อจะแสดงว่าโค้ด $\lfloor \bar{T}_X(x) \rfloor_{l(x)}$ เป็นเอกลักษณ์เฉพาะ ทั้งหมดที่เราต้องการจะทำนั้นก็เพื่อจะแสดงว่ามันบรรจุอยู่ในช่วง$[F_X(x-1), F_X(x))$ เพราะเราตัดตัวแทนที่เป็นไบนารีของ $\bar{T}_X(x)$ ก็จะกลายเป็น $\lfloor \bar{T}_X(x) \rfloor_{l(x)}$, $\lfloor \bar{T}_X(x) \rfloor_{l(x)}$ จะน้อยกว่าหรือเท่ากับค่า $\bar{T}_X(x)$ เราสามารถเจาะจงเข้าไปอีกได้ว่า

$$0 \le \bar{T}_X(x) - \lfloor \bar{T}_X(x) \rfloor_{l(x)} < \frac{1}{2^{l(x)}} \qquad (2.10)$$

ในขณะที่ $\bar{T}_X(x)$ มีค่าน้อยกว่า $F_X(x)$

$$\lfloor \bar{T}_X(x) \rfloor_{l(x)} < F_X(x)$$

เพื่อที่จะแสดงว่า $\lfloor \bar{T}_X(x) \rfloor_{l(x)} \ge F_X(x-1)$ เริ่มสังเกตจาก

$$\frac{1}{2^{l(x)}} = \frac{1}{2^{\left\lceil \log \frac{1}{P(x)} \right\rceil + 1}}$$

$$< \frac{1}{2^{\log\frac{1}{P(x)}+1}}$$

$$= \frac{1}{2\frac{1}{P(x)}}$$

$$= \frac{P(x)}{2}$$

จาก (2.6) เราจะได้ว่า

$$\frac{P(x)}{2} = \bar{T}_X(x) - F_X(x-1)$$

ดังนั้น

$$\bar{T}_X(x) - F_X(x-1) > \frac{1}{2^{l(x)}} \qquad (2.11)$$

รวม (2.10) และ (2.11) เราจะได้เป็น

$$\left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)} \geq F_X(x-1) \qquad (2.12)$$

ดังนั้นโค้ด $\left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)}$ คือ ตัวแทนที่มีเอกลักษณ์เฉพาะของ $\bar{T}_X(x)$

เพื่อแสดงว่าโค้ดนี้สามารถถอดรหัสได้โดยที่มีความเป็นเอกลักษณ์เฉพาะ เราจะต้องแสดงถึงโค้ดที่เป็นโค้ดส่วนหน้า (prefix code) นั่นคือ ไม่มีกลุ่มรหัสที่เป็นส่วนหน้าของอีกกลุ่มหนึ่ง เพราะอย่างนี้โค้ดส่วนหน้านั้นจึงสามารถถอดรหัสได้โดยมีความเป็นเอกลักษณ์เฉพาะเสมอ โดยถ้าเราแสดงได้ว่าโค้ดเลขคณิตนั้นเป็นโค้ดส่วนหน้าแล้ว เราก็จะสามารถบอกได้ว่ามันสามารถถอดรหัสได้โดยมีความเป็นเอกลักษณ์เฉพาะอยู่ด้วย ถ้าให้ค่า a อยู่ในช่วง [0,1) ด้วยตัวแทนที่เป็นไบนารี n บิท $[a_1a_2\ldots a_n]$ และค่า b ที่มีตัวแทนที่เป็นไบนารีคือ$[b_1b_2\ldots b_n]$ ในฐานะที่เป็นโค้ดส่วนหน้า ดังนั้น b ต้องถูกวางอยู่ในช่วง [a , $a+\frac{1}{2^n}$] (ดูปัญหาที่ 1)

ถ้า x และ y เป็นชุดข้อมูล 2 ชุดที่แตกต่างกัน เราจะรู้ว่า $\left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)}$ และ $\left\lfloor \bar{T}_X(y) \right\rfloor_{l(y)}$ วางอยู่ในสองช่วงที่ไม่เชื่อมต่อกันคือ $[F_X(x-1), F_X(x))$ และ $[F_X(y-1), F_X(y))$ ดังนั้นถ้าหากสามารถแสดงได้ว่าสำหรับชุดข้อมูล x ช่วง$[\left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)}, \left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)} + \frac{1}{2^{l(x)}})$ จะวางอยู่ภายในช่วง

ของ $[F_X(x-1), F_X(x))$ ทั้งหมด สิ่งนี้จะหมายความว่าโค้ดสำหรับชุดข้อมูลหนึ่งๆ ไม่สามารถเป็นโค้ดส่วนหน้าของโค้ดสำหรับชุดข้อมูลอื่นๆได้

เราได้แสดงว่า $\lfloor \bar{T}_X(x) \rfloor_{l(x)} \geq F_X(x-1)$ ไปเรียบร้อยแล้ว ดังนั้นเราจึงเหลือหน้าที่ที่ต้องแสดงว่า

$$F_X(x) - \lfloor \bar{T}_X(x) \rfloor_{l(x)} > \frac{1}{2^{l(x)}}$$

สิ่งนี้จะทำให้ได้ว่า

$$F_X(x) - \lfloor \bar{T}_X(x) \rfloor_{l(x)} > F_X(x) - \bar{T}_X(x)$$

$$= \frac{P(x)}{2}$$

$$> \frac{1}{2^{l(x)}}$$

โค้ดนี้เป็นโค้ดส่วนหน้าทั่วไป โดยการนำตัวแทนที่เป็นไบนารีของ $\bar{T}_X(x)$ และตัดมันให้มีความยาวเป็น $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ บิท เราก็จะได้โค้ดที่สามารถถอดรหัสได้โดยที่มีความเป็นเอกลักษณ์เฉพาะ

ขณะที่โค้ดนี้สามารถถอดรหัสได้โดยมีความเป็นเอกลักษณ์เฉพาะแล้ว จะมาดูว่าประสิทธิภาพของมันจะเป็นอย่างไร เราจะแสดงให้เห็นว่าจำนวนของบิท $l(x)$ ที่ต้องการที่จะนำไปใช้ $F_X(x)$ ด้วยความแม่นยำที่มากพอ ซึ่งโค้ดสำหรับค่าของ x ที่แตกต่างกันคือ

$$l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$$

จำไว้ว่า $l(x)$ คือ จำนวนของบิทที่ต้องการที่จะใช้ในการเข้ารหัสของชุดข้อมูล x ทั้งหมด ดังนั้นความยาวเฉลี่ยของโค้ดเลขคณิตสำหรับชุดข้อมูลที่มีความยาว m จะแทนโดย

$$l_A(m) = \sum P(x) l(x) \qquad (2.13)$$

$$= \sum P(x) \left[\left\lceil \log \frac{1}{P(x)} \right\rceil + 1 \right] \qquad (2.14)$$

$$< \sum P(x)\left[\log \tfrac{1}{P(x)} + 1 + 1\right] \quad (2.15)$$

$$= -\sum P(x)\log P(x) + 2\sum P(x) \quad (2.16)$$

$$= H\left(X^{(m)}\right) + 2 \quad (2.17)$$

ถ้าให้ความยาวเฉลี่ยมีค่ามากกว่าค่าเอ็นโทรปีเสมอ จะได้ขอบเขตของ $l_A(m)$ เป็น

$$H\left(X^{(m)}\right) \le l_A(m) \le H\left(X^{(m)}\right) + 2$$

ความยาวต่อสัญลักษณ์ l_A หรืออัตราของโค้ดเลขคณิตซึ่งคือค่า $\frac{l_A(m)}{m}$ ดังนั้นขอบเขตของ l_A คือ

$$\frac{H\left(X^{(m)}\right)}{m} \le l_A \le \frac{H\left(X^{(m)}\right)}{m} + \frac{2}{m} \quad (2.18)$$

$$H\left(X^{(m)}\right) = mH(X) \quad (2.19)$$

ดังนั้น

$$H(X) \le l_A \le H(X) + \frac{2}{m} \quad (2.20)$$

โดยการเพิ่มความยาวของชุดข้อมูล เราจะสามารถรับประกันอัตราให้ใกล้ค่าเอ็นโทรปี้เท่าที่เราต้องการได้

2.3.3.6 อัลกอริธึมในการที่จะนำไปประยุกต์ให้ใช้ได้จริง

อัลกอริธึมการเข้ารหัสและการถอดรหัสที่เราอธิบายมานั้นไม่สามารถที่จะนำไปใช้จริงตามแนวทางที่อธิบายไว้ได้ จำไว้ว่าเหตุผลและหลักการสำหรับการใช้ค่าในช่วง [0,1) ในฐานะที่เป็นค่าแท็กคือต้องมีค่าไม่จำกัดในช่วงนั้น อย่างไรก็ตามในทางปฏิบัติค่านั้นสามารถที่จะเป็นเอกลักษณ์เฉพาะได้โดยแสดงเป็นตัวอย่างบนเครื่องซึ่งถูกจำกัดโดยจำนวนบิทที่มากที่สุด เราสามารถใช้สำหรับการแทนค่าได้ พิจารณาค่าของ $l^{(n)}$ และ $u^{(n)}$ ในตัวอย่างการสร้างแท็ก ในขณะที่ n มีค่ามากขึ้นค่าเหล่านี้จะยิ่งแคบมากขึ้นเรื่อยๆ นั้นหมายความว่า มันเหมาะสมที่จะใช้แทนช่วงย่อยทั้งหมดที่มีความเป็นเอกลักษณ์เฉพาะ

และเราจำเป็นต้องเพิ่มความเที่ยงตรง (precision) ในขณะที่ความยาวของชุดข้อมูลเพิ่มมากขึ้น ในระบบที่ความเที่ยงตรงนั้นจำกัด ค่าทั้ง 2 นั้นจะถูกกำหนดขอบเขตลู่เข้าสู่ค่าหนึ่ง และเราจะสูญเสียข้อมูลทั้งหมดที่เกี่ยวกับชุดข้อมูลจากจุดที่ซึ่ง 2 ค่านี้ลู่เข้าสู่จุดหนึ่ง เพื่อจะหลีกเลี่ยงสถานการณ์นี้ เราจึงจำเป็นที่จะต้องปรับระดับของช่วงเอาใหม่ (rescale) อย่างไรก็ตามเราต้องทำมันในหนทางซึ่งจะช่วยรักษาข้อมูลที่จะใช้ส่งคล้ายกับการจะทำให้ส่วนการเข้ารหัสเพิ่มขึ้น (incremental encoding) เพื่อที่จะส่งส่วนของโค้ดที่เป็นชุดข้อมูลนั้นถูกพบเรื่อยๆ มันดีกว่าที่จะต้องรอจนกระทั่งชุดข้อมูลทั้งหมดถูกพบแล้วก่อนการส่งบิทแรกออกไป อัลกอริธึมที่เราจะอธิบายในหัวข้อนี้จะดูแลปัญหาในเรื่องการปรับระดับที่สอดคล้องกันและการเข้ารหัสที่เพิ่มขึ้น

ในขณะที่ช่วงนั้นจะค่อยๆแคบลงกว่าเดิม มันจะเป็นไปได้มากที่ช่วงที่บรรจุค่าแท็กจะถูกจำกัดอยู่ในครึ่งของช่วงบนหรือครึ่งช่วงล่างของช่วง [0,1) (เราจะแบ่งด้วยกรณีตามนี้ ซึ่งจะไม่เป็นจริงในเวลาต่อมา) ค่าแท็กที่จุดนี้จะถูกจำกัดอยู่ในครึ่งช่วงบนหรือครึ่งช่วงล่างของช่วงหนึ่งหน่วยเสมอ ดังนั้นบิทที่สำคัญที่สุด (most significant bit) ของค่าแท็กจะกลายเป็นตัวกำหนดไปเลย ถ้าค่าแท็กนั้นถูกกำหนดอยู่ในช่วงครึ่งบนของช่วงหนึ่งหน่วย ซึ่งค่าแท็กก็จะคือค่าที่มากกว่าหรือเท่ากับ 0.5 และบิทที่สำคัญที่สุดของค่าแท็กก็ต้องเป็น 1 แต่ถ้าค่าแท็กนั้นถูกกำหนดอยู่ในช่วงครึ่งล่างของช่วงหนึ่งหน่วย ซึ่งค่าแท็กก็จะคือค่าที่น้อยกว่า 0.5 และบิทที่สำคัญที่สุดของค่าแท็กก็จะต้องเป็น 0 ในสถานการณ์นี้เราสามารถชี้ถึงตัวถอดรหัสซึ่งค่าแท็กที่ถูกจำกัดโดยการส่ง 1 สำหรับครึ่งบน และส่ง 0 สำหรับช่วงครึ่งล่าง ซึ่งค่าไบนารีที่เราส่งก็จะคือบิทที่สำคัญที่สุดของค่าแท็ก

แต่ก่อนที่ตัวเข้ารหัสและตัวถอดรหัสรู้ถึงส่วนครึ่งที่บรรจุค่าแท็ก เราสามารถเพิกเฉยอีกครั้งหนึ่งของช่วงหนึ่งหน่วย ที่ไม่ได้บรรจุค่าแท็กและสนใจเฉพาะแต่ส่วนครึ่งที่บรรจุค่าแท็ก แต่ในฐานะที่โค้ดเลขคณิตนั้นเป็นแบบความเที่ยงตรงจำกัด เราจึงสามารถทำให้ดีที่สุดได้โดยการย้ายครึ่งช่วงที่บรรจุค่าแท็กไปให้เต็มช่วง [0,1) การย้ายทำได้ดังนี้

$$E_1 : [0, 0.5) \rightarrow [0,1);\ E_1(x) = 2x \qquad (2.21)$$

$$E_2 : [0.5, 1) \rightarrow [0,1);\ E_2(x) = 2(x - 0.5) \qquad (2.22)$$

การย้ายทั้งสองเงื่อนไขนี้ เราจะสูญเสียข้อมูลทั้งหมดเกี่ยวกับบิทตัวแรกไป อย่างไรก็ตามสิ่งนี้จะไม่จำเป็นเพราะเราได้ส่งบิทนั้นไปยังตัวถอดรหัสเรียบร้อยแล้ว เราสามารถทำกระบวนการนี้ได้อย่างต่อเนื่อง การสร้างบิทอีกตัวหนึ่งของค่าแท็กทุกๆครั้งที่ช่วงของค่าแท็กนั้นถูกจำกัดทั้งหมดอยู่ในช่วงครึ่งใดครึ่งหนึ่งของช่วงหนึ่งหน่วย กระบวนการสร้างบิทของค่าแท็กนี้ปราศจากการรอคอยที่จะต้องดูชุดข้อมูลทั้งหมดและถูกเรียกว่า การเข้ารหัสที่เพิ่มขึ้น

ตัวอย่าง : การสร้างค่าแท็กด้วยการขยายช่วง

ลองกลับไปดูที่ตัวอย่างการสร้างแท็ก จำได้ว่าเราต้องการที่จะเข้ารหัสชุดข้อมูล **1 3 2 1** ค่าความน่าจะเป็นมีดังนี้ $P(a_1) = 0.8$, $P(a_2) = 0.02$, $P(a_3) = 0.18$ เริ่มต้นด้วย $u^{(0)}$ เป็น 1 และ $l^{(0)}$ เป็น 0 ตัว (element)แรกของชุดข้อมูลคือ 1 จะทำให้เกิดผลลัพธ์ตามนี้

$$l^{(1)} = 0 + (1-0)(0) = 0$$

$$u^{(1)} = 0 + (1-0)(0.8) = 0.8$$

ช่วง [0,0.8) ไม่ถูกจำกัดอยู่ทั้งครึ่งบนและครึ่งล่างของช่วงหนึ่งหน่วย ดังนั้นเราจะรับค่าของตัวต่อไป ซึ่งค่าตัวที่ 2 ของชุดข้อมูลคือ 3 ผลลัพธ์ที่ได้เป็นตามนี้

$$l^{(2)} = 0 + (0.8-0)F_X(2) = 0.8 \times 0.82 = 0.656$$

$$u^{(2)} = 0 + (0.8-0)F_X(3) = 0.8 \times 1.0 = 0.8$$

ช่วง [0.656,0.8) ทั้งหมดบรรจุอยู่ในครึ่งบนของช่วงหนึ่งหน่วย ดังนั้นเราจะส่งโค้ดไบนารี 1 และทำการปรับช่วงเป็น

$$l^{(2)} = 2 \times (0.656 - 0.5) = 0.312$$

$$u^{(2)} = 2 \times (0.8 - 0.5) = 0.6$$

ค่าตัวที่ 3 คือค่า 2 จะทำให้เกิดผลลัพธ์ได้เป็น

$$l^{(3)} = 0.312 + (0.6-0.312)F_X(1) = 0.312 + 0.288 \times 0.8 = 0.5424$$

$$u^{(3)} = 0.312 + (0.6-0.312)F_X(2) = 0.312 + 0.288 \times 0.82 = 0.54816$$

ช่วงสำหรับค่าแท็กคือ [0.5424,0.54816) ซึ่งทั้งหมดบรรจุอยู่ในครึ่งบนของช่วงหนึ่งหน่วย เราจะทำการส่ง 1 และทำการปรับช่วงต่อไปอีกครั้ง

$$l^{(3)} = 2 \times (0.5424 - 0.5) = 0.0848$$

$$u^{(3)} = 2 \times (0.54816 - 0.5) = 0.09632$$

ช่วงนี้ทั้งหมดจะถูกบรรจุในครึ่งล่างของช่วงหนึ่งหน่วย ดังนั้นเราจะส่ง 0 และใช้การย้ายตามเงื่อนไข E_1 ทำการปรับช่วงได้เป็น

$$l^{(3)} = 2\times(0.0848) = 0.1696$$

$$u^{(3)} = 2\times(0.09632) = 0.19264$$

ทั้งหมดของช่วงนี้ยังคงอยู่ในครึ่งล่างของช่วงหนึ่งหน่วยอยู่ ดังนั้นเราจะส่ง 0 อีกครั้งและปรับช่วงอีกทีหนึ่งเป็น

$$l^{(3)} = 2\times(0.1696) = 0.3392$$

$$u^{(3)} = 2\times(0.19264) = 0.38528$$

เพราะช่วงที่บรรจุค่าแท็กยังคงอยู่ในครึ่งล่างของช่วงหนึ่งหน่วย เราจะส่ง 0 อีกตัวและปรับช่วงอีกครั้งหนึ่งได้เป็น

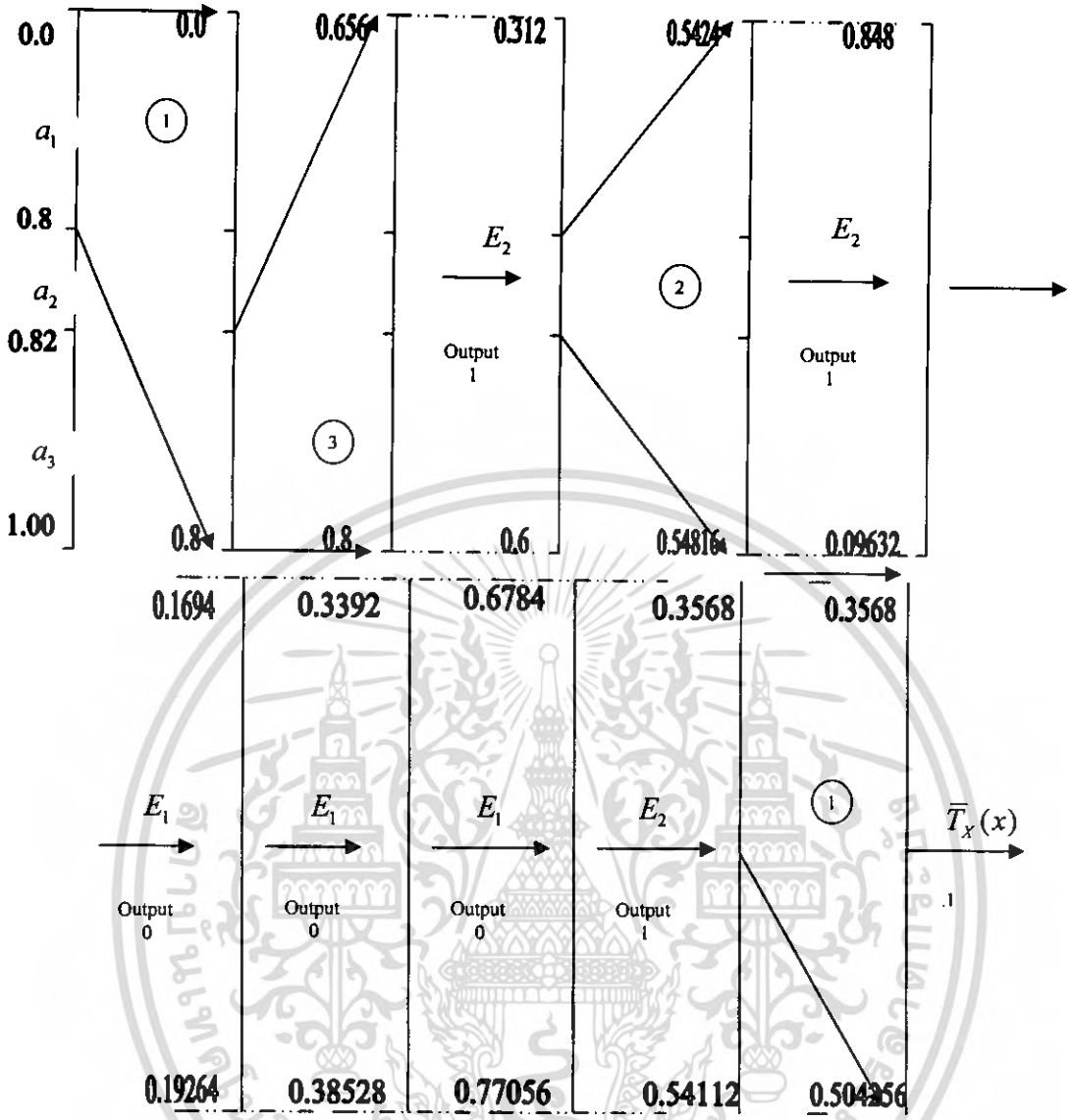
$$l^{(3)} = 2\times(0.3392) = 0.6784$$

$$u^{(3)} = 2\times(0.38528) = 0.77056$$

ตอนนี้ช่วงที่บรรจุค่าแท็กนั้นเป็นครึ่งบนของช่วงหนึ่งหน่วยแทน ดังนั้นเราจะส่ง 1 และปรับช่วงโดยการย้ายตามเงื่อนไข E_2 ได้เป็น

$$l^{(3)} = 2\times(0.6784-0.5) = 0.3568$$

$$u^{(3)} = 2\times(0.77056-0.5) = 0.54112$$



รูปที่ 2.5 แสดงการเข้ารหัสของการเข้ารหัสเลขคณิตแบบเลขฐานสอง

ที่แต่ละขั้น (stage) เราจะทำการส่งบิทที่สำคัญที่สุด ซึ่งคือตัวเดียวกันกับทั้งขอบเขตส่วนบนและขอบเขตส่วนล่างของช่วงของค่าแท็ก ถ้าบิทที่สำคัญที่สุดในขอบเขตส่วนบนและขอบเขตส่วนล่างเป็นตัวเดียวกันแล้วค่าของบิทที่จะส่งนี้จะเหมือนกันกับบิทที่สำคัญที่สุดของค่าแท็กเช่นกัน ดังนั้นเราจะส่งบิทที่สำคัญที่สุดของขอบเขตส่วนบนและขอบเขตส่วนล่างของค่าแท็กไปเรื่อยๆตราบใดที่มันยังคงเหมือนกันอยู่โดยเราจะส่งในรูปตัวแทนที่เป็นไบนารีของค่าแท็ก กระบวนการการปรับขยายช่วงนั้นสามารถทำได้โดยการเลื่อนบิทไปทางซ้ายซึ่งจะทำให้บิทที่สำคัญเป็นอันดับสองกลายเป็นบิทที่สำคัญที่สุด

ตามด้วยค่าตัวสุดท้ายขอบเขตบนและล่างของช่วงที่บรรจุค่าแท็ก คือ

$$l^{(4)} = 0.3568 + (0.54112 - 0.3568)F_X(0) = 0.3568 + 0.18422 \times 0.0 = 0.3568$$

$$u^{(4)} = 0.3568 + (0.54112 - 0.3568)F_X(1) = 0.3568 + 0.18422 \times 0.8 = 0.504256$$

ที่จุดนี้ถ้าเราต้องการหยุดการเข้ารหัส เราจำเป็นที่จะต้องทำรูปแบบสถานะสุดท้ายของค่าแท็กที่ตัวรับ โดยเราสามารถทำได้โดยการส่งตัวแทนที่เป็นไบนารีของค่าต่างๆในช่วงแท็กสุดท้าย ทั่วๆไปค่านี้จะเป็น $l^{(n)}$ ในเฉพาะตัวอย่างนี้มันเหมาะสมที่จะใช้ค่า 0.5 ตัวแทนที่เป็นไบนารีของ 0.5 คือ 0.10$\cdots\cdots$. ดังนั้นเราจะส่ง 1 ตามด้วย 0 มากเท่าที่ต้องการ โดยตามความยาวของการนำไปใช้

สังเกตว่าขนาดของช่วงแท็กที่ชั้นนี้จะประมาณ 64 เท่าของขนาดตอนที่เราใช้อัลกอริธึมที่ไม่ได้ดัดแปลงอะไรเลย ดังนั้นเทคนิคนี้จะแก้ปัญหาความเที่ยงตรงที่จำกัดได้ เราจะพบว่าบิทซึ่งเราส่งไปกับแต่ละการย้ายตามเงื่อนไขนั่นได้สร้างค่าแท็กออกมาโดยตัวของมันเอง ซึ่งเป็นค่าที่พอใจตามความต้องการของเราสำหรับการเข้ารหัสที่เพิ่มขึ้น ชุดข้อมูล,ที่เป็นไบนารีถูกสร้างขึ้นระหว่างกระบวนการเข้ารหัสในตัวอย่างก่อนหน้านี้คือ 1100011 เราสามารถพิจารณาง่ายๆว่าสิ่งนี้คือปริมาณที่ขยายออกที่เป็นไบนารีของค่าแท็ก ค่าที่เป็นไบนารี .1100011 จะได้เท่ากับค่าฐานสิบ คือ 0.7734375 เมื่อมองย้อนกลับไปดูที่ตัวอย่างการสร้างแท็ก สังเกตได้ว่าค่านี้วางอยู่ภายในช่วงของแท็กสุดท้าย ดังนั้นเราสามารถใช้ค่านี้ที่จะถอดรหัสของชุดข้อมูลได้

อย่างไรก็ตามเราต้องการที่จะทำการถอดรหัสที่เพิ่มขึ้นได้ดีเท่ากับการเข้ารหัสที่เพิ่มขึ้น ซึ่งจะต้องเริ่มตอบคำถาม 3 คำถามนี้

1. เราจะเริ่มการถอดรหัสอย่างไร
2. เราจะทำการถอดรหัสให้ต่อเนื่องได้อย่างไร
3. เราจะหยุดการถอดรหัสอย่างไร

คำถามที่ 2 นั้นง่ายที่จะตอบที่สุด การเริ่มต้นการถอดรหัส เราต้องทำการเลียนแบบอัลกอริธึมของการเข้ารหัสทั้งหมด นั้นคือเรารู้วิธีที่จะทำการถอดรหัสได้อย่างต่อเนื่อง เพื่อที่จะเริ่มกระบวนการถอดรหัส เราจำเป็นต้องมีข้อมูลเพียงพอที่จะถอดรหัสสัญลักษณ์ตัวแรกได้อย่างถูกต้อง เพื่อที่จะรับประกันการถอดรหัสที่ถูกต้อง จำนวนบิทที่รับควรจะเป็นจุดทศนิยมที่น้อยกว่าช่วงของค่าแท็กที่น้อยที่สุด เราสามารถกำหนดจำนวนของบิทที่ต้องการก่อนที่เราจะเริ่มกระบวนการถอดรหัสโดยยึดอยู่บนพื้นฐานของช่วงของค่าแท็กที่น้อยที่สุด ลองใช้ข้อมูลนี้เพื่อดูวิธีที่เราถอดรหัสข้อความที่สร้างในตัวอย่างก่อนหน้านี้

<u>ตัวอย่าง</u>

ช่วงของแท็กที่เล็กที่สุดคือ [0.8,0.82) ซึ่งมีขนาด 0.02 ดังนั้นจำนวนของบิท k ที่ต้องการใช้ในการถอดรหัสเพื่อให้สามารถเริ่มปฏิบัติการได้อย่างถูกต้อง ควรจะได้ $2^{-k} < 0.02$ หรือ $k = 6$ ขณะที่ใน

การเข้ารหัส เราเริ่มด้วยค่าเริ่มต้นของ $u^{(0)}=0$ และ $l^{(0)}=0$ ชุดข้อมูลของบิทที่รับได้คือ $110001100\cdots\cdots 0$ ค่า 6 บิทแรกแปลงเป็นฐานสิบจะได้ว่าค่าแท็กเท่ากับ 0.765625 ซึ่งหมายความว่าค่าแรกของชุดข้อมูลคือ 1 จะได้รับผลลัพธ์ตามนี้

$$l^{(1)}=0+(1-0)(0)=0$$

$$u^{(1)}=0+(1-0)(0.8)=0.8$$

ช่วง [0,0.8) จะถูกจำกัดอยู่ในทั้งส่วนครึ่งบนและครึ่งล่างของช่วงหนึ่งหน่วย ดังนั้นเราจะดำเนินการต่อ ค่าแท็ก 0.765625 วางในช่วงบน 18 % ของช่วง [0,0.8) ดังนั้นค่าที่ 2 ของชุดข้อมูลนี้คือ 3 เราจะได้ค่าช่วงของแท็กเป็นตามนี้

$$l^{(2)}=0+(0.8-0)F_X(2)=0.8\times 0.82=0.656$$

$$u^{(2)}=0+(0.8-0)F_X(3)=0.8\times 1.0=0.8$$

ช่วง [0.656,0.8) ทั้งหมดจะบรรจุอยู่ในครึ่งบนของช่วงหนึ่งหน่วย ที่การเข้ารหัสเราจะส่งบิท 1 และทำการปรับช่วงใหม่ แต่ที่ตัวถอดรหัสเราจะเลื่อน 1 ออกจากบัฟเฟอร์ของตัวรับและเคลื่อนบิทถัดไปเข้ามาอยู่ใน 6 บิทของค่าแท็กแทน เราจะได้ผลลัพธ์ช่วงของค่าแท็กเป็น

$$l^{(2)}=2\times(0.656-0.5)=0.312$$

$$u^{(2)}=2\times(0.8-0.5)=0.6$$

ในขณะที่ทำการเลื่อนบิทแล้วค่าแท็กที่เราได้รับคือ 0.546875 เมื่อเราเปรียบเทียบค่านี้กับช่วงของค่าแท็ก เราจะสามารถพบได้ว่าค่านี้วางอยู่ในระดับ 80-82 % ของช่วงของค่าแท็ก ดังนั้นเราจะถอดรหัสค่าตัวต่อไปของชุดข้อมูลซึ่งคือ 2 เราสามารถหาผลลัพธ์ช่วงของแท็กได้เป็น

$$l^{(2)}=0.312+(0.6-0.312)F_X(1)=0.312+0.288\times 0.8=0.5424$$

$$u^{(2)}=0.312+(0.8-0.312)F_X(2)=0.312+0.288\times 0.82=0.54816$$

ในฐานะที่ช่วงของค่าแท็กทั้งหมดนั้นยังคงบรรจุอยู่ในครึ่งบนของช่วงหนึ่งหน่วย เราสามารถปรับช่วงโดยใช้ตามเงื่อนไข E_2 จะได้ว่า

$$l^{(3)} = 2\times(0.5424 - 0.5) = 0.0848$$

$$u^{(3)} = 2\times(0.54816 - 0.5) = 0.09632$$

เราจะเลื่อนบิทออกจากค่าแท็กและเลื่อนบิทถัดไปเข้ามาแทน ตอนนี้ค่าแท็กจะคือ 000110 ช่วงของมันทั้งหมดจะบรรจุอยู่ครึ่งล่างของช่วงหนึ่งหน่วย ดังนั้นเราประยุกต์ตามเงื่อนไข E_1 และเลื่อนอีก 1 บิท ขอบเขตบนและขอบเขตล่างของช่วงแท็กจะกลายเป็น

$$l^{(3)} = 2\times(0.0848) = 0.1696$$

$$u^{(3)} = 2\times(0.09632) = 0.19264$$

และค่าแท็กจะกลายเป็น 001100 ช่วงทั้งหมดยังคงบรรจุอยู่ในครึ่งล่างของช่วงหนึ่งหน่วย ดังนั้นเราจะเลื่อน 0 อีกตัวออกไป และจะได้ค่าแท็กเป็น 011000 และปรับช่วงอีกครั้งได้เป็น

$$l^{(3)} = 2\times(0.1696) = 0.3392$$

$$u^{(3)} = 2\times(0.19264) = 0.38528$$

เพราะช่วงที่บรรจุค่าแท็กยังคงอยู่ในช่วงครึ่งล่างของช่วงหนึ่งหน่วย เราจึงเลื่อน 0 ออกอีก 1 ตัวจากค่าแท็ก ซึ่งค่าแท็กจะกลายเป็น 110000 และปรับช่วงอีกครั้งหนึ่งได้เป็น

$$l^{(3)} = 2\times(0.3392) = 0.6784$$

$$u^{(3)} = 2\times(0.38528) = 0.77056$$

ตอนนี้ช่วงที่บรรจุค่าแท็กถูกบรรจุอยู่ในส่วนครึ่งบนของช่วงหนึ่งหน่วยทั้งหมด ดังนั้นเราจะเลื่อน 1 ออกจากค่าแท็ก และปรับช่วงโดยการใช้การย้ายตามเงื่อนไข E_2 ได้

$$l^{(3)} = 2\times(0.6784 - 0.5) = 0.3568$$

$$u^{(3)} = 2\times(0.77056 - 0.5) = 0.54112$$

ตอนนี้เปรียบเทียบค่าแท็กกับช่วงของแท็กเพื่อจะถอดรหัสค่าตัวสุดท้าย แท็กเป็น 100000 ซึ่งแปลงเป็นค่าเลขฐานสิบได้เป็น 0.5 ค่านี้วางบน 80% แรกของช่วง ดังนั้นเราจะถอดรหัสได้ค่า 1

เราจะแสดงว่าการใช้การขยายช่วง จะสามารถดูแลปัญหาความเที่ยงตรงที่ถูกจำกัดได้ นั่นคือมีความเป็นไปได้ของขอบเขตบนและล่างของช่วงที่กำลังลู่เข้าสู่ค่าหนึ่ง อย่างไรก็ตามเราสามารถจัดการได้กับกรณีเหล่านี้เท่านั้น ซึ่งคือช่วงของค่าแท็กอยู่ในไม่ครึ่งล่างก็ครึ่งบนของช่วงหนึ่งหน่วย ตอนนี้เราจะพบวิธีที่จะดูแลกรณีที่ซึ่งช่วงของค่าแท็กคร่อมอยู่กึ่งกลางของช่วงหนึ่งหน่วย โดยตรวจสอบดูว่าถ้าช่วงของแท็กบรรจุอยู่ในช่วง [0.25,0.75) กรณีนี้จะเกิดขึ้นเมื่อ $l^{(n)}$ มีค่ามากกว่า 0.25 และ $u^{(n)}$ มีค่าน้อยกว่า 0.75 เมื่อเกิดเหตุการณ์แบบนี้ขึ้น เราจะขยายช่วงของแท็กโดยใช้การย้ายตาม

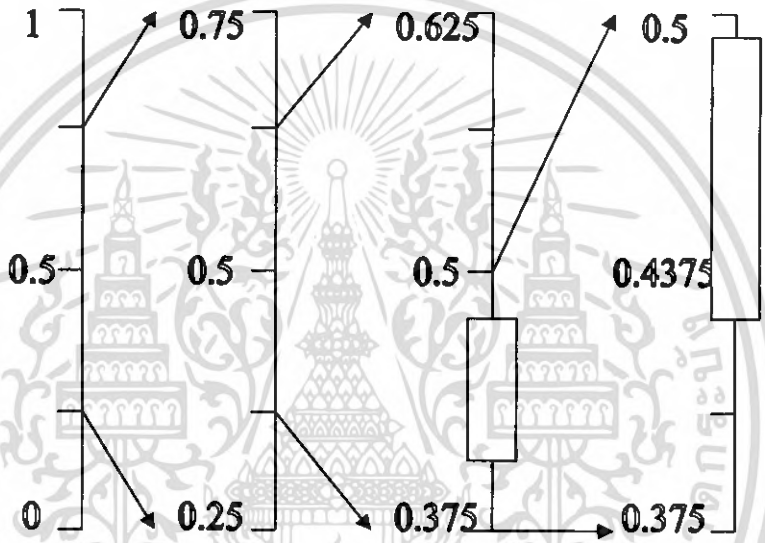
$$E_3 : [0.25, 0.75) \to [0,1); \quad E_3(x) = 2(x - 0.25) \tag{2.23}$$

เราใช้บิท 1 เพื่อส่งข้อมูลเกี่ยวกับการย้ายตามเงื่อนไข E_2 และบิท 0 เพื่อส่งข้อมูลเกี่ยวกับการย้ายตามเงื่อนไข E_1 เราจะส่งข้อมูลที่เกี่ยวกับการย้ายตามเงื่อนไข E_3 ให้กับตัวถอดรหัสได้อย่างไร เราจะใช้วิธีการที่แตกต่างออกไปในกรณีนี้ ที่เวลาเกิดการย้ายตามเงื่อนไข E_3 เราจะไม่ส่งข้อมูลใดๆออกไปยังตัวถอดรหัส แต่จะทำโดยการบันทึกแบบง่ายๆถึงข้อเท็จจริงของกรณีนี้ที่ตัวเข้ารหัส สมมติว่าหลังจากนี้ช่วงของแท็กถูกจำกัดอยู่ที่ครึ่งบนของช่วงหนึ่งหน่วย ที่จุดนี้เราจะใช้การย้ายตามเงื่อนไข E_2 และส่ง 1 ไปยังเครื่องรับ สังเกตว่าช่วงของแท็กที่ขั้นนี้จะอยู่ที่อย่างน้อย 2 เท่าของที่มันจะอยู่ถ้าไม่ใช้การย้ายตามเงื่อนไข E_3 มากไปกว่านั้นขอบเขตบนของช่วงของแท็กจะน้อยกว่า 0.75 ดังนั้นถ้าการย้ายตามเงื่อนไข E_3 ไม่ได้นำมาวางอย่างถูกต้องหลังจากการย้ายตามเงื่อนไข E_2 ช่วงของแท็กจะบรรจุอยู่ในครึ่งล่างทั้งหมดของช่วงหนึ่งหน่วย ที่จุดนี้เราจะใช้การย้ายตามเงื่อนไข E_1 เพื่อส่ง 0 ไปยังเครื่องรับ ในความเป็นจริงอิทธิพลของการย้ายตามเงื่อนไข E_3 สามารถลอกเลียนแบบได้ง่ายกว่าที่ตัวถอดรหัสตามการย้ายตามเงื่อนไข E_2 กับการย้ายตามเงื่อนไข E_1 ที่ตัวเข้ารหัส หลังจากที่เราส่ง 1 หลังจากที่เราส่ง 1 เพื่อประกาศว่าเป็นการย้ายตามเงื่อนไข E_2 เราจะส่ง 0 เพื่อให้ตัวถอดรหัสพบร่องรอยที่เปลี่ยนไปในช่วงแท็กที่ตัวถอดรหัส ถ้าการขยายระดับครั้งแรกหลังจากเกิดเหตุการณ์ย้ายตามเงื่อนไข E_3 เป็นการย้ายตามเงื่อนไข E_1 เราจะทำในการตรงข้ามแทนนั่นคือเราจะส่ง 0 ประกาศถึงการย้ายตามเงื่อนไข E_1 ตามด้วย1 เพื่อเลียนแบบผลกระทบของการย้ายตามเงื่อนไข E_3 ที่ตัวเข้ารหัส

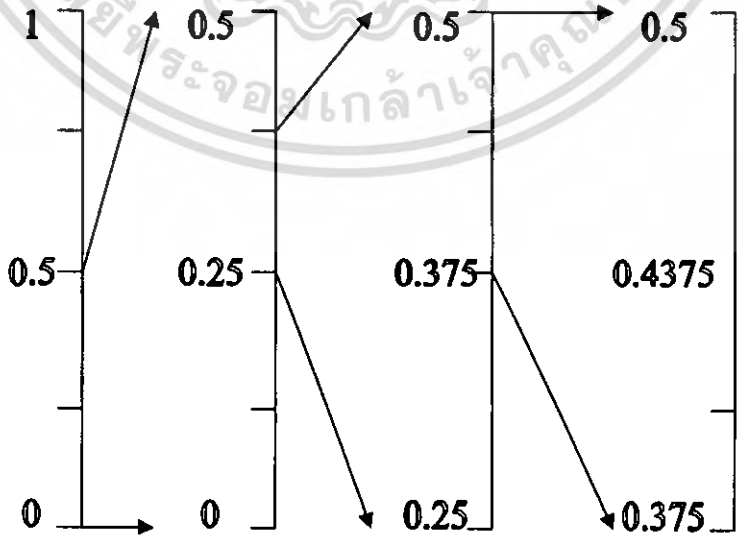
จะเกิดอะไรขึ้นถ้าเราต้องเกิดการย้ายตามเงื่อนไข E_3 อย่างต่อเนื่องที่ตัวเข้ารหัส เราจะแค่เก็บร่องรอยของจำนวนของการเกิดการย้ายตามเงื่อนไข E_3 และทำการส่งบิทที่ตรงกันข้ามมากมายไปเรื่อยๆ จนกว่าจะพบการย้ายตามเงื่อนไข E_1 หรือ E_2 ครั้งแรก ถ้าเราไปเจอการย้ายตามเงื่อนไข E_3 3 ครั้งที่ตัวเข้ารหัส ตามด้วยการย้ายตามเงื่อนไข E_2 เราจะส่ง 1 ตามด้วย 0 3 ตัว ในอีกด้านหนึ่งถ้าเราเจอการย้ายตามเงื่อนไข E_1 หลังจากการย้ายตามเงื่อนไข E_3 เราก็จะส่ง 0 ตามด้วย 1 3 ตัว ในเมื่อตัวถอดรหัสเลียนแบบตัวเข้ารหัส การย้ายตามเงื่อนไข E_3 ก็จะถูกประยุกต์ใช้ที่ตัวถอดรหัสด้วยเช่นกัน เมื่อช่วงของแท็กบรรจุอยู่ในช่วง [0.25,0.75]

การเชื่อมโยงนี้ถูกวาดเป็นภาพประกอบในรูปที่ 2.6 และ 2.7 ให้ A เป็นตัวหนังสือ A:={a,b,c,d,e} ใช้รูปแบบการกระจายความน่าจะเป็น รูปที่ 2.6 แสดงตัว c เป็นสัญลักษณ์ตัวแรก ช่วงที่เกี่ยวข้องด้วยคือ [0.4,0.6) ซึ่งครอบคลุมควอเตอร์ที่2 และ 3 เพราะฉะนั้นเราสามารถทำสเกล E3 และ ผลของช่วงที่ครอบคลุมควอเตอร์ที่2 และ 3 อีกทีหลังจากสเกล E3 ตัวถัดไป ช่วงจะครอบคลุมมากกว่า 2 ควอเตอร์ ดังนั้นเราต้องทำด้วยสัญลักษณ์ b ตัวถัดไป ผลของช่วงคือ [0.375,0.5) ซึ่งบรรจุค่าที่สมบูรณ์ในช่วงต่ำกว่าตรงกลาง สเกล E1 เก็บค่า 0 ในอนุกรมที่ออกมา ตามด้วย 1 บิท สำหรับ สเกล E3

รูปที่ 2.7 ทำไมการเก็บค่า 011 ถึงถูก เริ่มด้วยช่วง [0,1) เราทำสเกล E1 และ E2 ตามค่าบิทที่เก็บไว้ หมายความว่าหนึ่งสเกล E1 และ สอง สเกล E2 ผลของช่วงเหมือนกับรูปที่ 2.6 ซึ่งแสดงผลของ 2 สเกล E3 ตามโดยสเกล E1



รูปที่ 2.6 รูปแบบการใช้การวัดแบบ E3



รูปที่ 2.7 สำหรับการเปรียบเทียบ – ไม่มีการวัดแบบ E3

2.3.3.7 การนำไปใช้ให้เกิดผลโดยจำนวนเต็ม

2.3.3.7.1 การนำการเข้ารหัสไปใช้

สิ่งแรกที่ต้องพิจารณาคือความยาวของคำที่จะนำไปใช้ ให้เป็นความยาว m จะระบุค่าที่สำคัญในช่วง [0,1) ไปเป็นค่าเลขฐานสองในช่วง 2^m จุด 0 จะถูกระบุค่าเป็น

$$\overbrace{00...0}^{m \text{ times}},$$

ค่า 1 ก็จะถูกระบุค่าเป็น

$$\overbrace{11...1}^{m \text{ times}}.$$

ค่า 0.5 ก็จะถูกระบุค่าเป็น

$$1\overbrace{00...0}^{m \text{ times}}.$$

สมการที่อัพเดทแล้วจะยังคงคล้ายกับสมการ (2.8) และ (2.9) ในการเข้ารหัสเลขคณิตแบบจำนวนเต็ม จะต้องแทนที่ $F_X(x)$ ในสมการเหล่านี้

ค่า n_j แสดงถึงจำนวนครั้งที่สัญลักษณ์ j เกิดขึ้นในลำดับของความยาว Total Count ดังนั้น $F_X(k)$ จะถูกประมาณเป็น

$$F_X(k) = \frac{\sum_{i=1}^{k} n_i}{Total\ Count} \tag{2.24}$$

จะได้

$$Cum\_Count(k) = \sum_{i=1}^{k} n_i$$

จะสามารถแสดงสมการ (2.8) และ (2.9) เป็น

$$l^{(n)} = l^{(n-1)} + \left\lfloor \left(u^{(n-1)} - l^{(n-1)} + 1\right) \frac{Cum\_Count(x_n - 1)}{Total\ Count} \right\rfloor \tag{2.25}$$

$$u^{(n)} = l^{(n-1)} + \left\lfloor \left(u^{(n-1)} - l^{(n-1)} + 1\right)\frac{Cum\_Count(x_n)}{Total\ Count} \right\rfloor - 1 \qquad (2.26)$$

เมื่อ $x^{(n)}$ เป็นสัญลักษณ์ตัวที่ n ที่จะถูกเข้ารหัส $\lfloor x \rfloor$ จะเป็นจำนวนเต็มที่มากที่สุดที่น้อยกว่าหรือเท่ากับ x และการบวกและการลบหนึ่งนั้นเพื่อจัดการกับผลกระทบของการเข้ารหัสเลขคณิตแบบจำนวนเต็ม

เพราะวิธีที่ระบุค่าจุดสุดท้ายหรือจุดกึ่งกลางของช่วงหนึ่งหน่วยเมื่อทั้ง $l^{(n)}$ และ $u^{(n)}$ อยู่ในช่วงครึ่งบนหรือครึ่งล่างของช่วงพร้อมกัน บิทนำของ $l^{(n)}$ และ $u^{(n)}$ จะเหมือนกัน ถ้าบิทนำหรือบิทสูงสุด (most significant bit)เป็น 1 ดังนั้นค่าที่ระบุช่วง(tag interval)จะอยู่ในช่วงครึ่งบนของช่วง [00...0,11...1] ถ้าบิทสูงสุดเป็น 0 ค่าที่ระบุช่วงจะอยู่ในช่วงครึ่งล่าง จะนำการระบุตำแหน่ง E_1 และ E_2 มาใช้ในเงื่อนไขปกติ ซึ่งจะต้องเลื่อนบิทสูงสุดออกและเลื่อน 1 เข้ามาในรหัสจำนวนเต็มของ $u^{(n)}$ และเลื่อนบิท 0 เข้ามาในรหัส $l^{(n)}$ ตัวอย่างเช่น สมมติค่า m เป็น 6 $u^{(n)}$ เท่ากับ 54 $l^{(n)}$ เท่ากับ 33 เลขฐานสองที่แสดงค่า $u^{(n)}$ และ $l^{(n)}$ จะเป็น 110110 และ 100001 ตามลำดับ สังเกตว่าค่าบิทสูงสุดของทั้งสองจะเป็น 1 เมื่อทำตามวิธีข้างต้นจะส่งบิท 1 ออก(เพื่อส่งหรือเก็บไว้)และเลื่อน 1 เข้าสำหรับ $u^{(n)}$ และเลื่อน 0 สำหรับ $l^{(n)}$ จะได้ค่าใหม่ของ $u^{(n)}$ เป็น 101101 หรือ 45 และค่าใหม่สำหรับ $l^{(n)}$ เป็น 000010 หรือ 2 ซึ่งการระบุตำแหน่งของ E_2 ก็จะใช้วิธีเดียวกันนี้ จะเห็นว่าการระบุตำแหน่ง E_1 จะถูกแสดงโดยใช้กระบวนการเดียวกัน

หากต้องการระบุตำแหน่ง E_3 จะควบคุมบิทสูงสุดลำดับที่สอง(second most significant bit)ของ $u^{(n)}$ เป็น 0 และบิทสูงสุดลำดับที่สองของ $l^{(n)}$ เป็น 1 หมายความว่าค่าที่ระบุช่วงจะอยู่บนช่วงครึ่งกลางของช่วง[00...0,11...1] การระบุตำแหน่ง E_3 สำหรับการนำไปใช้จะคอมพลีเมนท์บิทสูงสุดลำดับที่สองใน $u^{(n)}$ และ $l^{(n)}$ และเลื่อนออกทางซ้าย เลื่อน 1 เข้าใน $u^{(n)}$ และเลื่อน 0 เข้าใน $l^{(n)}$ จำนวนการระบุตำแหน่ง E_3 จะเก็บไว้ใน Scale3

<u>ตัวอย่าง</u>

การเข้ารหัสลำดับ **1 3 2 1** ด้วยค่าในตาราง เริ่มแรกจำเป็นต้องเลือกความยาว m ของคำ $Cum\_Count(1)$ และ $Cum\_Count(2)$ จะต่างกันเพียง 1 จากเดิมที่ค่า Cum_Count จะถูกแปลงเป็นจุดสุดท้ายของช่วงย่อย

$Count(1) = 40$	$Cum\_Count(0) = 0$	$Scale3 = 0$
$Count(2) = 1$	$Cum\_Count(1) = 40$	
$Count(3) = 9$	$Cum\_Count(2) = 41$	
$Total\_Count = 50$	$Cum\_Count(3) = 50$	

ตารางที่ 2.8 ค่าสำหรับตัวอย่างการเข้ารหัสเลขคณิต

เพื่อให้แน่ใจว่าค่าความยาวที่เลือกจะให้ช่วงที่เพียงพอที่จะแสดงความต่างระหว่างช่วงที่น้อยที่สุดได้ ซึ่งจะมีการหาสเกลใหม่เมื่อช่วงนั้นเล็กลง เพื่อที่จะให้แน่ใจว่าจุดสุดท้ายของช่วงยังคงชัดเจนอยู่ จึงจำเป็นต้องทำให้ทุกค่าในช่วงจาก 0 ถึง $Total\_Count$ (เหมือนกับ $Cum\_Count(3)$)ยังถูกแสดงอย่างเป็นเอกลักษณ์ในช่วงที่เล็กที่จะไม่เกิดการหาสเกลใหม่ เมื่อ $l^{(n)}$ อยู่ต่ำกว่าจุดกึ่งกลางของช่วงและ $u^{(n)}$ อยู่ที่ควอเตอร์ที่สามของช่วง หรือเมื่อ $u^{(n)}$ อยู่ที่จุดกึ่งกลางพอดีและ $l^{(n)}$ อยู่ต่ำกว่าควอเตอร์แรกของช่วง นั่นคือช่วงที่เล็กที่สุด $\left[l^{(n)}, u^{(n)}\right]$ จะเป็นหนึ่งในสี่ของช่วงค่า 2^m ที่มีดังนั้น m ควรมีค่ามากพอที่จะวางค่าที่เป็นเอกลักษณ์ระหว่าง 0 ถึง $Total\_Count$

สำหรับตัวอย่างนี้ หมายถึงช่วงความกว้างทั้งหมดต้องมากกว่า 200 ค่า m ที่เหมาะสมกับความต้องการจึงเป็น 8 จากค่า m นี้เราจะได้

$$l^{(0)} = 0 = (00000000)_2$$

$$u^{(0)} = 255 = (11111111)_2$$

เมื่อ $(...)_2$ เป็นเลขฐานสองที่แทนที่เลข

ส่วนแรกของลำดับที่จะเข้ารหัสคือ 1 จะได้

$$l^{(1)} = 0 + \left\lfloor \frac{256 \times Cum\_Count(0)}{50} \right\rfloor = 0 = (00000000)_2$$

$$u^{(1)} = 0 + \left\lfloor \frac{256 \times Cum\_Count(1)}{50} \right\rfloor - 1 = 203 = (11001011)_2$$

ส่วนต่อไปของลำดับคือ 3 :

$$l^{(2)} = 0 + \left\lfloor \frac{204 \times Cum\_Count(2)}{50} \right\rfloor = 167 = (10100111)_2$$

$$u^{(2)} = 0 + \left\lfloor \frac{204 \times Cum\_Count(3)}{50} \right\rfloor - 1 = 203 = (11001011)_2$$

บิทสูงสุดของ $l^{(n)}$ และ $u^{(n)}$ เป็น 1 ทั้งคู่ ดังนั้นเราจึงเลื่อนบิทนี้ออกและส่งไปยังตัวถอดรหัส ทุกบิทก็เลื่อนไปทางซ้ายหนึ่งบิทจะได้

$$l^{(2)} = (01001110)_2 = 78$$

$$u^{(0)} = (10010111)_2 = 151$$

สังเกตว่าหากบิทสูงสุดของลิมิทแตกต่างกัน บิทสูงสุดลำดับที่สองของขอบบนเป็น 0 ในขณะที่บิทสูงสุดลำดับที่สองของขอบล่างเป็น 1 นั่นจะเป็นเงื่อนไขสำหรับการระบุตำแหน่ง E_3 เราจะคอมพลีเมนท์บิทสูงสุดลำดับที่สองของลิมิททั้งคู่และเลื่อนหนึ่งบิทออกไปทางซ้าย เลื่อน 0 เข้ามาที่บิทต่ำสุดของ $l^{(2)}$ และเลื่อน 1 สำหรับ $u^{(2)}$ จะได้

$$l^{(2)} = (01001110)_2 = 28$$

$$u^{(0)} = (10101111)_2 = 175$$

จะเพิ่ม Scale3 ไปเป็นค่า 1
ส่วนต่อไปของลำดับคือ 2 จะอัพเดทลิมิทเป็น

$$l^{(3)} = 28 + \left\lfloor \frac{148 \times Cum\_Count(1)}{50} \right\rfloor = 146 = (10010010)_2$$

$$u^{(3)} = 28 + \left\lfloor \frac{148 \times Cum\_Count(2)}{50} \right\rfloor - 1 = 148 = (10010100)_2$$

บิทสูงสุดทั้งสองสามารถระบุค่าได้ ดังนั้นจึงเลื่อนบิท 1 ออกและเลื่อนมาทางซ้ายหนึ่งบิท

$$l^{(3)} = (00100100)_2 = 36$$

$$u^{(3)} = (01010011)_2 = 41$$

ขณะที่ Scale3 เป็น 1 เราจะส่ง 0 และลดค่า Scale3 ลงเป็น 0 บิทลำดับสูงสุดของขอบบนและขอบล่างก็เป็น 0 ทั้งคู่ จึงเลื่อนออกและส่ง 0 :

$$l^{(3)} = (01001000)_2 = 72$$

$$u^{(3)} = (01010011)_2 = 83$$

บิทลำดับสูงสุดเป็น 0 ทั้งคู่อีก จึงเลื่อนออกและส่ง 0 :

$$l^{(3)} = (10010000)_2 = 144$$

$$u^{(3)} = (10100111)_2 = 167$$

ตอนนี้บิทลำดับสูงสุดของทั้งคู่เป็น 1 ดังนั้นจึงเลื่อนออกและส่ง 1 ลิมิทจะกลายเป็น

$$l^{(3)} = (00100000)_2 = 32$$

$$u^{(3)} = (01001111)_2 = 79$$

อีกครั้งที่บิทลำดับสูงสุดเหมือนกัน จะเลื่อนออกและส่ง 0 :

$$l^{(3)} = (01000000)_2 = 64$$

$$u^{(3)} = (10011111)_2 = 159$$

ขณะนี้บิทลำดับสูงสุดนั้นแตกต่างกัน บิทสูงสุดลำดับที่สองของขอบบนเป็น 1 และขอบล่างเป็น 0 ซึ่งเป็นกรณีของการระบุตำแหน่ง E_3 นำการระบุตำแหน่ง E_3 มาใช้โดยทำคอมพลีเมนท์บิทสูงสุดลำดับที่สอง และเลื่อนหนึ่งบิทออกไปทางซ้าย จะได้

$$l^{(3)} = (00000000)_2 = 0$$

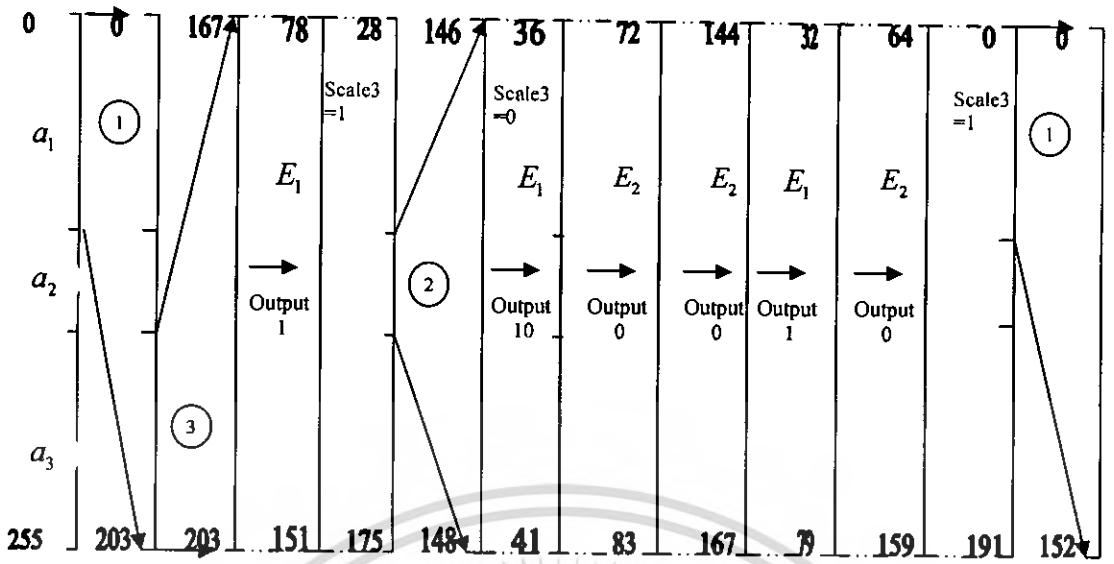
$$u^{(3)} = (10111111)_2 = 191$$

ส่วนถัดไปของลำดับที่จะเข้ารหัสคือ 1 ดังนั้น

$$l^{(4)} = 0 + \left\lfloor \frac{192 \times Cum\_Count(0)}{50} \right\rfloor = 0 = (00000000)_2$$

$$u^{(4)} = 0 + \left\lfloor \frac{192 \times Cum\_Count(1)}{50} \right\rfloor - 1 = 152 = (10011000)_2$$

กระบวนการเข้ารหัสจะดำเนินต่อไปดังนี้ ในจุดนี้เราได้สร้างลำดับเลขฐานสอง 1100010



รูปที่ 2.8 แสดงการเข้ารหัสของการเข้ารหัสเลขคณิตแบบจำนวนเต็ม

2.3.3.7.2 การนำไปใช้ของตัวถอดรหัส

การนำไปใช้ของตัวถอดรหัสเป็นเหมือนทั่วไป สำหรับการนำไปใช้แบบจำนวนเต็ม วิธีการถอดรหัสจะเป็นดังนี้

1. เริ่มต้น $l^{(0)} = 00...0$ และ $u^{(0)} = 11...1$
2. สำหรับแต่ละ k หาค่า $t^* = \left(tag - l^{(k-1)} + 1\right) / \left(u^{(k-1)} - l^{(k-1)} + 1\right)$
3. หาค่าของ $x^{(k)}$ สำหรับที่ $\frac{Cum\_Count(x^{(k)} - 1)}{Total\ Count} \le t^* < \frac{Cum\_Count(x^{(k)})}{Total\ Count}$
4. อัพเดท $u^{(k)}$ และ $l^{(k)}$
5. ดำเนินการต่อไปเรื่อยๆจนกว่าลำดับทั้งหมดถูกถอดรหัสแล้ว

2.3.4 การเปรียบเทียบระหว่างการเข้ารหัสแบบฮัฟแมน กับการเข้ารหัสเลขคณิต

เราอธิบายวิธีการของการเข้ารหัสตัวใหม่นี้ว่าสามารถทำการเข้ารหัสชุดข้อมูลของสัญลักษณ์แม้ว่าจะมีความยุ่งยากมากกว่าการเข้ารหัสแบบฮัฟแมน วิธีการเข้ารหัสนี้จะทำงานได้ดีนั้นขึ้นอยู่กับการนำไปใช้ สิ่งแรกลองใช้โค้ดนี้สำหรับแหล่งข้อมูลของการเข้ารหัสด้วยอันที่เรารู้ค่าของโค้ดแบบฮัฟแมน ดูที่ตัวอย่างแรกสุด ความยาวเฉลี่ยของโค้ดนี้คือ

$$l = 2 \times 0.5 + 3 \times 0.25 + 4 \times 0.125 + 4 \times 0.125$$

$$= 2.75 \text{ bits/symbol}$$

ทวนดูในก่อนหน้านี้ ซึ่งค่าเอ็นโทรปีของแหล่งข้อมูลคือ 1.75 bits/symbol และโค้ดฮัฟแมนก็ใช้ตามค่าเอ็นโทปี้นี้ เห็นได้ชัดว่าการเข้ารหัสเลขคณิตไม่เป็นแนวทางที่ดีถ้านำไปใช้ในการเข้ารหัสข้อความที่มีเพียงหนึ่งสัญลักษณ์ต่อหนึ่งครั้ง ลองทำซ้ำในตัวอย่างเดิมด้วยข้อความที่ประกอบด้วยสองสัญลักษณ์ (สังเกตว่าในแบบฝึกหัดเราจะไม่ใช้การเข้ารหัสเลขคณิตกับโค้ดที่มีชุดข้อมูลสั้น)

<u>ตัวอย่าง</u>

ถ้าเราเข้ารหัสสองสัญลักษณ์ต่อหนึ่งครั้ง ผลลัพธ์ของโค้ดจะถูกแสดงตามในตารางที่ 2.9 ความยาวเฉลี่ยต่อข้อความคือ 4.5 bits ดังนั้นการใช้สองสัญลักษณ์ต่อหนึ่งครั้งเราจะได้รับอัตราเป็น 2.25 bits/symbol(แน่นอนมันดีกว่า 2.75 bits/symbol แต่ยังคงไม่ดีเท่ากับอัตราที่ดีที่สุดคือ 1.75 bits/symbol) อย่างไรก็ตามเราจะพบว่าถ้าหากเราเพิ่มจำนวนของสัญลักษณ์ของข้อความมากขึ้นผลลัพธ์ที่ได้นั้นก็จะยิ่งดีขึ้นเรื่อยๆ

ข้อความ	*P(x)*	$\bar{T}_X(x)$	$\bar{T}_X(x)$ *ในรูปฐานสอง*	$\left\lceil \log \frac{1}{P(x)} \right\rceil + 1$	*โค้ด*
11	.25	.125	.001	3	001
12	.125	.3125	.0101	4	0101
13	.0625	.40625	.01101	5	01101
14	.0325	.46875	.01111	5	01111
21	.125	.5625	.1001	4	1001
22	.0625	.65625	.10101	5	10101
23	.03125	.703125	.101101	6	101101
24	.03125	.734375	.101111	6	101111
31	.0625	.78125	.11001	5	11001
32	.03125	.828125	.110101	6	101111
33	.015625	.8515625	.1101101	7	1101101
34	.015625	.8671875	.1101111	7	1101111
41	.0625	.90625	.11101	5	11101
42	.03125	.953125	.111101	6	111101
43	.015625	.9765625	.1111101	7	1111101
44	.015625	.984375	.1111111	7	1111111

ตารางที่ 2.9 รหัสเลขคณิต สำหรับ 2 สัญลักษณ์ต่อชุดข้อมูล

เราจะรู้ได้อย่างไรว่าจะใช้วิธีการเข้ารหัสเลขคณิตเพื่อจะให้ผลลัพธ์ที่ดีกว่าการเข้ารหัสแบบฮัฟแมนนั้นจะใช้เมื่อมีจำนวนสัญลักษณ์เท่าใด เราสามารถที่จะนึกโดยดูที่ขอบเขตของอัตราการเข้ารหัสขอบเขตบนความยาวเฉลี่ย l_A ของโค้ดแบบเลขคณิตคือ

$$H(X) \le l_A \le H(X) + \frac{2}{m}$$

มันไม่ต้องใช้สัญลักษณ์มากต่อชุดข้อมูลสำหรับโค้ดแบบเลขคณิตก็จะได้อัตราเข้ารหัสใกล้กับค่าเอ็นโทรปี อย่างไรก็ตามถ้าเราใช้ m สัญลักษณ์เท่ากัน อัตราเข้ารหัสของโค้ดแบบฮัฟแมนคือ

$$H(X) \le l_H \le H(X) + \frac{1}{m}$$

ข้อดีดูเหมือนว่าจะดูไม่จริงสำหรับโค้ดแบบฮัฟแมน แม้ว่าข้อดีจะลดลงเมื่อเพิ่มจำนวน m อย่างไรก็ตามจำไว้ว่าการสร้างชุดของโค้ดสำหรับชุดข้อมูลที่มีความยาว m การใช้กระบวนการฮัฟแมนต้องการการสร้างโค้ดทั้งหมดของชุดข้อมูลของความยาว m จะใช้วิธีการของฮัฟแมนซึ่งต้องใช้โค้ดทั้งหมดที่มีสำหรับชุดข้อมูลที่เป็นไปได้ทั้งหมดของความยาว m ถ้าขนาดของอักขระเป็น k และขนาดของโค้ดบุ๊กจะเป็น k^m ซึ่งถ้ากำหนดให้ค่าอย่างสมเหตุผลจะได้ค่าของ k=16 และ m=20 จะให้ขนาดของโค้ดบุ๊กเป็น 16^{20} สิ่งนี้ดูเหมือนจะไม่ใช่ทางเลือกที่เป็นไปได้นัก สำหรับกระบวนการถอดรหัสเลขคณิตเราจะไม่ต้องใช้ทั้งหมดของโค้ดบุ๊ก แต่เราจะทำให้โค้ดนั้นง่ายขึ้นโดยใช้ค่าแท็กแทนซึ่งจะต้องสอดคล้องกับชุดข้อมูล ดังนั้นมันจะสามารถนำมาใช้จริงได้สำหรับการสร้างโค้ดของชุดข้อมูลให้มีความยาว 20 หรือมากกว่านั้น ในทางปฏิบัติเราสามารถทำให้ m ใหญ่พอสำหรับตัวเข้ารหัสเลขคณิตและไม่พอสำหรับตัวเข้ารหัสฮัฟแมน สิ่งนี้หมายความว่าแหล่งข้อมูลที่เรามีทั้งหมดนี้ เราสามารถที่จะทำให้เข้าใกล้สู่เอ็นโทรปีด้วยการใช้การเข้ารหัสเลขคณิตมากกว่าการใช้การเข้ารหัสฮัฟแมน ยกเว้นว่าแหล่งข้อมูลนั้นมีค่าความน่าจะเป็นที่จะเกิดเลขยกกำลังสองซึ่งในกรณีนี้อักษรตัวเดียวของฮัฟแมน โค้ดสามารถที่จะได้ค่าเอ็นโทรปี และเราไม่สามารถจะใช้การเข้ารหัสเลขคณิตมาช่วยเพื่อให้ดีขึ้นกว่าเดิมได้ ไม่ว่าความยาวของชุดข้อมูลจะเป็นเท่าไหร่ก็ตาม

ปริมาณที่ได้มาจะขึ้นอยู่กับแหล่งข้อมูล จำได้ไหมว่าโค้ดฮัฟแมนนั้นเรารับประกันไว้ว่าจะได้อัตราอยู่ในช่วง $0.086+p_{max}$ ของเอนโทรปี ซึ่ง p_{max} คือค่าความน่าจะเป็นของตัวอักษร 1 ตัวในทั้งหมดที่น่าจะเหมาะสมที่สุดในอักขระ ถ้าขนาดของอักขระนั้นค่อนข้างใหญ่มาก และค่า ความน่าจะเป็นนั้นดูไม่น่าจะหาได้ ค่าความน่าจะเป็นสูงสุด p_{max} จะค่อนข้างเล็ก ในกรณีนี้ประโยชน์ของการเข้ารหัสเลขคณิตที่มีมากกว่าโค้ดฮัฟแมนนั้นจะน้อยลง และมันดูจะไม่มีค่าเพราะว่าการใช้การเข้ารหัสเลขคณิตนั้นดูจะซับซ้อนกว่าการเข้ารหัสฮัฟแมนมาก อย่างไรก็ตามมันก็ยังมีแหล่งข้อมูลอีกมากมาย ยกตัวอย่างเช่นแฟกซ์ ซึ่งมีขนาดอักขระเล็ก และความน่าจะเป็นจะไม่สมดุลมาก ในกรณีนี้นั้นการใช้การเข้ารหัสเลขคณิตจะมีประโยชน์ที่จะเพิ่มความซับซ้อนให้สมดุล

ข้อดีหลักของการเข้ารหัสเลขคณิตอีกอย่างก็คือ มันง่ายที่จะใช้ในระบบด้วยโค้ดเลขคณิตที่หลากหลาย ซึ่งข้อดีนี้ดูเหมือนจะขัดแย้ง เพราะว่าจากการที่เราเคยบอกว่าการเข้ารหัสเลขคณิตนั้นจะ

ซับซ้อนกว่าการเข้ารหัสฮัฟแมน เครื่องที่ใช้ในการคำนวณนั้นจะทำให้เกิดการเพิ่มขึ้นของความซับซ้อน เราจึงมีเครื่องที่ใช้ในการคำนวณเพื่อนำไปใช้ในโค้ดเลขคณิตหนึ่งๆ ทั้งหมดนี้เราต้องการที่จะนำไปใช้มากกว่าโค้ดเลขคณิตเดี่ยวๆ ซึ่งคือประโยชน์ที่ได้มากกว่าตารางความน่าจะเป็น ถ้าขนาดของอักขระของแหล่งข้อมูลนั้นเล็ก เช่นในกรณีของแหล่งข้อมูลที่เป็นไบนารี ซึ่งมันจะไปเพิ่มความซับซ้อนได้น้อยมากจริงๆ

ท้ายที่สุด มันง่ายมากที่จะปรับโค้ดเลขคณิตเมื่อเปลี่ยนข้อมูลทางสถิติที่จะส่ง เราจึงต้องทำการประมาณค่าความน่าจะเป็นของอักขระที่จะส่ง สิ่งนี้สามารถที่จะทำโดยการนับตัวอักษรของข้อมูลที่เราจะเข้ารหัส ไม่มีความจำเป็นที่จะต้องเก็บรักษากราฟรูปต้นไม้ เช่นเดียวกับโค้ดฮัฟแมนที่ปรับเปลี่ยนแล้วมากกว่านั้นไม่มีความจำเป็นที่ต้องสร้างโค้ดก่อน อย่างเช่นกรณีของโค้ดฮัฟแมน คุณสมบัตินี้อนุญาตให้แยกกระบวนการการเข้ารหัสและรูปแบบออกจากกัน ซึ่งไม่เหมาะที่จะใช้กับการเข้ารหัสฮัฟแมน การแยกกันนี้ทำให้มีความยืดหยุ่นที่ดีกว่าในการออกแบบระบบการบีบอัดข้อมูล

2.4 ภาษาวีเอชดีแอล

ความซับซ้อนและขนาดของระบบดิจิตอลในปัจจุบันได้เพิ่มมากขึ้นทุกขณะ ส่งผลให้มีการนำคอมพิวเตอร์เพื่อช่วยในการออกแบบมาใช้ในกระบวนการออกแบบฮาร์ดแวร์เพิ่มขึ้น อีกทั้งอุปกรณ์และวิธีการออกแบบใหม่ๆ ก็ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบมากขึ้นด้วย สำหรับภาษาบรรยายอุปกรณ์ฮาร์ดแวร์ เอชดีแอล (HDL: Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่องเพื่อช่วยในการปรับปรุงขบวนการออกแบบระบบดิจิตอลให้เป็นไปอย่างมีประสิทธิภาพ

2.4.1 ประวัติความเป็นมาของภาษาวีเอชดีแอล

วีเอชดีแอล ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และสามารถเขียนได้หลายรูปแบบด้วยเหตุผลนี้จึงทำให้ภาษาวีเอชดีแอล เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหาลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกรได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชั่นการทำงานของวงจรอย่างสังเขปโดยไม่คำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนี้วีเอชดีแอลยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มีความซับซ้อนได้ทั้งหมด ดังนั้น วีเอชดีแอลจึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งานเป็นอย่างยิ่ง สำหรับมาตรฐานของภาษาที่ใช้บรรยายพฤติกรรมวงจรหรือฮาร์ดแวร์ สามารถสรุปได้ดังนี้

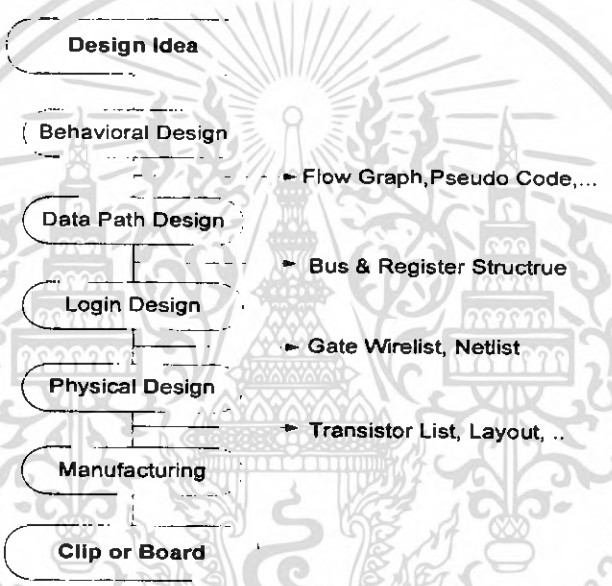
- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอลและมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่องคอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่น เดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรมภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า ภาษาโปรแกรมระดับสูง

2.4.2 การออกแบบระบบดิจิตอล

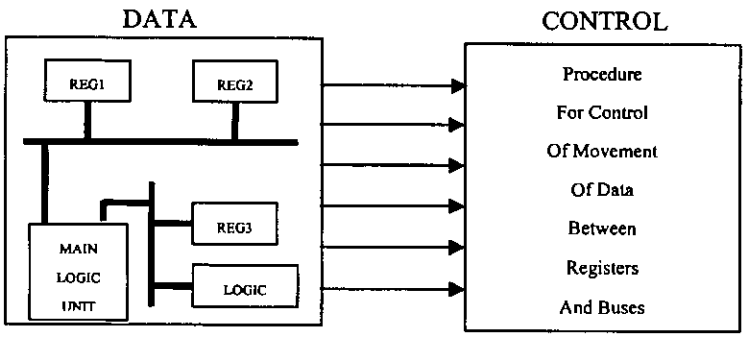
ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำเนิดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้จะต้องผ่านขั้นตอนต่างๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์ในแต่ละขั้นก่อนเข้าสู่กระบวนการออกแบบในขั้นต่อไป

รูปที่ 2.9 แสดงขั้นตอนปกติที่ใช้ในการออกแบบ แล้วทำการพัฒนาให้สามารถนำไปใช้ได้อย่างสมบูรณ์ ซึ่งภายในจะทำการสร้างรูปแบบเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจจะเป็นผังงาน ขั้นตอนนี้ผู้ออกแบบจำเป็นต้องสร้างรูปแบบระบบแสดงผล หรือรหัสคำสั่งเทียม (Psedo code) ก็ได้



รูปที่ 2.9 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์และวงจรลอจิกที่จำเป็นทั้งหมด เพื่อนำมาประกอบเป็นระบบที่สมบูรณ์ โดยแต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) กระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจรลอจิกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 2.10



รูปที่ 2.10 แสดงการออกแบบระบบเส้นทางข้อมูล

ขั้นตอนถัดมาเป็นการออกแบบวงจรลอจิกซึ่งจะเกี่ยวข้องกับการนำเกทดิจิตอลพื้นฐาน และฟลิปฟลอป (Flip – Flop) มาประกอบเป็นอุปกรณ์ย่อยต่าง ๆ เช่นรีจิสเตอร์เก็บข้อมูล บัสวงจรลอจิก และส่วนควบคุมฮาร์ดแวร์ ซึ่งผลลัพธ์ที่ได้ในขั้นตอนนี้จะเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง

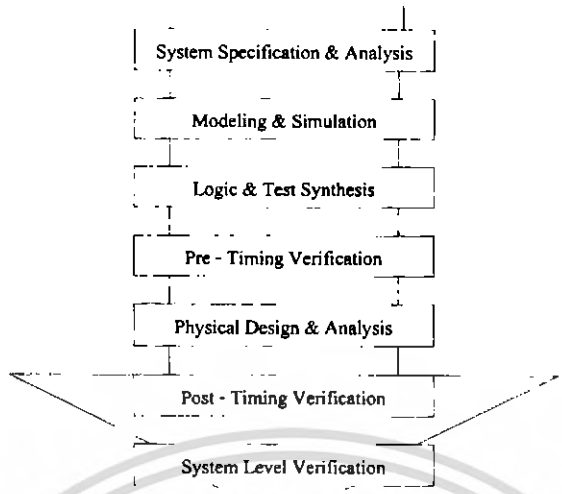
การออกแบบในขั้นตอนถัดไปเป็นการเปลี่ยนเครือข่ายการโยงใยที่ได้จากขั้นตอนที่แล้ว ให้เป็นลำดับของทรานซิสเตอร์ (Transistor List) และ โครงงาน (Layout) ซึ่งขั้นตอนนี้จะเกี่ยวข้องกันโดยตรงกับการจัดวางทรานซิสเตอร์หรือไลบรารีเซลล์เพื่อแทนเกทและฟลิปฟลอปต่าง ๆ

และในขั้นตอนสุดท้ายจะเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

2.4.3 การออกแบบจากบนลงล่าง (Top-Down Design)

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

วีเอชดีแอลกับหลักการออกแบบจากบนลงล่างจึงเป็นทางออกให้กับวิศวกรให้สามารถออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 2.11 แสดงขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.11 แ สดงถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อแตกต่างไปจากนี้บ้าง เล็กน้อยเนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอน การออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

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

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

3) ลอจิกและการทดสอบการสังเคราะห์ คือ หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้วหลักการนี้จะถูกเพิ่มเติมรายละเอียดลงมาเป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจรจริง หรือทำการสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่จะเขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของโครงข่ายการเชื่อมต่อ ที่สามารถนำไปผลิตอุปกรณ์อื่นได้

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

ทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไป หรือไม่สามารถที่จะทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

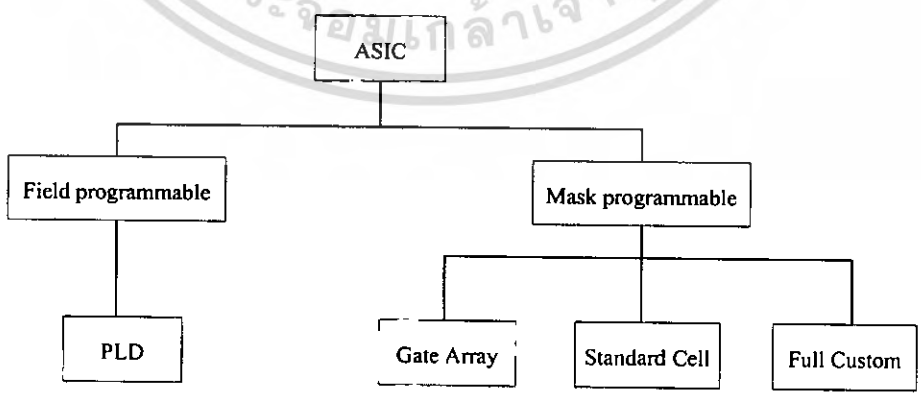
5) การออกแบบทางกายภาพและการวิเคราะห์ คือ ขั้นตอนในการผลิตเป็นวงจรจริง (Technology and device mapping) โดยจะนำข้อมูลที่ได้จากการสังเคราะห์ มาใช้ในการผลิตเป็นวงจรรวม ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวมเอซิก (ASIC)

6) การตรวจสอบเวลาหลังการออกแบบ คือ การทำการตรวจสอบการทำงานด้วยตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้ายก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้ วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุตและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

7) การตรวจสอบระบบ คือ การนำวงจรที่ออกแบบไว้ประกอบเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบที่สมบูรณ์ แล้วทำการทดสอบการทำงานทั้งระบบรวมกับอุปกรณ์อื่นๆ อีกครั้งเพื่อควบคุมคุณภาพของผลิตภัณฑ์

2.5 เอฟพีจีเอ

เทคโนโลยีความก้าวหน้าของอุตสาหกรรมอิเล็กทรอนิกส์ในปัจจุบัน ทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆ ซึ่งทำให้ลดค่าใช้จ่ายต่างๆได้มาก ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครโปรเซสเซอร์ และหน่วยความจำปัจจุบัน ทุกๆครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างระหว่างวงจรรวมและไอซีมาตรฐานมากขึ้น นักออกแบบอุปกรณ์ทางด้านดิจิตอล ได้พิจารณาถึงการผลิตให้มีขนาดมากขึ้นและการผลิตวงจรรวมเอซิก (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวมจะแบ่งตามโครงสร้างออกเป็น 2 กลุ่ม คือ ฟิลด์โปรแกรมเมเบิล (Field programmable) และ แมสโปรแกรมเมเบิล (Mask programmable) ดังแสดงในรูปที่ 2.12



รูปที่ 2.12 แสดงผังการแบ่งกลุ่มของวงจรรวมเอซิก

2.5.1 การออกแบบวงจรเชิงเลขด้วยชิพอุปกรณ์เอฟพีจีเอ

ชิพอุปกรณ์เอฟพีจีเอ เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์เอฟพีจีเอ มีฟังก์ชั่นการทำงานตามที่ออกแบบไว้ในการทำชิพอุปกรณ์ ซึ่งเป็นวิธีการออกแบบ ไอซี (IC : Integrated Circuit) แบบ เซมิคัสตัม (Semi custom) อีกวิธีหนึ่ง เมื่อเทียบกับการทำเอซิกแล้วนั้น ก็มีทั้งข้อดีและข้อเสีย คือ การทำชิพอุปกรณ์เอฟพีจีเอ จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในชิพอุปกรณ์เอฟพีจีเอจะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัด และการทำชิพอุปกรณ์เอฟพีจีเอ ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำชิพอุปกรณ์ก็คือ ระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (download) นั้นน้อยกว่าการทำเอซิกมากและการตรวจสอบหรือแก้ไขการออกแบบที่ทำได้สะดวก

การทำชิพอุปกรณ์เอฟพีจีเอ ในปัจจุบันมีประสิทธิภาพและความสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตชิพอุปกรณ์เอฟพีจีเอ ได้เพิ่มความสามารถของชิพอุปกรณ์เอฟพีจีเอ โดยเพิ่มจำนวนองค์ประกอบภายใน หรือปรับปรุงโครงสร้างสถาปัตยกรรมภายในและยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ พีพีอาร์ (PPR: Partitioning Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วย

สำหรับตัวชิพอุปกรณ์เอฟพีจีเอนั้นมีโครงสร้างพื้นฐานเทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรแกรมที่แตกต่างกันสำหรับผู้ผลิตแต่ละราย นอกจากนั้นชิพอุปกรณ์เอฟพีจีเอของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกันบางส่วน ในการใช้งานนั้นชิพอุปกรณ์เอฟพีจีเอสามารถนำไปประยุกต์ใช้งานได้ เช่น การประมวลผลสัญญาณเชิงเลข (DSP : Digital Signal Processing) การออกแบบไมโครคอนโทรลเลอร์ เป็นต้น

2.5.2 ปัจจัยที่ทำให้การออกแบบชิพอุปกรณ์เอฟพีจีเอ ทำได้ง่ายและสะดวกรวดเร็ว

1. ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพเพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึงภาษาแอสแซมบลี (Assembly) ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2. มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจรหรือเอชดีแอล เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มันจากนั้นตัวซอฟต์แวร์จะทำการสังเคราะห์ ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตรฐานเดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3. การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของคอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสียค่าใช้จ่ายเพิ่มเติมอย่าง

2.6 การประยุกต์นำไปใช้

การพิจารณาต้องถูกนำมาสู่การคำนวณในความเข้าใจเชิงปฏิบัติของการเข้ารหัสเลขคณิต นี้จะเป็นต้องใช้เลขคณิตที่มีความแม่นยำที่จำกัด เพื่อต้องการแน่ใจว่าตัวถอดรหัสที่ปลายทางจะอยู่ในจุดที่ถูก

2.6.1 ความแม่นยำที่จำกัด

จากด้านบนได้แสดงจำนวนที่เป็นตัวแทนของข้อความที่สามารถคำนวณไปเป็นจำนวนบิทเท่าที่จำเป็น โดยทั่วไปแล้วอุปกรณ์ที่คำนวณรหัสเลขคณิตจะมีรีจิสเตอร์ที่มีความแม่นยำที่จำกัดอยู่เท่านั้น มันจึงสามารถคำนวณส่วนย่อยๆของรหัสได้ในเวลาหนึ่งๆ และส่งข้อมูลออกมาเมื่อไม่ต้องมีการคำนวณอื่นๆอีก

จะสามารถทำได้โดยใช้กระบวนการดังนี้ เมื่อพิจารณาตัวเข้ารหัสที่มีรีจิสเตอร์ 16 บิท(LOW,HIGH) เริ่มต้นค่า LOW จะเป็น 0000000000000000 แสดงค่า 0 และ HIGH จะเป็น 1111111111111111 แทนค่า 1- δ เมื่อ δ เป็นค่าที่น้อยมากๆจนใกล้ค่าศูนย์ เมื่อตัวเข้ารหัสได้รับสัญลักษณ์ s_k จะเกิดการเปลี่ยนแปลงขึ้นดังนี้

$$\text{LOW} = \text{LOW} + c_{k-1} \times (\text{HIGH} - \text{LOW} + 0000000000000001)$$

$$\text{HIGH'} = \text{LOW} + c_{k-1} \times (\text{HIGH} - \text{LOW} + 0000000000000001) - 0000000000000001$$

(0000000000000001 ถูกเพิ่มสู่ความแตกต่างระหว่างรีจีสเตอร์และถูกลบจาก ผลของ HIGH เพราะว่าช่วงต้องถูกปฏิบัติเหมือนช่วงเปิด หลังจากนี้ MSB ของสองรีจิสเตอร์ถูกเปรียบเทียบถ้ามันเข้ากัน ค่านี้เป็นเอาท์พุท และบิทยังคงถูกเลื่อนไป 1 ตำแหน่ง LSB ของ LOW ถูกตั้งให้เป็น 0 และ HIGH ให้เป็น 1 ความต่อเนื่องนี้ยังคงเป็นจนกระทั่ง MSB ของสองรีจิสเตอร์ไม่เข้ากันแล้ว

บิทได้รับโดยตัวถอดรหัสถูกเก็บอยู่ในรีจิสเตอร์ 16 บิท CURRENT ถ้าค่าถูกเก็บอยู่ในรีจิสเตอร์วางอยู่ในช่วง $c_{k-1} \leq \text{CURRENT} \leq c_k$ สัญลักษณ์ s_k คือเอาท์พุท และ HIGH และ รีจิสเตอร์ LOW ถูกรีสเกลสำหรับเข้ารหัสสัญญาณนี้ ถ้า MSB ของ HIGH และ LOW เข้ากัน MSB ของรีจิสเตอร์ทุกตัวจะถูกละทิ้งและบิททุกตัวจะถูกเลื่อนหนึ่งตำแหน่ง LSB ของ HIGH ถูกตั้งให้เป็น 1 ซึ่ง LOW ถูกเซ็ตเป็น 0 และ CURRENT ด้วยข้อมูลจากสตรีมที่มา

ขบวนการนี้บางทีจะถูกอธิบายตามที่กล่าวมา ช่วงที่แสดงข้อความหลังจากให้หมายเลขของสัญลักษณ์ถูกเข้ารหัส ปกติวางอยู่บน ทั้งหมด ครึ่งหนึ่งของช่วง [0 , 1)อีกครึ่งนึงของช่วงไม่จำเป็นต้องถูกพิจารณา และบางทีอาจไม่ถูกสนใจ การอนุญาตินี้เป็นช่วงที่ยังคงเหลืออีกครึ่งนึง เพื่อที่จะถูกพิจารณาด้วยความแม่นยำที่มากกว่า มีรหัสอยู่ข้างหน้าจากการเข้าใกล้นี้ เนื่องจากการปัดค่าที่ผิดพลาด แต่มันยังน้อยมาก

กลไกมีประโยชน์เพื่อขึ้นมาซึ่งมันให้การเข้ารหัสและถอดรหัสอยู่เป็นเรียลทามได้ ที่เป็นกลไกใดๆได้ซึ่งต้องการข้อความทั้งหมดเพื่อที่จะถูกเข้ารหัสก่อนการส่งเพื่อจะสามารถพิจารณาดีเลย์ได้

2.6.2 UNDERFLOW

ลำดับของสัญลักษณ์ที่อาจเกิดขึ้นในข้อความซึ่งจะเป็นสาเหตุของการจำกัดช่วงที่ถูกทำให้กว้างขึ้นครึ่งหนึ่งซ้ำไปมานั้นอาจเกิดขึ้นได้ หากเกิดขึ้นแล้วบิทตำแหน่งสูงสุดของรีจิสเตอร์จะไม่ลู่เข้าและการคำนวณสัญลักษณ์ต่อๆไปก็จะมีความแม่นยำน้อยลงเรื่อยๆ ซึ่งอาจเกิดปัญหานี้ได้เมื่อ

HIGH = 1000000000000000

LOW = 0111111111111111

และไม่มีบิทเหลือสำหรับการคำนวณ การที่จะทำให้ข้อความถูกเข้ารหัสและถอดรหัสได้อย่างถูกต้องนั้น จะต้องป้องกันไม่ให้เกิดกรณีเช่นนี้

ทำได้โดย ตัวรหัสต้องตรวจสอบบิทตำแหน่งสูงสุดสองบิทของบัฟเฟอร์ หากว่า HIGH เป็น 10 และ LOW เป็น 01 แล้วตัวนับ x จะเพิ่มขึ้น บิทตำแหน่งสูงสุดลำดับที่สองจะถูกลบทิ้งและเลื่อนบิทถัดไปขึ้นมาหนึ่งตำแหน่ง บิทตำแหน่งตำแหน่งต่ำสุดของ HIGH และ LOW จะถูกเติมด้วย 1 และ 0 ตามลำดับ

เมื่อค่าบิทตำแหน่งสูงสุดของ HIGH และ LOW ลู่เข้าไปยังค่า b จะได้ค่านี้ออกมาเป็นเอาท์พุทและตามด้วย $\overline{b}$ จำนวนเท่ากับ x จากนั้น x จึงถูกตั้งค่าใหม่ให้เป็นศูนย์

เราจะพิจารณากระบวนการนี้ในส่วนต่อไป เมื่อบิทตำแหน่งสูงสุดของ HIGH และ LOW เป็น 10 และ 01 ตามลำดับทำให้รู้ว่าช่วงที่หาอยู่ในระยะ $[1/4,3/4)$ ดังนั้นเราจึงละทิ้งระยะ $[0,1/4)$ และ $[3/4,1)$ ได้และพิจารณาเฉพาะช่วง $[1/4,3/4)$ เพื่อให้เกิดความแม่นยำสูงสุด ถ้าสุดท้ายแล้วบิทตำแหน่งสูงสุดเกิดการลู่เข้ามากที่ 0 แล้วทำให้รู้ว่าช่วงนี้อยู่ในระยะ $[1/4,1/2)$ ดังนั้นบิทต่อไปที่จะส่งต้องเป็น 1 ถ้าบิทตำแหน่งสูงสุดลู่เข้า 1 แล้วเราจะรู้ได้ว่าช่วงนี้อยู่ที่ระยะ $[1/2,3/4)$ แล้วดังนั้นบิทต่อไปที่ส่งต้องเป็น 0

ในตัวถอดรหัส การทดสอบสองบิทที่มีตำแหน่งสูงสุดของ HIGH และ LOW และการเลื่อนบิทเพื่อป้องกันการสูญเสียความแม่นยำก็เป็นเหมือนในตัวเข้ารหัส แต่ไม่มีเก็บค่าที่นับบิทที่ถูกละทิ้งไป

2.6.3 OVERFLOW

ในทางปฏิบัติความน่าจะเป็นสะสม c_k ของสัญลักษณ์จะเก็บอยู่ในรูปจำนวนเต็ม หากเก็บเป็นทศนิยมจะยากแก่การสร้างเป็นอุปกรณ์และปฏิบัติได้ล่าช้า ยิ่งไปกว่านั้นการใช้เลขทศนิยมยังทำให้ความแม่นยำลดลงด้วย ความน่าจะเป็นสะสมจึงถูกแสดงในรูปอัตราส่วนด้วยตัวเศษ f_k และตัวส่วน f_{n-1} (เมื่อ n คือจำนวนที่เป็นไปได้ของสัญลักษณ์) ดังนั้น $f_k = c_k f_{n-1}$ เพื่อให้แน่ใจว่าการคำนวณจะทำให้ได้ความแม่นยำมากที่สุดเมื่อคำนวณหาสเกลเสร็จแล้วโดยจะทำการคูณก่อนที่จะหาร แต่ผลลัพธ์ที่ได้ในขั้นต้นอาจเกินค่าสูงสุดที่รีจิสเตอร์จะเก็บได้ เพื่อหลีกเลี่ยงปัญหานี้ผลลัพธ์จะถูกคำนวณในรีจิสเตอร์ที่ใหญ่พอที่จะไว้ค่าอย่างต่ำ MAX$\times f_{n-1}$ เมื่อ MAX คือค่าสูงสุดที่เก็บไว้ใน LOW หรือ HIGH ได้ ผลที่ได้จากการหารจะถูกส่งไปยังรีจิสเตอร์ที่เหมาะสมในภายหลัง สิ่งนี้จะต้องนำไปใช้การสร้างทั้งตัวเข้ารหัสและถอดรหัส

2.6.4 การสิ้นสุด

กลไกการให้สัญญาณกับตัวถอดรหัสเมื่อข้อความได้สิ้นสุดแล้วก็มีความสำคัญ ดังนั้นเพื่อที่จะหลีกเลี่ยงการที่ตัวถอดรหัสจะเพิ่มสัญลักษณ์สุ่มเข้ามายังข้อความ จึงมีกลไกที่ทำได้ 3 วิธีดังนี้ วิธีแรกคือการให้สัญญาณที่บอกถึงขนาดของข้อความที่จะถูกถอดรหัสไว้ก่อน เพื่อที่ว่าตัวถอดรหัสจะสามารถหยุดเมื่อจำนวนสัญลักษณ์ที่ต้องการได้ถูกถอดรหัสแล้ว วิธีนี้ต้องการสัญลักษณ์ที่จะถูกนับในการเตรียมเพื่อการเข้ารหัสซึ่งอาจสร้างการหน่วงเวลาเล็กน้อยในการเข้ารหัส วิธีที่สองคือการส่งข้อความที่ประกอบไปด้วยจำนวนสัญลักษณ์ที่จะถูกถอดรหัส ตัวถอดรหัสจะสามารถถอดรหัสสัญลักษณ์ตามจำนวนที่ต้องการได้โดยไม่มีการนับและส่งมาเตรียมไว้ล่วงหน้า สำหรับการประยุกต์ใช้การเข้ารหัสวีดีโอการใช้วิธีนี้จะเหมาะสมกับช่องสัญญาณที่ไม่มีความผิดพลาด อย่างไรก็ตามก็สามารถใช้ได้ในช่องสัญญาณที่มีความผิดพลาด จำนวนบิทที่ถูกเข้ารหัสก็ถูกส่งไปเช่นกัน ทั้งนี้ความผิดพลาดในบิทที่ต่อเนื่องนี้ก็ไม่ได้ทำให้ตัวถอดรหัสดำเนินการไปจนเลยขุดสิ้นสุดของเฟรม สุดท้ายคือตัวอักษร special end of file (EOF) ที่ถูกเข้ารหัสแล้วจะถูกส่งไปอยู่ส่วนท้ายของข้อมูล ตัวถอดรหัสจะหยุดเมื่อตัวอักษรนี้ถูกถอดรหัสออกมาแล้ว วิธีนี้จะมีประโยชน์เมื่อไม่ทราบความยาวของข้อมูลล่วงหน้าอย่าเช่นข้อมูลตัวอักษร แต่เนื่องจากตัวอักษร EOF มีโอกาวมน่าจะเป็นในการเกิดน้อยมากจึงค่อนข้างสิ้นเปลืองในการส่ง

ไม่ว่าวิธีใดที่ใช้ในการให้สัญญาณการสิ้นสุดของข้อความ จะต้องแน่ใจว่าสัญลักษณ์สุดท้ายของข้อความได้ถูกส่งอย่างสมบูรณ์แล้วและสามารถถูกถอดรหัสได้อย่างถูกต้องด้วย

2.6.5 ข้อดีของการใช้ระบบเลขฐานสอง

เท่าที่เราได้พิจารณาอักษรสัญลักษณ์อย่างไรก็ได้ อย่างไรก็ตามสัญลักษณ์อักษรถูกจำกัดเป็นฐาน 2 ด้วยสัญญาลักษณ์ 1 กับ 0 เราพบว่าเราต้องการเพียงแค่เก็บค่าความน่าจะเป็น $p_0 = c_0$ โดยการระบุ c_1=1 ก่อนหน้านี้ ในกรณีที่เป็น 0 จะถูกเข้ารหัส

$$LOW' = low + c_{-1}(HIGH - LOW + 0000000000000001) = LOW$$

Since $c_{-1} = 0$

$$LOW' = low + c_0(HIGH - LOW + 0000000000000001) - 0000000000000001$$

และ ที่ 1 ต้องถูกเข้ารหัส

$$HIGH' = LOW + c_1(HIGH - LOW + 0000000000000001) - 0000000000000001 = HIGH$$

ก่อนหน้านี้ในแต่ละกรณี หนึ่งของรีจิสเตอร์ไม่ถูกเลือก และการคำนวณเพียงครั้งเดียวต้องการ carried out นอกจากนี้ ปกติมันจะเหมือนในการคำนวณแต่ละกรณี อักษรสัญลักษณ์ของฐานสอง

เพราะฉะนั้นยิ่งมีการบีบอัดที่ดีเท่าไหร่ความซับซ้อนของตัวเข้ารหัสและตัวถอดรหัสก็ยิ่งเพิ่มมากขึ้น เพราะฉะนั้นตัวที่ถูกเข้ารหัสจะถูกแปลงไปเป็นตัวอักษรไบนารี่เพื่อการเข้ารหัส

2.6.6 ADAPTIVE PROBABILITIES

จากข้างต้นเราได้สมมติให้ทราบค่าความน่าจะเป็นมาก่อนหน้าแล้วซึ่งจะเหมาะสมกับในบางกรณีเท่านั้น เช่น การเข้ารหัสบทอ่านในภาษาที่ทราบอยู่แล้วแต่โดยทั่วไปจะไม่เป็นดังกรณีเช่นนี้ ดังนั้นตัวเข้ารหัสนั้นจำเป็นจะต้องเรียนรู้ความน่าจะเป็นของสัญลักษณ์จากข้อมูลปัจจุบัน สำหรับตัวเข้ารหัสที่ไม่ทำงานในเวลาจริงก็จะสามารถนับความน่าจะเป็นของสัญลักษณ์ของข้อความทั้งหมดก่อนที่จะส่งไปยังตัวถอดรหัสได้ แต่จะทำให้เกิดการเลื่อนเวลาที่ไม่สามารถยอมรับได้ในแอพพลิเคชั่นที่ทำงานในเวลาจริง และส่งความน่าจะเป็นนั้นเป็นโอเวอร์เฮดในการส่งด้วย ทางเลือกอื่นคือการเข้ารหัสแต่ละสัญลักษณ์โดยใช้โมเดลสถิติที่ขึ้นอยู่กับการนับสัญลักษณ์ก่อนหน้านี้ สำหรับตัวเข้ารหัสเลขคณิตที่ได้อธิบายไปข้างต้นจะมีการนับสัญลักษณ์ค่าเริ่มต้นเป็น $f_0 = 1, f_1 = 2$ เมื่อสัญลักษณ์ถูกเข้ารหัสแล้ว f_i จะเพิ่มขึ้น และเมื่อสัญลักษณ์เป็น 0 แล้ว f_0 ก็ถูกเพิ่มค่าขึ้นด้วย ตัวถอดรหัสจะปรับปรุงค่าความน่าจะเป็นไปตามการถอดรหัสสัญลักษณ์ ส่วนความแตกต่างระหว่างความน่าจะเป็นที่ใช้ในตัวเข้ารหัสและความน่าจะเป็นปัจจุบันคือ

$$\frac{1-2p_0}{f_1} = \frac{1-2p_0}{N+2}$$

เมื่อ N คือจำนวนของสัญลักษณ์ที่ถูกเข้ารหัสแล้ว ดังนั้นค่าความน่าจะเป็นจะใกล้เคียงค่าจริงขณะที่สัญลักษณ์ถูกเข้ารหัสและอาจมีโอเวอร์เฮดเล็กน้อยที่เกิดจากความไม่แม่นยำเริ่มต้นของความน่าจะเป็นแต่เป็นค่าน้อย

ขณะที่จำนวนสัญลักษณ์ที่ถูกเข้ารหัสเพิ่มขึ้นนั้นความสามารถของตัวเข้ารหัสที่จะปรับตัวเข้ากับการเข้ารหัสนั้นจะลดลง ผลกระทบนี้อาจทำให้ลดลงได้โดยการรีเฟรชการนับสัญลักษณ์อย่างเป็นคาบ ตัวอย่างเช่น ลดทุกการนับสัญลักษณ์ให้เหลือครึ่งเดียวเมื่อ f_i ใกล้ถึงค่าสูงสุด สิ่งนี้จะหมายถึงว่าความเร็วที่มีตัวเข้ารหัสปรับให้เข้ากับสถิติของสัญลักษณ์จะเป็นไปอย่างไม่ต่อเนื่อง

2.6.7 CONTEXT MODELING

จากที่เราได้สมมติว่าความน่าจะเป็นของสัญญาลักษณ์ที่ต่อเนื่องกันโดยไม่ขึ้นต่อกัน ในหลายข้อความ นี่จะไม่เป็นกรณี ยกตัวอย่างเช่นถ้าเราจะโค้ดดิ้งตัวหนังสือภาษาอังกฤษและสัญลักษณ์ q ที่เจอปกติเราจะรู้ด้วยความแน่นอนว่าสัญลักษณ์ต่อไปจะเป็น ทำให้ใช้ความรู้นี้ เราจะสามารถที่จะถอดรหัสสัญลักษณ์มีประสิทธิภาพมากกว่ากรณีอื่นๆ

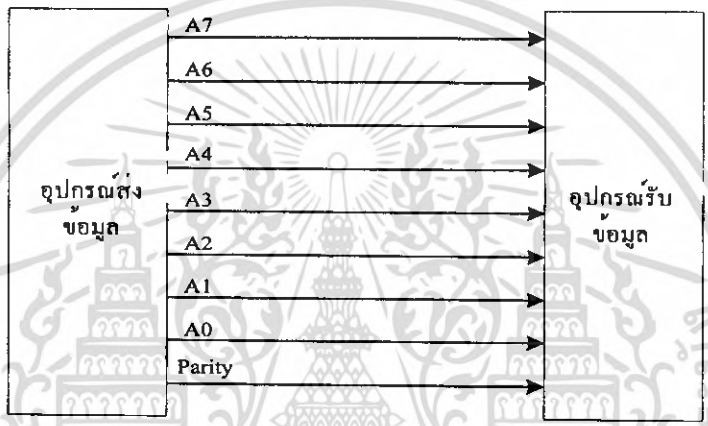
รหัสเลขคณิตที่มีประสิทธิภาพมากกว่าสามารถสำเร็จโดยการรักษาโมเดลสภาพความน่าจะเป็นการนับสัญลักษณ์แบบแยกใช้นี้ สำหรับจำนวน context ที่แตกต่างกัน และเลือกตัวหนึ่งที่เหมาะสมที่จะเข้ารหัสสำหรับการเข้ารหัสอ้างถึงรหัสที่เกิดขึ้น

2.7 พอร์ตอนุกรม

2.7.1 การสื่อสารข้อมูล

2.7.1.1 การสื่อสารข้อมูลแบบขนาน (Parallel Communication)

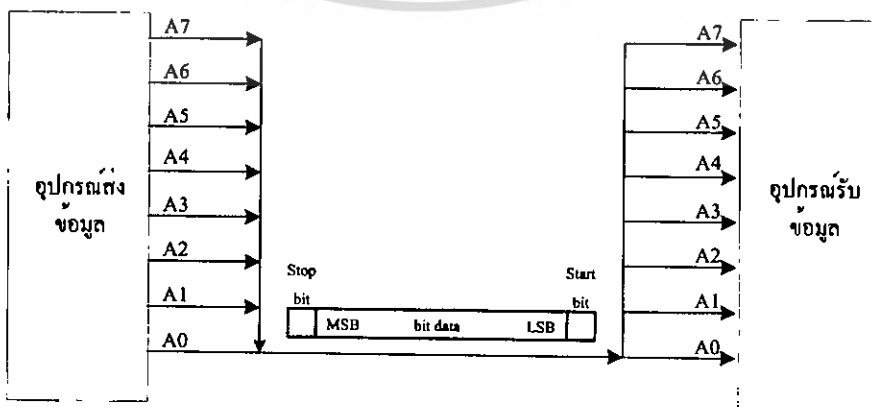
การสื่อสารข้อมูลแบบขนานคือการสื่อสารแบบที่ส่งข้อมูลพร้อมๆกัน n บิทผ่านสายสัญญาณ n เส้น สามารถแสดงรูปแบบการสื่อสารข้อมูลแบบขนานได้ดังรูปที่ 2.13



รูปที่ 2.13 แสดงบล็อกไดอะแกรมรูปแบบการสื่อสารข้อมูลแบบขนาน

2.7.1.2 การสื่อสารข้อมูลแบบอนุกรม (Serial Communication)

การสื่อสารข้อมูลแบบอนุกรม คือ การสื่อสารแบบที่ส่งข้อมูลที่ละบิท ผ่านสายสัญญาณเส้นเดียวจนครบจำนวนข้อมูลที่ต้องการ โดยเฟรมของการสื่อสารข้อมูลแบบอนุกรมประกอบด้วย สตาร์ทบิท (start bit) , สตอบบิท (stop bit) , ข้อมูล (data bit) สามารถแสดงรูปแบบการสื่อสารข้อมูลแบบอนุกรมได้ดังรูปที่ 2.14



รูปที่ 2.14 แสดงบล็อกไดอะแกรมรูปแบบการสื่อสารแบบอนุกรม

อัตราบอดเรด (Baud Rate) ในการสื่อสาร คือ ความเร็วในการรับ-ส่งข้อมูลแบบอนุกรมมีหน่วยเป็นบิทต่อวินาที (bit/sec) ซึ่งจะบอกถึงจำนวนบิทที่รับ-ส่ง ในเวลา 1 วินาที เช่น ส่งข้อมูลด้วยอัตรา 9600 บิทต่อวินาที หมายถึง เวลา 1 วินาที รับ-ส่งข้อมูลได้ 9600 บิท รวมทั้งบิทข้อมูล (Data bit) สตาร์ทบิท (Start bit) สตอบบิท (Stop bit) ด้วย

2.7.2 การอินเตอร์เฟสตามมาตรฐาน RS-232

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

มาตรฐาน RS-232 จะใช้สัญญาณเส้นเดียวในการส่งสัญญาณ โดยจะสัญญาณจะส่งไปในทิศทางเดียวกัน สำหรับการแทนค่าแรงดันในการส่งสัญญาณเป็นดังนี้

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

การเชื่อมต่อกับพอร์ตสื่อสารของคอมพิวเตอร์ส่วนบุคคลจะเลือกใช้พอร์ตสื่อสารแบบอนุกรม 9 ขา (DB-9) ซึ่งสามารถทำการส่งสัญญาณข้อมูลได้ตามมาตรฐาน RS-232 โดยลักษณะของคอนเน็คเตอร์แบบ DB-9 สามารถแสดงได้ดังรูปที่ 2.15 และการเชื่อมต่อของพอร์ตสื่อสารสำหรับคอนเน็คเตอร์แบบ DB-9 สามารถแสดงได้ดังตารางที่ 2.10

รูปที่ 2.15 แสดงลักษณะของคอนเน็กเตอร์แบบ DB-9

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

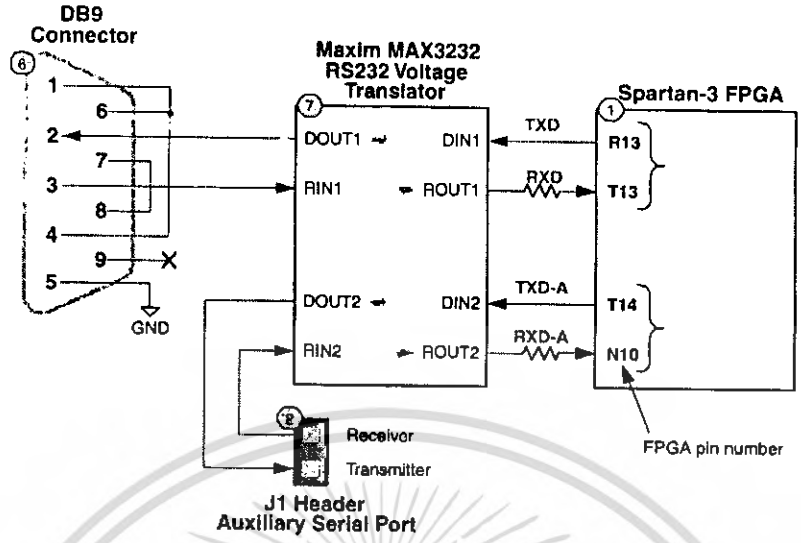
ตารางที่ 2.10 การเชื่อมต่อของพอร์ตสื่อสารสำหรับคอนเน็คเตอร์แบบ DB-9

เปรียบเทียบข้อดีข้อเสียของการสื่อสารข้อมูลแบบอนุกรมและขนาน

- การสื่อสารข้อมูลแบบอนุกรมสามารถสื่อสารได้ระยะทางที่ไกลกว่า
- การสื่อสารข้อมูลแบบอนุกรมใช้สายสัญญาณที่ประหยัดกว่า
- การสื่อสารข้อมูลแบบขนานสามารถสื่อสารข้อมูลได้ทีละมากๆ และรวดเร็วกว่า

2.7.3 การเชื่อมต่อระหว่าง DB-9 กับ FPGA

โครงงานนี้ได้ใช้บอร์ด FPGA สปาร์ตัน 3 สตาร์เตอร์คิท (Spartan-3 Starter Kit Board) การเชื่อมต่อระหว่างคอนเน็คเตอร์แบบ DB-9 และ บอร์ด FPGA จะต้องผ่าน MAX3232 ก่อนเพื่อปรับแรงดันที่รับมาได้จากพอร์ต RS232 ให้เหมาะสมกับระดับแรงดันภายในบอร์ด ซึ่งสามารถแสดงการเชื่อมต่อระหว่างคอนเน็คเตอร์แบบ DB-9 และ บอร์ด FPGA โดยผ่าน MAX3232 ได้ดังรูปที่ 2.16



รูปที่ 2.16 แสดงการเชื่อมต่อระหว่างคอนเน็คเตอร์แบบ DB-9 และ บอร์ด FPGA โดยผ่าน MAX3232

บทที่ 3
การคำนวณและการสร้าง

เราได้แบ่งงานออกเป็นสองส่วนหลักๆ คือ

1. ส่วนที่เป็นซอฟท์แวร์ของอัลกอริธึมนี้ ซึ่งเขียนขึ้นด้วยโปรแกรมแมทแลบมีประโยชน์ในการใช้ในการตรวจสอบผลลัพธ์ที่เกิดขึ้นจากส่วนของผลการจำลองการทำงานของภาษา VHDL ที่จะนำไปลง FPGA

2. ส่วนที่เป็นฮาร์ดแวร์ซึ่งคือ บอร์ด FPGA ที่ถูกโปรแกรมด้วยภาษา VHDL และใช้การแสดงผลต่างๆผ่านทางโปรแกรมแมทแลบ

3.1 ส่วนซอฟท์แวร์ ในการจำลองผลการทำงานของการเข้ารหัสเลขคณิตด้วยโปรแกรมแมทแลบ

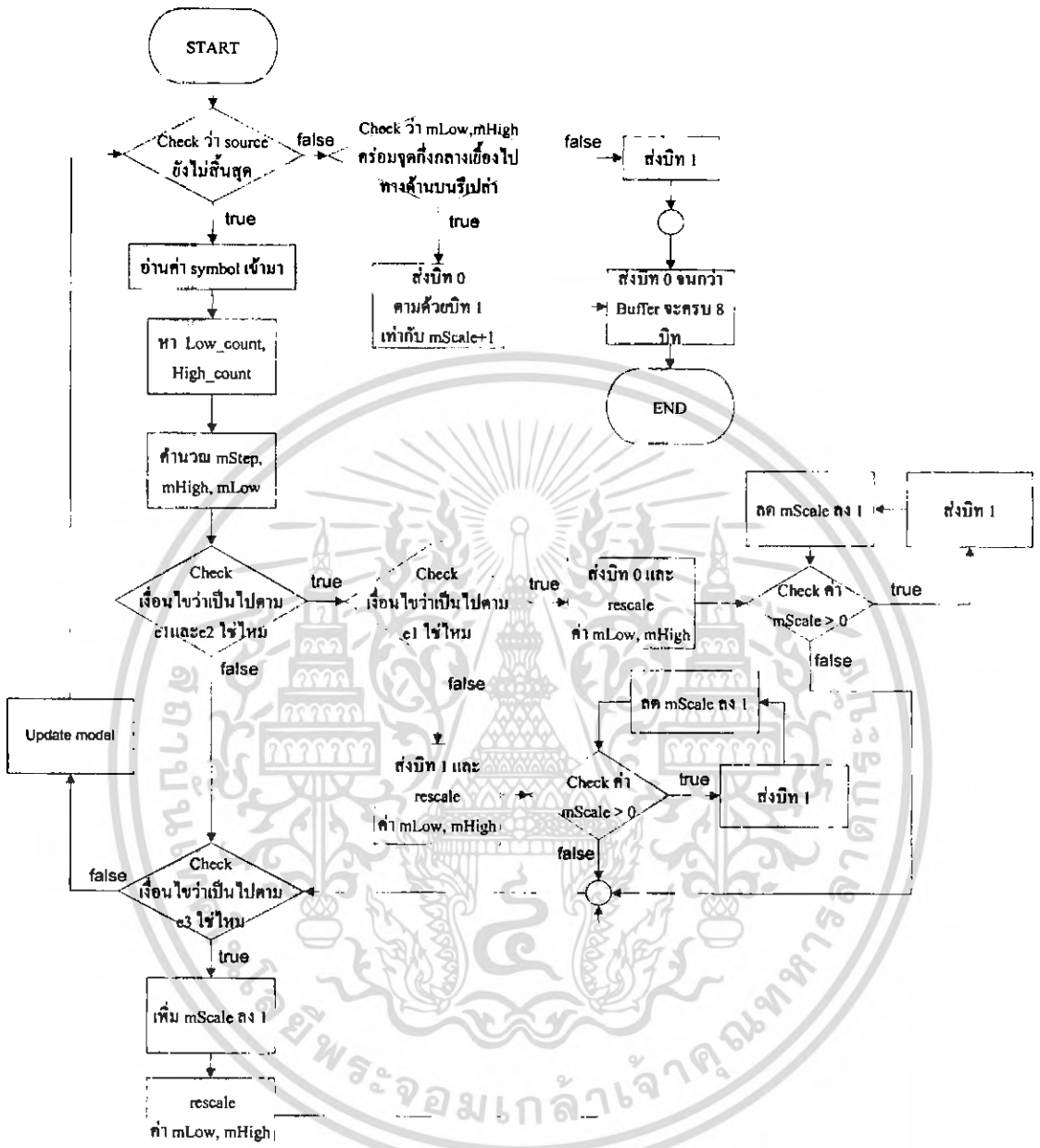
จะแบ่งออกเป็น 2 ส่วนย่อย คือ ส่วนในการเข้ารหัสเลขคณิต และส่วนในการถอดรหัสเลขคณิต

รูปที่ 3.1 แสดงแผนผังการทำงานของโปรแกรมการเข้ารหัสเลขคณิตในภาพรวม

เริ่มจากมี DATA_IN เข้ามาในส่วนของการเข้ารหัสแล้วนำผลลัพธ์จากส่วนการเข้ารหัสไปเข้าส่วนของการถอดรหัสก็จะได้ผลลัพธ์ออกมาเป็น DATA_OUT ซึ่งจะต้องตรงกับ DATA_IN ที่ใส่เข้าในตอนแรก

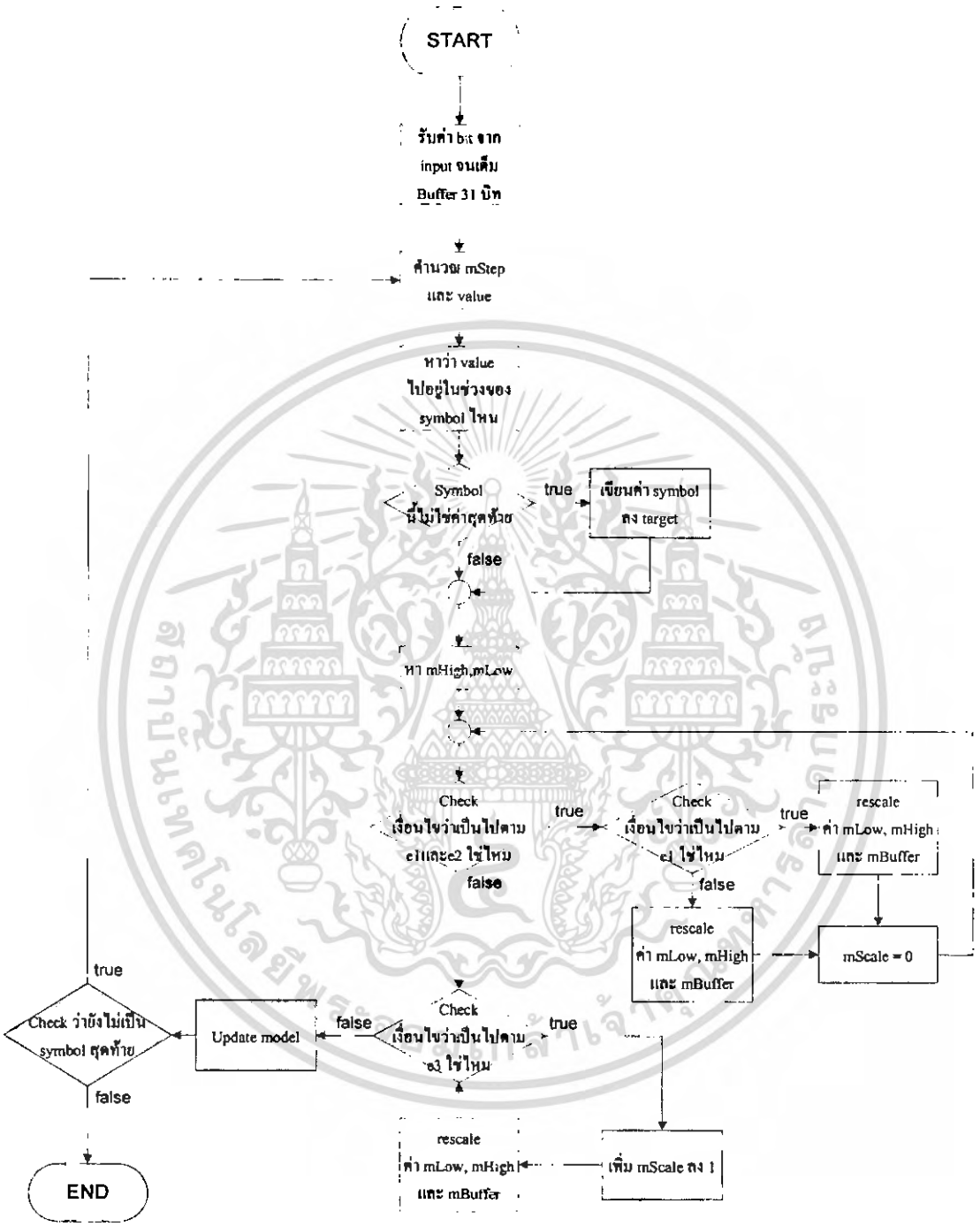
โดยขั้นตอนในโปรแกรมนั้นจะเขียนให้คล้ายคลึงกับการทำงานของตรงส่วนที่เขียนด้วยภาษา VHDL ให้มากที่สุด เพราะบางการคำนวณจะมีข้อจำกัดในการทำเป็นฮาร์ดแวร์ ให้ใช้ในเวลาจริง (realtime)ได้ ยกตัวอย่างอย่างการคำนวณการหารในโปรแกรม VHDL นั้น จะเก็บเป็นบิท 0,1 เท่านั้น (เราเลือกใช้ 16 บิท) ยิ่งถ้าเราต้องการให้ละเอียดเท่าใดก็ต้องยิ่งใช้จำนวนบิทในการรองรับมากขึ้นเท่านั้น ส่วนการทำงานในโปรแกรมแมทแลบ นี้จะแสดงทศนิยมได้มากประมาณ 15 ตำแหน่งในเลขฐานสิบ ซึ่งให้ความละเอียดที่มากกว่า ดังนั้นผลที่ได้อาจจะไม่ตรงกับภาษา VHDL ได้ แต่เราต้องการใช้ในการจำลองผลการทำงานเพื่อเปรียบเทียบจึงต้องเขียนแมทแลบ ในลักษณะเช่นนี้

3.1.1 แผนผังการทำงานของโปรแกรมการเข้ารหัส



รูปที่ 3.2 แสดงแผนผังการทำงานของโปรแกรมการเข้ารหัส

3.1.2 แผนผังการทำงานของโปรแกรมการถอดรหัส



รูปที่ 3.3 แสดงแผนผังการทำงานของโปรแกรมการถอดรหัส

จะเห็นได้ว่าทั้งส่วนการเข้ารหัสและส่วนการถอดรหัสนั้นจะเน้นการทำงานหลักอยู่ตรงที่การย้ายตามเงื่อนไข ซึ่งมีทั้งหมด 3 กรณี ดังนี้

การย้ายตามเงื่อนไข E1 : mHigh<g_Half

- ส่งบิท 0
- mLow = mLow * 2
- mHigh = mHigh * 2 + 1

การย้ายตามเงื่อนไข E2 : mLow >= g_Half

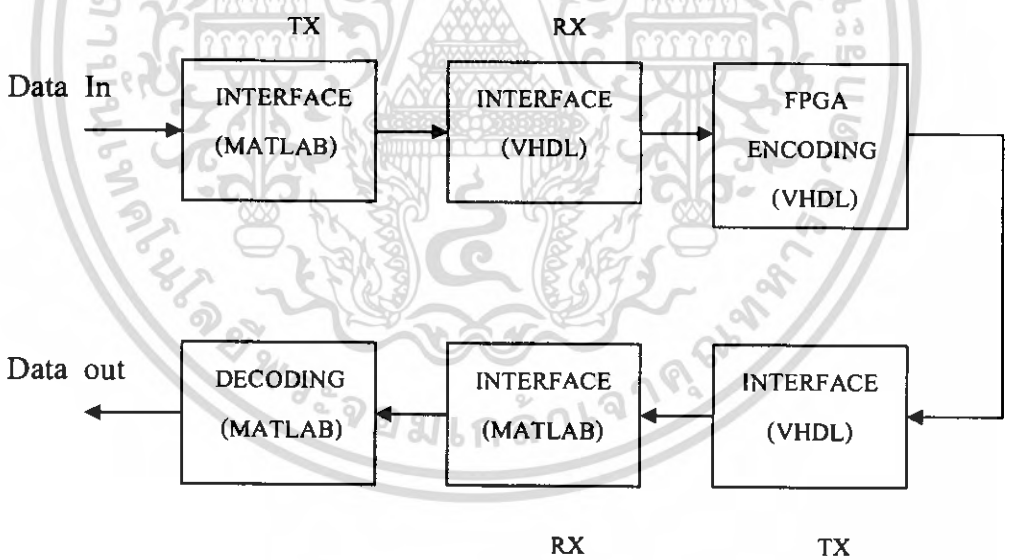
- ส่งบิท 1
- mLow = (mLow-g_Half) * 2
- mHigh = (mHigh-g_Half) * 2 + 1

การย้ายตามเงื่อนไข E3 : (g_FirstQuater <= mLow) และ (mHigh < g_ThirdQuater)

- mScale = mScale + 1
- mLow = (mLow-g_ FirstQuater) * 2
- mHigh = (mHigh-g_ FirstQuater) * 2 + 1

3.2 ส่วนฮาร์ดแวร์ ที่เขียนด้วยภาษา VHDL และใช้การแสดงผลต่างๆผ่านทางโปรแกรมแมทแลบ

ส่วนนี้เราจะทำเพียงเฉพาะการเข้ารหัสเท่านั้น โดยตรงการถอดรหัสจะให้ใช้โปรแกรมแมทแลบในส่วนแรกแทน โดยการแสดงผลต่างๆจะใช้ผ่านทางโปรแกรมแมทแลบเช่นกัน ดังนั้นจะต้องมีส่วนอินเตอร์เฟสทั้งของแมทแลบ และ VHDL



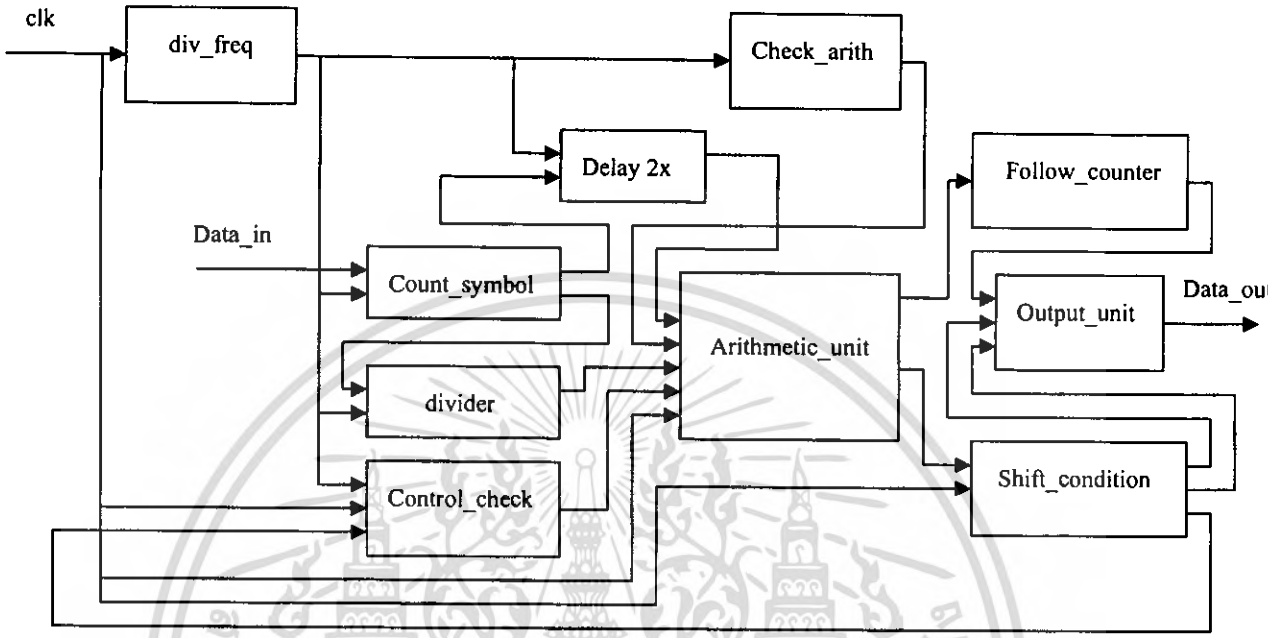
รูปที่ 3.4 แสดงบล็อกไดอะแกรมการทำงานหลักของฮาร์ดแวร์

3.2.1 สถาปัตยกรรมการเข้ารหัส

ตัวเข้ารหัสเลขคณิตมี 3 อินพุท ได้แก่ DATA_IN, RESET และ CLK หรือก็คือขา CLOCK กับอีก 1 เอาท์พุท คือ DATA_OUT

เมื่อ CLOCK มีค่าเป็น '1' ข้อมูลชุดใหม่จะสามารถอ่านได้ที่ DATA_INและทำตามกระบวนการด้วยตัวเข้ารหัส ส่วนสัญญาณ RESET จะถูกใช้เพื่อทำให้สถานะภายในตัวเข้ารหัสเป็นไปตามเงื่อนไข

ตามเงื่อนไขตอนเริ่มต้น กระบวนการของตัวเข้ารหัสนั้นจะถูกทำให้เป็นจังหวะเดียวกันด้วยสัญญาณ CLOCK และข้อมูลที่ถูกเข้ารหัสจะสามารถอ่านได้จาก DATA_OUT สถาปัตยกรรมทั้งหมดของการเข้ารหัสถูกแสดงได้ดังรูป



รูปที่ 3.5 แสดงบล็อกไดอะแกรมของส่วนการเข้ารหัส

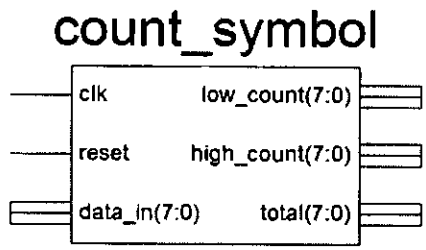
3.2.2 ส่วนประกอบต่างๆ (components) ของตัวเข้ารหัส

ตัวเข้ารหัสประกอบด้วยส่วนประกอบดังต่อไปนี้

- COUNT_SYMBOL ทำหน้าที่ในการอัพเดทโมเดล
- DIV_FREQ ทำหน้าที่หารความถี่ที่เข้ามาให้ลดลง 32 เท่า
- DELAY 2X ทำหน้าที่นำข้อมูลเลื่อนออกไป 1 clock
- DIVIDER ทำหน้าที่ในการคิดค่าผลหาร
- CHECK_ARITH ทำหน้าที่เป็นสัญญาณควบคุมบอกให้ทำการคำนวณในบล็อก ARITHMETIC_UNIT ได้
- SHIFT_CONDITION ใช้เป็นตัวเปรียบเทียบค่า LOW และ HIGH ว่าตรงกับเงื่อนไขในกรณีใด
- ARITHMETIC_UNIT ใช้แสดงการคำนวณค่าของ LOW และHIGH อันใหม่ที่ใช้ในตัวเข้ารหัส
- CONTROL_CHECK ทำหน้าที่ในการกำหนดสัญญาณขาเข้าให้วงจร ARITHMETIC_UNIT และกำหนดสัญญาณควบคุมให้วงจร SHIFT_CONDITION
- FOLLOW_COUNTER ทำหน้าที่ในการนับค่า mScale3
- OUTPUT_UNIT ใช้แสดงค่าเอาท์พุท

องค์ประกอบเหล่านี้สามารถอธิบายรายละเอียดในการทำงานที่มากขึ้นได้ดังนี้

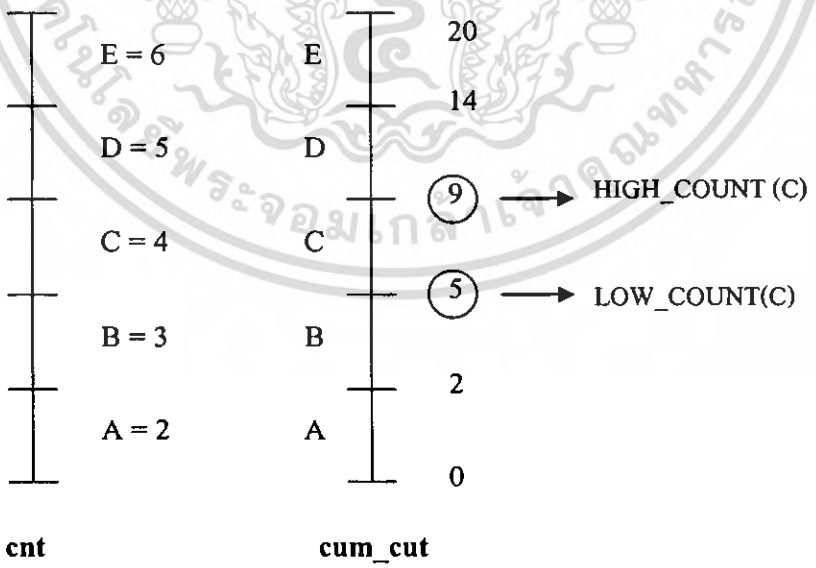
3.2.2.1 COUNT_SYMBOL



รูปที่ 3.6 บล็อกไดอะแกรมแสดงวงจรของ COUNT_SYMBOL

COUNT_SYMBOL จะมีอินพุททั้งหมด 3 อินพุท โดยมีขนาด 1 บิทอยู่ 2 อินพุท คือ RESET และ CLK ขนาด 8 บิทอยู่ 1 อินพุท คือ DATA_IN และมีเอาท์พุท 8 บิทอยู่ 3 เอาท์พุทคือ HIGH_COUNT , LOW_COUNT และ TOTAL

โดยการทำงานภายในนั้นจะเริ่มตรวจสอบที่ RESET ก่อนว่าเท่ากับ '1' หรือไม่ ถ้าเท่ากับ '1' ค่า HIGH_COUNT, LOW_COUNT, TOTAL ก็จะถูกเซทให้เป็นค่าตั้งต้น แต่ถ้า RESET เท่ากับ '0' ก็จะไปตรวจสอบที่ขา CLOCK ทุกครั้งที่ CLOCK เปลี่ยนแปลงเป็น '1' ก็จะอ่านค่า DATA_IN เข้ามาซึ่งในโปรแกรมนี้จะกำหนดให้ใช้ได้แค่ A,B,C,D,E เท่านั้น โดยจะให้เริ่มต้นว่าทุกตัวมีอยู่แล้วอย่างละ 1 ตัว ซึ่งจะเก็บอยู่ในตัวแปร cnt ค่า TOTAL จะนับว่ามีข้อมูลอยู่กี่ตัวแล้ว โดยเริ่มต้นเป็นค่า 5 เนื่องจากเราให้ A,B,C,D,E ทุกตัวมีอย่างละ 1 ตัวอยู่แล้ว ส่วน LOW_COUNT และ HIGH_COUNT จะอธิบายโดยแสดงให้เห็นจากภาพเพื่อให้ง่ายต่อความเข้าใจ

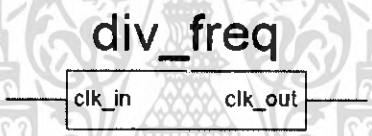


รูปที่ 3.7 แสดงการนับค่าของตัวแปร cnt และ cum_cnt

จากรูปทางด้านซ้ายจะเห็นได้ว่าตัวแปร cnt จะแสดงว่าตัวอักษรนั้นมีอยู่กี่ตัว ซึ่งจะได้ว่าค่า cnt = [2 3 4 5 6] ส่วนรูปทางด้านขวาจะแสดงเหมือนเอาจำนวนตัวของแต่ละตัวอักษรมาบวกเพิ่มเข้าไปเรื่อยๆจาก A ถึง E คล้ายๆกับค่านับสะสม ซึ่งจะเก็บในตัวแปร cum_cnt ซึ่งในตัวอย่างนี้ cum_cnt = [0 2 5 9 14 20] และ LOW_COUNT ของ C จะได้เท่ากับ 5 ส่วน HIGH_COUNT ของ C จะได้เท่ากับ 9จะสังเกตได้ว่า LOW_COUNT ก็คือค่าจำนวนตัวที่นับมาได้ก่อนที่จะนับจำนวนของตัวมัน ส่วน HIGH_COUNT ก็คือจำนวนตัวที่นับมาได้โดยรวมการนับจำนวนตัวของมันเข้าไปด้วย หลังจากมีการรับ DATA_IN ไปแล้ว ก็จะมีการอัพเดทโมเดลด้วยการเพิ่มจำนวนของเลขนั้นเพิ่มอีก 1 และค่าของ cum_cnt ก็จะต้องเปลี่ยนแปลงไปด้วย

และจะมีตัวสุดท้ายในการบอกให้หยุดการทำงานซึ่งคือ DATA_IN ค่า "00000000" โดยมันจะถูกเพิ่มเข้าไปที่ตัวสุดท้ายของข้อมูลที่เราใส่เข้าไปตั้งแต่ในส่วนเข้ารหัสโปรแกรมแมทแลบ ซึ่งจะให้ผลในการคำนวณค่า LOW_COUNT และ HIGH_COUNT เป็น "00000001" เท่ากัน ซึ่งเป็นกรณีที่ไม่สามารถเป็นไปได้จริง จึงนำเอามาเป็นค่าที่ใช้บอกให้โปรแกรมหยุดการทำงานได้ในบล็อก CHECK_ARITH

3.2.2.2 DIV_FREQ

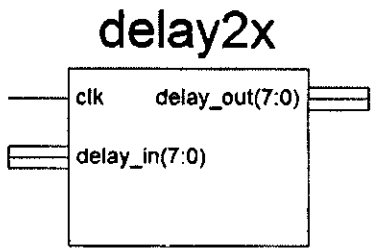


รูปที่ 3.8 บล็อกไดอะแกรมแสดงวงจรของ DIV_FREQ

DIV_FREQ จะมีอินพุท 1 อินพุท ขนาด 1 บิท คือ CLK_IN และมีเอาท์พุทขนาด 1 บิทอยู่ 1 เอาท์พุทคือ CLK_OUT

DIV_FREQ จะทำหน้าที่ในการหารความถี่ลง 32 เท่า เนื่องจากว่า COUNT_SYMBOL และ SHIFT_CONDITION นั้นทำงานคนละจังหวะกัน COUNT_SYMBOL ใช้ 1 คล็อก(clock) เท่ากับข้อมูล 1 ชุด ส่วน SHIFT_CONDITION นั้น ใช้ 32 คล็อก เท่ากับ ข้อมูล 1 ชุด จึงต้องมี DIV_FREQ มาช่วยทำให้จังหวะของ 2 ตัวนี้สอดคล้องกัน โดยการทำงานของ DIV_FREQ ก็คือจะนับจำนวนของ CLK_IN โดยเริ่มจาก 0 จนถึง 15 จังหวะนี้จะให้ CLK_OUT เป็น '0' แล้ว 15 CLK_IN ถัดไป จะได้ CLK_OUT เป็น '1' สลับกันอย่างนี้ไปเรื่อยๆ จะเห็นได้ว่า CLK_IN จะเร็วกว่า CLK_OUT บางครั้งจะใช้เรียก CLK_OUT ตัวนี้ว่า CLK_SLOW

3.2.2.3 DELAY_2X

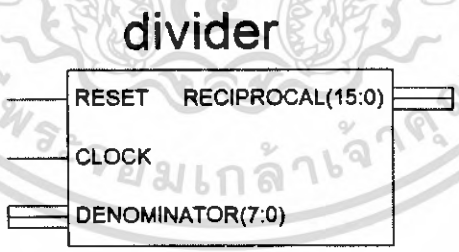


รูปที่ 3.9 บล็อกไดอะแกรมแสดงวงจรของ DELAY_2X

DELAY_2X จะมีอินพุทขนาด 8 บิท อยู่ 1 อินพุท คือ DELAY_IN และอินพุทขนาด 1 บิท อีก 1 อินพุท ซึ่งคือ CLK ส่วนเอาท์พุท จะมีขนาด 8 บิท อยู่ 1 อินพุท คือ DELAY_OUT

DELAY_2X จะทำหน้าที่ในการเลื่อนข้อมูลอินพุทให้ออกไปช้ากว่าเดิม 1 คล็อก โดยหลักการทำงานโดยใช้บัฟเฟอร์มาช่วย คือเมื่อเจอ CLK = '1' ก็ให้ส่งค่า DELAY_IN ไปที่ bufer1 แล้วพอเจอ CLOCK = '0' ก็ให้ส่งค่าจาก buffer1 ไปยัง buffer2 แล้วพอเจอ CLK = '1' ซึ่งเป็นของลูกใหม่ ก็จะส่งค่าจาก buffer2 ไปให้ DELAY_OUT ได้ใน 1 คล็อก สาเหตุที่ต้องมีบล็อกนี้ เนื่องจากว่า ARITHMETIC_UNIT นั้นจะต้องรับค่าอินพุท RECIPROCAL ที่เป็นเอาท์พุทของ DIVIDER และอินพุทHIGH_COUNT, LOW_COUNT ซึ่งเป็นเอาท์พุทของ COUNT_SYMBOL ในจังหวะที่พร้อมกัน ขา RECIPROCAL นั้นจะได้จาก TOTAL ซึ่งเป็นเอาท์พุทของ COUNT_SYMBOL ไปต่อเข้ากับ DIVIDER ดังนั้น RECIPROCAL จึงจะช้ากว่าHIGH_COUNT และ LOW_COUNT อยู่ 1 คล็อก ทำให้ต้องใช้ DELAY_2X ต่อท้ายกับ HIGH_COUNT และ LOW_COUNT เพื่อให้จังหวะพอดีกัน

3.2.2.4 DIVIDER



รูปที่ 3.10 บล็อกไดอะแกรมแสดงวงจรของ DIVIDER

DIVIDER จะมีอินพุททั้งหมด 3 อินพุท โดยมีขนาด 1 บิท อยู่ 1 อินพุท ซึ่งคือ CLOCK กับ RESET และขนาด 8 บิท อยู่ 1 อินพุท คือ DENOMINATOR ส่วนเอาท์พุท มีขนาด 16 บิท อยู่ 1 เอาท์พุท คือ RECIPROCAL

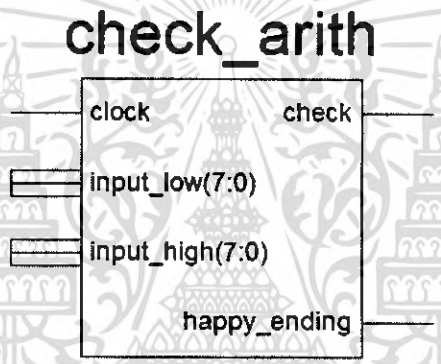
DIVIDER มีหน้าที่ในการหาค่าผลหารที่เกิดจากส่วนกลับของ DENOMINATOR (ซึ่งคือค่าที่รับมาจาก TOTAL จาก COUNT_SYMBOL) แล้วก็จะแสดงออกมาที่ RECIPROCAL โดยการทำงานของ

เราจะใช้ DENOMINATOR เพื่อชี้ตำแหน่งในตารางฐานข้อมูล(lookup table)ซึ่งได้คำนวณค่าผลหารที่เป็นทศนิยมให้แทนด้วยเลขฐานสองลงในตารางฐานข้อมูลเรียบร้อย

<u>หมายเหตุ</u> เมื่อคำนวณแล้ว จะพบว่าการใช้ค่า RECIPROCAL 16 บิท จะทำให้เราแทนค่าทศนิยมของส่วนกลับของ DENOMINATOR ตั้งแต่ค่าเริ่มต้นที่ 1/5 ไปเป็น 1/6, 1/7, 1/8, ...จะได้ถึง 1/256 ที่ทำให้ค่าทศนิยม 16 บิท ไม่ซ้ำกัน ดังนั้นเราจะใช้ได้ถึงค่า DENOMINATOR สูงสุดคือ 256 ซึ่งค่า DENOMINATOR เริ่มต้นจาก 5 ดังนั้นเราจะรับค่าได้สูงสุด 251 ตัวเท่านั้น ถ้าอยากจะมากกว่านี้ก็ต้องเพิ่มจำนวนบิทให้กับ RECIPROCAL โดยจำนวนสูงสุด (Y ตัว) ที่จะเข้ารหัสได้ เมื่อใช้ RECIPROCAL จำนวน X บิท สามารถหาได้จากสมการ

$$1/(Y-1) - 1/Y = 2^{-X}$$

3.2.2.5 CHECK_ARITH



รูปที่ 3.11 บล็อกไดอะแกรมแสดงวงจรของ CHECK_ARITH

CHECK_ARITH จะมีอินพุททั้งหมด 3 อินพุทโดยมีขนาด 8 บิท จำนวน 2 อินพุท คือ INPUT_LOW และมีขนาด 1 บิท 1 อินพุท คือ CLOCK ส่วนเอาท์พุทมี 2 เอาท์พุท ขนาด 1 บิททั้งคู่ คือ CHECK และ HAPPY_ENDING

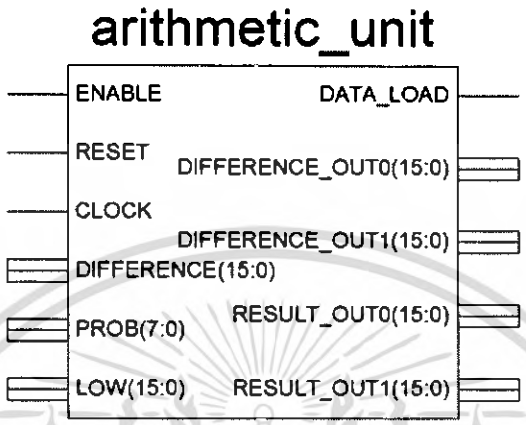
CHECK_ARITH นั้นจะทำหน้าที่อยู่ 2 อย่างคือ

1. การควบคุมการทำงานของ ARITHMETIC_UNIT ด้วยขา CHECK ที่จะไปต่อกับ ENABLE ของ ARITHMETIC_UNIT ให้เริ่มทำการคำนวณค่าได้ โดยการทำงานนั้นเพียงแค่เราตรวจสอบว่าค่า INPUT_HIGH และ INPUT_LOW นั้นไม่เป็นค่า "00000000" และ "00000001 " ซึ่ง INPUT_LOW และ INPUT_HIGH นั้นจะไม่มีทางเป็น "00000000" หรือ "00000001" พร้อมกัน ยกเว้นกรณีค่าเริ่มต้นที่เป็น "00000000" และค่าตัวสุดท้ายที่เป็น "00000001" เท่านั้น ค่า CHECK ถึงจะส่งออกไป ซึ่งมีลักษณะเหมือน CLOCK_SLOW ทุกประการ ยกเว้นแต่เริ่มช้ากว่าเพราะไม่เริ่มตั้งแต่ค่าเริ่มต้นและจะไม่มีในค่าหลังจากค่าสุดท้ายถูกส่งไป

2. การควบคุมให้หยุดการทำงานเพื่อให้บล็อกสุดท้าย คือ OUTPUT นั่นได้ทำการตรวจสอบเงื่อนไขสุดท้าย ด้วยขา HAPPY_ENDING ซึ่งจะตรวจสอบว่าขา INPUT_LOW และ INPUT_HIGH มีค่า

เป็น "00000001" เท่ากันหรือไม่ ถ้าใช่ค่าของขา HAPPY_ENDING ก็จะส่งสัญญาณกระตุ้นขึ้นมาเพื่อให้รู้ว่าข้อมูลสิ้นสุดหยุดการคำนวณค่าต่อ และให้ตรวจสอบเงื่อนไขสุดท้ายแทน

3.2.2.6 ARITHMETIC_UNIT



รูปที่ 3.12 บล็อกไดอะแกรมแสดงวงจรของ ARITHMETIC_UNIT

ARITHMETIC_UNIT จะมีอินพุททั้งหมด 8 อินพุทโดยมีขนาด 8 บิท อยู่ 2 อินพุท คือ HIGH_-COUNT และLOW_COUNT ขนาด 32 บิท อยู่ 2 อินพุทคือ mHigh และ mLow ขนาด 16 บิท อยู่ 1 อินพุท คือ RECIPROCAL ขนาด 1 บิท อยู่ 3 อินพุท คือ CLOCK,ENABLE และ RESET ส่วนเอาท์พุทจะมี 2 เอาท์พุท ขนาด 32 บิททั้งคู่ คือ mHigh_new และ mLow_new

ARITHMETIC_UNIT ทำหน้าที่ในการคำนวณค่า mHigh และ mLow ตัวใหม่ โดยตั้งค่าเริ่มต้นของ mHigh, mLow และ new คือ

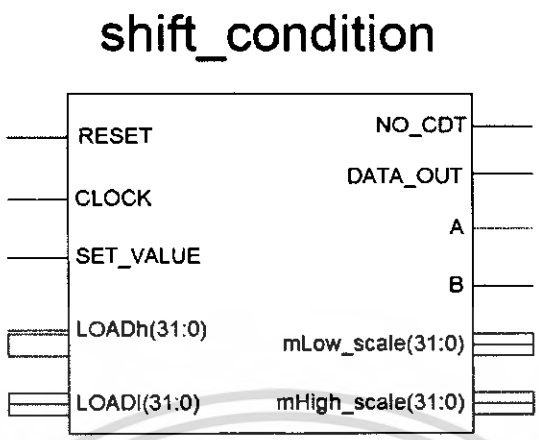
mHigh = '11111111111111111111111111111111'

mLow = '00000000000000000000000000000000'

ENABLE ทำหน้าที่ในการบอก ARITHMETIC_UNIT ให้เริ่มคำนวณได้ ค่าที่ได้มาถูกต้องแล้ว โดยการคำนวณนี้จะเสร็จสิ้นต้องใช้สัญญาณนาฬิกาทั้งสิ้นจำนวน 4 คล็อกซึ่งจะมีสัญญาณDATA_LOAD ที่คอยควบคุมอยู่ในตัว ARITHMETIC_UNIT ว่าค่าที่คำนวณได้นั้นถูกต้องแล้ว ถึงสามารถที่จะให้ค่าที่คำนวณเสร็จสิ้นแล้วออกมาทาง mHigh_new และ mLow_new

การคำนวณนี้จะใช้ค่า RECIPROCAL ซึ่งจริงๆคือค่าทศนิยม ดังนั้นเมื่อนำไปคูณกับค่าตัวไหนผลที่ได้ที่จะนำไปใช้ก็จะต้องถูกตัดบิทในช่วงท้ายออกไป 16 บิทเท่ากับขนาดของ RECIPROCAL ซึ่งก็เหมือนกับว่าได้ค่าเป็นจำนวนเต็ม และ ในการคำนวณตรงนี้จะมีการเพิ่มบิท 0 เข้าไปด้านหน้าของค่า mHigh , mLow เพื่อป้องกันการเกิดโอเวอร์โฟลว์

3.2.2.7 SHIFT_CONDITION

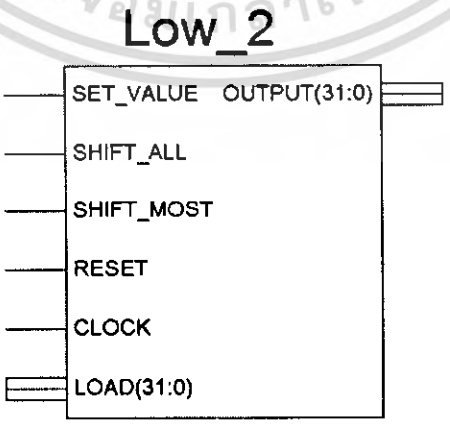


รูปที่ 3.13 บล็อกไดอะแกรมแสดงวงจรของ SHIFT_CONDITION

SHIFT_CONDITION จะมีอินพุททั้งหมด 5 อินพุท โดยมีขนาด 32 บิท 2 อินพุท คือ LOADh และ LOADl และขนาด 1 บิท 3 อินพุท คือ CLOCK, RESET และ SET_VALUE ส่วนเอาท์พุทจะมีทั้งหมด 6 เอาท์พุท โดยมีขนาด 32 บิท 2 เอาท์พุท คือ mHigh_scale และ mLow_scale ขนาด 1 บิท 4 เอาท์พุท คือ A, B, DATA_OUT และ NO_CDT

SHIFT_CONDITION จะทำหน้าที่ในการตรวจสอบเงื่อนไขของ LOADh , LOADl ที่มาจาก mHigh_new และ mLow_new ว่า HSB และ SECOND_HSB ของทั้งสองขานั้น มีเงื่อนไขตรงกับกรณีใดบ้าง ซึ่งมี 2 กรณี คือ กรณีที่ บิทแรกเหมือนกัน A จะได้เป็น 1 แต่ถ้าบิทแรกของ LOADh เป็น 1 และ SECOND_MSB เป็น 0 ส่วนบิทแรกของ LOADl เป็น 0 และ SECOND_LSB เป็น 1 ก็จะทำให้ B เป็น 1 เช่นกัน และถ้าหากไม่ตรงทั้ง 2 กรณี ก็จะเกิด NO_CDT เป็น 1 แทน โดยบล็อกนี้จะประกอบไปด้วยบล็อกการทำงานภายในอีก 3 ตัวคือ

3.2.2.7.1 LOW



รูปที่ 3.14 บล็อกไดอะแกรมแสดงวงจรของ HIGH และ LOW

LOW มีอินพุททั้งหมด 6 อินพุท มีขนาด 32 บิท อยู่ 1 อินพุท คือ LOAD นอกนั้นอีก 5 อินพุท มีขนาด 1 บิท คือ CLOCK , RESET , SET_VALUE , SHIFT_ALL และSHIFT_MOST

LOW ทำหน้าที่ในการเลื่อนบิทตามเงื่อนไขของ SHIFT_ALL และ SHIFT_MOST ขา SHIFT_-ALL แสดงว่าให้เลื่อนบิททั้ง 32 บิทของ LOW ไปทางซ้าย 1 บิท แล้วนำบิท 0 ไปแทนที่บิท LSB ของ LOW ส่วนขา SHIFT_MOST แสดงว่าให้เลื่อนบิทตั้งแต่บิท SECOND_HSB ถึง บิท LSB (31 บิท)ไปทางซ้าย 1 บิท โดยให้ บิท HSB อยู่อย่างเดิม แล้วก็เช่นกันนำบิท 0 ไปแทนที่บิท LSB ที่เลื่อนไป โดยการทำงานนี้จะเริ่มทำงานได้ด้วยขา SET_VALUE ซึ่งต้องมีค่าเท่ากับ 1 จึงจะเริ่มการทำงาน

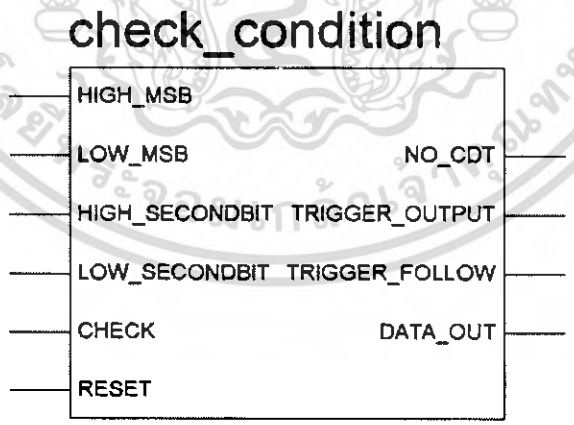
ขา LOAD ของ LOW ก็คือ LOADl ของ SHIFT_CONDITION ขา SET_VALUE ก็คือขาเดียวกันกับของ SHIFT_CONDITION ส่วนขา SHIFT_ALL จะต่อกับ TRIGGER_OUTPUT ของ CHECK_CONDITION และขา SHIFT_MOST จะต่อกับขา TRIGGER_FOLLOW ของ CHECK_-CONDITION โดยค่า LOAD นี้จะถูกตั้งให้เป็น '00000000000000000000000000000000' ตั้งแต่เริ่มต้น

3.2.2.7.2 HIGH

บล็อกทุกอย่างจะเหมือนกันกับของ LOW แต่แค่บิทที่จะถูกนำมาแทนที่บิท LSB ของ HIGH นั้นจะเป็นบิท 1 แทน และค่าเริ่มต้นของ LOAD จะเท่ากับ '11111111111111111111111111111111' และต่อเชื่อมกับ LOADh ของ SHIFT_CONDITION

โดยสาเหตุที่ให้ค่า LOADh และ LOADl เป็นเท่านี้เนื่องจากว่าค่าของ 2 ขานี้ จะถูกส่งไปที่ CHECK_CONDITION แล้วจะทำให้ไม่เกิดค่า OUTPUT ออกมา เพราะเป็นกรณีที่อยู่นอกเงื่อนไขจะรับค่าตัวใหม่มาอ่านแทน ดังนั้นค่าเริ่มต้นก็จะไม่มีผลอะไร

3.2.2.7.3 CHECK_CONDITION



รูปที่ 3.15 บล็อกไดอะแกรมแสดงวงจรของ CHECK_CONDITION

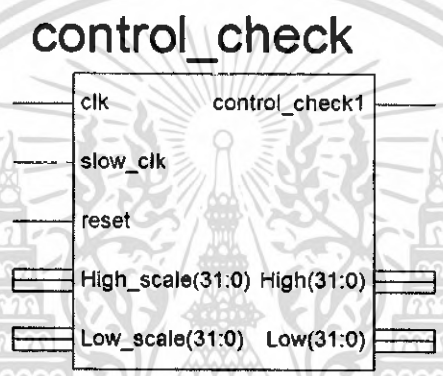
CHECK_CONDITION มีอินพุททั้งหมด 6 อินพุท ซึ่งมีขนาด 1 บิททั้งหมด คือ CHECK , HIGH_MSB , HIGH_SECONDBIT, LOW_MSB, LOW_SECONDBIT และRESET ส่วนเอาท์พุทมีทั้งหมด 4 เอาท์พุท ขนาด 1 บิททั้งหมด ได้แก่ DATA_OUT, NO_CDT, TRIGGER_FOLLOW และ TRIGGER_OUTPUT

ทั้งหมด 4 เอาท์พุท ขนาด 1 บิททั้งหมด ได้แก่ DATA_OUT, NO_CDT, TRIGGER_FOLLOW และ TRIGGER_OUTPUT

CHECK_CONDITION ทำหน้าที่ในการตรวจสอบเงื่อนไขจากขา HIGH_SECONDBIT HIGH_MSB , LOW_MSB, LOW_SECONDBIT เพื่อดูว่าเป็นกรณีใดแล้วจะทำการขยายช่วงอย่างไร โดย TRIGGER_OUTPUT ก็คือ A ของ SHIFT_CONDITION ส่วน TRIGGER_FOLLOW ก็คือ B ของ SHIFT_CONDITION

การเลื่อนบิทก็คือการขยายขอบเขตของช่วงให้กว้างขึ้น ซึ่งปกติแล้วตามทฤษฎีการขยายช่วงนั้นไม่ว่ากรณีใดก็จะมีการคูณด้วย 2 ซึ่งมันก็คือการเลื่อนบิทไปทางซ้าย 1 บิทนั่นเอง

3.2.2.8 CONTROL_CHECK



รูปที่ 3.16 บล็อกไดอะแกรมแสดงวงจรของ CONTROL_CHECK

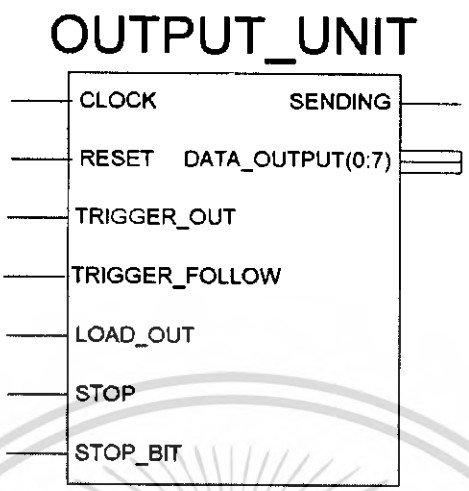
CONTROL_CHECK มีอินพุททั้งหมด 5 อินพุท โดยมีขนาด 32 บิท 2 อินพุท และขนาด 1 บิท อีก 3 อินพุท ส่วนเอาท์พุทมีขนาด 32 บิท 2 เอาท์พุท คือ High และ Low และขนาด 1 บิท อีก 1 เอาท์พุท คือ CONTROL_CHECK1

CONTROL_CHECK ทำหน้าที่ 2 ส่วน คือ

1. ทำหน้าที่ควบคุมการทำงานของ SHIFT_CONDITION โดย CONTROL_CHECT1 จะไปต่อกับ SET_VALUE ของ SHIFT_CONDITION

2. ทำหน้าที่จัดเรียงให้จังหวะของ mHigh_scale และ mLow_scale เอาท์พุทของ SHIFT_-CONDITIONให้ตรงกันกับจังหวะของ mHigh และ mLow ซึ่งเป็นอินพุทของ ARITHMETIC_UNIT

3.2.2.9 OUTPUT_UNIT



รูปที่ 3.17 บล็อกไดอะแกรมแสดงวงจรของ OUTPUT_UNIT

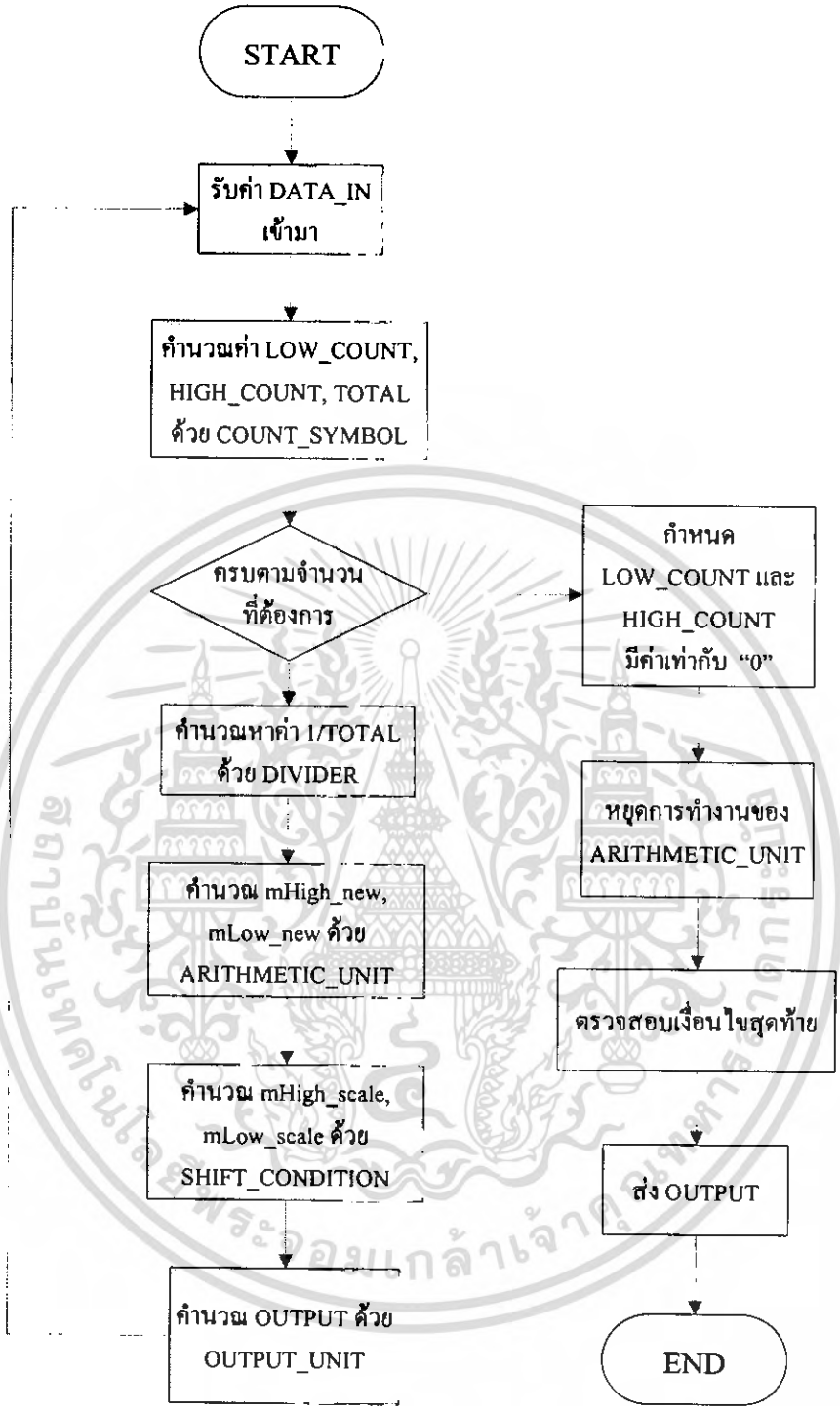
OUTPUT_UNIT ประกอบไปด้วยอินพุทขนาด 1 บิทจำนวน 6 อินพุท คือ TRIGGER_OUT, TRIGGER_FOLLOW, CLOCK, RESET, LOAD_FOLLOW และ LOAD_OUT เอาท์พุท 1 ตัวขนาด 1 บิทคือ SENDING และ เอาท์พุทขนาด 7 บิท อีก 1 เอาท์พุท คือ DATA_OUT

หน้าที่หลักของ OUTPUT_UNIT มีอยู่ 3 อย่าง คือ

1. จัดเตรียมเอาท์พุททั้งหมดที่เกิดขึ้นทั้งหมดทั้งในกรณี E1, E2 และ E3 โดยเรียงต่อติดกันให้เป็นขนาด 8 บิท หากข้อมูลที่เก็บใน DATA_OUT ครบ 8 บิทแล้วจะมีสัญญาณ SENDING เป็น 1 และเมื่อจบข้อมูลทั้งหมดจะทำการเติมบิทสิ้นสุดลงไปโดยเติม STOP_BIT ที่ STOP เท่ากับ 1 เกิดขึ้น 1 บิทและบิทคอมพลีเมนท์ของ STOP_BIT จำนวนเท่ากับจำนวนที่นับ TRIGGER_FOLLOW + 1 และสิ้นสุดด้วยการเติมบิท '0' ไปจนครบ 8 บิท

2. นับจำนวนครั้งที่เกิดกรณี E3 และสร้างบิทที่ต้องนำไปแทรกกับชุดข้อมูลถัดไปในกรณี E3 โดยบิทที่จะนำไปแทรกนี้คือ บิทคอมพลีเมนท์ของเอาท์พุทในกรณี E1 หรือ E2 ที่จะเกิดถัดไป ซึ่งต้องแทรกบิทดังกล่าวไปเป็นจำนวนเท่ากับค่าที่ได้นับการเกิดกรณี E3 ก่อนหน้านี้

3. เลือกเอาท์พุทที่จะแสดงออกมาซึ่งขึ้นอยู่กับสัญญาณ TRIGGER_OUT และ TRIGGER-_FOLLOW ซึ่งสัญญาณ TRIGGER_OUT จะแสดงการเกิดกรณี E1 และ E2 จะเลือกอินพุทจาก LOAD_OUT ส่วนสัญญาณ TRIGGER_FOLLOW จะแสดงการเกิดกรณี E3 และพิจารณาข้อมูล LOAD_OUT มาทำคอมพลีเมนท์



รูปที่ 3.18 โฟลว์ชาร์ตแสดงการทำงานของการเข้ารหัสเลขคณิต ในส่วนของการจำลองการทำงาน

บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดลอง

การทดลองก็ได้แบ่งออกเป็นสองส่วนเช่นเดียวกัน

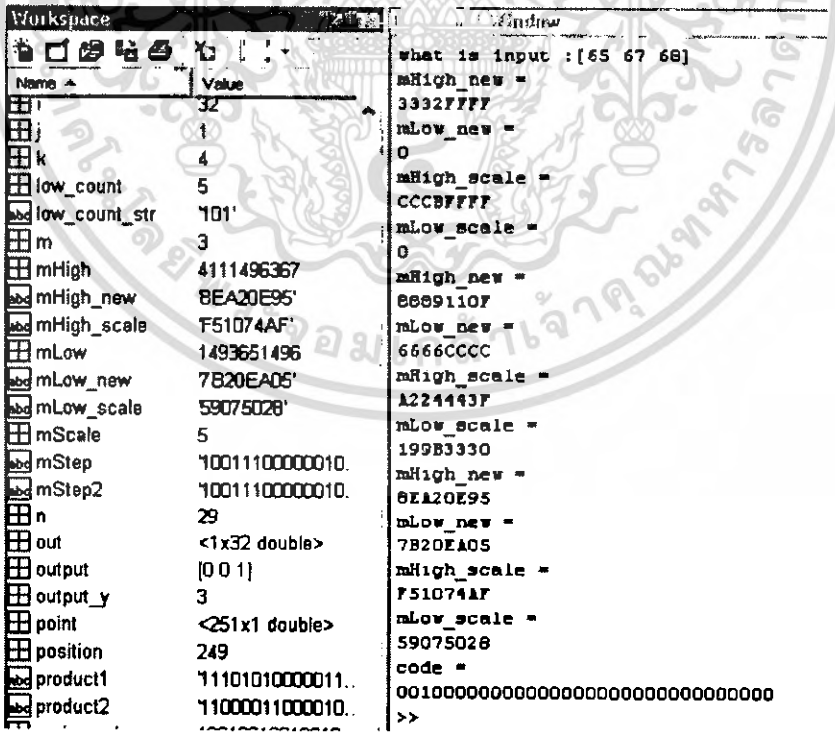
ส่วนแรก ส่วนที่เป็นซอฟท์แวร์ของการเข้ารหัสเลขคณิตนี้ ซึ่งเขียนขึ้นด้วยโปรแกรม แมทแลบ โดยตรวจสอบว่าโปรแกรมแมทแลบที่เขียนขึ้นนั้นให้ผลลัพธ์ที่ถูกต้องทั้งในส่วนของการเข้ารหัสและการถอดรหัส ว่ามีการการบีบอัดข้อมูล และผลลัพธ์ที่ได้ออกมาเป็นค่าเดิมหรือไม่

ส่วนที่สอง ส่วนที่เป็นซอฟท์แวร์ของการเข้ารหัสเลขคณิตนี้ ซึ่งเขียนขึ้นด้วยโปรแกรม VHDL โดยจะแสดงการจำลองผลการทำงานในแต่ละบล็อกไดอะแกรม ว่าให้ผลถูกต้องตรงกับหน้าที่ที่ต้องการหรือไม่ และตรวจสอบผลลัพธ์ในแต่ละส่วนกับโปรแกรมแมทแลบที่เขียนไว้ จะได้ทราบว่าภาษา VHDL ที่เขียนนั้นให้ผลที่ถูกต้อง แล้วนำผลที่ได้จากการเข้ารหัสมาเข้าในส่วนการถอดรหัสของโปรแกรม แมท - แลบ เพื่อตรวจสอบว่าได้ข้อมูลกลับเป็นตัวเดิมหรือไม่

4.2 ผลการทดลอง

4.2.1 ส่วนแรก โปรแกรมแมทแลบ

- **การเข้ารหัส** ผลที่ได้จากการเข้ารหัสผ่านโปรแกรมแมทแลบ โดยให้อินพุทเป็น [65 67 68]



รูปที่ 4.1 แสดงหน้าผลของส่วนการเข้ารหัสของโปรแกรมแมทแลบ

จากภาพที่แสดงจะเห็นได้ว่ามันจะแสดงผลของการหาค่าของ mHigh_new, mLow_new และ mHigh_scale, mLow_scale ในแต่ละครั้ง ซึ่งในตัวอย่างมีอินพุทเข้ามา 3 ตัว ก็จะมีค่าของ mHigh_new, mLow_new และ mHigh_scale, mLow_scale อย่างละ 3 ค่า เช่นกัน โดยค่า mHigh_new , mLow_new และ mHigh_scale, mLow_scale ชุดแรก จะเป็นผลของอักษรตัวที่หนึ่งซึ่งคือ '65' สามารถอธิบายได้ว่าเมื่อรับค่า '65' แล้ว จะได้ผลของช่วงใหม่ดังนี้

- mHigh_new = '3332FFF' (เป็นผลที่แสดงในรูปฐานสิบหก ซึ่งก็คือฐานสอง 32 บิทเหมือนเดิม)

- mLow_new = '0'

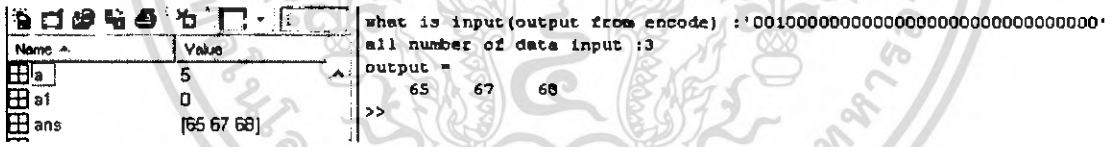
นำค่า mHigh_new, mLow_new ที่ได้ไปขยายช่วง จะได้ผลดังนี้

- mHigh_scale = 'CCCBFFF'

- mLow_scale = '0'

ส่วนอีก 4 ตัวชุดต่อไปก็จะคือค่าของ '67' ที่เกิดต่อจากการนำผลลัพธ์ของค่า '65' มาคิดต่อ และทำอย่างนี้ไปเรื่อยๆจนตัวสุดท้าย ก็จะได้โค้ดที่ถูกเข้ารหัสแล้วเป็น '001000000' ซึ่งจริงๆแล้วรหัสที่ถูกคำนวณได้จริงๆนั้นจะแค่ '001000000' ส่วนที่มี 0 เติมต่อมาทั้งหมดนั้นเพื่อให้ครบ 32 บิทเป็นอย่างน้อย เพราะตรงส่วนถอดรหัสนั้นจะต้องรับค่าอย่างน้อย 32 บิทก่อน ซึ่งถ้าค่าที่เข้ารหัสจริงๆคือ '001000000' ก็จะมีจำนวนบิทที่ถอดได้เป็น 9 บิท ซึ่งก็เท่ากับ 1 ตัวอักษร (8 บิท) กับอีก 1 บิท เท่านั้น ซึ่งแสดงได้ว่ามันบีบอัดได้จริง

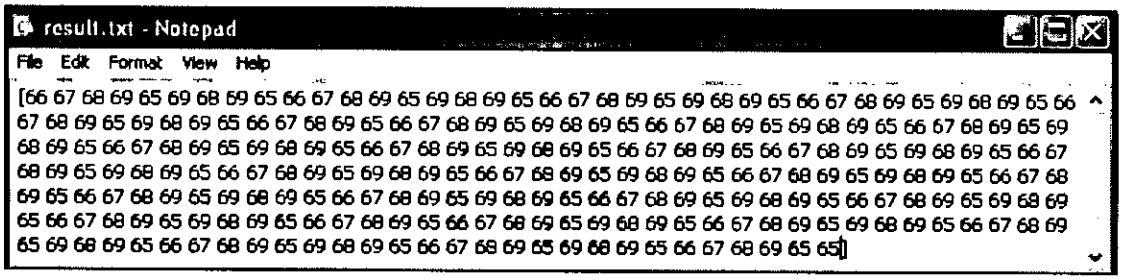
- การถอดรหัส ผลที่ได้จากการถอดรหัสผ่านโปรแกรมแมทแลบ โดยให้อินพุทเป็น ['00100000000000000000000000000000'] และป้อนจำนวนตัวอักษรของข้อมูลที่ต้องการให้ถอดได้



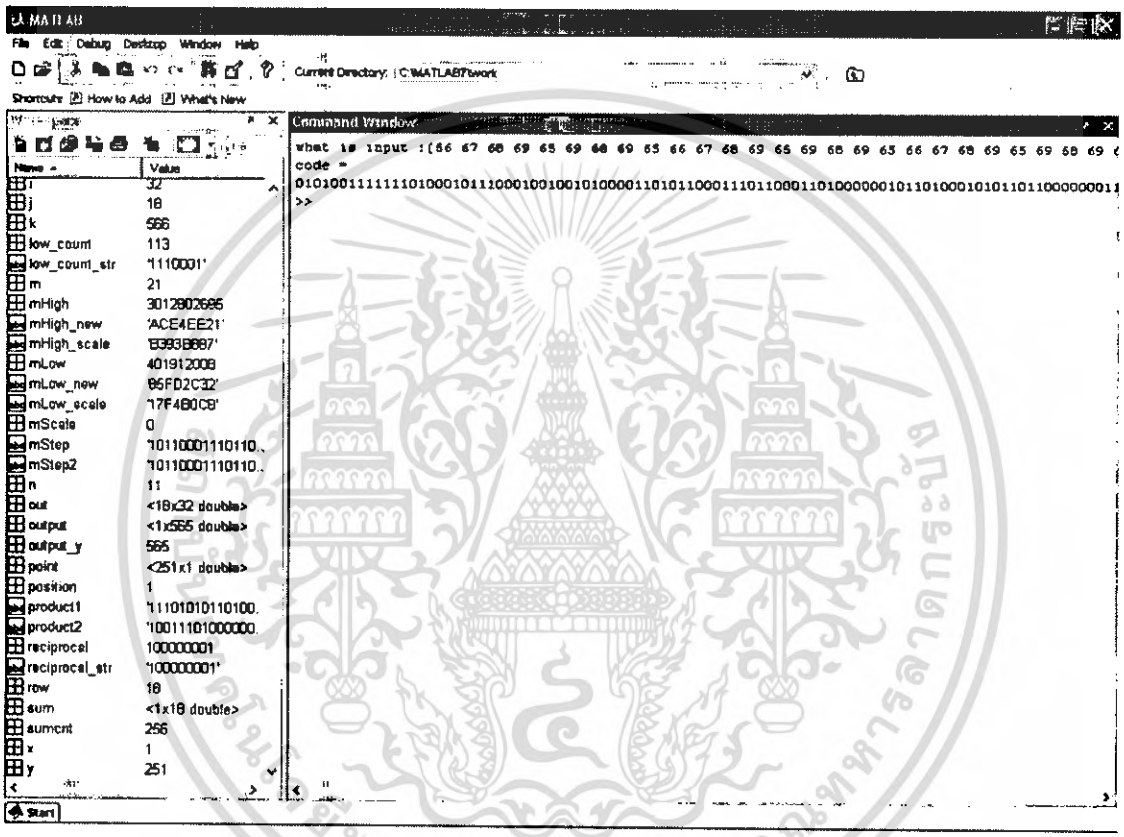
รูปที่ 4.2 แสดงหน้าผลของส่วนการถอดรหัสของโปรแกรมแมทแลบ

อินพุทที่เราใส่เข้าไปนั้นจะต้องบอกจำนวนตัวอักษรที่ต้องการให้ถอดรหัสด้วย หรือก็คือจำนวนตัวอักษรที่ป้อนเข้าในส่วนเข้ารหัสนั่นเอง ซึ่งที่เราต้องระบุก็เพื่อให้โปรแกรมในส่วนถอดรหัสรู้ว่าจะหยุดการคำนวณเมื่อไหร่ และจากตัวอย่างในรูปจะเห็นว่าผลลัพธ์ที่ได้จากการถอดรหัสก็คือ [65 67 68] ซึ่งก็ตรงกับข้อมูลที่ป้อนเข้าในส่วนเข้ารหัส

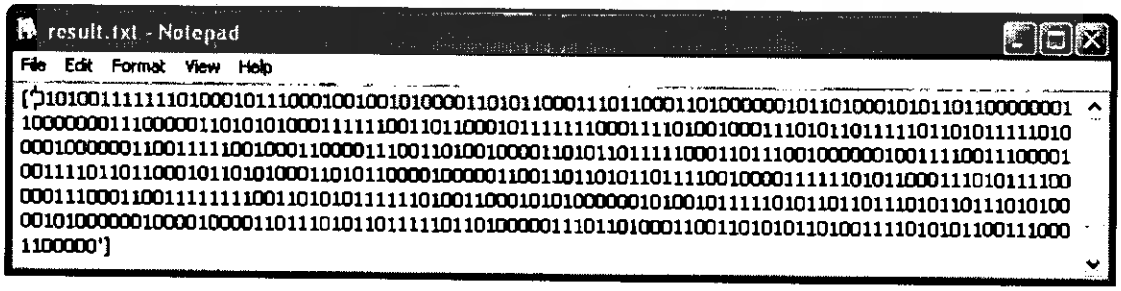
เราจะลองให้อินพุทมีทั้งหมด 251 ตัว ซึ่งเป็นจำนวนที่มากที่สุดที่โปรแกรมรองรับได้ว่ายังคงสามารถถอดรหัสได้ถูกต้องหรือไม่



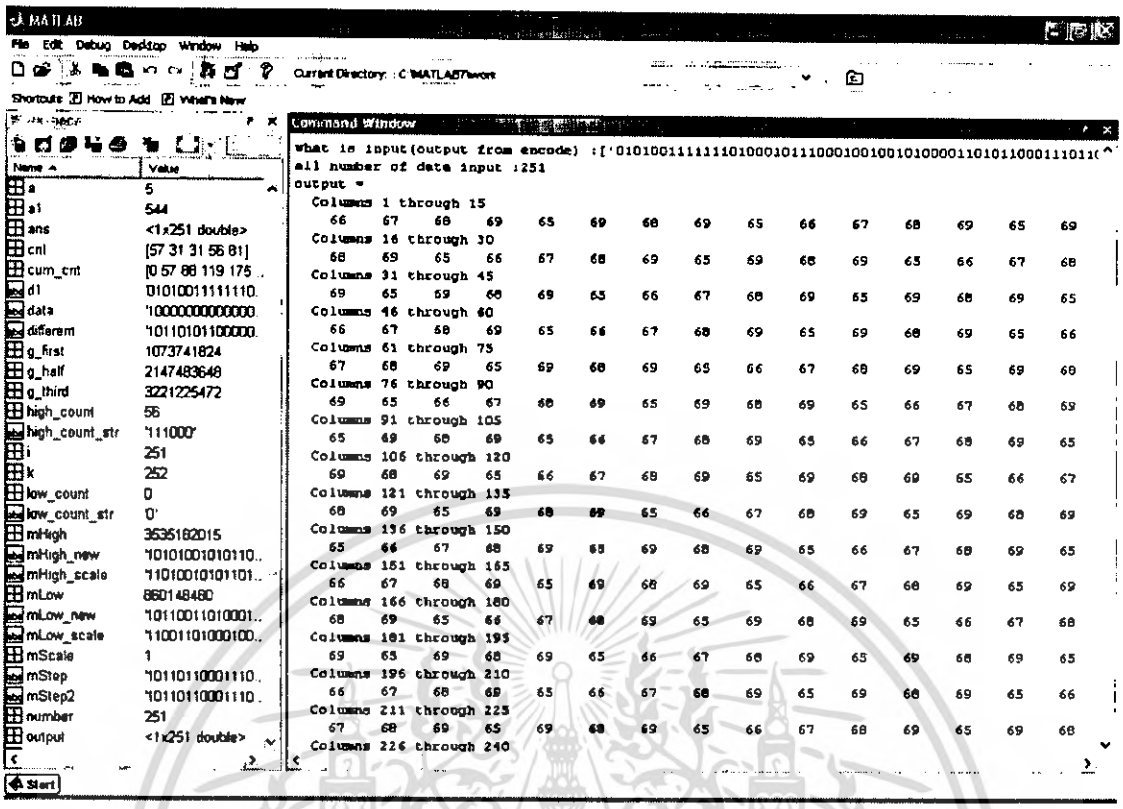
รูปที่ 4.3 แสดงอินพุท 251 ตัว ที่จะใช้ป้อนเข้าไปในโปรแกรมแมทแลบ



รูปที่ 4.4 แสดงผลลัพธ์ที่ได้จากการเข้ารหัสผ่านโปรแกรมแมทแลบ ป้อนอินพุท 251 ตัว จากรูปจะเห็นว่าค่า y = 251 ซึ่งเป็นค่าของจำนวนอินพุทที่ป้อนเข้าไป

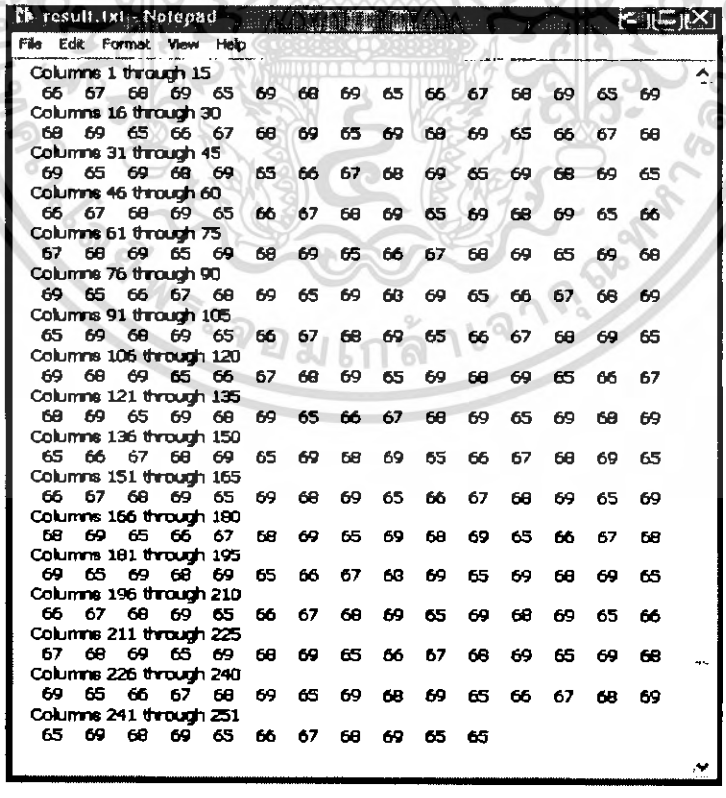


รูปที่ 4.5 แสดงผลลัพธ์ที่ได้จากการเข้ารหัสผ่านโปรแกรมแมทแลบ ป้อนอินพุททั้ง 251 ตัว



รูปที่ 4.6 แสดงผลลัพธ์ที่ได้จากการถอดรหัสผ่านโปรแกรมแมทแลบ

เมื่อป้อนค่าที่ถูกเข้ารหัสแล้วจะได้ผลลัพธ์เหมือนค่าก่อนเข้ารหัส ซึ่งสามารถทำได้สูงสุดถึง 251 ตัว

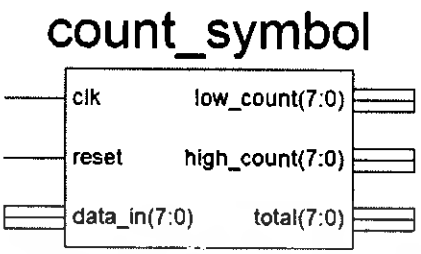


รูปที่ 4.7 แสดงผลลัพธ์ที่ได้จากการถอดรหัสผ่านโปรแกรมแมทแลบ ได้เอาท์พุท 251 ตัว

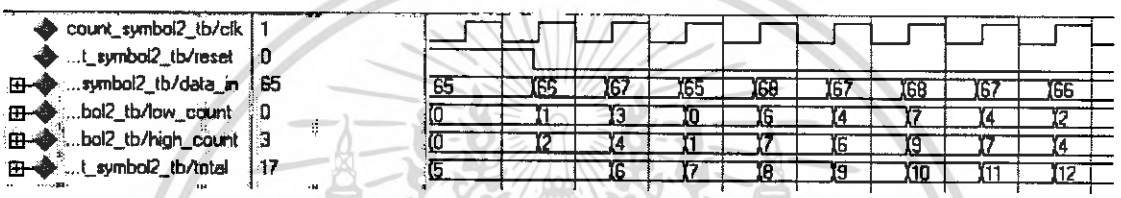
4.2.2 ส่วนที่สอง โปรแกรม VHDL

ผลการทดลองจากการจำลองการทำงานของแต่ละบล็อกไดอะแกรมได้ดังนี้

4.2.2.1 COUNT_SYMBOL



รูปที่ 4.8 บล็อกไดอะแกรมแสดงวงจรของ COUNT_SYMBOL

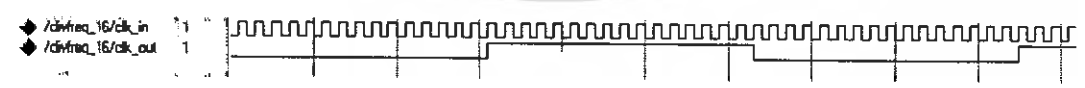


รูปที่ 4.9 แสดงผลจำลองการทำงานของวงจร COUNT_SYMBOL

จากผลการทดลองจะเห็นได้ว่า COUNT_SYMBOL ทำหน้าที่ในการอัพเดท ค่า LOW_COUNT, HIGH_COUNT , TOTAL โดยค่า TOTAL จะเพิ่มขึ้นทีละหนึ่งทุกครั้งที่มี DATA_IN ตัวใหม่เข้ามา ส่วนค่า LOW_COUNT, HIGH_COUNT ขึ้นอยู่กับ DATA_IN ที่รับเข้ามา

4.2.2.2 DIV_FREQ

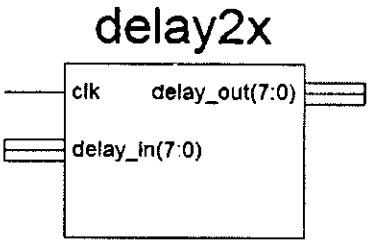
รูปที่ 4.10 บล็อกไดอะแกรมแสดงวงจรของ DIV_FREQ



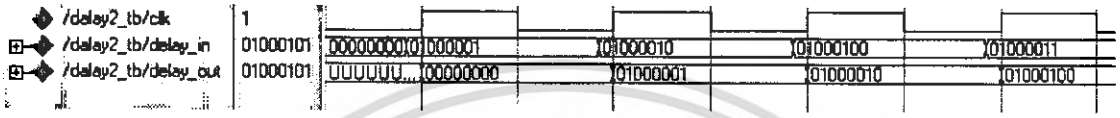
รูปที่ 4.11 แสดงผลจำลองการทำงานของวงจร DIV_FREQ

จากผลการทดลองจะเห็นได้ว่า CLK_OUT จะมีความถี่ลดลง 32 เท่า คือ CLK_IN 1 ไซเคิล (cycle) จะเท่ากับ CLK_OUT 32 ไซเคิล

4.2.2.3 DELAY_2X



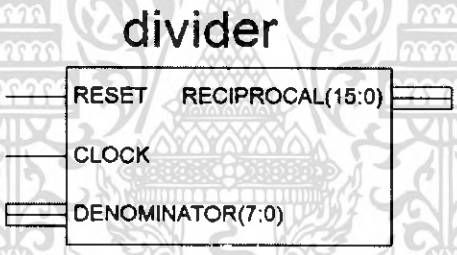
รูปที่ 4.12 บล็อกไดอะแกรมแสดงวงจรของ DELAY_2X



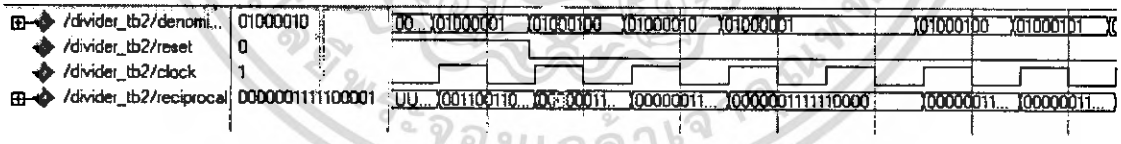
รูปที่ 4.13 แสดงผลจำลองการทำงานของวงจร DELAY_2X

จากผลการทดลองจะเห็นได้ว่าข้อมูลใน DELAY_OUT จะถูกเลื่อนเวลาออกไปจาก DELAY_IN 1 คล็อก

4.2.2.4 DIVIDER



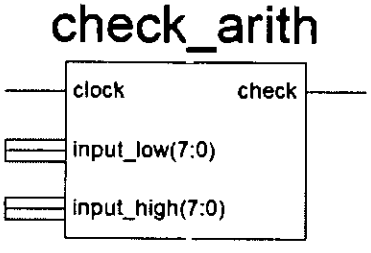
รูปที่ 4.14 บล็อกไดอะแกรมแสดงวงจรของ DIVIDER



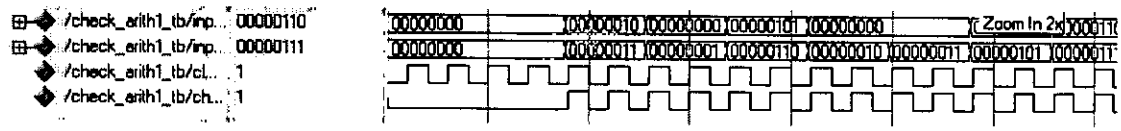
รูปที่ 4.15 แสดงผลจำลองการทำงานของวงจร DIVIDER

จากผลการทดลองจะเห็นได้ว่าสามารถที่จะชี้ค่าของการหารได้ตรงตำแหน่งในตารางฐานข้อมูล

4.2.2.5 CHECK_ARITH



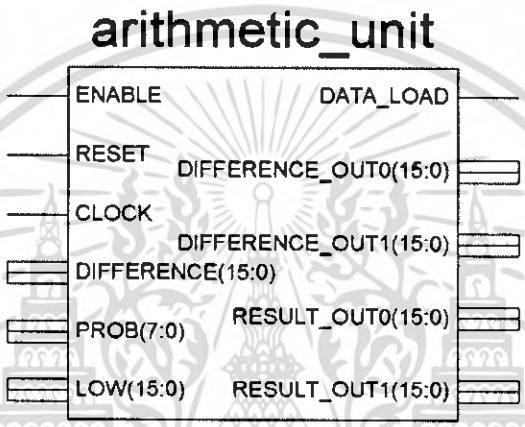
รูปที่ 4.16 บล็อกไดอะแกรมแสดงวงจรของ CHECK_ARITH



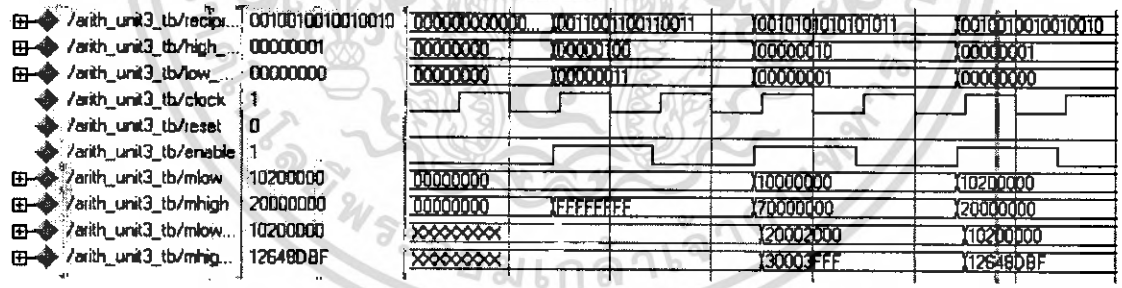
รูปที่ 4.17 แสดงผลจำลองการทำงานของวงจร CHECK_ARITH

จากผลการทดลองจะเห็นได้ว่า สัญญาณ check จะเหมือนสัญญาณนาฬิกา เพียงแต่จะเริ่มตรงตำแหน่งที่ INPUT_LOW, INPUT_HIGH มีค่าไม่เป็น ' 00000000' พร้อมกัน

4.2.2.6 ARITHMETIC_UNIT



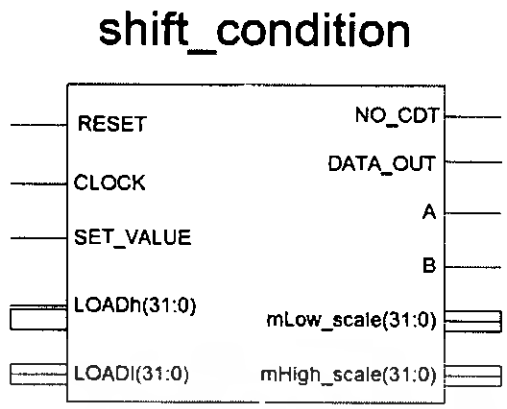
รูปที่ 4.18 บล็อกไดอะแกรมแสดงวงจรของ ARITHMETIC_UNIT



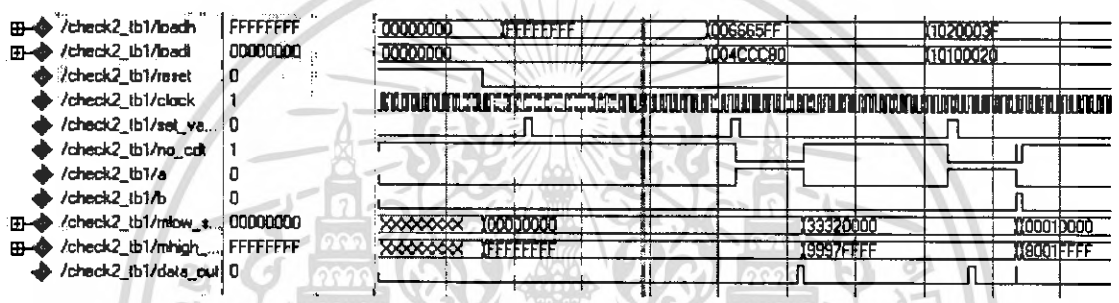
รูปที่ 4.19 แสดงผลจำลองการทำงานของวงจร ARITHMETIC_UNIT

จากผลการทดลองจะเห็นได้ว่า การคำนวณของ ARITHMETIC_UNIT จะเริ่มที่ตำแหน่ง ENABLE เป็น '1'

4.2.2.7 SHIFT_CONDITION



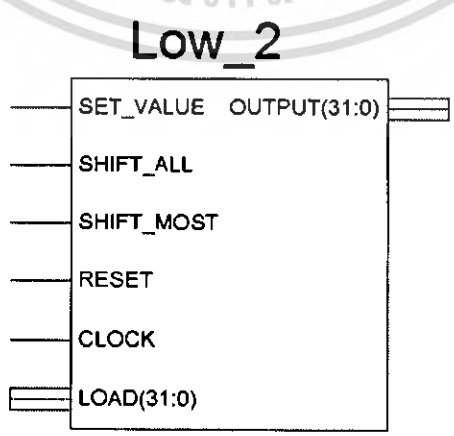
รูปที่ 4.20 บล็อกไดอะแกรมแสดงวงจรของ SHIFT_CONDITION



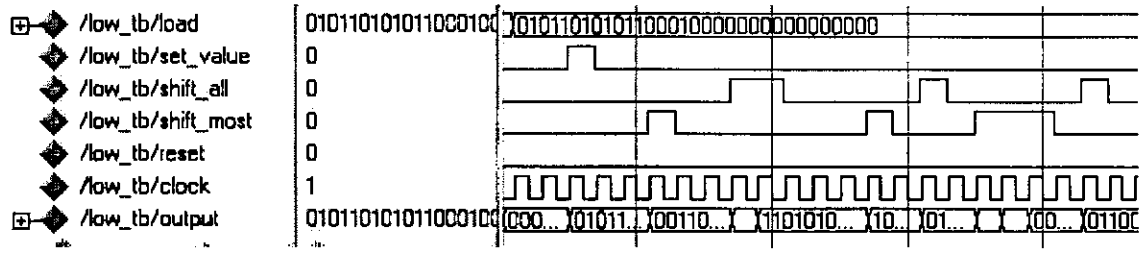
รูปที่ 4.21 แสดงผลจำลองการทำงานของวงจร SHIFT_CONDITION

จากผลการทดลองจะเห็นได้ว่า SET_VALUE = '1' ซึ่งจะเป็นตำแหน่งที่ให้เริ่มทำการตรวจสอบเงื่อนไขในการเลื่อนบิทได้ และมันจะเลื่อนไปจากตำแหน่งที่ LOADh และ LOADl เริ่มอ่านค่า 4 คล็อก เนื่องจากว่าในความเป็นจริงมันจะต้องคอยค่า LOAD ทั้ง 2 มาจากบล็อก ARITHMETIC_UNIT ซึ่งจะทำการคำนวณเสร็จสิ้นใช้จำนวน 4 ไซเคิล แล้วหลังจากนี้ถึงจะมีผลที่เกิดจากการตรวจสอบเงื่อนไขได้ ซึ่งนั่นก็คือขา A และ B

4.2.2.8 LOW



รูปที่ 4.22 บล็อกไดอะแกรมแสดงวงจรของ HIGH และ LOW

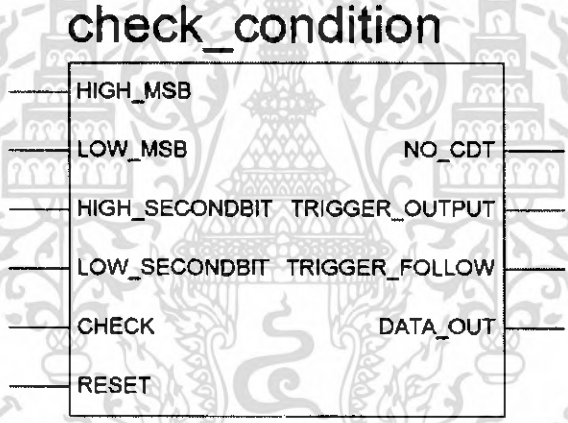


รูปที่ 4.23 แสดงผลจำลองการทำงานของวงจร LOW

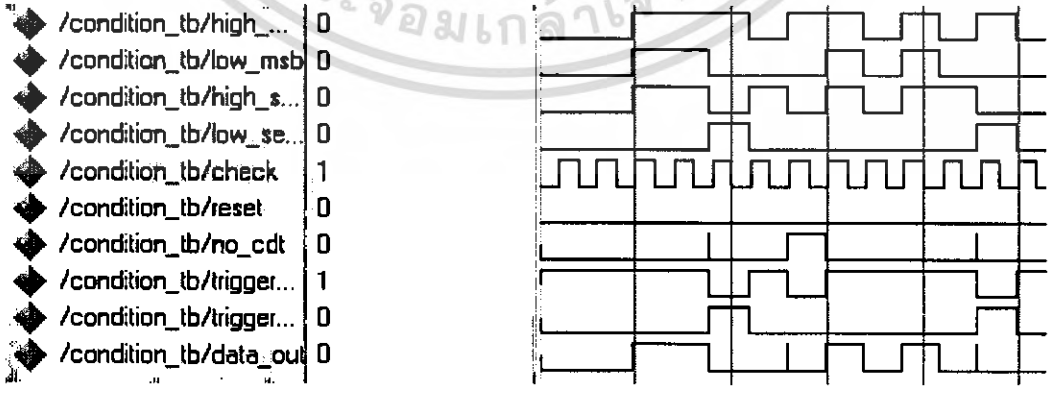
จากผลการทดลองจะเห็นได้ว่า จะเริ่มทำการเลื่อนบิทในตำแหน่ง SET_VALUE = '1' ค่าของ OUTPUT ถึงจะเริ่มเก็บค่ามาแทนค่าที่ตั้งไว้ ('00000000000000000000000000000000')

- เมื่อเจอ SHIFT_ALL จะให้ OUTPUT เป็นการเลื่อนบิททุกบิทไปทางขวา 1 บิท
- เมื่อเจอ SHIFT_MOST จะให้ OUTPUT เป็นการเลื่อนบิททุกบิทยกเว้นบิท MSB ไปทางขวา 1 บิท ทั้งหมด

4.2.2.9 CHECK_CONDITION



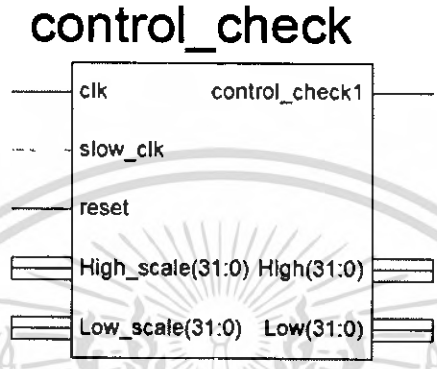
รูปที่ 4.24 บล็อกไดอะแกรมแสดงวงจรของ CHECK_CONDITION



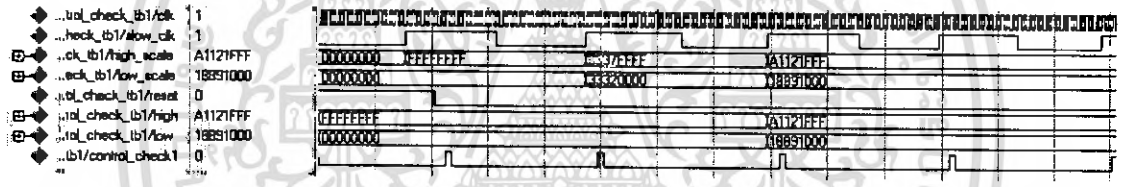
รูปที่ 4.25 แสดงผลจำลองการทำงานของวงจร CHECK_CONDITION

จากผลการทดลองจะเห็นได้ว่าเมื่อ LOW_MSB และ HIGH_MSB มีค่าเท่ากัน TRIGGER_OUTPUT จะเป็น 1 แต่ถ้า LOW_MSB = '0' , HIGH_MSB = '1', HIGH_SECOND_MSB = '0' และ LOW_SECOD_MSB = '1'TRIGGER_FOLLOW จะเป็น 1และหากไม่ตรงเงื่อนไขใดๆ แล้ว NO_CDT = '1'

4.2.2.10 CONTROL_CHECK



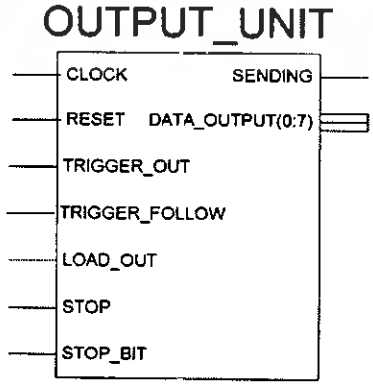
รูปที่ 4.26 บล็อกไดอะแกรมแสดงวงจรของ CONTROL_CHECK



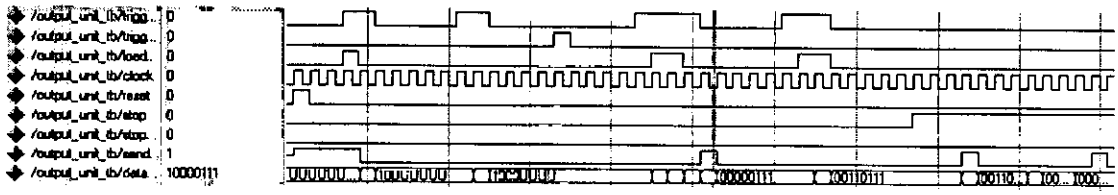
รูปที่ 4.27 แสดงผลจำลองการทำงานของวงจร CONTROL_CHECK

จากผลการทดลองจะเห็นได้ว่าจะสร้าง CONTROL_CHECK ที่นำไปใช้ในการควบคุมการทำงานของ SHIFT_CONDITION และทำหน้าที่กำหนดจังหวะของ HIGH_SCALE และ LOW_SCALE ให้ตรงกับ CLOCK

4.2.2.11 OUTPUT_UNIT



รูปที่ 4.28 บล็อกไดอะแกรมแสดงวงจรของ OUTPUT_UNIT

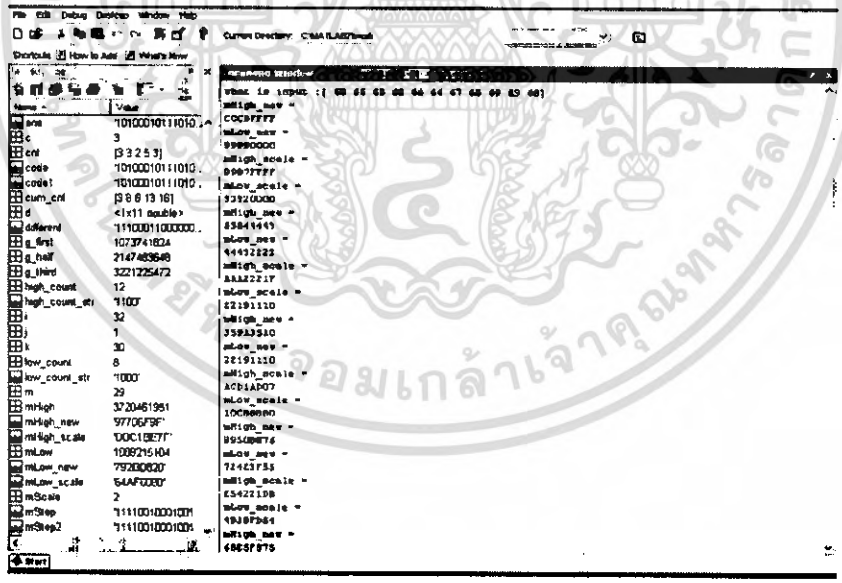


รูปที่ 4.29 แสดงผลจำลองการทำงานของวงจร OUTPUT_UNIT

จากผลการทดลองจะเห็นได้ว่าช่วงที่มีสัญญาณ TRIGGER_OUT ข้อมูลจะถูกดึงจาก LOAD_OUT มาเก็บไว้ที่ DATA_OUT และหากมีการเกิดสัญญาณ TRIGGER_FOLLOW แล้วหลังจากนั้นจะเกิด TRIGGER_OUT จะส่ง LOAD_OUT ออกไป DATA_OUT 1 บิท แล้วตามด้วยสัญญาณนี้ที่ถูกคอมพลีเมนท์ไปเป็นจำนวนเท่ากับจำนวนสัญญาณ TRIGGER_OUT ที่นับไว้ จากนั้นจึงส่ง LOAD_OUT ที่ถูกแทรกตามไป

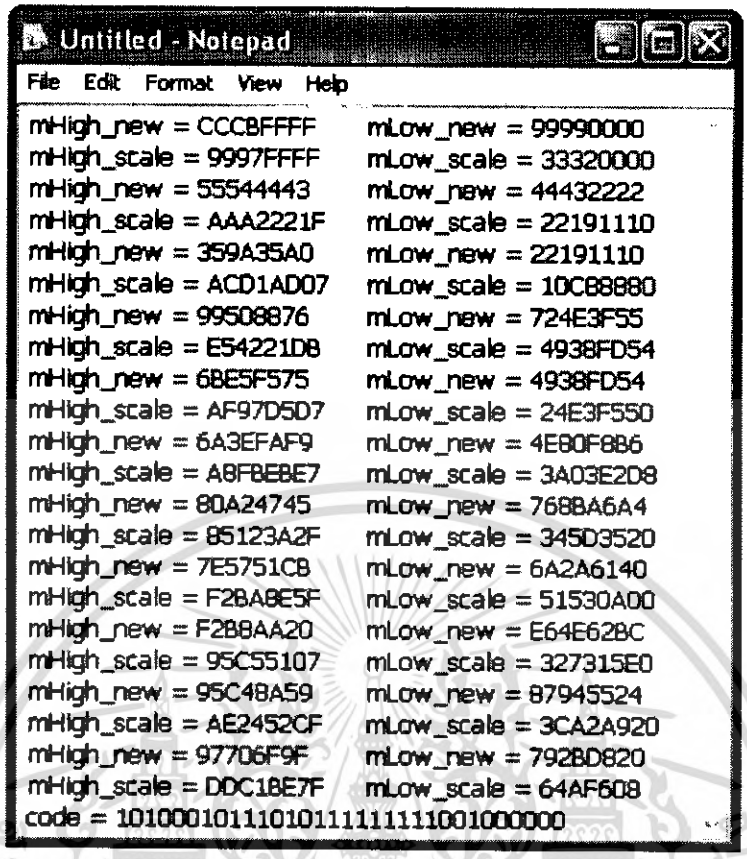
หากข้อมูลที่เก็บใน DATA_OUT ครบ 8 บิทแล้วจะมีสัญญาณ SENDING = '1' และเมื่อจบข้อมูลทั้งหมดจะทำการเติมบิทสิ้นสุดลงไปโดยเติม STOP_BIT ที่ STOP = '1' 1 บิทและบิทคอมพลีเมนท์ของ STOP_BIT จำนวนเท่ากับจำนวนที่นับ TRIGGER_FOLLOW + 1 และสิ้นสุดด้วยการเติมบิท '0' ไปจนครบ 8 บิท

สุดท้ายจะทำการตรวจสอบผลจากโปรแกรม VHDL เทียบกับโปรแกรมแมทแลบ ว่าได้ตรงกันหรือไม่ โดยสมมติว่าเราใช้ให้ อินพุท เป็น [68 66 65 68 65 66 67 68 69 69 68]

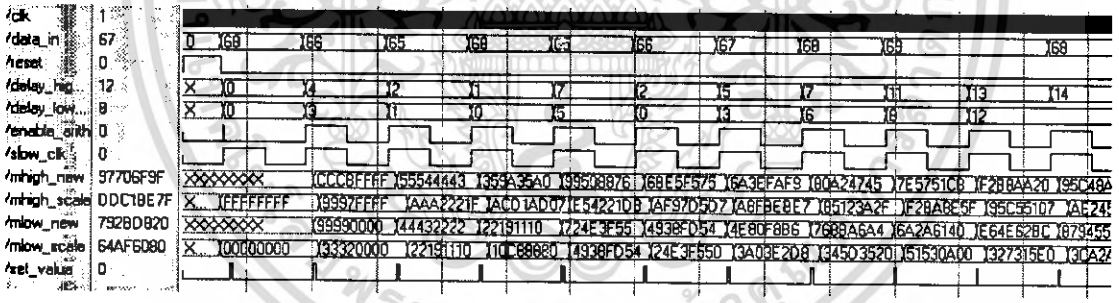


รูปที่ 4.30 เอาท์พุทที่จะนำไปใช้ในการตรวจสอบค่าในภาษา VHDL

แต่เนื่องจากผลที่แสดงในแมทแลบ นั้นไม่สามารถแสดงได้ครบทุกตัวในหน้าจอเดียว จึงได้ทำการนำคำตอบมาจัดรูปใหม่ให้อยู่ในหน้าเดียวโดยใช้โปรแกรมโน๊ตแพด(notepad)

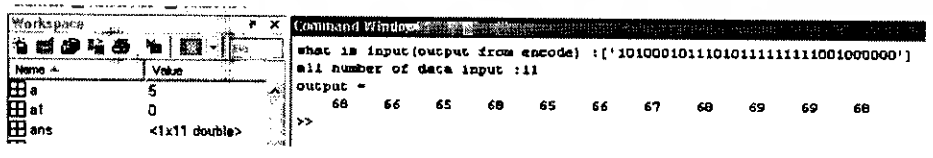


รูปที่ 4.31 เอาท์พุทที่ได้จากส่วนการเข้ารหัสในแมทแลบทั้งหมด



รูปที่ 4.32 เอาท์พุทที่ได้จากการเข้ารหัสในภาษา VHDL

ซึ่งเมื่อนำเอาท์พุททั้งสองมาเปรียบเทียบกันก็จะได้ผลลัพธ์ที่ตรงกัน



รูปที่ 4.33 เอาท์พุทที่ได้จากการนำโค้ดที่ได้จากการเข้ารหัสมาใส่ในส่วนถอดรหัส

บทที่ 5
บทวิจารณ์และบทสรุป

จากการออกแบบตัวบีบอัดข้อมูลโดยใช้วิธีการเข้ารหัสเลขคณิต โดยมีการปรับให้เหมาะสมกับการประยุกต์ใช้กับฮาร์ดแวร์โดยใช้การเข้ารหัสเลขคณิตแบบจำนวนเต็ม ทำการจำลองผลโดยเขียนโปรแกรมภาษาแมทแลบ ซึ่งแสดงการบีบอัดข้อมูลที่เป็นตัวอักษร โปรแกรมภาษาแมทแลบที่เขียนขึ้นจะบีบอัดข้อมูลนั้นให้มีขนาดเล็กลงและเข้ารหัสที่ทำให้ไม่เหมือนข้อมูลชุดเดิม เมื่อนำผลที่เข้ารหัสแล้วมาทำการถอดรหัสจากโปรแกรมนี้ก็จะได้ข้อมูลเดิมที่ถูกต้องกลับมา

เราจะใช้แนวคิดจากการเขียนโปรแกรมภาษาแมทแลบนี้ และแนวคิดในการประยุกต์ใช้ทำเป็นฮาร์ดแวร์ไปเป็นแนวทางในการเขียนภาษาวีเอชดีแอลเพื่อทำการอิมพลีเมนท์ลงบนบอร์ดเอฟพีจีเอซึ่งทำให้มีผลลัพธ์บางอย่างที่ไม่สามารถที่จะหาค่าได้อย่างแม่นยำ ตัวอย่างเช่นการหารที่เราไม่สามารถที่จะเก็บทศนิยมได้จนตำแหน่งสุดท้าย เราจึงใช้เพียงแค่จากการประมาณ ซึ่งตรงจุดนี้ทำให้เราต้องคิดวิธีการใหม่ที่จะนำมาใช้โดยยังให้ผลลัพธ์ที่ถูกต้องคงเดิม

ดังนั้นในโปรแกรมภาษาวีเอชดีแอลที่ประยุกต์แล้วนั้นจึงยังคงมีข้อจำกัดบางอย่างอยู่ดังนี้

1. จำกัดว่าจะใส่ข้อมูลได้เพียงแค่ 5 ตัวเท่านั้น ซึ่งได้แก่ A, B, C, D ,E ซึ่งทำให้ค่า SUMCNT เริ่มต้น เป็น 5 เสมอ ทำให้สามารถที่จะใช้ RECIPROCAL เพียง 16 บิทเพื่อให้พอเพียงแก่ผลหารของ SUMCNT ตั้งแต่ 5 ไปจนถึง 256

2. จำกัดว่าจะรับข้อมูลได้สูงสุด 251 ตัว เนื่องจากว่า RECIPROCAL 16 บิทนั้น สามารถใช้ได้กับ SUMCNT สูงสุดได้เพียง 256 จากค่าเริ่มต้นซึ่งคือ 5 ดังนั้นจึงรับข้อมูลได้ (256-5 = 251)

นอกจากข้อจำกัดตรงนี้แล้วยังมีปัญหาที่บล็อกไดอะแกรมของ OUTPUT_UNIT ซึ่งมีความซับซ้อนมากจึงยังไม่สามารถเขียนให้ครอบคลุมผลลัพธ์ที่ถูกต้องได้ทุกเงื่อนไข เราจึงได้ตรวจสอบผลลัพธ์ได้ถึงบล็อกไดอะแกรมของ SHIFT_CONDITION ซึ่งสามารถตรวจสอบกับผลของโปรแกรมแมทแลบแล้วว่ามีความถูกต้อง จึงนำค่าที่ได้จากบล็อกนี้ไปคำนวณในแมทแลบต่อเพื่อหาผลลัพธ์ที่ถูกต้องแทนได้

แนวทางในการพัฒนาต่อ

1. เราสามารถที่จะเพิ่มจำนวนบิทของ RECIPROCAL ให้มากขึ้นได้ก็จะสามารถที่จะทำให้ตัวอักษรที่ใช้ส่งและจำนวนข้อมูลที่ส่งเพิ่มมากขึ้น แต่ก็ต้องแลกกับการคำนวณค่าที่เก็บใน lookup table ที่ยุ่งยากซับซ้อนกว่าเดิม

2. การใช้ระบบเลขฐานสองในการส่งข้อมูล จะทำให้การคำนวณหา mHigh และ mLow ง่ายขึ้นกว่าเดิมมาก

3. หากเพิ่มส่วนการคำนวณหาชุดข้อมูล context ได้ ก็จะสามารถให้ค่าความน่าจะเป็นของชุดข้อมูลนั้นคงที่ และใกล้เคียงความจริงมากขึ้น ก็จะสามารถบีบอัดข้อมูลได้มากยิ่งขึ้น

หนังสืออ้างอิง

[1] Khalid Sayood , "Introduction to Data compression (third edition)" university of Nabraska,Elsevier Inc.,2006

[2] Khalid sayood , "Lossless Compression Handbook"university of Nabraska ,Elsevier Science(USA), 2003

[3]Mark Nelson ,Jean Loup-Gailly , "The Data Compression Book"The second edition,M&T Books,1996

[4]Stephen J. Chapman , "MATLAB Programming for engineers"second edition,BROOKS/COLES ,2002

[5] ยุทธนา ลีลาวัฒนกุล , "เริ่มต้นการเขียนโปรแกรมภาษา C++" ,หจก.ไทยเจริญการพิมพ์ ,มิถุนายน 2547

[6] ชำนาญ ปัญญาใส ,วัชรากร หนูทอง , "ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล" ,กรุงเทพฯ:ซี-เอ็ดยูเคชั่น ,2547