การสร้างตัวเข้ารหัสเสียงพูดแบบ ADPCM ด้วย FPGA

FPGA IMPLEMENTATION OF ADPCM SPEECH CODEC

โดย

นางสาวกนกพร จันทร์แดง

นางสาวกาญจนากร ปันวารี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ผ่านการตรวจชั้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

การสร้างตัวเข้ารหัสเสียงพูดแบบ ADPCM ด้วย FPGA

FPGA IMPLEMENTATION OF ADPCM SPEECH CODEC

โดย

นางสาวกนกพร จันทร์แดง 45010002

นางสาวกาญจนากร ปันวารี 45010039

อาจารย์ที่ปรึกษา

รศ.ดร. ยุทธพงษ์ รังสรรค์เสรี

รศ.ดร. ปัญญา ฐิติมัชฌิมา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การสร้างตัวเข้ารหัสเสียงพูดแบบ ADPCM ด้วย FPGA**

FPGA IMPLEMENTATION OF ADPCM SPEECH CODEC

ผู้จัดทำ

1. นางสาวกนกพร จันทร์แดง 45010002

2. นางสาวกาญจนากร ปันวารี 45010039

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ยุทธพงษ์ รังสรรค์เสรี)

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ปัญญา ฐิติมัชฌิมา)

การสร้างตัวเข้ารหัสเสียงพูดแบบ ADPCM ด้วย FPGA

FPGA IMPLEMENTATION OF ADPCM SPEECH CODEC

โดย นางสาวกนกพร จันทร์แดง 45010002
นางสาวกาญจนากร ปันวารี 45010039

อาจารย์ที่ปรึกษา รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี
รศ.ดร.ปัญญา ฐิติมัชฌิมา

บทคัดย่อ

โครงงานนี้มีวัตถุประสงค์เพื่อทำการศึกษาและสร้างตัวเข้ารหัสสัญญาณเสียง โดยวิธี Adaptive Differential Pulse Code Modulation (ADPCM) ซึ่งใช้การจัดระดับกับค่าผลต่างระหว่างแซมเปิ้ลที่อยู่ติดกัน ทำให้สามารถเข้ารหัสได้โดยใช้จำนวนบิตที่ลดลง ในส่วนของการออกแบบจะใช้วิธีการออกแบบด้วยภาษา VHDL แล้วจำลองการทำงานรวมทั้งสังเคราะห์วงจรลงบนอุปกรณ์ FPGA เพื่อใช้ทดสอบการทำงาน และดูผลลัพธ์

ABSTRACT

The objective of this project is to study and construct audio coder by Adaptive Differential Pulse Code Modulation (ADPCM). In ADPCM, the difference between the values of the sample is coded with an adaptive quantizes. Consequent, we can decrease the word length of channel sample. The hardware design is used VHDL. After that the circuit design will be simulated, synthesized and programmed on FPGA for testing all results .

สารบัญ

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

บทที่ 1
บทนำ

การส่งข้อมูลข่าวสารในปัจจุบันนั้นได้มีการพัฒนาไปอย่างรวดเร็ว มีการนำเอาเทคโนโลยีต่างๆ มาใช้ เพื่อเพิ่มประสิทธิภาพให้กับการส่งข้อมูล ซึ่งปัจจุบันนิยมใช้การส่งสัญญาณแบบดิจิตอล เพราะมีข้อดีในด้านความทนทานต่อสัญญาณรบกวนจึงสามารถแก้ปัญหาการพร่าเพี้ยน (Distortion) และลดทอนได้ นอกจากนี้สัญญาณดิจิตอลยังเหมาะสมกับการบริการในรูปแบบต่างๆทั้งเสียง ข้อมูลและภาพอีกด้วย

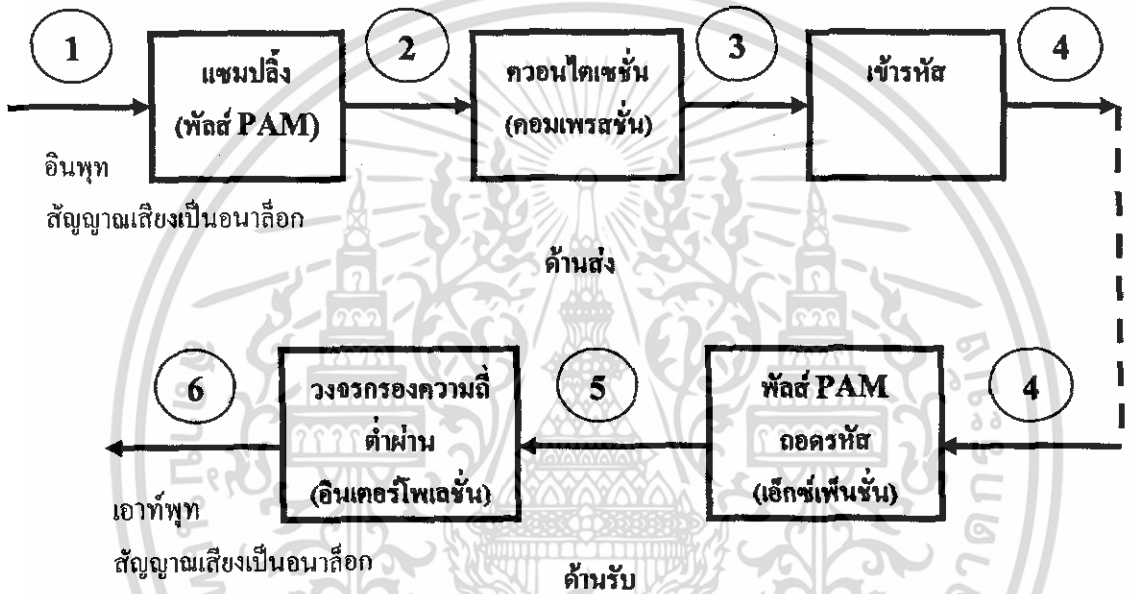
วิธีที่ใช้กันอย่างแพร่หลายก็คือ พัลส์โค้ดมอดูเลชั่น (Pulse Code Modulation: PCM) ซึ่งถูกนำไปใช้อย่างกว้างขวางโดยเฉพาะกับสัญญาณเสียง แต่ยังมีการเข้ารหัสอีกแบบหนึ่งที่ทำการส่งด้วยรหัสที่น้อยกว่า 8 บิตคือการเข้ารหัสสัญญาณแบบอแด็พทีฟว์ดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น (Adaptive Differential Pulse Code Modulation: ADPCM) วิธีนี้จะใช้ค่าผลต่างระหว่างแซมเปิ้ลที่อยู่ข้างเคียงกันซึ่งมีค่าน้อย อีกทั้งยังเพิ่มคุณสมบัติในการปรับขั้นระดับของการควอนไตซ์ตามขนาดของสัญญาณผลต่างเข้าไปด้วยและใช้อัตราการส่งสัญญาณที่ต่ำลงได้

CCITT ได้กำหนดมาตรฐาน G.721 ADPCM 32 kbps ซึ่งสัญญาณเสียงที่กู้กลับมาได้นั้นมีคุณภาพเกือบเทียบเท่ากับ PCM 64 kbps จากนั้นได้มีการกำหนดมาตรฐาน G.726 และ G.727 ขึ้น โดย G.726 ได้อธิบายถึงอัลกอริทึมของ ADPCM ที่แปลงสัญญาณ PCM 64 kbps เป็น 40, 32, 24 หรือ 16 kbps

บทที่ 2
ทฤษฎีหรือหลักการ

2.1 การเข้ารหัสของสัญญาณเสียงแบบพัลส์โค้ดมอดูเลชั่น (Pulse Code Modulation: PCM)

วิธีการเข้ารหัสสัญญาณเสียงในระบบสื่อสารแบบดิจิตอลที่ได้รับการพัฒนาและนำมาใช้อย่างกว้างขวางก็คือ พัลส์โค้ดมอดูเลชั่น (Pulse Code Modulation: PCM) ซึ่งเป็นวิธีการหนึ่งในการเปลี่ยนสัญญาณเสียงจากอนาล็อกให้เป็นดิจิตอล โดยจะประกอบด้วยขั้นตอนต่างๆ ดังนี้

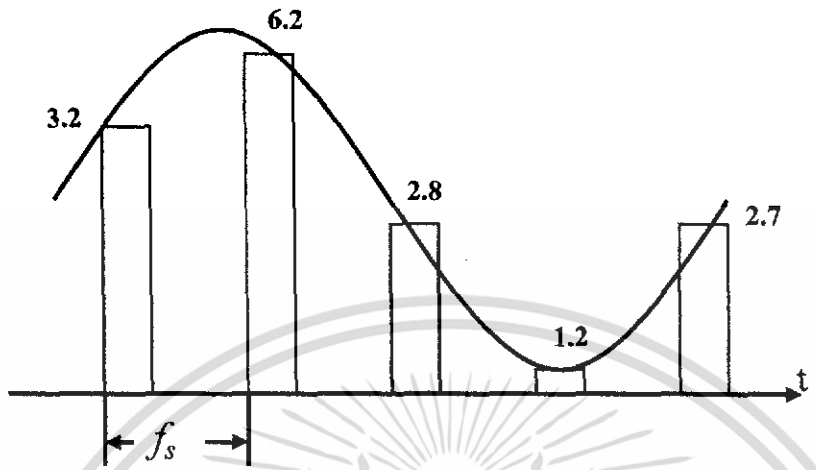


รูปที่ 2.1 กระบวนการเข้ารหัสและถอดรหัสของระบบพีซีเอ็ม

จากรูปที่ 2.1 แสดงขั้นตอนการประมวลสัญญาณ เพื่อให้ได้สัญญาณพีซีเอ็ม กล่าวอย่างกว้างๆก็คือ ระบบพีซีเอ็ม เป็นการจัดการกับสัญญาณพัลส์แอมปลิจูดมอดูเลชั่น (Pulse Amplitude Modulation : PAM) โดยการนำสัญญาณพีเอเอ็มไปทำการเข้ารหัส (Coding) เป็นสัญญาณดิจิตอล แล้วจึงนำสัญญาณดิจิตอลที่ได้นั้นส่งผ่านระบบต่อไปและทางด้านรับก็จะทำการถอดรหัส (Decoding) เป็นสัญญาณพีเอเอ็มแล้วนำไปดีมอดูเลตเพื่อให้ได้สัญญาณเดิมกลับคืนมา

ในการส่งโดยระบบพีซีเอ็มนั้นต้องส่งด้วยจำนวนบิต 8 บิต แต่ยังมีการเข้ารหัสอีกแบบหนึ่งที่ทำการส่งด้วยรหัสน้อยกว่า 8 บิตคือ การเข้ารหัสเสียงแบบอแด็พทิฟว์ดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น ซึ่งมีข้อได้เปรียบกว่าระบบพีซีเอ็มหลายประการ ดังจะกล่าวต่อไป

2.2 การชักตัวอย่าง (Sampling)

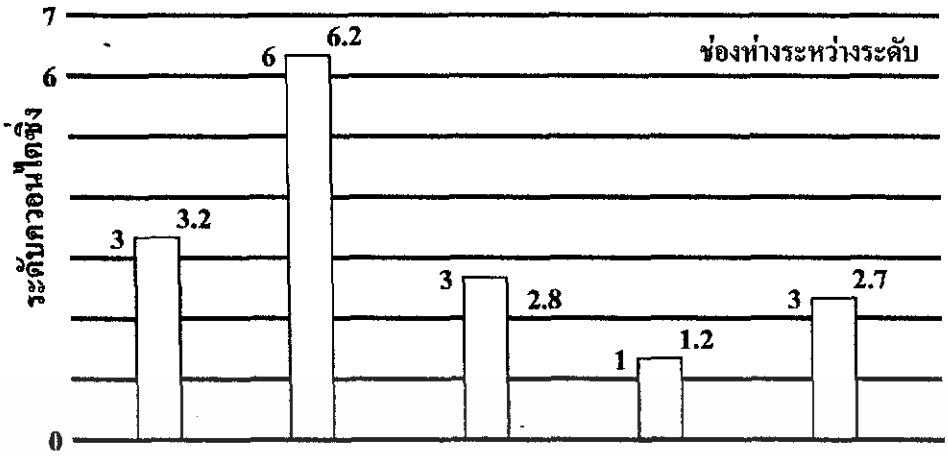


รูปที่ 2.2 การชักตัวอย่างสัญญาณเสียง

การชักตัวอย่าง (Sampling) คือการทำสัญญาณซึ่งมีค่าต่อเนื่องให้เป็นแบบดีสครีตในช่วงเวลาที่เท่าๆกัน จากทฤษฎีการชักตัวอย่าง ถ้าเก็บแซมเปิ้ล (Sample) ด้วยอัตรา 2 เท่าหรือมากกว่าความถี่สูงสุดของสัญญาณอนาล็อก แล้วจะทำให้สัญญาณเดิมกลับคืนมาได้ เนื่องจากสัญญาณเสียงที่ใช้ในระบบโทรศัพท์นั้นถูกจำกัดให้มีความถี่ระหว่าง 300-3400 Hz ดังนั้นอัตราการชักตัวอย่างต่ำสุดจะต้องเท่ากับ 6.8 kHz สำหรับในทางปฏิบัติจะใช้ 8 kHz คือการชักค่าตัวอย่างทุกๆ 125 us

2.3 การจัดระดับ (Quantizing)

ขบวนพัลส์พีเอเอ็มที่ผ่านการชักค่าตัวอย่างมาแล้ว ยังถือว่าเป็นชนิดอนาล็อกอยู่คือจะมีแอมปลิจูดที่เปลี่ยนแปลงอย่างต่อเนื่องไปกับเวลาที่เป็นช่วงๆ การจัดระดับคือกระบวนการที่เปลี่ยนแปลงแอมปลิจูดของพัลส์พีเอเอ็มเหล่านั้นให้เป็นค่าตัวเลขแบบดีสครีต ตามที่แสดงไว้ในรูปที่ 2.3



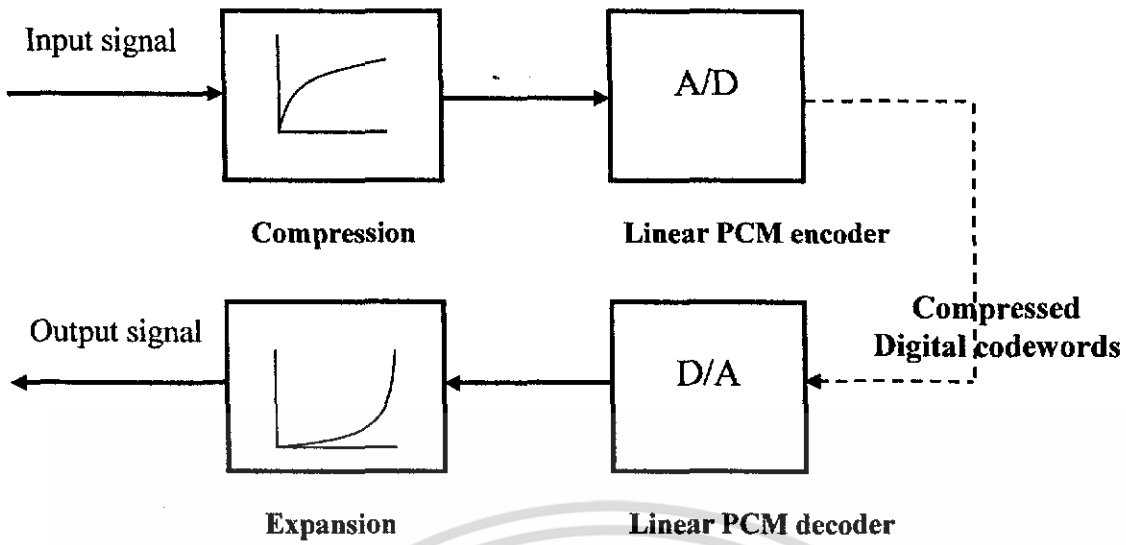
รูปที่ 2.3 การจัดระดับสัญญาณพีเอเอ็ม

จากรูปที่ 2.3 แอมปลิจูดของการชักตัวอย่างทุกตัวของพีเอเอ็มจะถูกจัดให้เป็นระดับซึ่งเรียกว่า ระดับการควอนไตซ์ (quantize level) โดยมีระยะห่างระหว่างระดับข้างเคียง เรียกว่า ควอนไตซ์ซิ่งอินเทอร์วัล (quantizing interval) เท่ากัน กรณีนี้เรียกว่าเป็นการจัดแบบยูนิฟอร์ม (uniform quantizing) ขนาดของแซมเปิ้ลทุกตัวจะแสดงด้วยค่าระดับการควอนไตซ์ที่ใกล้เคียงที่สุด เช่น ขนาดของแซมเปิ้ลที่ $t = t_1$ คือ 3.2 จะจัดให้เป็นระดับ 3 หรือค่าแซมเปิ้ลที่ $t = t_2$ มีขนาด 6.2 จะจัดให้เป็น 6 เป็นต้น จะเห็นได้ว่าสัญญาณพีเอเอ็มที่ถูกจัดระดับแล้วนี้จะเป็นเพียงค่าโดยประมาณของสัญญาณอนาล็อกเท่านั้น ดังนั้นส่วนเกินและส่วนขาดจากการจัดระดับจึงเป็นค่าผิดพลาดระหว่างสัญญาณเดิมและค่าที่จัดระดับ ซึ่งค่าผิดพลาดนี้เรียกว่า ควอนไตซ์ซิ่งนอยส์ (quantizing noise) หรือค่าความผิดเพี้ยนจากการควอนไตซ์ (quantizing distortion)

จากหลักการที่กล่าวมานี้ ในทางปฏิบัติจะไม่สามารถหลีกเลี่ยงควอนไตซ์ซิ่งนอยส์ได้ แต่เพื่อรักษาคุณภาพเสียงในการสนทนาให้ดีจึงจำเป็นต้องทำควอนไตซ์ซิ่งนอยส์นี้ให้ลดลง ในเบื้องต้นคือลดควอนไตซ์ซิ่งอินเทอร์วัลให้แคบลง ก็สามารถลดควอนไตซ์ซิ่งนอยส์ได้ในระดับหนึ่ง

2.4 คอมแพนดิ้ง (Companding)

ตามที่ได้กล่าวแล้วว่าเราไม่สามารถหลีกเลี่ยงควอนไตซ์ซิ่งนอยส์ได้ แต่เราสามารถทำให้มันลดลงได้โดยการเพิ่มควอนไตซ์ซิ่งอินเทอร์วัลหรือการเพิ่มจำนวนระดับนั่นเอง แต่เมื่อจำนวนระดับเพิ่มขึ้นแล้วจำนวนบิตที่ใช้ก็เพิ่มขึ้นด้วย จึงจำเป็นต้องใช้ความเร็วในการส่งสัญญาณดิจิตอลสูงขึ้น ตามปกติควอนไตซ์ซิ่งนอยส์จะเกิดขึ้นสม่ำเสมอในทุกอินเทอร์วัลโดยไม่เกี่ยวข้องกับแอมปลิจูดของสัญญาณเดิม ในกรณีที่สัญญาณมีระดับสูง คุณภาพของการเข้ารหัสของสัญญาณเสียงจะดีกว่าของสัญญาณซึ่งมีระดับต่ำ ดังนั้นจึงจำเป็นต้องพิจารณาควอนไตซ์ซิ่งนอยส์ในบริเวณที่สัญญาณมีระดับต่ำ การจัดระดับแบบนอนยูนิฟอร์ม (Non-uniform quantizing) คือบริเวณที่สัญญาณมีแอมปลิจูดต่ำจะควอนไตซ์ซิ่งอินเทอร์วัลแคบๆ และในทางตรงกันข้ามบริเวณที่สัญญาณมีแอมปลิจูดสูงจะใช้ควอนไตซ์ซิ่งอินเทอร์วัลกว้างๆ ซึ่งการทำให้เป็นแบบนอนยูนิฟอร์มนั้นจะใช้หลักการคอมแพนดิ้งเข้าช่วย

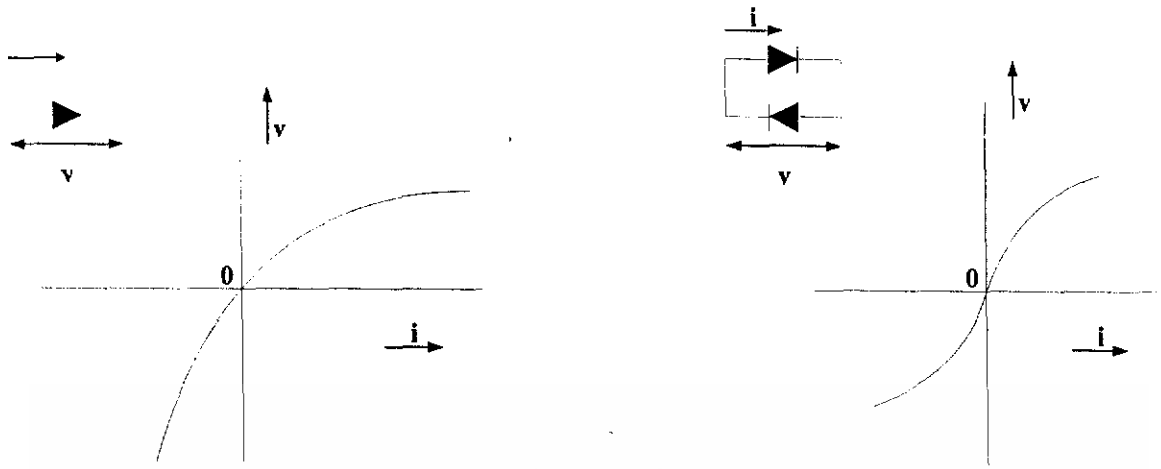


รูปที่ 2.4 การแทรกขั้นตอนของการอัดสัญญาณและการยืดสัญญาณลงในระบบพีซีเอ็ม

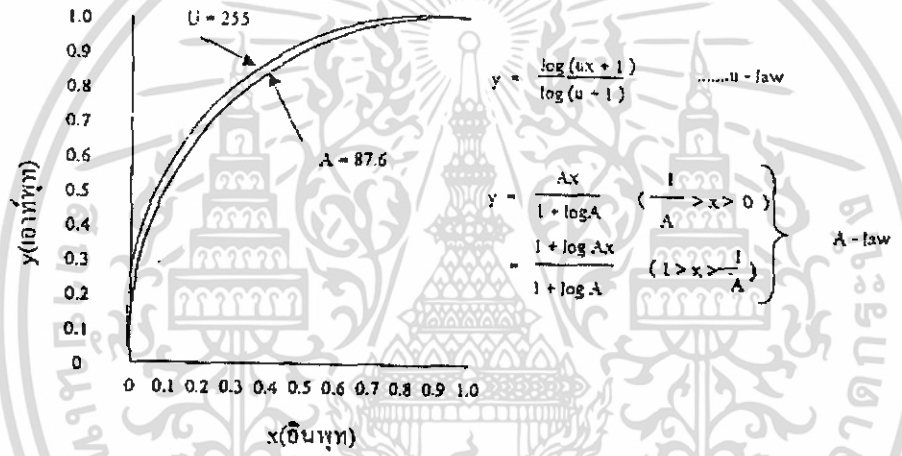
การทำคอมแพนดิ้งเมื่อนำมาใช้ในระบบพีซีเอ็ม จะทำให้ประสิทธิภาพของการใช้จำนวนบิตในการเข้ารหัสสูงขึ้น กรณีที่จัดระดับแบบยูนิฟอร์มนั้นจะใช้ประมาณ 2,000 ระดับ จึงจะสามารถรักษาระดับคุณภาพของเสียงให้ดีและในการเข้ารหัสต้องใช้ถึง 11 บิต ต่อ 1 แซมเปิ้ล แต่ถ้าใช้แบบนอนยูนิฟอร์มแล้วจะใช้เพียง 7 บิต ซึ่งมีระดับเพียง 128 ระดับ ก็เพียงพอที่จะทำให้อัตราส่วนของสัญญาณต่อควอนไตซ์ซึ่งนอยส์ใกล้เคียงกับการจัดระดับแบบยูนิฟอร์ม CCITT กำหนดให้ใช้ 8 บิตต่อ 1 แซมเปิ้ล และระดับการควอนไตซ์เท่ากับ 256 ระดับ จึงจะเป็นการรับรองว่าเสียงพูดจะมีคุณภาพดี

ลักษณะของการคอมเพรสเซอร์นั้นจะเป็นแบบลอการิทึม รูปแบบโดยทั่วไปจะใช้คุณสมบัติของ V-I ของไดโอดตามรูปที่ 2.5 กรณีที่ใช้คอมเพรสเซอร์จะมีกระแส I เป็นอินพุท โวลเตจ V เป็นเอาท์พุท สำหรับกรณีที่ใช้เป็นเอกซ์แพนเดอร์จะมีโวลเตจเป็นอินพุท และกระแสเป็นเอาท์พุท

คุณลักษณะของคอมเพรสเซอร์ที่ใช้สำหรับประกอบการเข้ารหัสสัญญาณเสียงในปัจจุบัน คือ μ-law ซึ่งใช้ในทวีปอเมริกาเหนือและญี่ปุ่นเป็นหลัก โดยค่า μ ที่ใช้คือ 255 สำหรับในทวีปยุโรปและประเทศไทยใช้ A-law ซึ่งโดยหลักการแล้วจะเหมือนกับกฎ μ-law แต่จะแตกต่างกันในรายละเอียด โดยค่า A ที่ใช้คือ 87.6 คุณลักษณะทั้งสองนี้แสดงไว้ในรูปที่ 2.6 (เฉพาะกรณี μ=255 และ A=87.6)



รูปที่ 2.5 คุณลักษณะคอมเพรสชั่นของไดโอด



รูปที่ 2.6 แบบอย่างคุณลักษณะของการคอมเพรสชั่น

กรณีที่ μ=100 จะใช้คุณลักษณะของคอมเพรสเซอร์ตามรูปที่ 2.5 แต่กรณีที่ μ=255 และ A=87.6 จะใช้คุณลักษณะของคอมเพรสเซอร์ที่มีคุณลักษณะเป็นเส้นตรง โดยแยกเป็นส่วนๆเรียกว่า เซกเมนต์ (Segment)

2.5 การเข้ารหัสแบบดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น (Differential pulse code modulation: DPCM)

สัญญาณประเภทเสียงที่ถูกสุ่มตัวอย่าง (Sampling) แล้วจะมีความสัมพันธ์กันอย่างมากและจากข้อเท็จจริงนี้เองทำให้เราสามารถทำนายหรือคาดคะเน (Predict) แต่ละตัวอย่าง (Sample) โดยใช้ข้อมูลพื้นฐานจากตัวอย่างที่ผ่านมาแล้วเพียงแต่เข้าโค้ดที่แตกต่างระหว่างค่าที่ได้ทำนายไว้กับค่าตัวอย่าง แล้วส่งไปซึ่งเรียกวิธีการนี้ว่า "Differential Encoding" ซึ่งจะช่วยลดปริมาณของข้อมูลที่จำเป็นต้องใช้

อัลกอริทึมเบื้องต้น (The Basic Algorithm)

ในการทำนายอย่างง่ายเราจะทำนายอย่างง่าย เราจะทำนายว่า ตัวอย่างถัดไปจะเท่ากับตัวอย่างปัจจุบัน

$p_n = X_{n-1}$ เมื่อ p_n คือ ค่าที่ทำนายตัวอย่างที่ n

X_{n-1} คือ ตัวอย่างที่ n-1

แม้ว่าในการเข้าโค้ดของผลต่างจะใช้จำนวนบิตน้อยกว่าแบบเดิม (PCM) แต่กล่าวได้ว่ามันสามารถครอบคลุมได้ในเรื่องของการถอดรหัสที่ยอมรับได้ของลำดับสัญญาณจากค่าแตกต่างที่ถูกควอนไตซ์ (Quantize) แล้ว

หากสนใจการบีบอัดข้อมูลแบบไม่มีการสูญเสียข้อมูล (Lossless Compression) จะพบว่าถ้าทำการเข้าโค้ดความแตกต่างระหว่างตัวอย่าง จะสามารถคลอบคลุมลำดับเดิมได้อย่างไม่สูญเสีย ตัวอย่างเช่นลำดับต่อไปนี้

6.2 9.7 13.2 5.9 8 7.4 4.2 1.8

เราจะได้ลำดับของผลต่างระหว่างตัวอย่างดังนี้

6.2 3.5 3.5 -7.3 2.1 -0.6 -3.2 -2.4

จากนั้นลองมาดูว่าจะเกิดอะไรขึ้น ถ้าผลต่างถูกเข้าโค้ดแบบสูญเสียข้อมูล (Lossy) สมมุติว่าเรามี ตัว ควอนไตซ์เจ็ดระดับ ด้วยค่าเอาท์พุท -6,-4,-2, 0,2,4,6 จะได้ลำดับที่ถูกควอนไตซ์ เป็น

6 4 4 -6 2 0 -4 -2

เมื่อเราทำตามกระบวนการเดิมสำหรับสร้างสัญญาณคืน เราจะได้ลำดับดังนี้

6 10 14 8 10 10 6 4

ค่าผิดพลาดของสัญญาณที่สร้างคืนได้เมื่อเทียบกับสัญญาณเดิม มีค่าดังนี้

0.2 -0.3 -0.8 -2.1 -2 -2.6 -1.8 -2.2

สังเกตว่า ค่าผิดพลาดจะมีค่าน้อยสำหรับลำดับต้นๆ (0.2, 0.3) แต่จะเพิ่มขึ้นเมื่อลำดับเพิ่มมากขึ้น (2.6, 1.8, 2.2) ในการพิสูจน์นี้สามารถพิสูจน์ได้ด้วยสมการพิจารณาลำดับตัวอย่าง $\{X_n\}$ ลำดับผลต่าง $\{d_n\}$ ซึ่งเกิดจาก $X_n - X_{n-1}$ ลำดับผลต่างนี้ถูกควอนไตซ์ได้ $\{\hat{d}_n\}$

$$\{\hat{d}_n\} = Q[d_n] = d_n + q_n \tag{2.1}$$

โดย q_n คือ ค่าความผิดพลาดที่เกิดจากการควอนไตซ์ (Quantization Error) และที่เครื่องรับ $\{\hat{X}_n\}$ ถูกสะสมโดยการบวกค่า $\hat{d}$ กับค่าสะสมก่อนหน้า $\hat{X}_{n-1}$

$$\hat{X}_n = \hat{X}_{n-1} + \hat{d}_n \tag{2.2}$$

สมมุติว่าสัญญาณตัวอย่างเริ่มต้นด้วย X_0 $(\hat{X}_0 = X_0)$ จะได้

$$d_1 = X_1 - X_0 \tag{2.3}$$

$$\hat{d}_1 = Q[d_1] = d_1 + q_1 \quad \text{จากการเข้ารหัส} \tag{2.4}$$

$$\hat{X}_1 = \hat{X}_0 + \hat{d}_1 = X_0 + d_1 + q_1 = X_1 + q_1 \quad \text{จากการถอดรหัส} \tag{2.5}$$

$$d_2 = X_2 - X_1 \tag{2.6}$$

$$\hat{d}_2 = Q[d_2] = d_2 + q_2 \tag{2.7}$$

$$\hat{X}_2 = \hat{X}_1 + \hat{d}_2 = X_1 + q_1 + d_2 + q_2 \quad (2.8)$$

$$= X_2 + q_1 + q_2 \quad (2.9)$$

เมื่อกระบวนการนี้มีต่อเรื่อยๆ ไปจนถึงลำดับที่ n จะได้ว่า

$$\hat{X}_n = X_n + \sum_{k=1}^{n} q_k \quad \text{ซึ่ง } \sum_{k=1}^{n} q_k \text{ คือค่าของ Error} \quad (2.10)$$

จะพบว่าเมื่อลำดับมากขึ้นค่าความผิดพลาดจะมากขึ้น ก็เนื่องจาก $\sum_{k=1}^{n} q_k$ จะมีค่าสะสมมากขึ้นเรื่อยๆ ปัญหานี้จะแก้ไขโดยการแทน $d_n = X_n - \hat{X}_{n-1}$ $(\hat{X}_0 = X_0)$ จะได้

$$d_1 = X_1 - \hat{X}_0 \quad (2.11)$$

$$\hat{d}_1 = Q[d_1] = d_1 + q_1 \quad (2.12)$$

$$\hat{X}_1 = \hat{X}_0 + \hat{d}_1 = X_0 + d_1 + q_1 = X_1 + q_1 \quad (2.13)$$

$$d_2 = X_2 - \hat{X}_1 \quad (2.14)$$

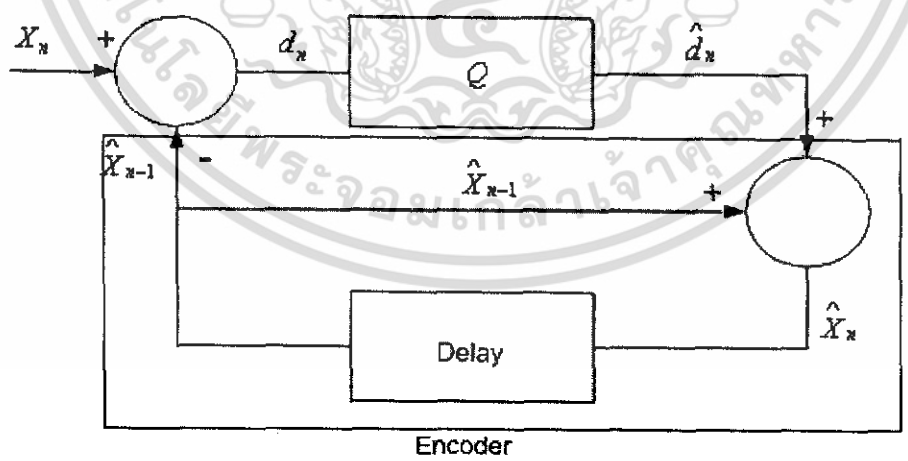
$$\hat{d}_2 = Q[d_2] = d_2 + q_2 \quad (2.15)$$

$$\hat{X}_2 = \hat{X}_1 + \hat{d}_2 = \hat{X}_1 + d_2 + q_2 \quad (2.16)$$

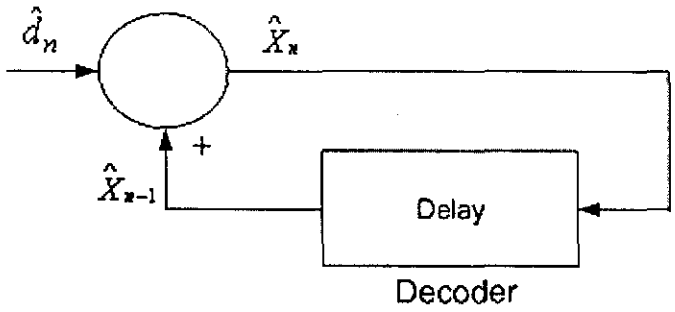
$$= X_2 + q_2 \quad (2.17)$$

เราจะพบว่า $\hat{X}_n = X_n + q_n \quad (2.18)$

ค่าผิดพลาดจากการควอนไตซ์จะไม่เพิ่มขึ้นเมื่อลำดับเพิ่มขึ้น ซึ่งเทคนิคที่กล่าวมาสามารถเขียนเป็นระบบการเข้ารหัส และถอดรหัสได้ดังนี้

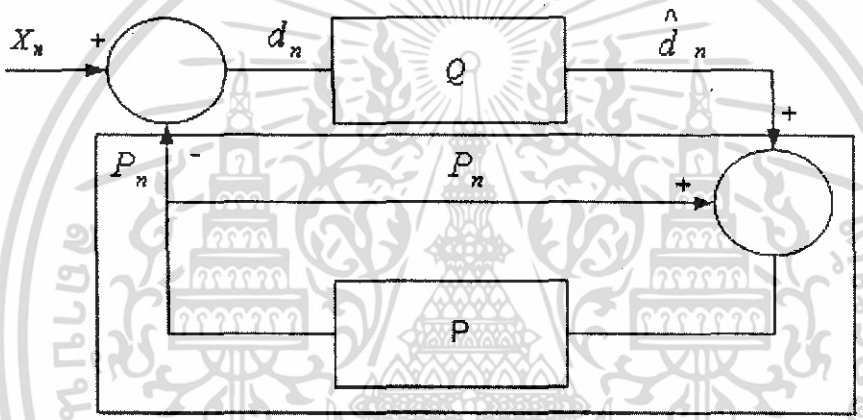


รูป 2.7 แสดงระบบการเข้ารหัสแบบผลต่าง

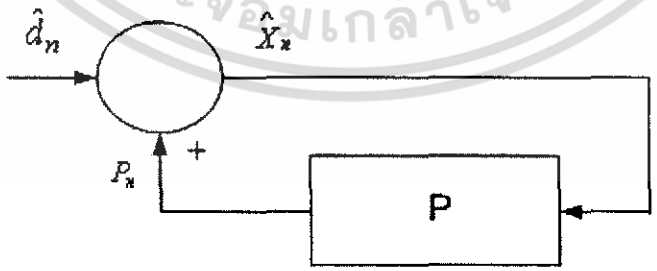


รูปที่ 2.8 แสดงระบบการถอดรหัสแบบผลต่าง

จากรูปที่ 2.7 และ 2.8 เป็นตัวอย่างของระบบเข้ารหัสและถอดรหัสเมื่อทำนายว่า $X_n = \hat{X}_{n-1}$ ซึ่งเป็นรูปแบบหนึ่งของระบบดังรูป 2.9 และ 2.10



รูป 2.9 แสดงระบบการเข้ารหัสแบบผลต่าง



รูปที่ 2.10 แสดงระบบการถอดรหัสแบบผลต่าง

จากรูปที่ 2.9 และ 2.10 นำมาเขียนเป็นสมการได้ดังนี้

$$\hat{d}_n = Q[d_n] = d_n + q_n \tag{2.19}$$

$$d_n = X_n - p_n \tag{2.20}$$

$$\hat{X}_n = \hat{d}_n + p_n = d_n + q_n + p_n = X_n + q_n \tag{2.21}$$

โดย p_n คือค่าที่ทำนายได้โดย

$$p_n = f(\hat{X}_{n-1}, \hat{X}_{n-2}, ..., \hat{X}_0) \tag{2.22}$$

2.6 การเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น (Adaptive Differential Pulse Code Modulation: ADPCM)

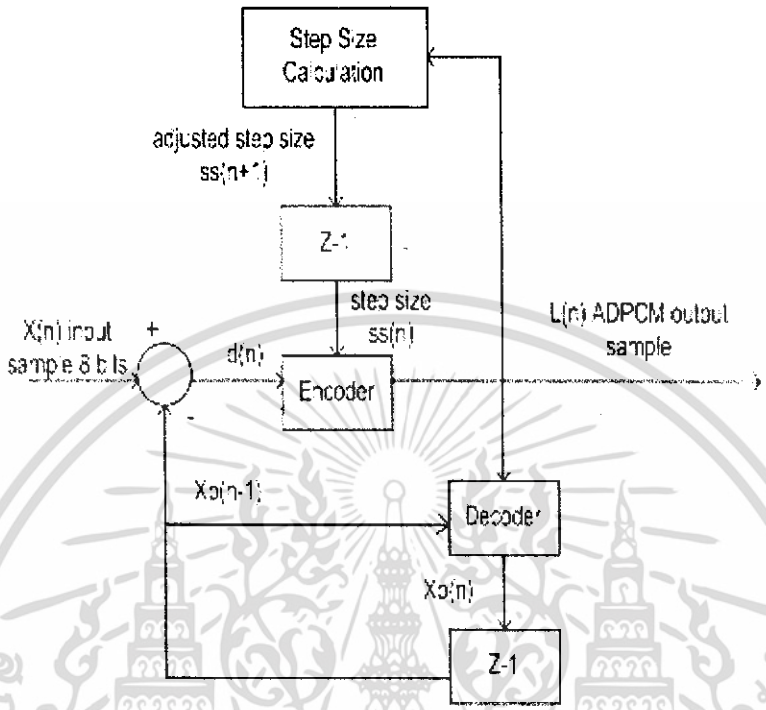
ในการใช้งานพีซีเอ็ม มีอัตราการส่งมาตรฐานคือ 64 Kbit/s ซึ่งต้องการช่องสัญญาณในการส่งที่มีแบนด์วิธกว้างในการนำไปใช้งาน เช่นงานเกี่ยวกับความปลอดภัยของสัญญาณเสียงในการส่งในสายส่งบนช่องสัญญาณวิทยุซึ่งมีความจุต่ำ การประยุกต์ใช้งานแบบนี้เป็นที่ต้องการสำหรับการเข้ารหัสสัญญาณเสียงที่มีอัตราการส่งต่ำ จากการศึกษาได้พบว่าการเข้ารหัสสัญญาณแบบดีพีซีเอ็ม เป็นวิธีการบีบอัดสัญญาณวิธีหนึ่งที่ถูกนำมาใช้ในการลดจำนวนบิตของการเข้ารหัสสัญญาณ แต่ความต้องการในการลดจำนวนบิตในการเข้ารหัสนั้นยังคงมีอยู่เรื่อยๆ จึงได้เกิดระบบการเข้ารหัสสัญญาณเสียงอีกแบบหนึ่งขึ้นมา คือ การเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่นดูเลชั่น

หลักการของการเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น หรือ เอดีพีซีเอ็ม จะคล้ายคลึงกับระบบดีพีซีเอ็ม แต่มีการเพิ่มคุณสมบัติในการปรับขั้นของการควอนไตซ์ และวิธีการเปรียบเทียบสัญญาณเข้าไปในระบบดีพีซีเอ็ม ซึ่งทำให้ลดจำนวนบิตในการเข้ารหัสได้มากขึ้น มาตรฐานของระบบเอดีพีซีเอ็ม จะอธิบายถึงการคอมเพรซซิ่งและเอ็กแพนดิ้ง ค่าความยาวของแซมเปิ้ลจาก 8 บิต จะลดลงเหลือ 3,4 หรือ 5 บิต โดยมีอัตราการบีบอัดเป็น 2.67,2 และ 1.6 ตามลำดับ และมีอัตราการส่งสัญญาณเท่ากับ 24,32 และ 40 Kbit/s ตามลำดับ เมื่อมีอัตราการแซมปลิ้งเท่ากับ 8 KHz โดยใช้ควอนไตซ์เซอร์เป็นแบบอแด็พทีฟควอนไตซ์เซอร์ (adaptive quantizer) ซึ่งค่าระดับการควอนไตซ์จะเปลี่ยนตามค่าระดับของสัญญาณแบบนอนยูนิฟอร์มแล้วค่าสเต็ปไซต์ของระดับการควอนไตซ์ยังปรับตามค่าระดับสัญญาณผลต่างที่เข้ามา คือ เมื่อระดับของสัญญาณผลต่างมีค่าสูงค่าสเต็ปไซต์ก็จะปรับเป็นค่าสูง แต่ถ้าระดับของผลต่างมีค่าต่ำค่าสเต็ปไซต์ก็จะปรับเป็นค่าต่ำเพื่อให้จำนวนบิตที่ใช้ในการเข้ารหัสสัญญาณสามารถครอบคลุมค่าระดับของผลต่างของสัญญาณได้ทั้งหมด ส่วนการเปรียบเทียบสัญญาณจะใช้อแด็พทีฟพรีดิกเตอร์ (adaptive predictor) เป็นตัวหาค่าประมาณของสัญญาณอินพุทเพื่อนำมาใช้เปรียบเทียบกับสัญญาณอินพุทที่เข้ามาตัวถัดไป

2.6.1 หลักการของการเข้ารหัส เอดีพีซีเอ็ม

จากรูปที่ 2.11 เป็นบล็อกไดอะแกรมของเอดีพีซีเอ็ม ซึ่งสามารถแบ่งเป็นขั้นตอนการทำงานได้ดังนี้ คือ ทำการแปลงสัญญาณเสียงอินพุท 64 Kbit/s ให้เป็นสัญญาณแบบยูนิฟอร์มพีซีเอ็มซึ่งได้ค่า

แอมปลิจูดของสัญญาณ แล้วใช้เป็นสัญญาณอินพุทในการเข้ารหัสสัญญาณ จากนั้นหาค่าผลต่างของสัญญาณ (difference signal) ซึ่งได้จากการลบระหว่างค่าคาดคะเนสัญญาณอินพุทปัจจุบันกับค่าสัญญาณอินพุทปัจจุบัน



รูปที่ 2.11 แสดงบล็อคไดอะแกรมการเข้ารหัส เอดีพีซีเอ็ม

เมื่อกำหนดให้

$d(n)$ = สัญญาณผลต่าง (Difference signal)

$X(n)$ = สัญญาณอินพุท (Input signal)

$Xp(n-1)$ = ค่าคาดคะเนสัญญาณอินพุทก่อนหน้า (Previous signal estimate)

จะได้ว่า

$$d(n) = X(n) - Xp(n-1) \qquad (2.23)$$

ค่าสัญญาณผลต่างที่ได้จะถูกนำไปทำการจัดระดับการควอนไตซ์ตามค่าสเต็ปไซต์ปัจจุบัน [ss (n)] ซึ่งแทนลอจิกการเข้ารหัส และผลที่ได้คือ รหัสเอดีพีซีเอ็ม ซึ่งเป็นเอาท์พุทเพื่อนำไปทำการถอดรหัสให้ได้สัญญาณเสียงที่ใกล้เคียงกับสัญญาณเสียงเดิมกลับมา

ค่ารหัสเอดีพีซีเอ็มที่ได้จะถูกนำกลับไปใช้ในการคำนวณค่าสเต็ปไซต์ต่อไป : $ss(n+1)$ และอีกส่วนจะถูกป้อนเข้าไปยังอินเวอร์สอแด็พทีฟควอนไตซ์เซอร์ (Inverse adaptive quantizer) เพื่อสร้างสัญญาณผลต่างที่ถูกควอนไตซ์ที่ใกล้เคียงกับสัญญาณผลต่างตัวเดิมกลับคืนมา

เมื่อกำหนดให้

$v(n)$ = สัญญาณผลต่างที่ถูกควอนไตซ์ (Quantized difference signal)

$q(n)$ = ค่าความผิดพลาดของการควอนไตซ์ (Quantization error)

จะได้ว่า

$$v(n) = d(n) + q(n) \tag{2.24}$$

และเนื่องจากการจัดระดับเป็นแบบอแด็พทีฟ การหาค่าสัญญาณผลต่างที่ถูกควอนไตซ์จึงเป็นแบบอแด็พทีฟ จากนั้นสัญญาณผลต่างที่ถูกควอนไตซ์จะถูกรวมเข้ากับค่าคาดคะเนสัญญาณอินพุทตัวก่อน (Previous signal estimate) เพื่อสร้างสัญญาณกลับคืนมา (Reconstructed signal) ป้อนให้กับอแด็พทีฟพรีดิกเตอร์ (adaptive predictor) ร่วมกับสัญญาณผลต่างที่ถูกควอนไตซ์ เพื่อทำนายค่าคาดคะเนสัญญาณอินพุทตัวถัดไป

เมื่อกำหนดให้

$r(n)$ = สัญญาณที่สร้างกลับคืนมา (Reconstructed signal)

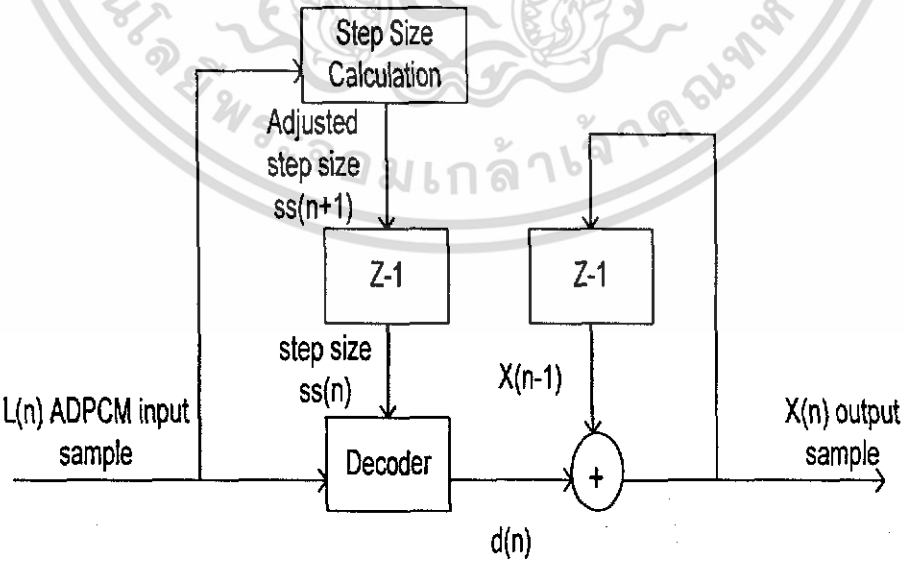
จะได้ว่า

$$r(n) = Xp(n-1) + v(n) \tag{2.25}$$

ค่าคาดคะเนสัญญาณอินพุทที่ได้จะนำไปเปรียบเทียบกับสัญญาณอินพุทปัจจุบันที่เข้ามาใหม่ เพื่อหาค่าสัญญาณผลต่างตัวถัดไป แล้วนำไปสร้างเป็นรหัสเอดีพีซีเอ็มตัวถัดไป

2.6.2 การถอดรหัสเอดีพีซีเอ็ม

ในรูปที่ 2.12 จะแสดงในส่วนของบล็อกไดอะแกรมของกระบวนการถอดรหัสเอดีพีซีเอ็ม โดยโครงสร้างของกระบวนการถอดรหัสสัญญาณจะเหมือนกับส่วนป้อนกลับของสัญญาณที่กระบวนการเข้ารหัสสัญญาณ ซึ่งจะรับค่ารหัสเอดีพีซีเอ็ม และค่าสเต็ปไซต์ โดยเป็นการคำนวณผลของค่าความแตกต่างซ้ำไปอีกครั้ง และจะทำการสะสมค่าที่ประมาณได้ให้อยู่ในค่าของสัญญาณอินพุทที่กระบวนการเข้ารหัส (x)



รูปที่ 2.12 แสดงบล็อคไดอะแกรมการถอดรหัส เอดีพีซีเอ็ม

เมื่อกำหนดให้

$\hat{X}$ = สัญญาณที่ได้จากการถอดรหัสของเอดีพีซีเอ็ม

จะได้ว่า

$$\hat{X}(n) = Xp(n-1) + d(n) \tag{2.26}$$

2.7 การวัดคุณภาพของสัญญาณจากการบีบอัด

นิยมใช้การหาอัตราส่วนของอินพุทกับสัญญาณรบกวน (Signal to noise ratio: SNR) ดังสมการ

$$SNR = 10\log_{10}[\frac{1}{N}\sum_{i=1}^{N} X^2 \Big/ MSE] \tag{2.27}$$

เมื่อกำหนดให้

X = สัญญาณอินพุท

โดย ค่า Mean square error: MSE มีค่าดังนี้

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(X - \hat{X}\right)^2 \tag{2.28}$$

เมื่อกำหนดให้

$\hat{X}$ = สัญญาณที่ได้จากการถอดรหัสของเอดีพีซีเอ็ม

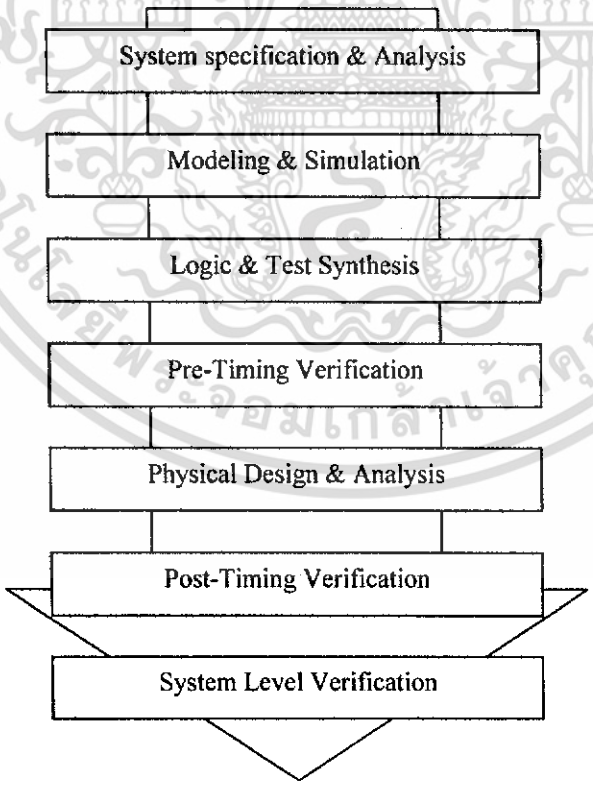
2.8 ภาษาวีเอชดีแอล (VHDL)

การออกแบบวงจรเชิงเลข (Digital Circuit) นั้น ในปัจจุบันก้าวหน้าไปอย่างมาก โดยการใช้ภาษาบรรยายการทำงานของวงจร (Hardware Description Language : HDL) ซึ่งเป็นภาษาที่ใช้สำหรับออกแบบฮาร์ดแวร์ โดยภาษาที่เป็นมาตรฐานสากลเช่น Verilog หรือ VHDL (VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit)) หรือภาษาที่ไม่เป็นมาตรฐาน เช่น AHDL (Altera Hardware Description Language) หรือ PHDL (Philips Hardware Description Language) เป็นต้น มาบรรยายการทำงานของวงจรที่ได้ออกแบบไว้ ตัวอย่างเช่นการใช้ภาษาวีเอชดีแอล (VHDL) มาทำการออกแบบวงจรกรองสัญญาณเชิงเลข (Digital Filter) ทำให้ลดความยุ่งยากในการนำเอาอุปกรณ์มาเชื่อมต่อให้เป็นวงจร รวมทั้งลดเวลาที่ใช้ในการออกแบบและทดสอบการทำงาน ซึ่งมีความแตกต่างเป็นอย่างมาก เมื่อเปรียบเทียบกับการออกแบบในอดีตที่ผ่านมา คือผู้ออกแบบจะต้องนำเอาอุปกรณ์แต่ละตัวที่ทำการออกแบบไว้มาทำการต่อทดลองในวงจรจริงและทำการทดสอบวงจรเพื่อหาข้อผิดพลาดซึ่งต้องใช้เวลานานกับการแก้ปัญหาแต่ละอย่างที่เกิดขึ้น แต่ในการออกแบบด้วยภาษาวีเอชดีแอล (VHDL) ผู้ออกแบบเพียงแต่เขียนซอสโค๊ด (Source Code) บรรยายการทำงานของวงจร หลังจากนั้นก็ทำการคอมไพล์ (Compile) แล้ว

จำลองการทำงาน (Simulate) ดูว่าฟังก์ชั่นการทำงานและไทม์มิ่ง (Timing) ตามที่ต้องการหรือไม่ จากนั้นก็ทำการนำซอสโค้ดที่ได้ไปทำการสังเคราะห์ด้วยโปรแกรมสังเคราะห์ (Synthesis Tool) สุดท้ายนำวงจรที่ได้จากการสังเคราะห์ไปทำการแม็ป (map) ลงไปยังFPGA (Field Programmable Gate Array) เพื่อเป็นชิป (Chip) ต้นแบบสำหรับการนำไปทดสอบการทำงาน

2.8.1 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมเชิงเลขขนาดใหญ่ที่มีความซับซ้อน ผู้ออกแบบมักมองการออกแบบให้อยู่ในรูปของบล็อกไดอะแกรมก่อน จากนั้นจึงวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษาวีเอชดีแอล (VHDL) นั้นอนุญาตให้อธิบายการทำงานของในแต่ละบล็อก และวิเคราะห์การทำงาน แก้ไขปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ โดยการออกแบบในลักษณะนี้เรียกว่าหลักการทำงานจากบนลงล่าง (Top-Down Design) ซึ่งถ้าเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom-Up design) จะเห็นได้ว่าการออกแบบจากล่างขึ้นบนจะใช้เวลาในการออกแบบมากกว่าเพราะเป็นการวาดรูปด้วยอุปกรณ์ต่างๆ (Schematic Capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบจำลองการทำงาน ตรวจสอบความถูกต้องซึ่งใช้เวลามาก และถ้าวงจรที่ต้องการออกแบบมีความซับซ้อนก็จะเป็นเรื่องที่ยากมากในการออกแบบลักษณะนี้ ดังนั้นการใช้ภาษาวีเอชดีแอล (VHDL) กับหลักการออกแบบจากบนลงล่างจึงเป็นวิธีการที่เหมาะสมสำหรับการออกแบบและพัฒนาวงจรที่มีความซับซ้อนมากขึ้นทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบ



รูปที่ 2.13 แสดงขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.13 แสดงให้เห็นถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในการปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย โดยขั้นตอนการออกแบบจากบนลงล่างมีรายละเอียด 7 ขั้นตอนดังนี้

1. ขั้นตอนการสร้างข้อกำหนดของความต้องการและวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and concept) ในการแก้ปัญหา

2. ขั้นตอนการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษาวีเอชดีแอล (VHDL) สำหรับบรรยายพฤติกรรมการทำงานพร้อมทั้งจำลองการทำงาน เพื่อตรวจสอบความถูกต้องของข้อกำหนด

3. ขั้นตอนการสังเคราะห์ ซึ่งจะต้องทำการกำหนดเทคโนโลยีที่จะมารองรับวงจรที่ออกแบบและระบบช่วยออกแบบจำลองทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์หรือวงจรในระดับเกท (Gate Level) และการเชื่อมต่อกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของเน็ทลิสต์ (Net List) ที่สามารถนำไปผลิตลงอุปกรณ์อื่นได้

4. หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือเน็ทลิสต์ ข้อมูลที่ได้นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงานในเรื่องของความถูกต้องของฟังก์ชั่นแล้ว ยังมีข้อมูลที่เกี่ยวกับเวลาด้วย ซึ่งจากความจริงที่ว่าอุปกรณ์อิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของการเคลื่อนผ่าน (Propagation Delay Time) เสมอถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับนาโนวินาที แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชั่นต่างๆจำนวน 10,000 เกทขึ้นไป เวลาดังกล่าวนี้จะสะสมมากขึ้นจนอาจจะทำให้การทำงานของวงจรทั้งหมดผิดไปหรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาสูงๆได้

5. ขั้นตอนของการผลิตเป็นวงจรจริง (Technology and Device Mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิตซึ่งอาจจะอยู่ในรูปของอุปกรณ์ (FPGA) หรือวงจรรวม ASICs

6. หลังจากที่ได้วงจรจริงมาแล้วยังต้องมีความจำเป็นที่จะต้องตรวจสอบการทำงานที่คำนึงถึงเวลาของวงจรก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นๆให้เป็นระบบ เพราะในขั้นตอนนี้วงจรที่ออกแบบจะประกอบด้วยอินพุทและเอาท์พุทแพด (Pad) ซึ่งเป็นจุดเชื่อมต่อสำหรับรับและส่งสัญญาณกับภายนอก

7. หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่นๆให้เป็นระบบแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นๆอีกครั้งซึ่งเป็นการทดสอบการทำงานจริงขั้นสุดท้าย

2.8.2 การออกแบบวงจรเชิงเลขด้วยอุปกรณ์ FPGA

อุปกรณ์ FPGA (Field Programmable Gate Array) เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการออกแบบตามที่กำหนดไว้ ซึ่งเป็นวิธีการออกแบบ IC (Integrated Circuit) แบบ Semi custom อีกวิธีหนึ่ง เมื่อเทียบกับการทำ ASICs (Application Specific Integrated Circuits) แล้วนั้นจะมีทั้งข้อดีและข้อเสีย คือ การทำ FPGA จะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัดและการทำ FPGA ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือผลิตในปริมาณต่ำ ส่วนข้อดี

ของการทำ FPGA ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (download) นั้นน้อยกว่าการทำ ASIC มากและการตรวจสอบหรือแก้ไขการออกแบบก็ทำได้สะดวก

การทำ FPGA ในปัจจุบันมีประสิทธิภาพและสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากบริษัทผู้ผลิตอุปกรณ์ FPGA ได้เพิ่มความสามารถของอุปกรณ์ FPGA โดยเพิ่มจำนวนองค์ประกอบภายในหรือปรับปรุงโครงสร้างสถาปัตยกรรมภายในและยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ PPR (Partitioning Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วยลักษณะของตัว FPGA และการนำไปใช้งาน

สำหรับตัวอุปกรณ์ FPGA นั้นก็มีโครงสร้างพื้นฐาน เทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรแกรมที่แตกต่างกันสำหรับผู้ผลิตแต่ละราย นอกจากนั้นอุปกรณ์ FPGA ของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกันบางส่วน ในการใช้งานนั้นอุปกรณ์ FPGA สามารถนำไปประยุกต์ใช้งานได้หลากหลาย เช่น การประมวลผลสัญญาณเชิงเลข (DSP: Digital Signal Processing) การออกแบบไมโครคอนโทรลเลอร์ เป็นต้น

2.8.3 การออกแบบโดยใช้ภาษาอธิบายการทำงานของฮาร์ดแวร์

ในการออกแบบวงจรเชิงเลขนั้นทำได้โดยการวาดวงจรหรือใช้ภาษาอธิบายฮาร์ดแวร์ ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วย FPGA และ ASICs ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์ แต่ในกรณีที่ออกแบบโดยวิธีการวาดวงจรจะแตกต่างกันโดยที่การทำวิธีนี้จะต้องคำนึงถึงเทคโนโลยีที่ใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป จะเห็นว่าการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ทำได้สะดวกกว่า เพราะการทำด้วยวิธีนี้ไม่ต้องคำนึงถึงเทคโนโลยีที่จะใช้ (Technology Independence) และที่สำคัญการออกแบบด้วยวิธีนี้สามารถจะแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกเพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี

ในการเขียนโค้ด สิ่งที่ต้องคำนึงถึงคือเขียนอย่างไรจึงจะสามารถสังเคราะห์เป็นวงจรได้และให้คุณสมบัติของวงจรตามกำหนดเพราะลักษณะการเขียนโค้ดจะมีผลโดยตรงกับวงจรที่ได้ เนื่องจากในการสังเคราะห์วงจรนั้นซอร์ฟแวร์สังเคราะห์วงจร (Synthesis Tools) จะทำการสังเคราะห์ตามโค้ดที่เขียนถ้าอธิบายการทำงานของวงจรเดียวกันแต่เขียนโค้ดในลักษณะที่ต่างกันเมื่อสังเคราะห์แล้วจะได้วงจรที่ต่างกันและจากวงจรที่ต่างกันเมื่อนำไปทำต้นแบบด้วย FPGA หรือการทำ ASICs แล้วจะได้ไอซีที่มีคุณสมบัติต่างกันทั้งในด้านขนาดและความเร็ว (area and time) ส่วนการเขียนโค้ดลักษณะใดเพื่อให้ได้ผลลัพธ์ที่ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ในการออกแบบ

2.8.4 การจำลองการทำงานของวงจร (Simulation)

ขั้นตอนการนี้เป็นขั้นตอนที่สำคัญเพราะเป็นขั้นตอนที่ใช้ตรวจสอบฟังก์ชั่นการทำงานของวงจรว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อที่จะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะใช้ซอร์ฟแวร์

สำหรับทำการจำลองการทำงานของวงจร เช่น V-System และ ModelSim ของบริษัท Model Technology

2.8.5 การสังเคราะห์วงจร (Synthesis)

ในขั้นตอนนี้จะใช้ซอฟแวร์สังเคราะห์วงจร (Synthesis tools) ทำการสังเคราะห์โค้ดเพื่อให้ได้เป็นวงจรขึ้นมา แล้วตรวจสอบว่าซอฟแวร์นั้นๆสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการใช้หรือไม่ โดย FPGA ที่นิยมใช้งาน เช่น บริษัท Xilinx ตระกูล XC4000 และบริษัท Altera ตระกูล FLEX 10k ซอฟแวร์สังเคราะห์วงจรที่นิยมใช้เช่นโปรแกรม Leonardo Spectrum ของบริษัท Exemplar Logic ซึ่งในขั้นตอนนี้ซอร์ฟแวร์สังเคราะห์วงจรจะแปลงโค้ดและทำการออปติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่ใช้ นอกจากนี้ยังสามารถกำหนดข้อบังคับสำหรับวงจรได้เช่น ข้อบังคับในเรื่องของเวลา (time constraints) หรือข้อบังคับในเรื่องของพื้นที่ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอนออปติไมซ์เพื่อให้วงจรที่ได้เป็นไปตามกำหนด ส่วนสำคัญในการออปติไมซ์คือการเทียบ (Mapping) วงจรเข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างสถาปัตยกรรมภายในอุปกรณ์ FPGA ในกรณีของ Xilinx ตระกูล XC4000 และ Altera ตระกูล FLEX 10k จะเทียบโดยใช้วิธี LUT (Look Up Table) เมื่อทำการสังเคราะห์วงจรเสร็จแล้วซอร์ฟแวร์สังเคราะห์วงจรก็จะแสดงผลว่าวงจรที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีความหน่วง (delay) เท่าไร ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้างเป็นต้น

2.8.6 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ให้เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่แยกออกจากกันมีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จำได้เพื่อช่วยลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (Routing) ในขั้นตอนนี้จะใช้ซอร์ฟแวร์ทำ โดยซอฟแวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิปฟลอป (flipflop) ลงในทรัพยากรต่างๆที่มีอยู่ภายในอุปกรณ์ FPGA (CKBs, IOBs, BUFT และ edge decoder) หลังจากทำขั้นตอนนี้เสร็จแล้วสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนซอฟแวร์ที่ใช้ในขั้นตอนนี้ขึ้นอยู่กับตัว FPGA ที่ใช้งาน FPGA ของบริษัท Xilinx จะใช้ Xilinx Foundation Series 2.11 ซึ่งซอฟแวร์ตัวนี้จะรวมเอาซอฟแวร์ย่อยอื่นๆอีกเพื่อให้การทำ PPR (Partitioning, Placement and Routing) เป็นไปอย่างต่อเนื่อง ส่วน FPGA ของบริษัท Altera จะใช้ Altera MAX+II

2.8.7 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งสำหรับแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาว่าควรจะอยู่ในตำแหน่งใดในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่นส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทาง (route) ได้ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมี

ความสำคัญ เพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วทำให้ความหน่วงเพิ่มขึ้นหรือตัว router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด

2.8.8 การเชื่อมต่อสัญญาณ (Route)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆภายในอุปกรณ์ FPGA เช่น ระหว่าง CLBs หรือระหว่าง CLBs กับ IOBs ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีจะทำการเชื่อมต่อสัญญาณด้วยตัวเอง (Manual layout) ก็ได้ แต่ทางที่ดีควรใช้ซอฟแวร์ซึ่งทำได้ดีกว่าโดยทำการค้นหาเส้นทางหลายๆครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการกำหนดข้อบังคับทางเวลา (time constraints) จะช่วยให้ได้ผลที่ได้จากการเชื่อมต่อสัญญาณขึ้นได้

2.8.9 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่วงจรผ่านขั้นตอนต่างๆจนกระทั่งผ่านการทำ PPR แล้วนั้นถึงตอนนี้สามารถที่จะดาวน์โหลดลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้ให้เป็นข้อมูลวงจร (Configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (Bit-stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามวงจรที่ออกแบบไว้

จากที่อธิบายมาทั้งหมดจะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นสามารถทำได้สะดวกกว่าการทำ ASICs มากเพราะใช้เวลาน้อยกว่ามาก ส่วนสำคัญที่ใช้ในการทำ FPGA คือซอฟแวร์ที่ใช้ตั้งแต่การเขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงในอุปกรณ์ FPGA ซึ่งซอฟแวร์ที่ใช้ต้องเป็นซอฟแวร์ที่ใช้ต้องทำงานต่อเนื่องกัน

บทที่ 3
หลักการคำนวณและการออกแบบการเข้ารหัสแบบเอดีพีซีเอ็ม

3.1 การเข้ารหัส

3.1.1 รับสัญญาณเสียงอินพุทเข้ามาเก็บไว้ในไฟล์

3.1.2 หาค่าผลต่างของสัญญาณอินพุทกับค่าคาดคะเนสัญญาณอินพุท : d (n)

3.1.3 นำค่าผลต่างที่ได้มาพิจารณาตามค่าสเต็ปไซด์ : ss (n) โดยแบ่งออกเป็น

1) บีบอัดเหลือ 3 บิต

มีเงื่อนไขดังนี้

$B2 = B1 = B0 = 0$

ถ้า $d(n) < 0$

แล้ว $B2 = 1$ และ $d(n) = ABS(d(n))$

ถ้า $d(n) >= ss(n)$

แล้ว $B1 = 1$ และ $d(n) = d(n) - ss(n)$

ถ้า $d(n) >= ss(n)/2$

แล้ว $B0 = 1$

ซึ่งค่ารหัสของเอดีพีซีเอ็ม :

$L(n) = (2\text{^}2 * B2) + (2\text{^}1 * B1) + B0$

2) บีบอัดเหลือ 4 บิต

มีเงื่อนไขดังนี้

$B3 = B2 = B1 = B0 = 0$

ถ้า $d(n) < 0$

แล้ว $B3 = 1$ และ $d(n) = ABS(d(n))$

ถ้า $d(n) >= ss(n)$

แล้ว $B2 = 1$และ $d(n) = d(n) - ss(n)$

ถ้า $d(n) >= ss(n)/2$

แล้ว $B1 = 1$และ $d(n) = d(n) - ss(n)/2$

ถ้า $d(n) >= ss(n)/4$

แล้ว $B0 = 1$

ซึ่งค่ารหัสของเอดีพีซีเอ็ม :

$L(n) = (2\text{^}3 * B3) + (2\text{^}2 * B2) + (2\text{^}1 * B1) + B0$

3) บีบอัดเหลือ 5 บิต

มีเงื่อนไขดังนี้

$B4 = B3 = B2 = B1 = B0 = 0$

ถ้า $d(n) < 0$

แล้ว $B4 = 1$ และ $d(n) = ABS(d(n))$

ถ้า $d(n) >= ss(n)$

แล้ว $B3 = 1$ และ $d(n) = d(n) - ss(n)$

ถ้า $d(n) >= ss(n)/2$

แล้ว $B2 = 2$ และ $d(n) = d(n) - ss(n)/2$

ถ้า $d(n) >= ss(n)/4$

แล้ว $B1 = 1$ และ $d(n) = d(n) - ss(n)/4$

ถ้า $d(n) >= ss(n)/8$

แล้ว $B0 = 0$

ซึ่งค่ารหัสของเอดีพีซีเอ็ม :

$$L(n) = (2\text{^}4 * B4) + (2\text{^}3 * B3) + (2\text{^}2 * B2) + (2\text{^}1 * B1) + B0$$

3.1.4 การคำนวณค่าสเต็ปไซต์

จากทั้งกระบวนการเข้ารหัสและกระบวนถอดรหัสเอดีพีซีเอ็มในอัลกอริทึมของเอดีพีซีเอ็มนั้น จะเป็นการปรับค่าของการควอนไตซ์สเต็ปไซต์โดยอยู่บนพื้นฐานของค่าเอดีพีซีเอ็มก่อนหน้า ซึ่งค่าสเต็ปไซต์ในแซมเปิ้ลถัดไปจะเป็น $[ss(n+1)]$

โดยคำนวณค่าได้ดังสมการ

$$ss(n+1) = ss(n) + 1.1M(L(n)) \qquad (3.1)$$

เมื่อกำหนดให้

$L(n)$ = ค่ารหัสเอดีพีซีเอ็ม

ซึ่งจากสมการที่ 3.1 สามารถนำมาทำการปรับปรุงค่าเพื่อให้มีประสิทธิภาพขึ้นโดยแสดงดังตารางแสดงค่าแมกนีจูดของค่ารหัสเอดีพีซีเอ็ม ซึ่งใช้เป็นดัชนีบ่งชี้สำหรับการปรับค่าตัวประกอบ

L(n)	M(L(n))
111 or 011	8
110 010	4
101 001	-1
100 000	-1

ตารางที่ 3.1 แสดงค่าแมกนีจูดของเอดีพีซีเอ็มแบบ 3 บิต ซึ่งใช้เป็นดัชนีบ่งชี้สำหรับการปรับค่าตัวประกอบ

L(n)	M(L(n))
1111 or 0111	+8
1110 0110	+6
1101 0101	+4
1100 0100	+2
1011 0011	-1
1010 0010	-1
1001 0001	-1
1000 0000	-1

ตารางที่ 3.2 แสดงค่าแมกนีจูดของเอดีพีซีเอ็มแบบ 4 บิต ซึ่งใช้เป็นดัชนีบ่งชี้สำหรับการปรับค่าตัวประกอบ

L(n)	M(L(n))
11111 or 01111	8
11110 01110	7
11101 01101	6
11100 01100	5
11011 01011	4
11010 01010	3
11001 01001	2
11000 01000	1
10111 00111	-1
10110 00110	-1
10101 00101	-1
10100 00100	-1
10011 00011	-1
10010 00010	-1
10001 00001	-1
10000 00000	-1

ตารางที่ 3.3 แสดงค่าแมกนีจูดของเอดีพีซีเอ็มแบบ 5 บิต ซึ่งใช้เป็นดัชนีบ่งชี้สำหรับการปรับค่าตัวประกอบ

3.2 การถอดรหัส

3.2.1 รับค่ารหัสของเอดีพีซีเอ็ม : $L(n)$

3.2.2 นำค่า $L(n)$ มาพิจารณาหาค่าสเต็ปไซด์ตามสมการที่ 3.1

3.2.3 นำค่าสเต็ปไซด์ที่ได้มาพิจารณาหาค่าผลต่างของสัญญาณ : $d(n)$ ดังนี้

1) บีบอัดเหลือ 3 บิต

มีเงื่อนไขดังนี้

$$d(n) = (ss(n) * B1) + (ss(n)/2 * B0) + (ss(n)/4)$$

ถ้า $B2 = 1$

แล้ว $d(n) = d(n) * (-1)$

2) บีบอัดเหลือ 4 บิต

มีเงื่อนไขดังนี้

$$d(n) = (ss(n) * B2) + (ss(n)/2 * B1) + (ss(n)/4 * B0) + ss(n)/8)$$

ถ้า $B3 = 1$

แล้ว $d(n) = d(n) * (-1)$

3) บีบอัดเหลือ 5 บิต

มีเงื่อนไขดังนี้

$$d(n) = (ss(n) * B3) + (ss(n)/2 * B2) + (ss(n)/4 * B1) + (ss(n)/8 * B0) + (ss(n)/16)$$

ถ้า $B4 = 1$

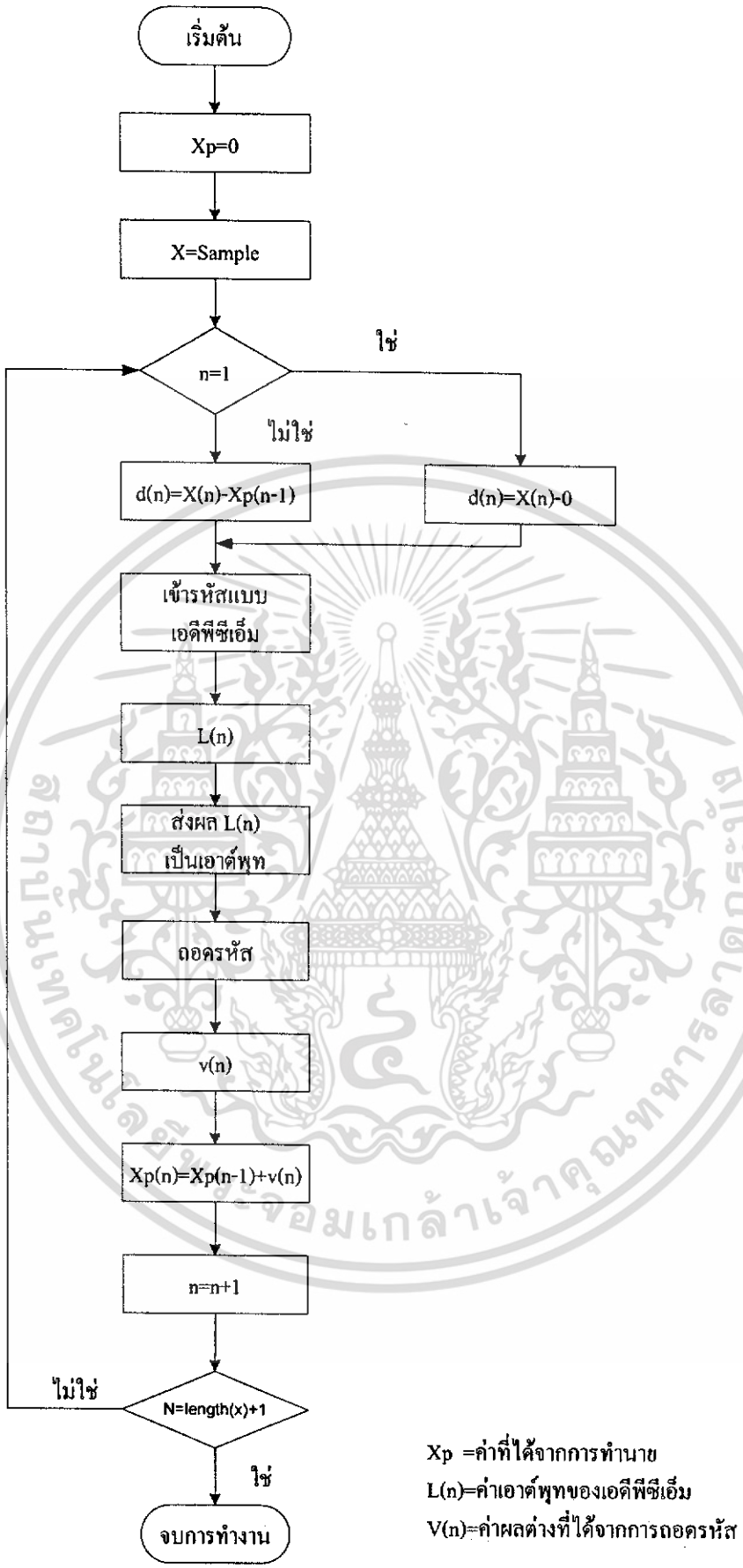
แล้ว $d(n) = d(n) * (-1)$

3.2.4 หาค่าแซมเปิ้ลจากการถอดรหัส: $X(n) = X(n-1) + d(n)$

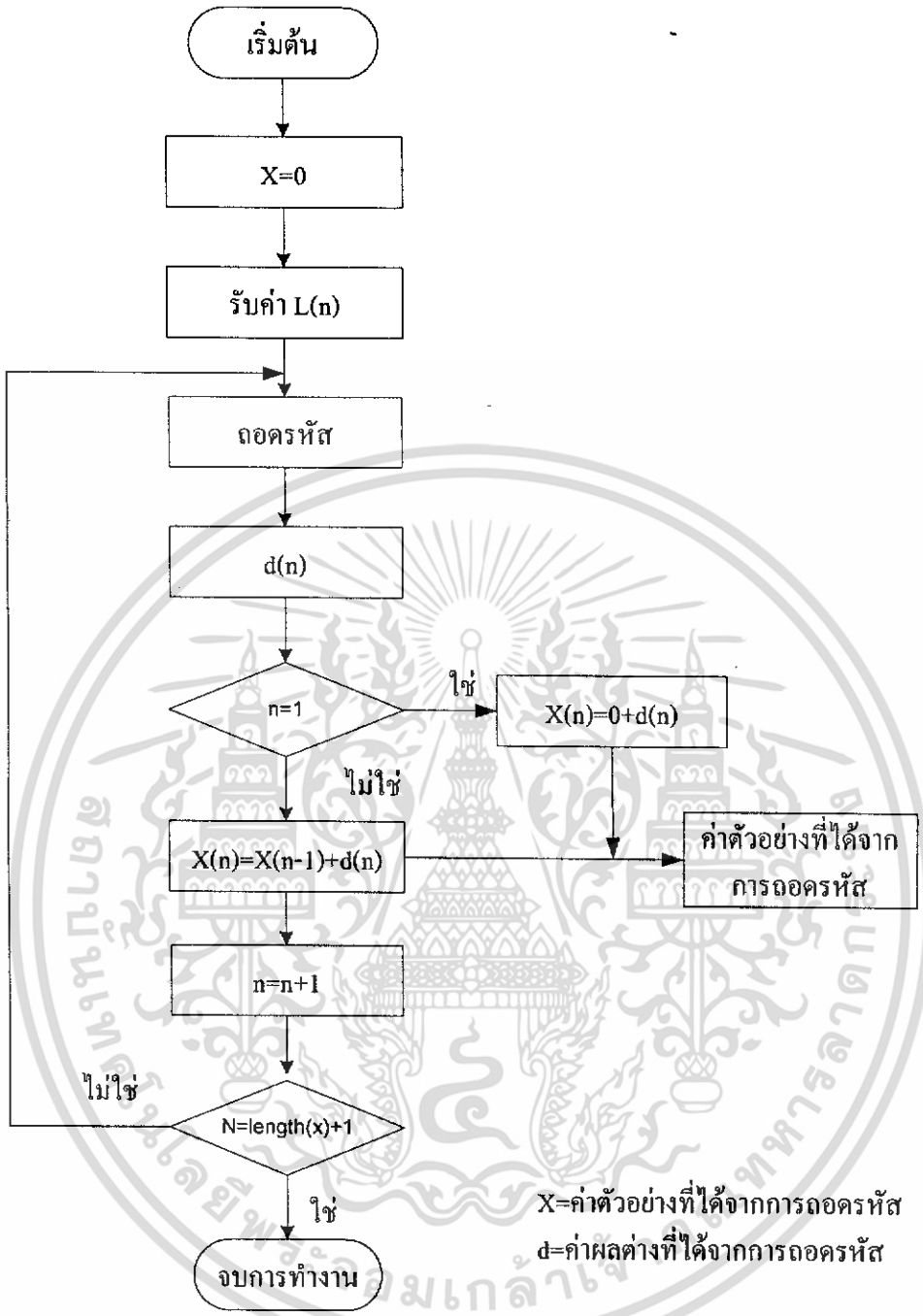
สำหรับวิธีการเอดีพีซีเอ็มนี้ เป็นวิธีการที่เหมาะสมสำหรับการปรับค่าในรูปแบบของคลื่นแบบไซนูซอยคอลแต่ไม่เหมาะสำหรับคลื่นในแบบอื่น

3.3 กระบวนการเข้ารหัสและถอดรหัสสัญญาณ (Flowchart)

จากการคำนวณนำค่าอัลกอริทึมมาแสดงกระบวนการการทำงานในการเข้ารหัสสัญญาณ และการถอดรหัสสัญญาณแบบเอดีพีซีเอ็ม ดังแสดงในรูปที่ 3.1 และ รูปที่ 3.2



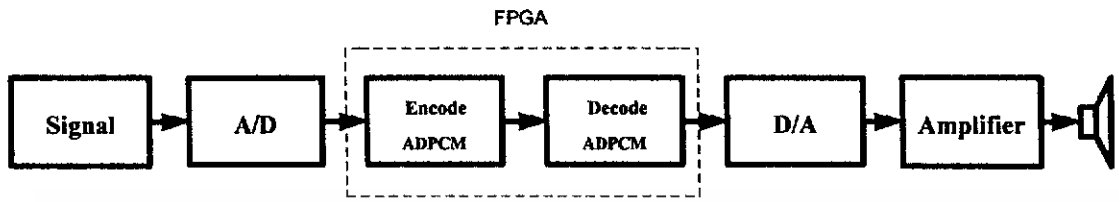
รูปที่ 3.1 แสดงกระบวนการเข้ารหัสเสียงแบบเอดีพีซีเอ็ม



รูปที่ 3.2 แสดงกระบวนการถอดรหัสข้อมูลเสียงแบบเอดีพีซีเอ็ม

3.4 การออกแบบวงจรในการเข้า-ถอดรหัสด้วยเอฟพีจีเอ (FPGA)

เราจะทำการออกแบบเอฟพีจีเอโดยทำการเขียนโปรแกรมการทำงาน โดยใช้ภาษาวีเอชดีแอล (VHDL) ทำการคอมไพล์ (Compile) แล้วจำลองการทำงาน (Simulation) ของโปรแกรมแต่ละส่วนที่ได้เขียนขึ้นโดยแบ่งออกเป็นส่วนๆ ดังนี้



รูปที่ 3.3 แสดงบล็อกไดอะแกรมการทำงานของวงจร

3.4.1 ภาคส่งข้อมูล

ภาคส่งข้อมูลประกอบด้วยสัญญาณเสียงเป็นสัญญาณอินพุทและวงจร A/D converter ซึ่งจะทำหน้าที่การแปลงสัญญาณเสียงจากสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอลขนาด 8 บิต

3.4.2 เอฟพีจีเอ (FPGA)

ประกอบด้วยการเข้ารหัส เอดีพีซีเอ็ม และการถอดรหัสเอดีพีซีเอ็ม

3.4.2.1 การเข้ารหัสเอดีพีซีเอ็ม

การเข้ารหัสจะประกอบด้วยวงจรผลต่างและค่าสัญญาณก่อนหน้า วงจรเข้ารหัส วงจรจัดระดับสัญญาณ โดยวงจรผลต่างจะทำการหาค่าผลต่าง (dout) ระหว่างสัญญาณปัจจุบัน (x_in) และสัญญาณก่อนหน้า (x_pre)

$$dout = x\_in - x\_pre$$

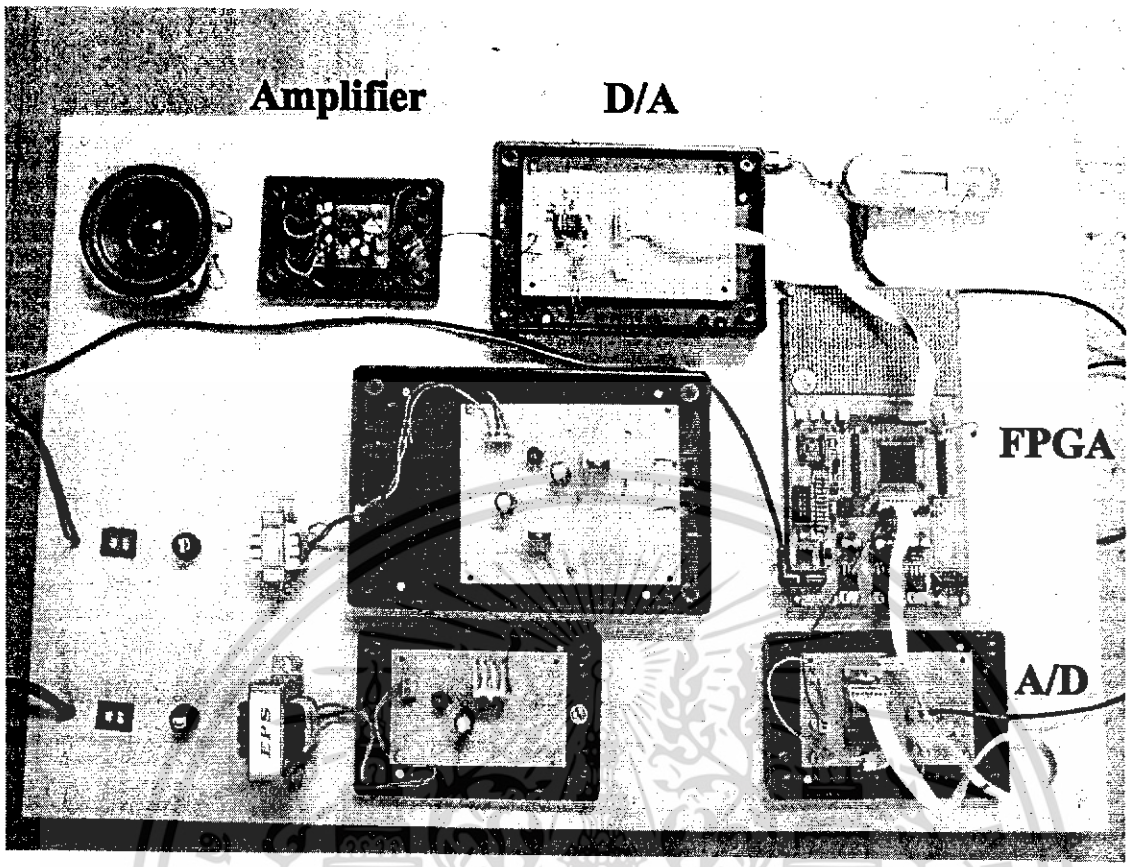
จากนั้นนำค่าสัญญาณผลต่างที่ได้ไปเข้าวงจรเข้ารหัส เพื่อไปทำการจัดระดับการควอนไตซ์ตามค่าสเต็ปไซต์ปัจจุบัน (ss) หลังจากการเข้ารหัสสัญญาณทำให้มีจำนวนบิตที่ลดลงซึ่งจากการทำงานเราจะทำการเข้ารหัสสัญญาณจาก 8 บิตให้เป็น 4 บิต

3.4.2.2 การถอดรหัสเอดีพีซีเอ็ม

การถอดรหัสจะประกอบด้วยวงจรถอดรหัส วงจรกู้สัญญาณกลับ โดยวงจรถอดรหัสก็จะทำการถอดรหัสโดยขึ้นกับค่าสเต็ปไซต์เช่นกันเพื่อจะทำให้สัญญาณกลับมาเป็น 8 บิตเหมือนเดิมหลังจากนั้นก็จะนำไปเข้าวงจรกู้สัญญาณกลับโดยสัญญาณที่ได้จะใกล้เคียงกับสัญญาณเดิม

3.4.3 ภาคแสดงผลข้อมูล

ภาคแสดงผลข้อมูลประกอบด้วยวงจร D/A converter ซึ่งจะทำหน้าที่แปลงสัญญาณดิจิตอลขนาด 8 บิตให้เป็นสัญญาณอนาลอก และวงจรขยายสัญญาณแล้วส่งผ่านไปที่ลำโพงเพื่อทำการกระจายเสียง



รูปที่ 3.4 วงจรในการเข้า-ถอดรหัสด้วยเอฟพีจีเอ (FPGA)

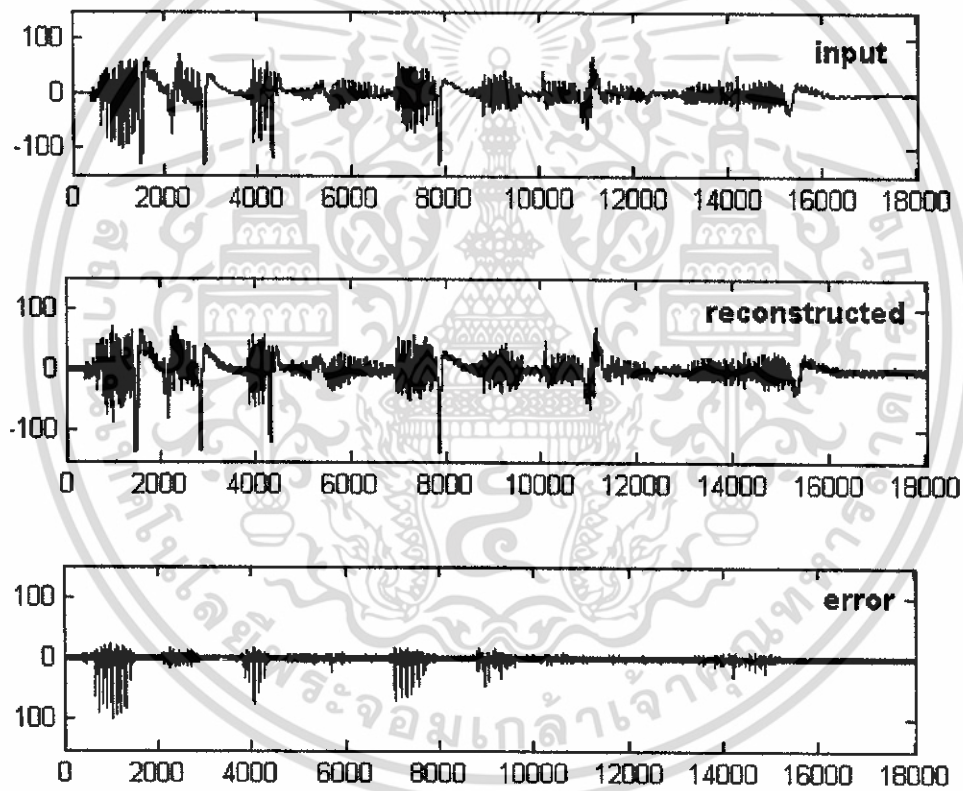
บทที่ 4
การทดลองและผลการทดลอง

โครงงานนี้ได้ทำการศึกษาการเข้ารหัสสัญญาณเสียง โดยใช้หลักการเอดีพีซีเอ็มทำการบีบอัดข้อมูลเสียงจากความยาวแซมเปิ้ล 8 บิต ให้เหลือ 3, 4 หรือ 5 บิต แล้วทำการเปรียบเทียบค่าความผิดพลาด (error)ระหว่างสัญญาณอินพุทกับสัญญาณเอาท์พุทที่ได้ แล้วคำนวณอัตราส่วนของสัญญาณต่อสัญญาณรบกวน (Signal to noise ratio : SNR) รวมทั้งการจำลองการทำงานของการเข้ารหัสสัญญาณเสียงโดยใช้หลักการเอดีพีซีเอ็มด้วยภาษา VHDL และการทำงานของวงจรในการเข้า-ถอดรหัสด้วยเอฟพีจีเอ (FPGA)

4.1 ผลจากการทำงานด้วยโปรแกรม Matlab

โดยแบ่งให้สัญญาณเสียงอินพุทที่เข้ามามีลักษณะแตกต่างกัน ดังนี้

1. สัญญาณเสียงของผู้ชายที่พูดคำว่า "การบีบอัดเสียงแบบเอดีพีซีเอ็ม"

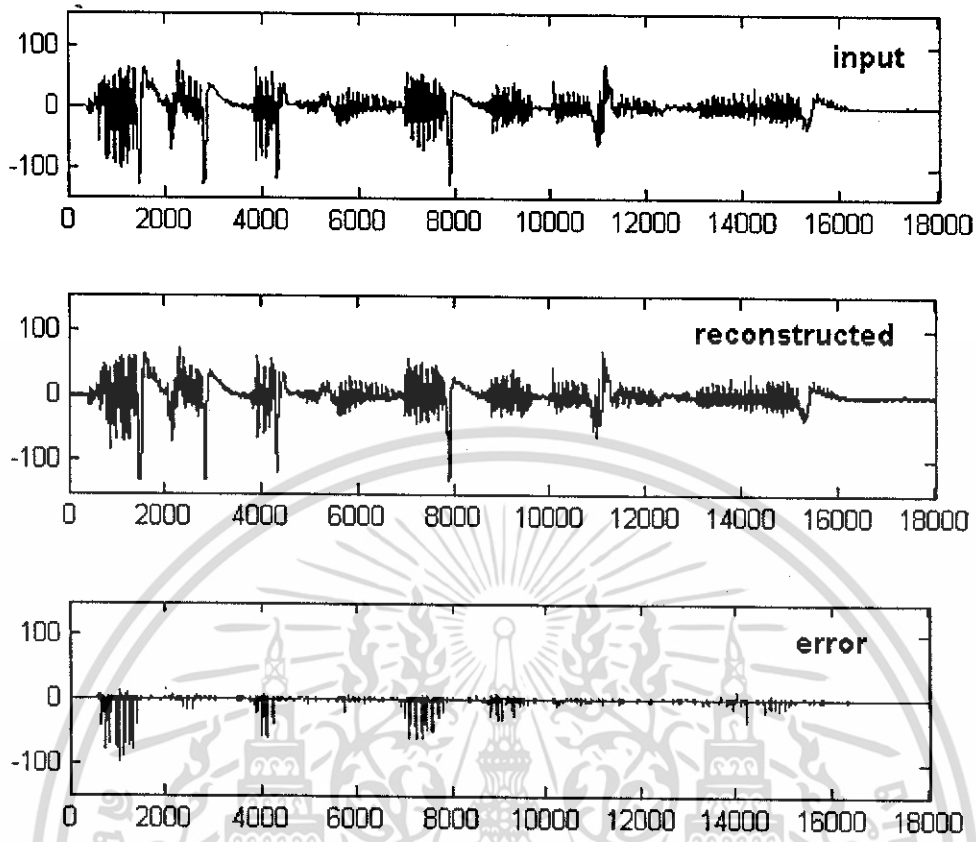


รูปที่ 4.1 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 3 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.1 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 12.2351$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.67

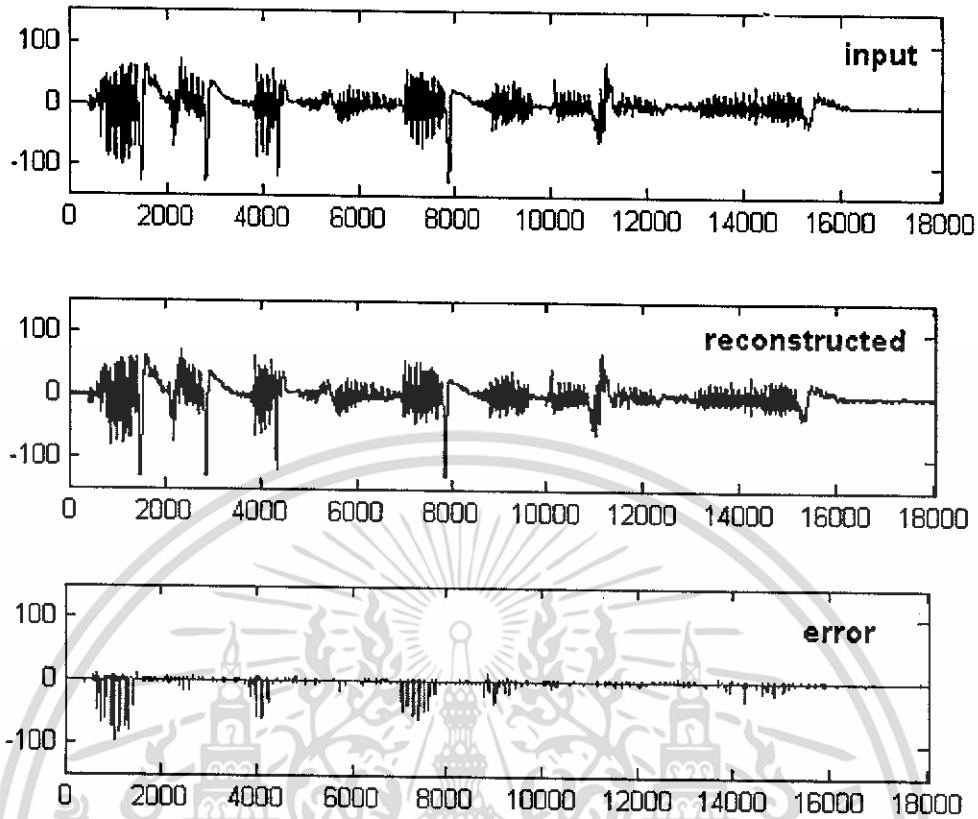


รูปที่ 4.2 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 4 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.2 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 15.4884$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00



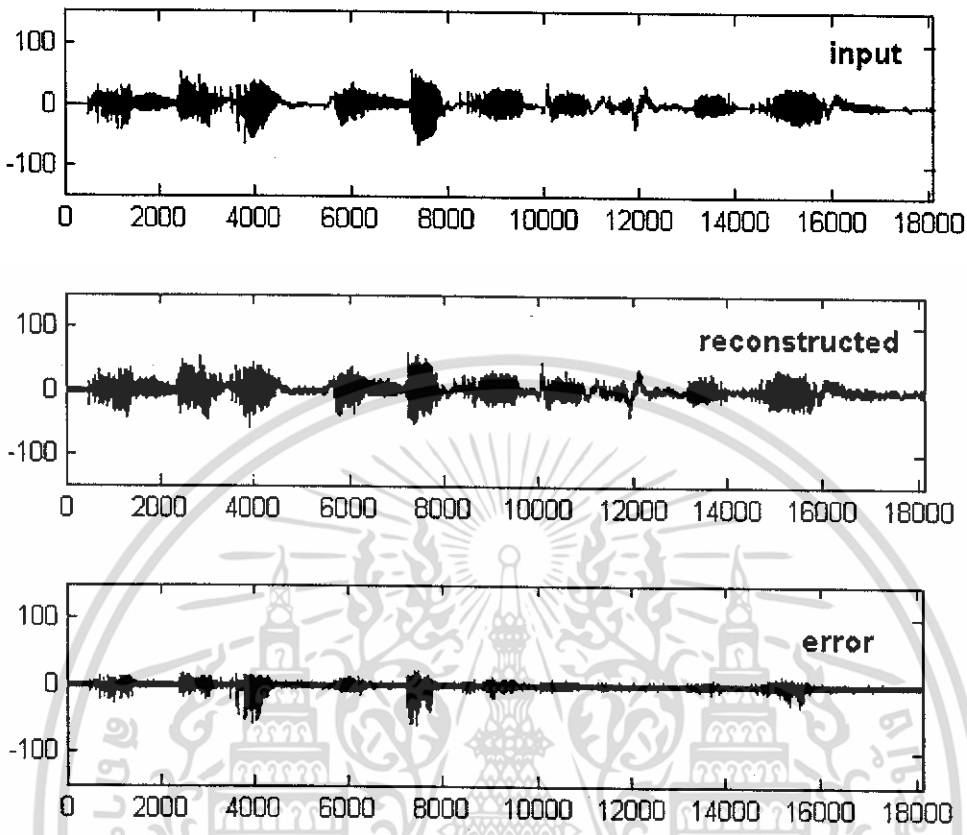
รูปที่ 4.3 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแชมเปิล 8 บิตเหลือ 5 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.3 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 15.7644$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 1.6

2. สัญญาณเสียงของผู้หญิงที่พูดคำว่า "การบีบอัดเสียงแบบเอดีพีซีเอ็ม"

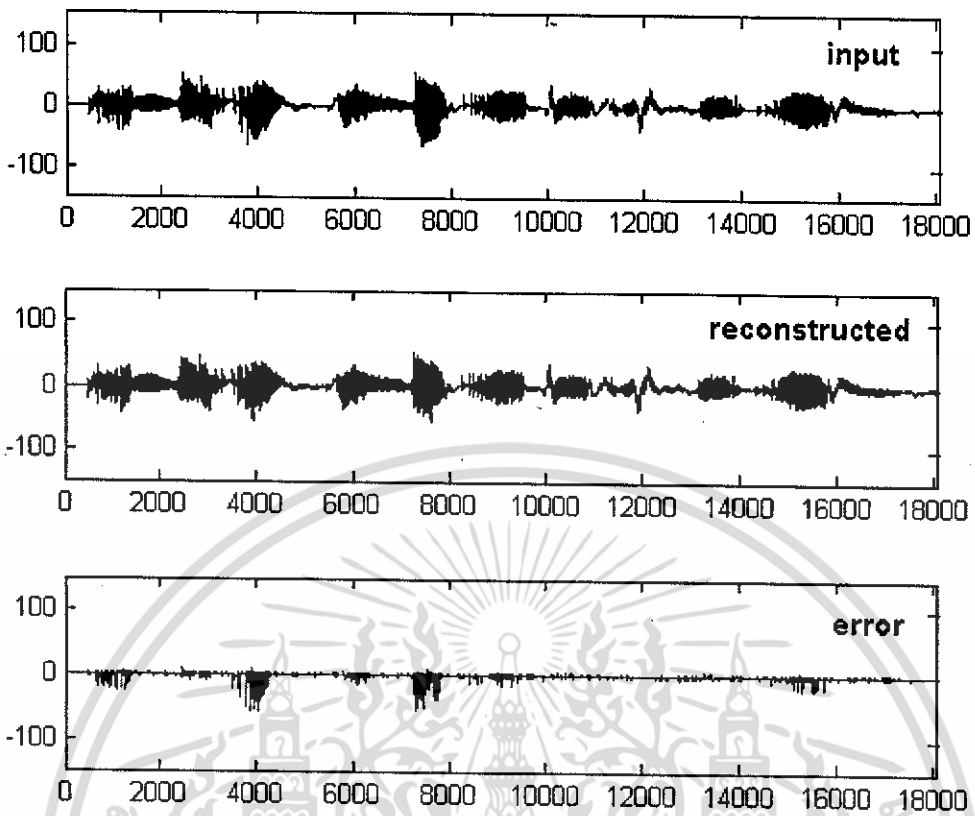


รูปที่ 4.4 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 3 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.4 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 5.9904$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.67

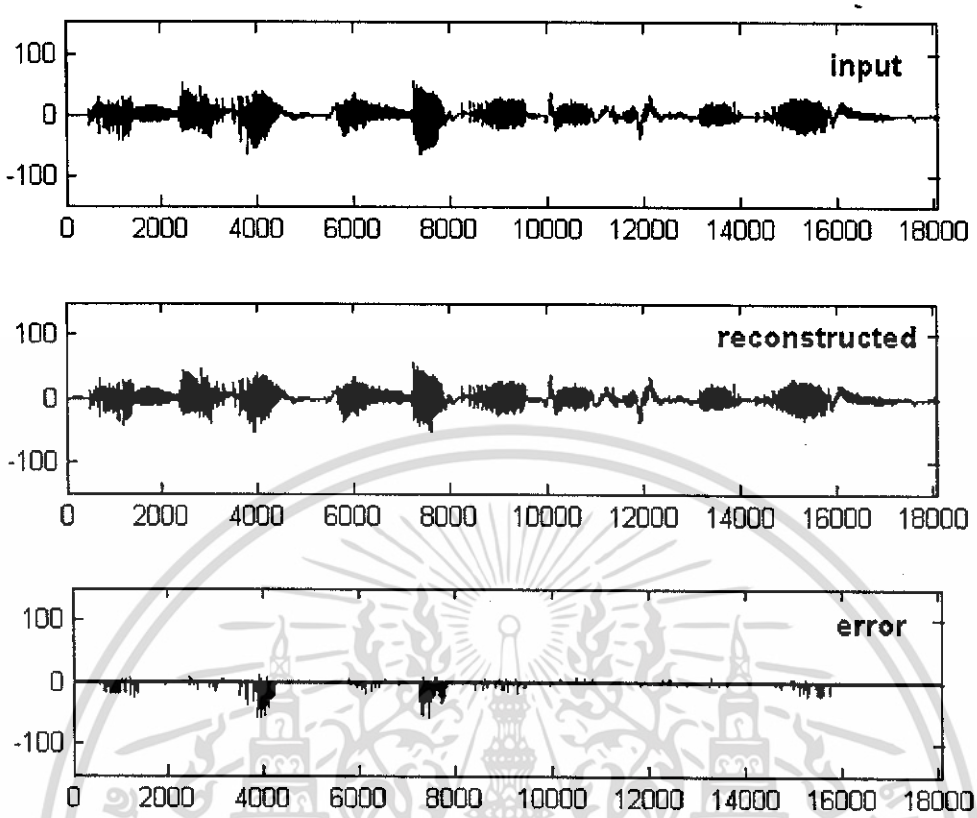


รูปที่ 4.5 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแชมเปิล 8 บิตเหลือ 4 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.5 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 12.4782$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00



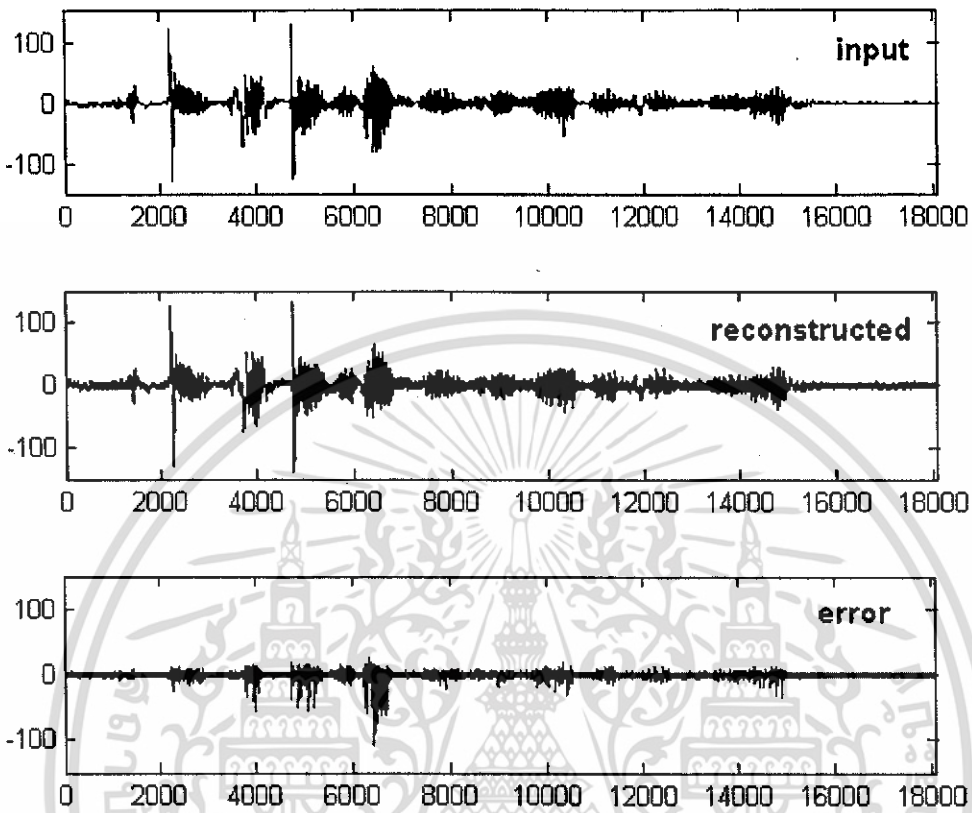
รูปที่ 4.6 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแชมเปิล 8 บิตเหลือ 5 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.6 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 13.1016$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 1.60

3. สัญญาณเสียงของผู้ชายที่พูดภาษาอังกฤษ “Speech compression using ADPCM”

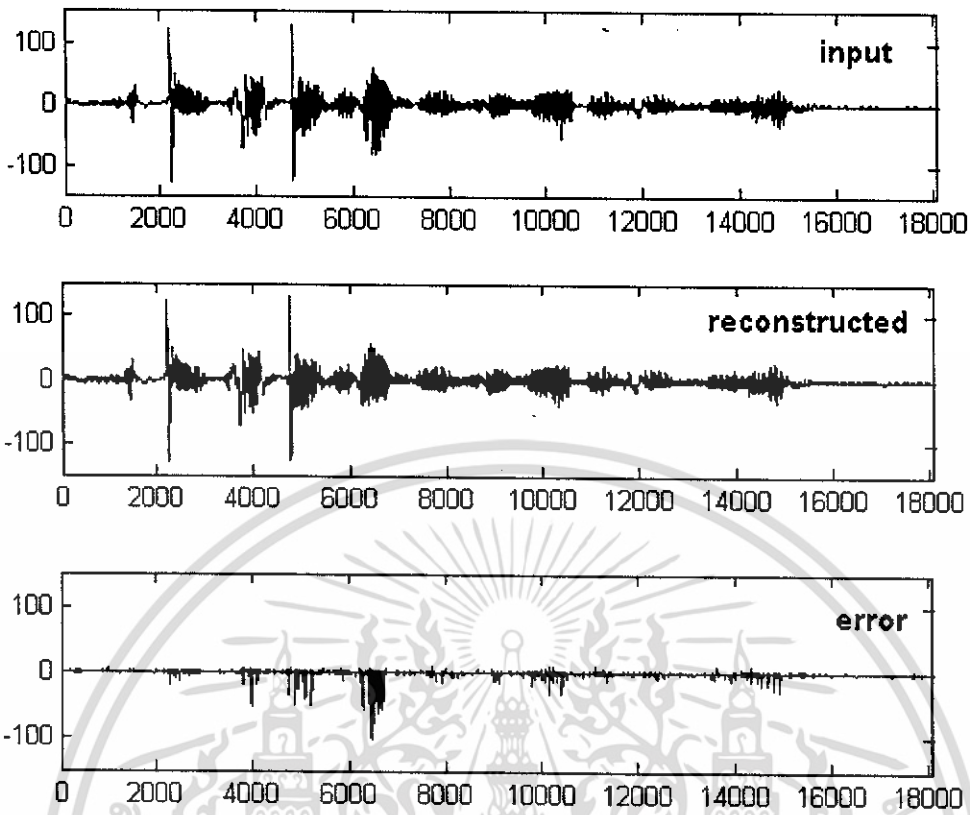


รูปที่ 4.7 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 3 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.7 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 9.2304$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.67

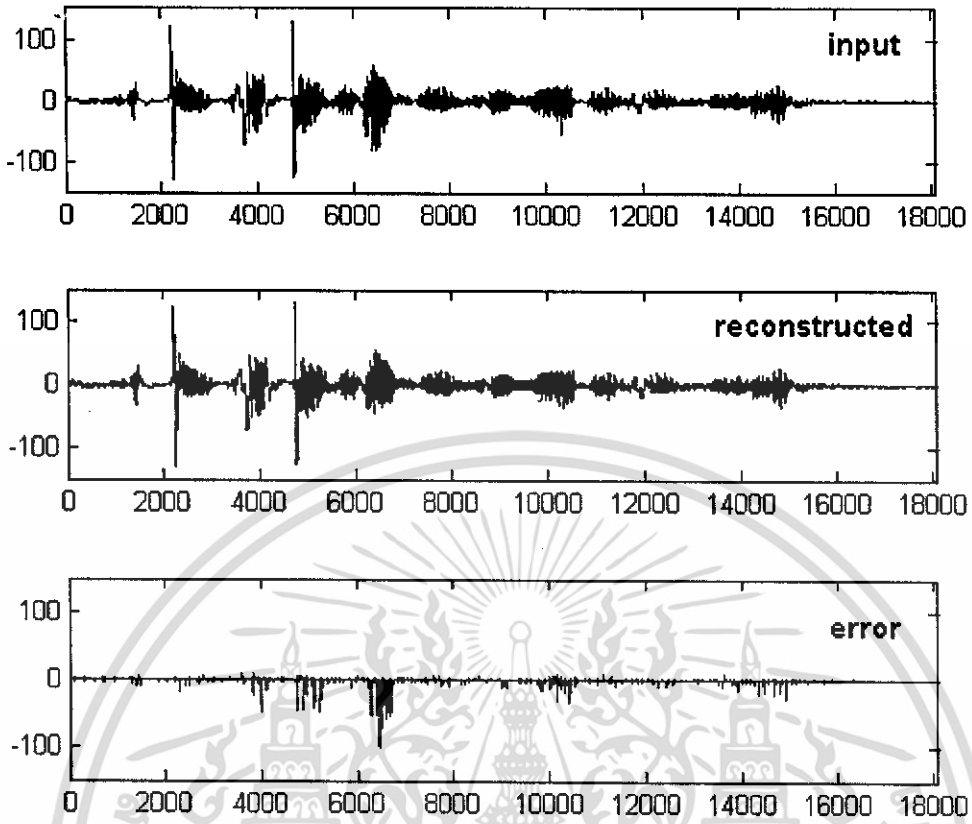


รูปที่ 4.8 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 4 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.8 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 13.5957$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00



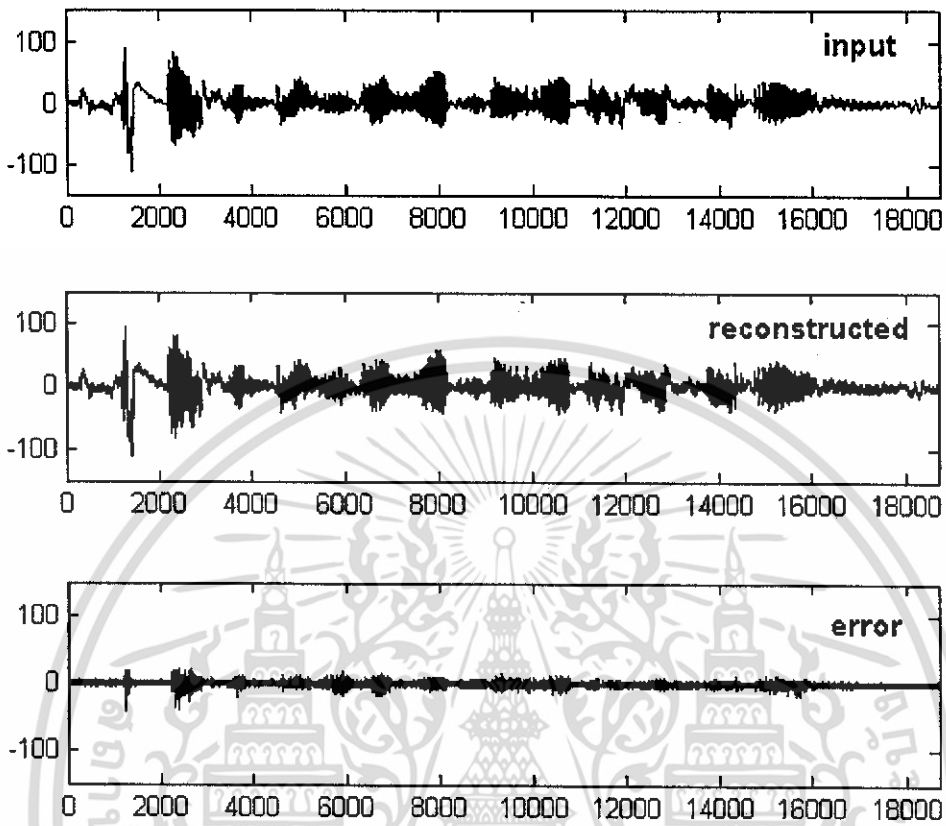
รูปที่ 4.9 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 5 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.9 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 13.9578$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 1.6

4. สัญญาณเสียงของผู้หญิงที่พูดภาษาอังกฤษ "Speech compression using ADPCM"

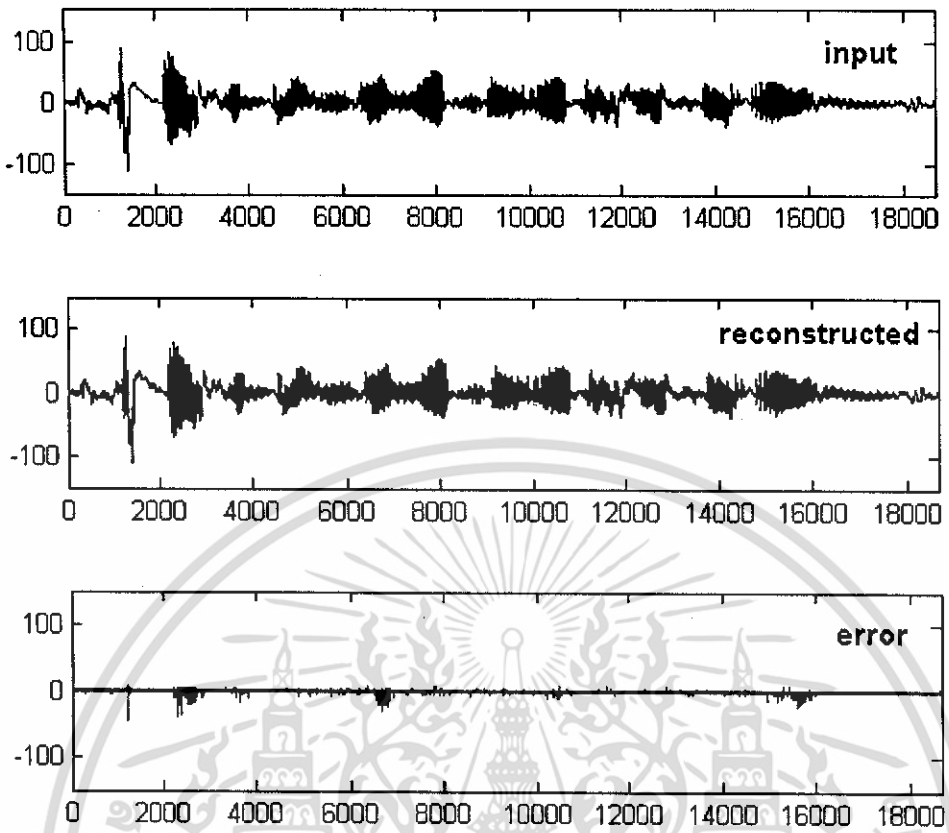


รูปที่ 4.10 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 3 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.10 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 11.3204$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.67

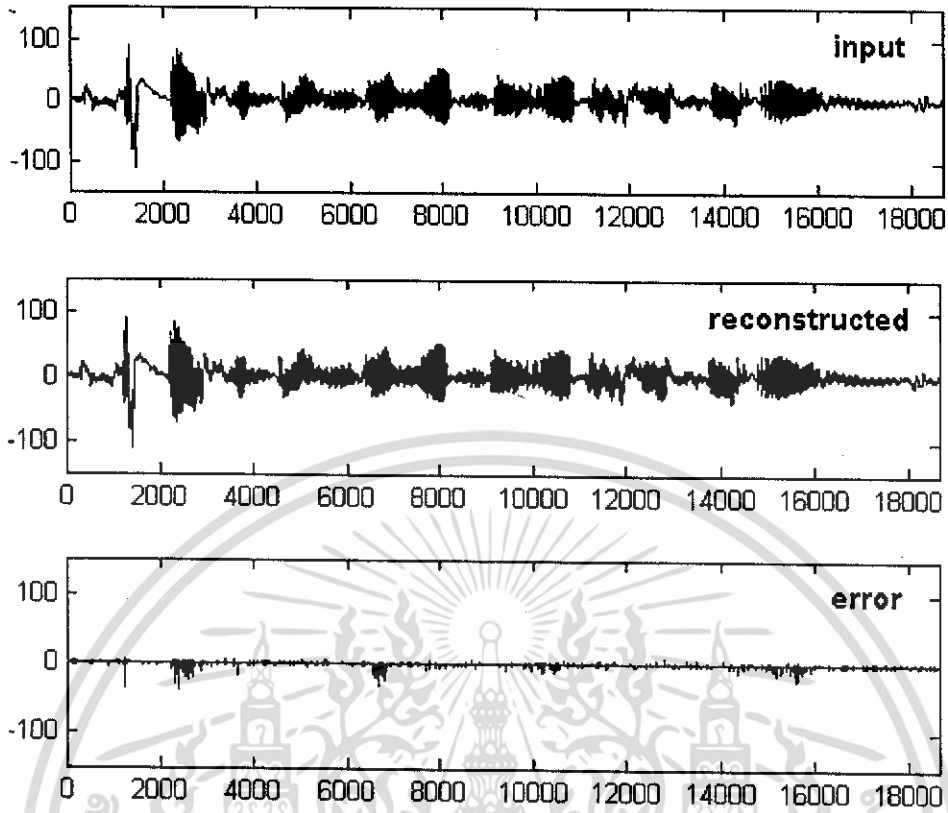


รูปที่ 4.11 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 4 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.11 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 19.8729$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00



รูปที่ 4.12 แสดงการเปรียบเทียบสัญญาณอินพุทก่อนทำการเข้ารหัสความยาวแซมเปิล 8 บิตเหลือ 5 บิต สัญญาณที่ได้จากการถอดรหัส และค่าความผิดพลาด

จากรูปที่ 4.12 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 21.2631$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 1.6

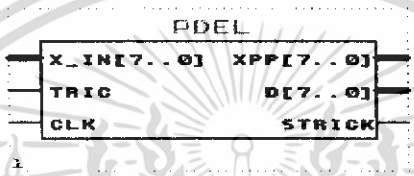
4.2 ผลจากการจำลองการทำงานด้วยภาษา VHDL

จากการออกแบบ สามารถทำการเขียนโปรแกรมการทำงานโดยใช้ภาษาวีเอชดีแอล (VHDL) ทำการคอมไพล์ (Compile) แล้วจำลองการทำงาน (Simulation) ของโปรแกรมแต่ละส่วนที่ได้เขียนขึ้น โดยแบ่งออกเป็นส่วนๆดังนี้

4.2.1 การเข้ารหัส

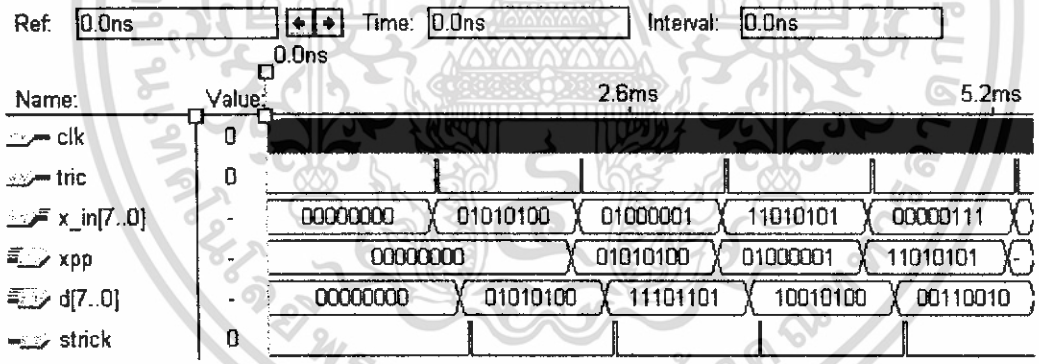
การเข้ารหัสจะประกอบด้วยวงจรผลต่างและค่าสัญญาณก่อนหน้า วงจรเข้ารหัส วงจรจัดระดับสัญญาณ

1) วงจรผลต่างและค่าสัญญาณก่อนหน้า



รูปที่ 4.13 แสดงสัญลักษณ์ของวงจรผลต่างและค่าสัญญาณก่อนหน้า

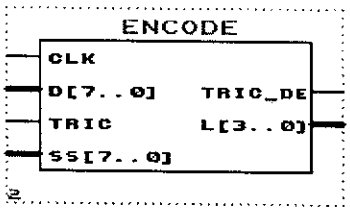
สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.14



รูปที่ 4.14 แสดงผลการจำลองการทำงานของวงจรผลต่างและค่าสัญญาณก่อนหน้า

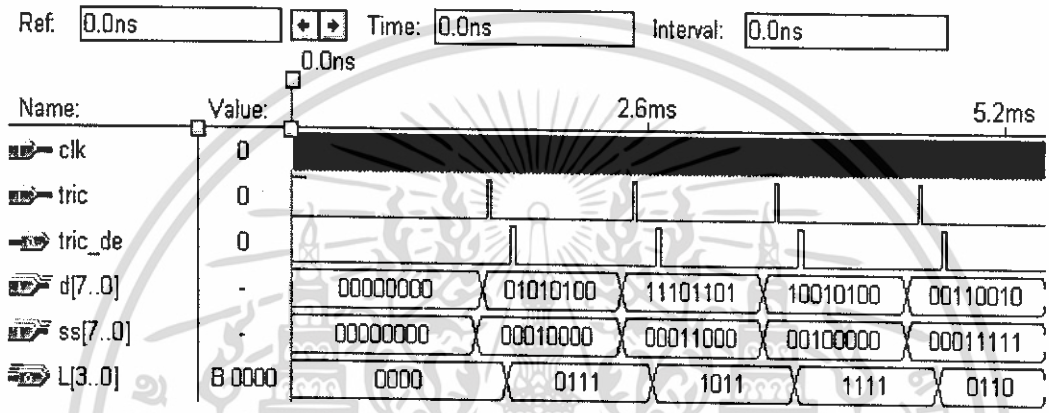
จากการทดลองพบว่า เมื่อสัญญาณ (x_in[7..0])มีค่าเป็น 00000000 ค่าสัญญาณก่อนหน้า (xpp) ก็จะเป็น 00000000 เนื่องจากยังไม่มีค่าสัญญาณ เมื่อสัญญาณมีค่าเป็น 01010100 ค่าสัญญาณก่อนหน้าก็จะเป็นค่า 00000000 และเมื่อสัญญาณมีค่าเป็น 01000001 สัญญาณก่อนหน้าก็จะเป็น 01010100 ซึ่งเป็นไปตามการออกแบบ และค่าผลต่าง (d[7..0] = x_in[7..0] - xpp) พบว่าเมื่อหาค่าผลต่างระหว่าง 01000001 กับ 01010100 จะได้เป็น 11101101 ซึ่งการจำลองการทำงานได้เป็นไปตามที่ออกแบบไว้ดังรูปที่ 4.14

2) วงจรเข้ารหัส



รูปที่ 4.15 แสดงสัญลักษณ์ของวงจรเข้ารหัส

สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.16



รูปที่ 4.16 แสดงผลการจำลองการทำงานของวงจรเข้ารหัส

จากวิธีการออกแบบการเข้ารหัสเอดีพีซีเอ็ม โดยการบีบอัดเหลือ 4 บิต มีเงื่อนไขดังนี้

$B3 = B2 = B1 = B0 = 0$

ถ้า $d(n) < 0$

แล้ว $B3 = 1$ และ $d(n) = ABS(d(n))$

ถ้า $d(n) >= ss(n)$

แล้ว $B2 = 1$ และ $d(n) = d(n) - ss(n)$

ถ้า $d(n) >= ss(n)/2$

แล้ว $B1 = 1$ และ $d(n) = d(n) - ss(n)/2$

ถ้า $d(n) >= ss(n)/4$

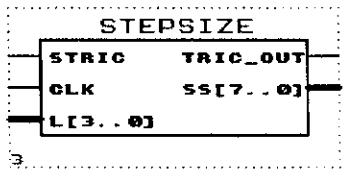
แล้ว $B0 = 1$

ซึ่งค่ารหัสของเอดีพีซีเอ็ม:

$L(n) = (2\text{^}3 * B3) + (2\text{^}2 * B2) + (2\text{^}1 * B1) + B0$

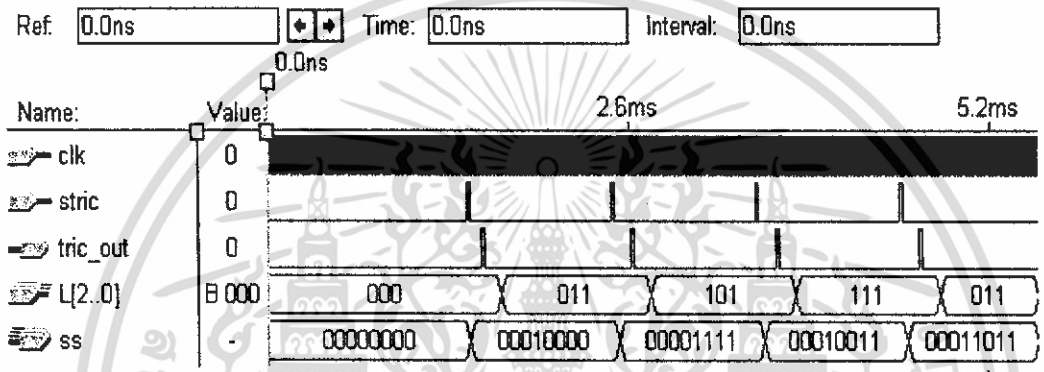
โดยเมื่อนำค่าผลต่าง (d[7..0]) ซึ่งมีค่า 01010100 จากเงื่อนไขข้างบนจะทำให้เราได้ค่ารหัสเอดีพีซีเอ็ม (L[3..0]) ซึ่งเป็นสัญญาณที่ถูกบีบอัด มีค่าเป็น 0111 และเมื่อนำค่าผลต่าง (d[7..0]) ซึ่งมีค่า 11101101 จากเงื่อนไขข้างบนจะทำให้เราได้ค่ารหัสเอดีพีซีเอ็ม (L[3..0]) เป็น 1011 ซึ่งจากการจำลองการทำงานดังรูปที่ 4.16 ค่าที่ได้เป็นไปตามค่าที่ออกแบบไว้

3) วงจรจัดระดับสัญญาณ



รูปที่ 4.17 แสดงสัญลักษณ์ของวงจรจัดระดับสัญญาณ

สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.18



รูปที่ 4.18 แสดงผลการจำลองการทำงานของวงจรจัดระดับสัญญาณ

จากการจำลองการทำงาน เมื่อค่ารหัสเอดีพีซีเอ็ม (L[2..0]) เป็น 101 จะได้ค่าสเต็ปไซต์ในแซมเปิ้ลถัดไปมีค่าเป็น 00010011

ซึ่งจากการคำนวณจะได้ค่าสเต็ปไซต์ในแซมเปิ้ลถัดไปมีค่าเป็น $[ss(n+1)]$

โดยคำนวณค่าได้ดังสมการ

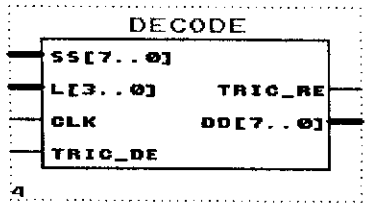
$$ss(n+1) = ss(n) + 1.1M(L(n))$$

เมื่อค่ารหัสเอดีพีซีเอ็ม (L[2..0]) เป็น 101 จะได้ค่า M(L(n)) จากตารางที่ 3.2 มีค่าเป็น +4 จะได้ค่าสเต็ปไซต์ในแซมเปิ้ลถัดไปมีค่าเป็น 00010011 ซึ่งผลที่ได้แสดงให้เห็นว่าค่าที่ได้จากการจำลองการทำงานดังรูปที่ 4.18 มีค่าเช่นเดียวกับค่าที่ได้คำนวณไว้

4.2.2 ส่วนการถอดรหัส

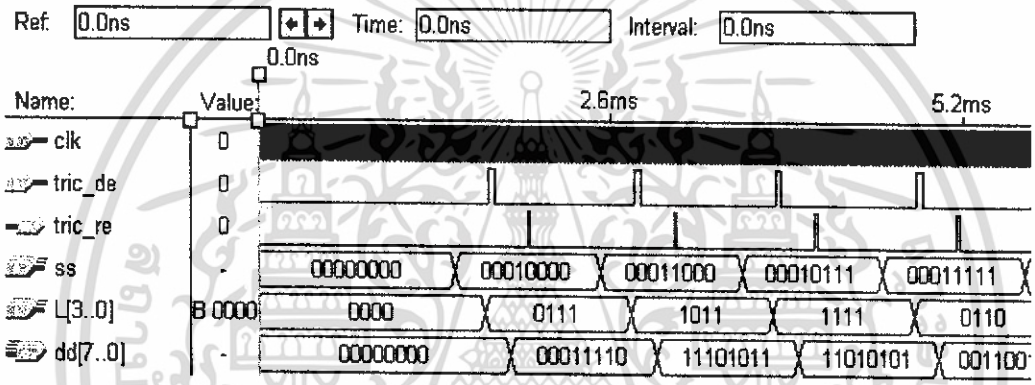
การถอดรหัสจะประกอบด้วยวงจรถอดรหัส วงจรกู้สัญญาณกลับ

1) วงจรถอดรหัส



รูปที่ 4.19 แสดงสัญลักษณ์ของวงจรถอดรหัส

สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.20



รูปที่ 4.20 แสดงผลการจำลองการทำงานของวงจรถอดรหัส

จากการจำลองการทำงานเมื่อ ค่ารหัสเอดีพีซีเอ็ม (L[3..0]) มีค่าเป็น 0111 จะได้ค่าที่ถอดรหัส (dd[7..0]) เป็น 00011110 เมื่อนำมาเปรียบเทียบกับค่าที่ได้จากวิธีการถอดรหัสจากค่ารหัสเอดีพีซีเอ็ม 4 บิต โดยมีเงื่อนไขดังนี้

$$d(n) = (ss(n) * B2) + (ss(n)/2 * B1) + (ss(n)/4 * B0) + ss(n)/8)$$

ถ้า $B3 = 1$

แล้ว $d(n) = d(n) * (-1)$

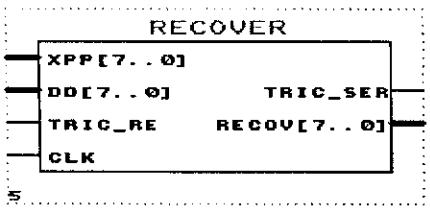
โดย ค่าสเต็ปไซต์ (ss) 00010000 มีค่าเป็น 16 จะได้

$$d(n) = (16*1) + (8*1) + (4*1) + (2)$$

$$= 30$$

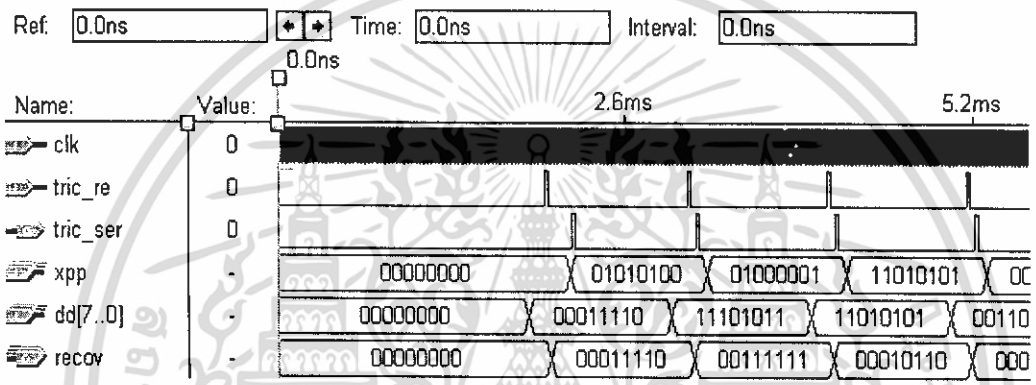
แปลงเป็น binary code จะได้เป็น 00011110 แสดงให้เห็นว่าการจำลองการทำงานดังรูปที่ 4.20 เป็นไปตามวิธีการถอดรหัสที่ได้คำนวณไว้

2) วงจรกู้สัญญาณกลับ



รูปที่ 4.21 แสดงสัญลักษณ์ของวงจรกู้สัญญาณกลับ

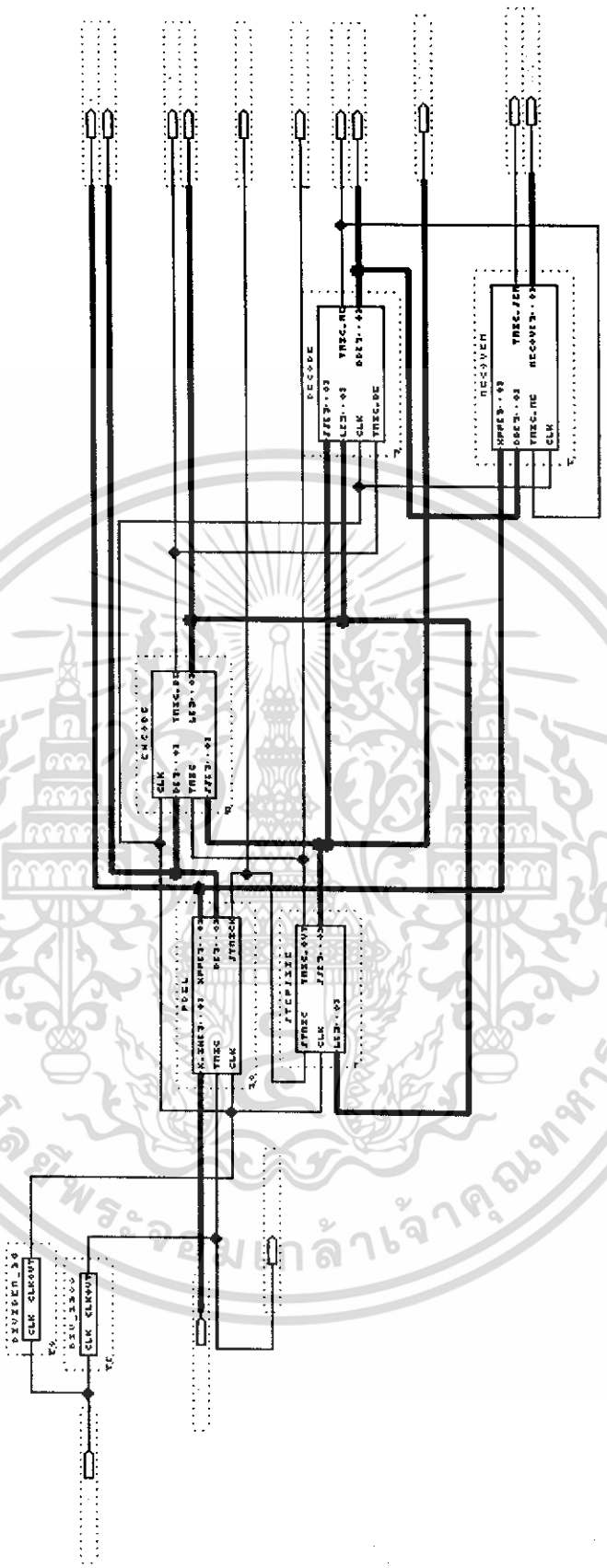
สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.22



รูปที่ 4.22 แสดงผลการจำลองการทำงานของวงจรกู้สัญญาณกลับ

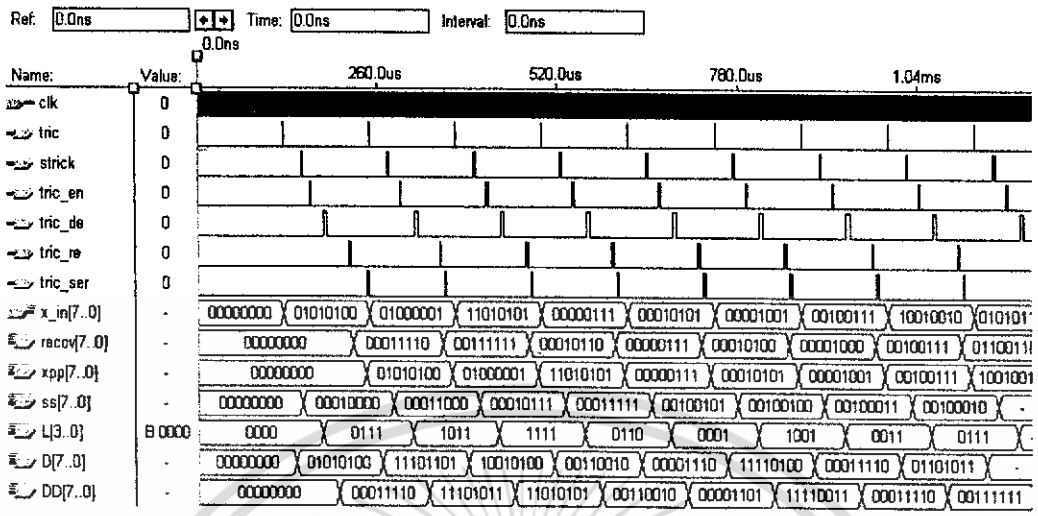
ค่าที่ได้จากการกู้สัญญาณกลับ (recov = xpp[7..0] + dd[7..0]) คือการนำค่าสัญญาณก่อนหน้า (xpp[7..0])รวมกับค่าผลต่าง (dd[7..0]) ที่ได้จากการถอดรหัสซึ่งเมื่อ ค่าสัญญาณก่อนหน้า (xpp[7..0]) มีค่าเป็น 01010100 และค่าผลต่าง (dd[7..0]) มีค่าเป็น 11101011 จะได้ค่าที่กู้สัญญาณกลับเป็น 00111111 ซึ่งจากการจำลองการทำงานดังรูปที่ 4.22 มีผลที่ได้เช่นเดียวกับที่ได้ออกแบบไว้

นำวงจรในแต่ละส่วนการทำงานมาต่อรวมกัน เป็นวงจรรวมดังรูปที่ 4.23



รูปที่ 4.23 แสดงวงจรรวม

สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.24

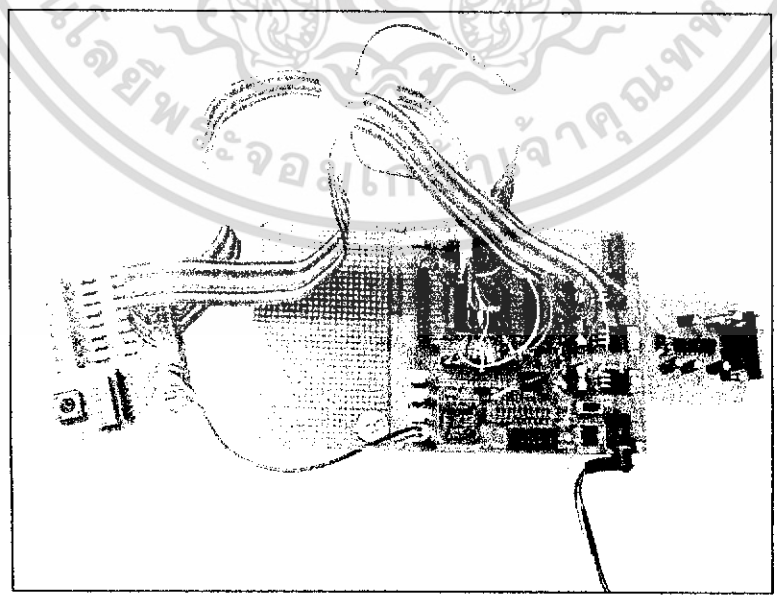


รูปที่ 4.24 แสดงผลการจำลองการทำงานของวงจรรวม

จากวงจรรวมเมื่อเราทำการเข้ารหัสสัญญาณและถอดรหัสสัญญาณ จะพบว่าค่าที่ได้นั้นจะมีค่าเท่ากับค่าเดิมหรือใกล้เคียงค่าเดิม เนื่องจากเป็นการเข้ารหัสเสียงแบบเอดีพีซีเอ็มซึ่งเป็นการบีบอัดสัญญาณที่มีการสูญเสียจึงทำให้ข้อมูลที่ได้มีค่าเท่ากับค่าเดิมหรือใกล้เคียงค่าเดิมได้

4.3 ผลการทำงานของวงจรในการเข้า-ถอดรหัสด้วยเอฟพีจีเอ (FPGA)โดยใช้มอนิเตอร์บอร์ด

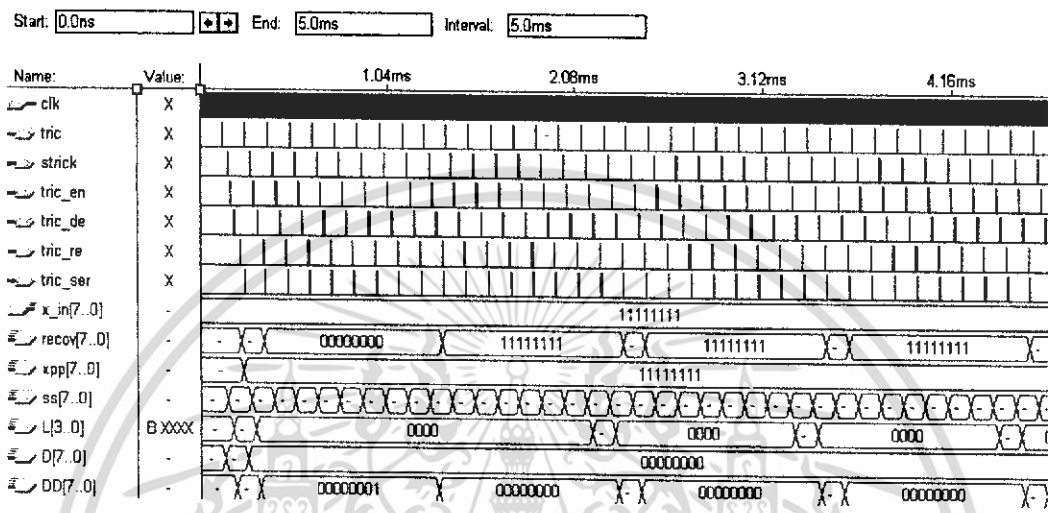
ทำการทดสอบการทำงานโดยการนำโปรแกรมที่ได้ทำการโปรแกรมลงบนบอร์ด FPGA แล้วทดสอบโดยการป้อนข้อมูลผ่านมอนิเตอร์บอร์ดและแสดงผลบนมอนิเตอร์บอร์ด



รูปที่ 4.25 แสดงบอร์ด FPGA

ผลการทดลองส่วนการทำงานบนบอร์ด FPGA

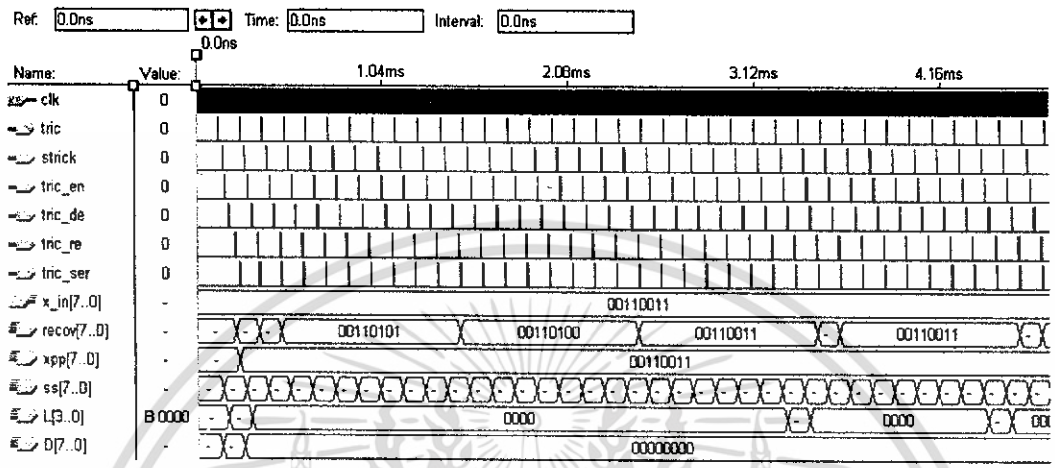
1) เมื่อทำการป้อนค่า 11111111 ผลที่ได้จากการจำลองการทำงาน ดังรูปที่ 4.26 จะมีค่าที่เป็นไปได้เป็น 11111111 หรือ 11111110 และเมื่อทำการทดลองโดยแสดงผลบนมอนิเตอร์บอร์ด จะได้เป็น 11111111 ดังรูปที่ 4.27 ผลที่ได้มีสัญญาณตรงตามที่ป้อนค่าซึ่งเป็นไปตามการจำลองการทำงาน



รูปที่ 4.26 แสดงผลการจำลองการทำงานเมื่อป้อนค่า 11111111

รูปที่ 4.27 แสดงผลเมื่อป้อนค่า 11111111

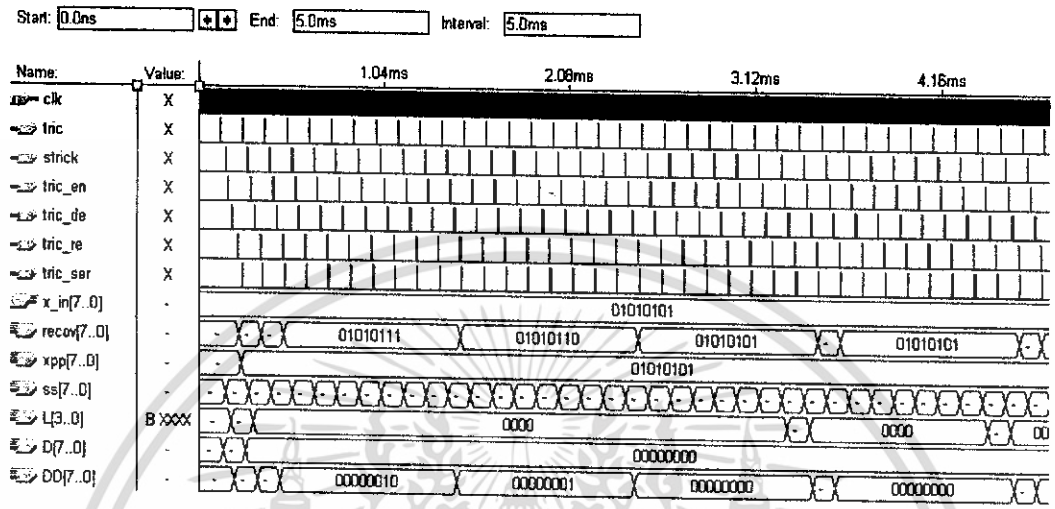
2) เมื่อทำการป้อนค่า 00110011 ผลที่ได้จากการจำลองการทำงานดังรูปที่ 4.28 ซึ่งมีค่าที่เป็นไปได้เป็น 00110101, 00110100 หรือ 00110011 และเมื่อทำการทดลองโดยแสดงผลบนมอนิเตอร์บอร์ดจะได้เป็น 00110011 ดังรูปที่ 4.29 ผลที่ได้มีสัญญาณตรงตามที่ป้อนค่าซึ่งเป็นไปตามการจำลองการทำงาน



รูปที่ 4.28 แสดงผลการจำลองการทำงานเมื่อป้อนค่า 00110011

รูปที่ 4.29 แสดงผลเมื่อป้อนค่า 00110011

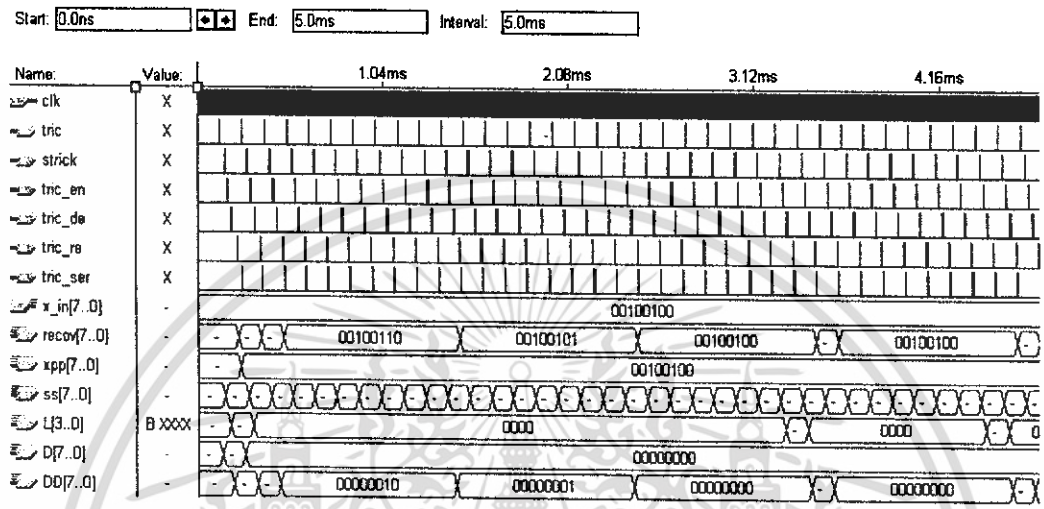
3) เมื่อทำการป้อนค่า 01010101 ผลที่ได้จากการจำลองการทำงานดังรูปที่ 4.30 ซึ่งมีค่าที่เป็นไปได้เป็น 01010111, 01010110 หรือ 01010101 และเมื่อทำการทดลองโดยแสดงผลบนมอนิเตอร์บอร์ดจะได้เป็น 01010101 ดังรูปที่ 4.31 ผลที่ได้มีสัญญาณตรงตามที่ป้อนค่าซึ่งเป็นไปตามการจำลองการทำงาน



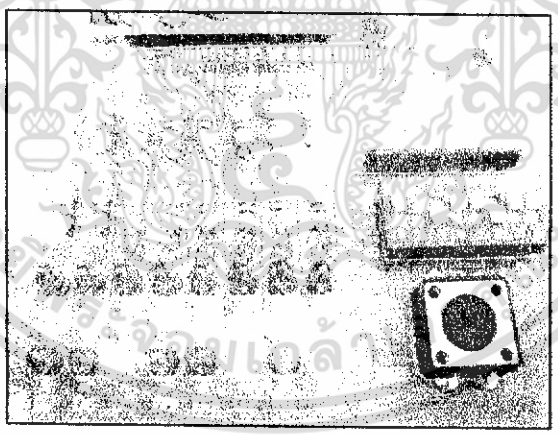
รูปที่ 4.30 แสดงผลการจำลองการทำงานเมื่อป้อนค่า 01010101

รูปที่ 4.31 แสดงผลเมื่อป้อนค่า 01010101

4) เมื่อทำการป้อนค่า 00100100 ผลที่ได้จากการจำลองการทำงานดังรูปที่ 4.32 ซึ่งมีค่าที่เป็นไปได้เป็น 00100110, 00100101 หรือ 00100100 และเมื่อทำการทดลองโดยแสดงผลบนมอนิเตอร์บอร์ดจะได้เป็น 00100101 ดังรูปที่ 4.33 ผลที่ได้มีค่าสัญญาณไม่ตรงตามที่ป้อนแต่ก็มีค่าใกล้เคียงกับผลการจำลองการทำงาน

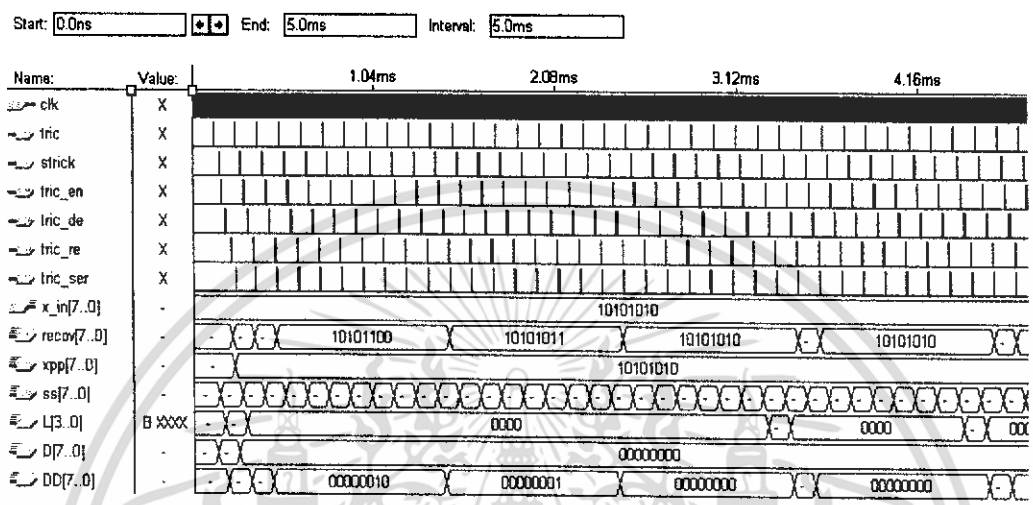


รูปที่ 4.32 แสดงผลการจำลองการทำงานเมื่อป้อนค่า 00100100



รูปที่ 4.33 แสดงผลเมื่อป้อนค่า 00100100

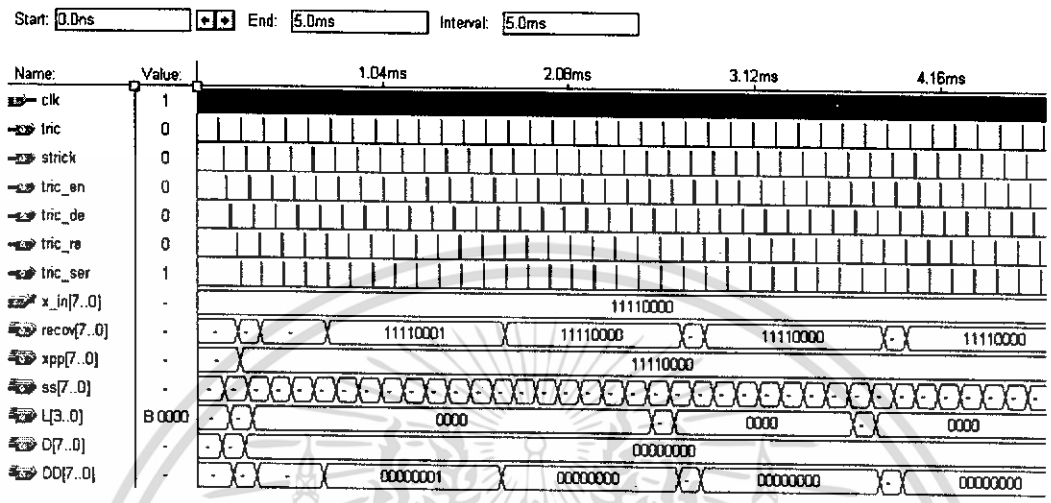
5) เมื่อทำการป้อนค่า 10101010 ผลที่ได้จากการจำลองการทำงานดังรูปที่ 4.34 ซึ่งมีค่าที่เป็นไปได้เป็น 10101100, 10101011 หรือ 10101010 และเมื่อทำการทดลองโดยแสดงผลบนมอนิเตอร์บอร์ดจะได้เป็น 10101011 ดังรูปที่ 4.35 ผลที่ได้มีค่าสัญญาณไม่ตรงตามที่ป้อนแต่ก็มีค่าใกล้เคียงกับผลการจำลองการทำงาน



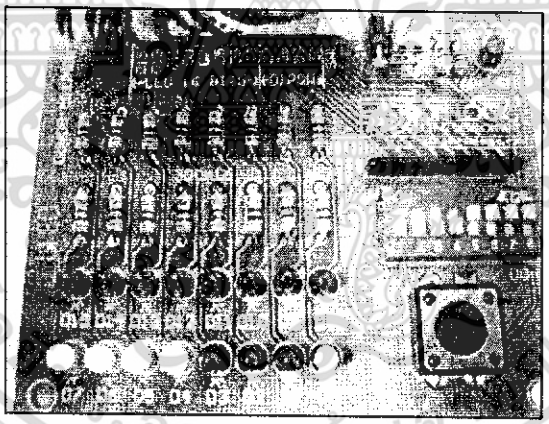
รูปที่ 4.34 แสดงผลการจำลองการทำงานเมื่อป้อนค่า 10101010

รูปที่ 4.35 แสดงผลเมื่อป้อนค่า 10101010

6) เมื่อทำการป้อนค่า 11110000 ผลที่ได้จากการจำลองการทำงาน ดังรูปที่ 4.36 จะมีค่าที่เป็นไปได้เป็น 11110001 หรือ 11110000 และเมื่อทำการทดลองโดยแสดงผลบนมอนิเตอร์บอร์ด จะได้เป็น 11110001 ดังรูปที่ 4.37 ผลที่ได้มีค่าสัญญาณไม่ตรงตามที่ป้อนแต่ก็มีค่าใกล้เคียงกับผลการจำลองการทำงาน



รูปที่ 4.36 แสดงผลการจำลองการทำงานเมื่อป้อนค่า 11110000

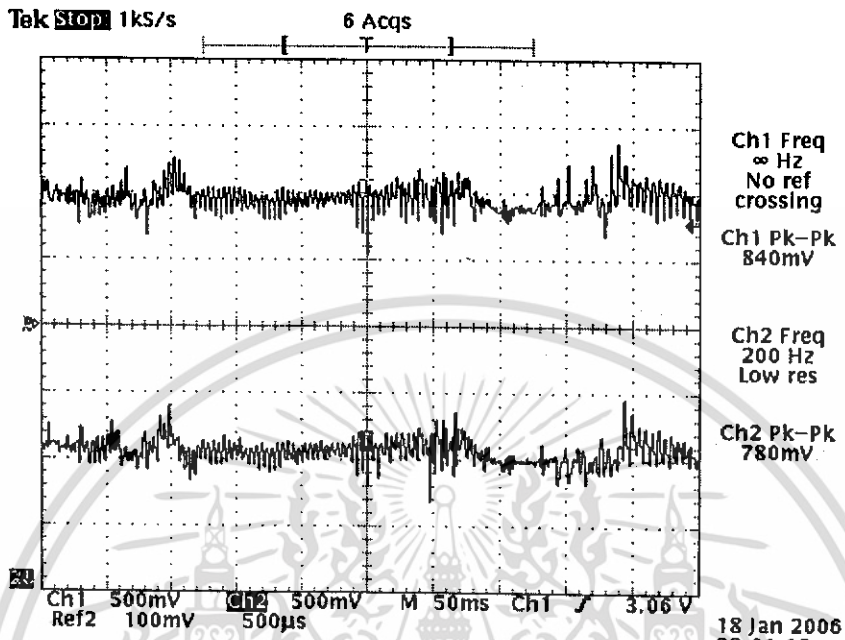


รูปที่ 4.37 แสดงผลเมื่อป้อนค่า 11110000

4.4 ผลการทำงานของวงจรในการเข้า-ถอดรหัสด้วยเอฟพีจีเอ (FPGA)

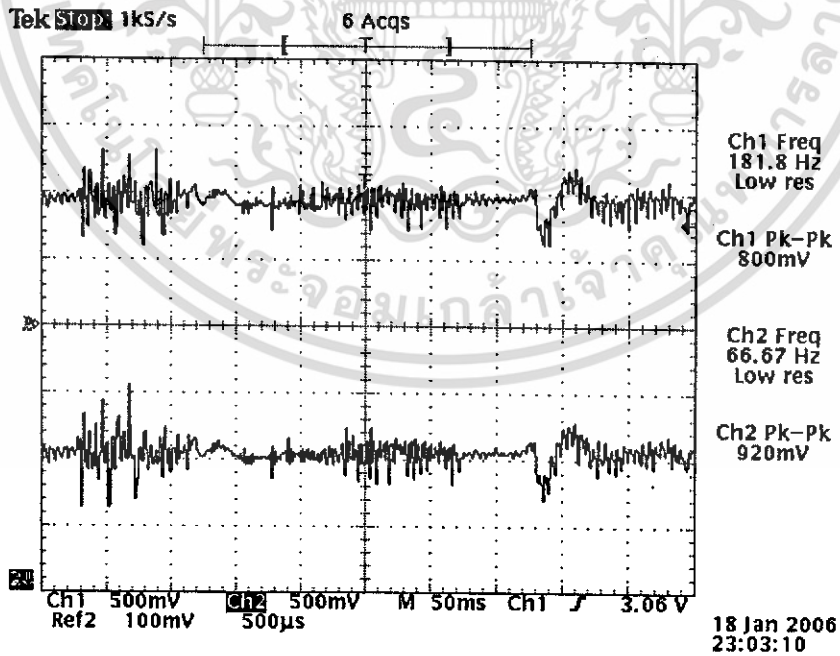
โดยแบ่งให้สัญญาณเสียงอินพุทที่เข้ามามีลักษณะแตกต่างกัน ดังนี้

1. สัญญาณเสียงของผู้ชายที่พูดคำว่า "การบีบอัดเสียงแบบเอดีพีซีเอ็ม"



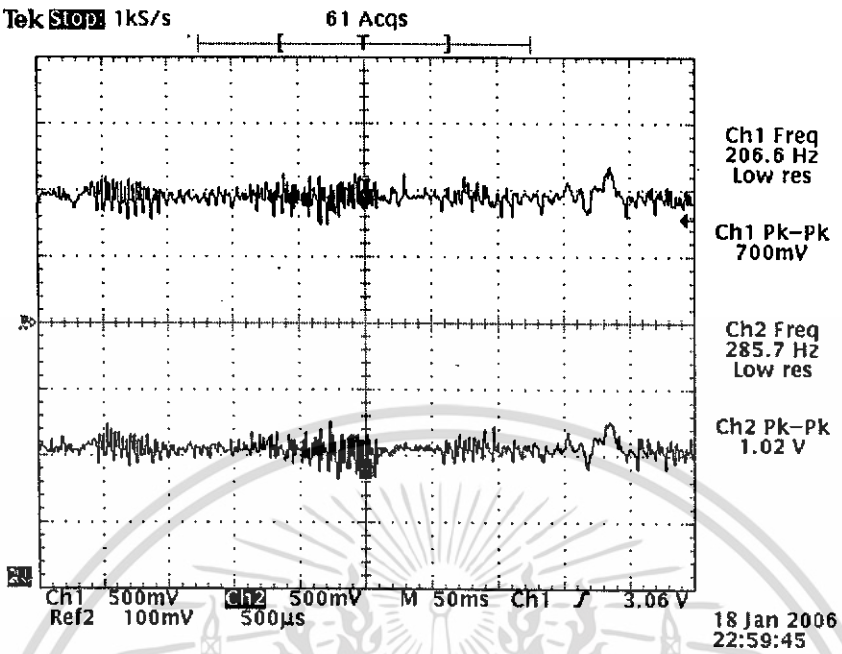
รูปที่ 4.38 แสดงสัญญาณอินพุทเทียบกับสัญญาณเอาท์พุตที่ผ่านการเข้า-ถอดรหัสด้วยเอฟพีจีเอ(FPGA)

2. สัญญาณเสียงของผู้หญิงที่พูดคำว่า "การบีบอัดเสียงแบบเอดีพีซีเอ็ม"



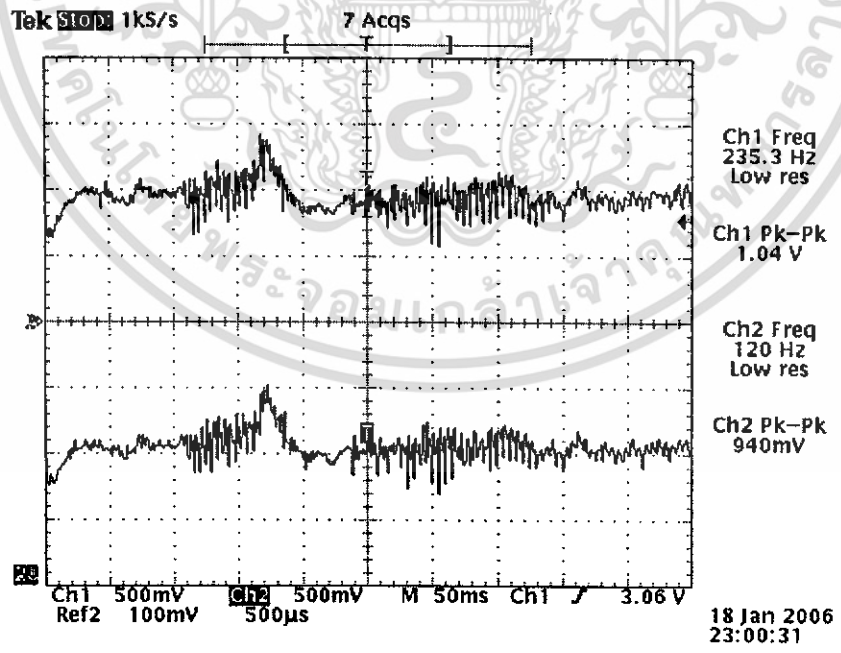
รูปที่ 4.39 แสดงสัญญาณอินพุทเทียบกับสัญญาณเอาท์พุตที่ผ่านการเข้า-ถอดรหัสด้วยเอฟพีจีเอ(FPGA)

3. สัญญาณเสียงของผู้ชายที่พูดภาษาอังกฤษ “Speech compression using ADPCM”



รูปที่ 4.40 แสดงสัญญาณอินพุทเทียบกับสัญญาณเอาท์พุตที่ผ่านการเข้า-ถอดรหัสด้วยเอฟพีจีเอ(FPGA)

4. สัญญาณเสียงของผู้หญิงที่พูดภาษาอังกฤษ “Speech compression using ADPCM”



รูปที่ 4.41 แสดงสัญญาณอินพุทเทียบกับสัญญาณเอาท์พุตที่ผ่านการเข้า-ถอดรหัสด้วยเอฟพีจีเอ(FPGA)

บทที่ 5
สรุป

จากที่ได้ทำการทดลองการเข้ารหัสแบบเอดีพีซีเอ็ม เมื่อทำการบีบอัดสัญญาณให้มีจำนวนบิตน้อยลงพบว่าคุณภาพของสัญญาณเสียงที่ได้จะแย่ลง ซึ่งเห็นได้จากค่าอัตราส่วนของสัญญาณต่อสัญญาณที่เกิดความผิดพลาด (SNR)

เมื่อเราบีบอัดข้อมูลจากความยาวแซมเปิ้ล 8 บิต ให้เหลือ 3, 4 และ 5 บิต ผลการทดลองเป็นดังตาราง

จำนวนบิตที่ได้จากการบีบอัด (บิต)	ค่า SNRของADPCM
3	12.2351
4	15.4848
5	15.7644

ตารางที่ 5.1 แสดงค่า SNR ที่ได้จากการบีบอัดสัญญาณเสียงของผู้ชายที่พูดคำว่า "การบีบอัดเสียงแบบเอดีพีซีเอ็ม"

จำนวนบิตที่ได้จากการบีบอัด (บิต)	ค่า SNRของADPCM
3	7.0881
4	12.4782
5	12.1016

ตารางที่ 5.2 แสดงค่า SNR ที่ได้จากการบีบอัดสัญญาณเสียงของผู้หญิงที่พูดคำว่า "การบีบอัดเสียงแบบเอดีพีซีเอ็ม"

จำนวนบิตที่ได้จากการบีบอัด (บิต)	ค่า SNRของADPCM
3	3.2304
4	13.5957
5	13.9578

ตารางที่ 5.3 แสดงค่า SNR ที่ได้จากการบีบอัดสัญญาณเสียงของผู้ชายที่พูดคำว่า "Speech compression using ADPCM"

จำนวนบิตที่ได้จากการบีบอัด (บิต)	ค่า SNRของADPCM
3	11.3204
4	19.8729
5	21.2631

ตารางที่ 5.4 แสดงค่า SNR ที่ได้จากการบีบอัดสัญญาณเสียงของผู้หญิงที่พูดคำว่า "Speech compression using ADPCM"

ซึ่งเห็นได้ว่าคุณภาพของสัญญาณที่ได้จากการบีบอัดให้เหลือ 5 บิตจะมีคุณภาพดีที่สุดและคุณภาพจากสัญญาณที่ได้จากการบีบอัดให้เหลือ 3 บิตจะมีคุณภาพแย่ที่สุด

และจากการเข้ารหัส - ถอดรหัสแบบเอดีพีซีเอ็มบนบอร์ด FPGA เมื่อทำการป้อนสัญญาณอินพุทเพื่อทำการเข้ารหัสและทำการถอดรหัส ค่าสัญญาณเอาท์พุทที่ได้มีค่าใกล้เคียงกับสัญญาณอินพุทซึ่งแสดงว่าการทำงานที่ได้ออกแบบไว้สามารถที่จะใช้ได้จริงในการบีบอัดสัญญาณเพื่อส่งสัญญาณด้วยจำนวนบิตที่ลดลงโดยได้สัญญาณเดิมกลับมามากที่สุด

ข้อได้เปรียบของระบบการเข้ารหัสสัญญาณแบบเอดีพีซีเอ็ม ที่เหนือกว่าระบบการเข้ารหัสสัญญาณแบบพีซีเอ็มและแบบดีพีซีเอ็ม คือ

1. สามารถลดจำนวนบิตที่ใช้ในการเข้ารหัสสัญญาณได้มากกว่า เนื่องจากใช้การเข้ารหัสจากค่าผลต่างของสัญญาณที่อยู่ข้างเคียงกันซึ่งมีค่าน้อย และมีการปรับขั้นระดับของการควอนไตซ์ตามขนาดของสัญญาณผลต่าง จึงสามารถลดจำนวนบิตที่ใช้เข้ารหัสลงได้มากกว่าระบบพีซีเอ็มและดีพีซีเอ็ม ซึ่งเท่ากับว่าเป็นการลดขนาดของข้อมูลลงด้วย ทำให้ประหยัดหน่วยความจำที่ใช้ในการเก็บข้อมูลลงได้

2. มีคุณสมบัติในการใช้สายส่งสัญญาณได้อย่างมีประสิทธิภาพมากกว่าระบบพีซีเอ็ม และระบบดีพีซีเอ็ม เนื่องจากอัตราเร็วที่ใช้ในการส่งสัญญาณจะลดลงเมื่อขนาดของข้อมูลลดลง จึงสามารถเพิ่มจำนวนของสัญญาณที่จะส่งไปในสายส่งได้มากขึ้น

เอกสารอ้างอิง

[1] ถวิล กิ่งทอง, "เทคโนโลยีการส่งสัญญาณดิจิตอล", กรุงเทพฯ: ตำราชุดวิศวกรรมศาสตร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

[2] มนัส สังวรศิลป์, วรรัตน์ ภัทรอมรกุล, "คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์", สำนักพิมพ์อินโฟเพรส พิมพ์ครั้งที่ 1 พ.ศ.2543

[3] วิวัฒน์ กิรานนท์, "วิศวกรรมสื่อสาร", สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

[4] CCITT Recommendation G.726, "CCITT 40, 32, 24, 16 kbps ADPCM"

[5] Kishan Shanoi, "Digital Signal Processing In Telecommunication" United State of America: Prentice-Hall International Editions

ภาคผนวก

โปรแกรมที่เขียนด้วยวีเอชดีแอล (VHDL)

โปรแกรม DIVIDER_1200

```
library ieee;
use ieee.std_logic_1164.all;
entity divider_1200 is
port(clk : in std_logic;
      clkout : out std_logic);
end divider_1200;
architecture rtl of divider_1200 is
signal count : integer range 0 to 1199;
begin
process (clk)
begin
if (clk'event and clk = '1') then
      if (count < 1199) then
      count <= count+1;
      clkout <= '0';
elsif(count <= 1199)then
            count <=0;
            clkout <= '1';
      end if;
end if;
end process;
end rtl;
```

โปรแกรม DIVIDER_24

```
library ieee;
use ieee.std_logic_1164.all;
entity divider_24 is
port(clk : in std_logic;
      clkout : out std_logic);
end divider_24;
architecture rtl of divider_24 is
signal count : integer range 0 to 23;
begin
process (clk)
begin
if (clk'event and clk = '1') then
      if (count < 1) then
      count <= count+1;
      clkout <= '1';
elsif(count <= 23)then
      count <= count+1;
      clkout <= '0';
      else
      count <= 0;
      end if;
end if;
end process;
end rtl;
```

โปรแกรม PDEL

```
library ieee;
use ieee.std_logic_1644.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
Entity pdel is
port(x_in  : in  std_logic_vector(7 downto 0);
       tric,clk : in  std_logic;
       xpp,d    : out std_logic_vector(7 downto 0);
       strick   : out std_logic);
end;
Architecture rtl of Pdel is
signal buff : std_logic_vector (7 downto 0);
begin
process (x_in,tric,clk)
           variable delay : integer range 0 to 50:=0;
           variable delay1,delay2 :integer range 0 to 50:=0;
           variable state : std_logic_vector(1 downto 0) :="00";
           variable x_pre,buff,xp,buf : std_logic_vector(7 downto 0)
:="00000000";
           variable dout : std_logic_vector(7 downto 0):="00000000";
begin
if clk'event and clk ='1' then
           case state is
           when "00"=>
                if tric = '1' then
                     state  := "01";
                     strick <= '0';
                end if;
           when "01"=>
                if delay = 5 then
                     x_pre  := buff;
                     buff   := x_in;
                     dout   := x_in - x_pre;
                     state  := "10";
                     else
                     delay := delay+1;
                end if;
           when "10"=>
                 d  <= dout;
                if delay1 = 3 then
                     strick <= '1';
                     state  := "11";
                else
                     strick <= '0';
                     delay1 := delay1+1;
                  end if;
           when "11"=>
                strick <= '0';
                if delay2 = 35 then
                          xp := buf;
                          buf := x_pre;
                          xpp <= xp;
                          delay  :=0;
                          delay1 :=0;
                          delay2 :=0;
                          state  := "00";
```

```
                    else
                              delay2 := delay2+1;
                              state  := "11";
                    end if;
                    when others =>
                          state  := "00";
              end case;
      end if;
end process;
end rtl;
```

โปรแกรม ENCODE

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity encode is
port( clk : in  std_logic;
        d  : in  std_logic_vector(7 downto 0);
        tric : in  std_logic;
        ss : in  std_logic_vector(7 downto 0);
        tric_de : out std_logic;
        L : out std_logic_vector(3 downto 0);
end encode ;
architecture con of encode IS
      signal sd,sh,shh :std_logic_vector(9 downto 0);
      signal inp1,inp2,inp_1,inp_a,inp_b:std_logic_vector(9 downto 0);
begin
      process (clk,d,ss)
            variable a,B3,B2,B1,B0:std_logic := '0';
            variable state : std_logic_vector (2 downto 0) :="000";
            variable delay,delay1:integer range 0 to 10;
      begin
            if clk'event and clk = '1' then
            case state is
                  when "000"=>
                  if tric = '1' then
                        tric_de <= '0';
                        inp1  <= d&"00";
                        inp2  <= ss&"00";
                        state := "001";
                  end if;
                  when "001"=>
                              a := inp1(9);
                        if    a = '1' then
                              sd <= (inp1 nor "000000000")+ '1';
                              B3 := '1' ;
                        else
                              B3 := '0';
                        end if;
                              state := "110";
                  when "110"=>
                        if B3 = '1' then
                           inp_1 <= sd;
                        else
                           inp_1 <= inp1;
                        end if;
                              state:= "010";
                  when "010"=>
                        if inp_1 >= inp2 then
                              B2 :='1';
                              inp_a  <= inp_1-inp2;
                        else
                              B2 :='0';
                              inp_a <= inp_1;
                        end if;
                              sh <= '0'&inp2(9 downto 1);
                              state := "011";
```

```
                when "011"=>
                      if inp_a >=sh then
                            B1 :='1';
                            inp_b <= inp_a-sh;

                      else
                            B1:='0';
                            inp_b<= inp_a;
                      end if;
                            shh <="00"&inp2(9 downto 2);
                            state  := "100";
                      when "100"=>
                      if inp_b >=shh then
                            B0 :='1';
                      else
                            B0 :='0';
                      end if;
                      state := "101";
                when "101"=>
                      L <= B3&B2&B1&B0;
                      state := "111";
                when "111"=>
                      tric_de <= '1';
                      if delay = 2  then
                            tric_de <= '0';
                            delay := 0;
                            state := "000";
                      else
                          delay := delay+1;
                      end if;
                when others =>
                      state:= "000";
                end case;
end if;
end process;
end;
```

โปรแกรม STEPSIZE

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
Entity stepsize is
port(stric,clk  : in  std_logic;
       L : in  std_logic_vector(3 downto 0);
       tric_out  : out std_logic;
       ss: out std_logic_vector(7 downto 0);
end stepsize;
Architecture size of stepsize is
      signal M :std_logic_vector(7 downto 0);
      signal sn :std_logic_vector(7 downto 0);
      signal buf_1:std_logic_vector(7 downto 0);
      signal Lo:std_logic_vector(3 downto 0);
      begin
      process(L,clk)
          variable state:std_logic_vector(2 downto 0):="000";
            variable tric_p  : integer range 0 to 2000 :=0;
            variable tric_pp : integer range 0 to 2000 :=0;
            variable delay   : integer range 0 to 10 :=0;
            variable delay1  : integer range 0 to 10 :=0;
      begin
      if clk'event and clk ='1' then
          if L(2)= '0' then
                  M <= "11111111";
                  tric_out <= '0';
      else
                  tric_out <= '0';
              if   L(1)='1' then
                 if L(0)='1' then
                  M <= "00001000";
                 else
                  M <= "00000110";
                 end if;
                  else
                 if L(0)='1' then
                  M <= "00000100";
                 else
                  M <= "00000010";
                 end if;
              end if;
          end if;
          case state is
          when "000"=>
                if stric = '1' then
                     tric_p := tric_p+1;
                     if tric_p = 1 then
                          if   L(2)='0' then
                               M <= "00000000";
                               tric_out <= '0';
                          end if;
                               sn <= "00010000";
                               ss <= "00010000";
                               tric_out <= '0';
                               state := "100";
```

```
                    else
                                tric_out <= '0';
                                state    := "001";
                    end if;
                    tric_out <= '0';
              end if;
        when "001"=>
                    buf_1<= sn+M;
                    tric_out <= '0';
                    state :="010";
        when "010"=>
              sn <= buf_1;
              ss <= buf_1;
              tric_out <= '0';
              state :="110";
        when "100"=>
              if delay = 3 then
                    tric_out <= '1';
                    state  := "000";
                    delay  := 0;
              else
                    tric_out <= '0';
                    delay := delay+1;
              end if;
        when "110"=>
              if delay1=3 then
                    tric_out <= '1';
                    state := "000";
                    delay1 :=0;
              else
                    tric_out <= '0';
                    delay1 := delay1+1;
              end if;
        when others =>
              state :="000";
  end case;
  end if;
end process;
end size;
```

โปรแกรม DECODE

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity decode is
port(ss : in std_logic_vector(7 downto 0);
        L : in std_logic_vector(3 downto 0);
        clk,tric_de: in std_logic;
        tric_re   : out std_logic;
        dd : out std_logic_vector(7 downto 0);
end decode;
Architecture cod of decode is
      signal  s,st,st_1,st_2,st_3,d_1: std_logic_vector(9 downto 0);
      signal  d_2: std_logic_vector(7 downto 0);
begin
      process(ss,L,clk)
      variable state : std_logic_vector (3 downto 0) :="0000";
      variable delay : integer range 0 to 10;
      begin
            if clk'event and clk ='1' then
            case state is
            when "0000"=>
                  tric_re <= '0';
                  if  tric_de  = '1' then
                        s <= ss&"00";
                        state := "0001";
                  end if;
            when "0001"=>
                  if L(2)='1' then
                        st<= s;
                  else
                        st<= "0000000000";
                  end if;
            state:= "0010";
            when "0010"=>
                  if L(1)= '1' then
                        st_1<= "0"&s(9 downto 1);
                  else
                        st_1<= "0000000000";
                  end if;
                  state:= "0011";
            when "0011"=>
                  if L(0)= '1' then
                        st_2<= "00"&s(9 downto 2);
                  else
                        st_2<= "0000000000";
                  end if;
                  state:= "0110";
            when "0110"=>
                  st_3  <= "000"& s(9 downto 3);
                  state:= "0100";
            when "0100"=>
                  d_1<= st+st_1+st_2+st_3;
                  state:="0101";
```

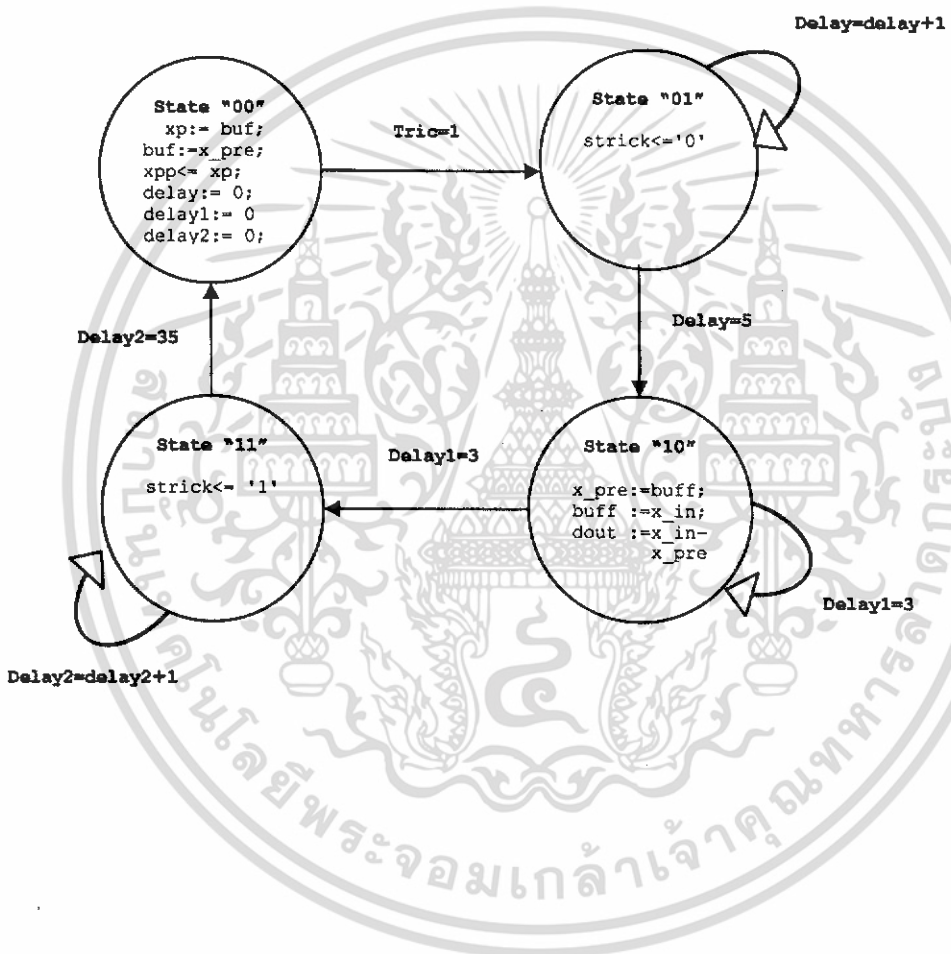
```
            when "0101"=>
                  d_2<= d_2(9 downto 2);
                  state:= "1000";
            when "1000"=>
                  if L(3)='1' then
                        dd <= (d_2 nor "00000000")+'1';
                  else
                        dd <= d_2;
                  end if;
                  if    delay = 5 then
                        state := "0111";
                        delay := 0 ;
                  else
                        delay := delay+1;
                  end if;
            when "0111"=>
                  tric_re <= '1';
                  state := "0000";
            when others =>
                  state := "0000";
            end case;
            end if;
end process;
end cod;
```

โปรแกรม RECOVER

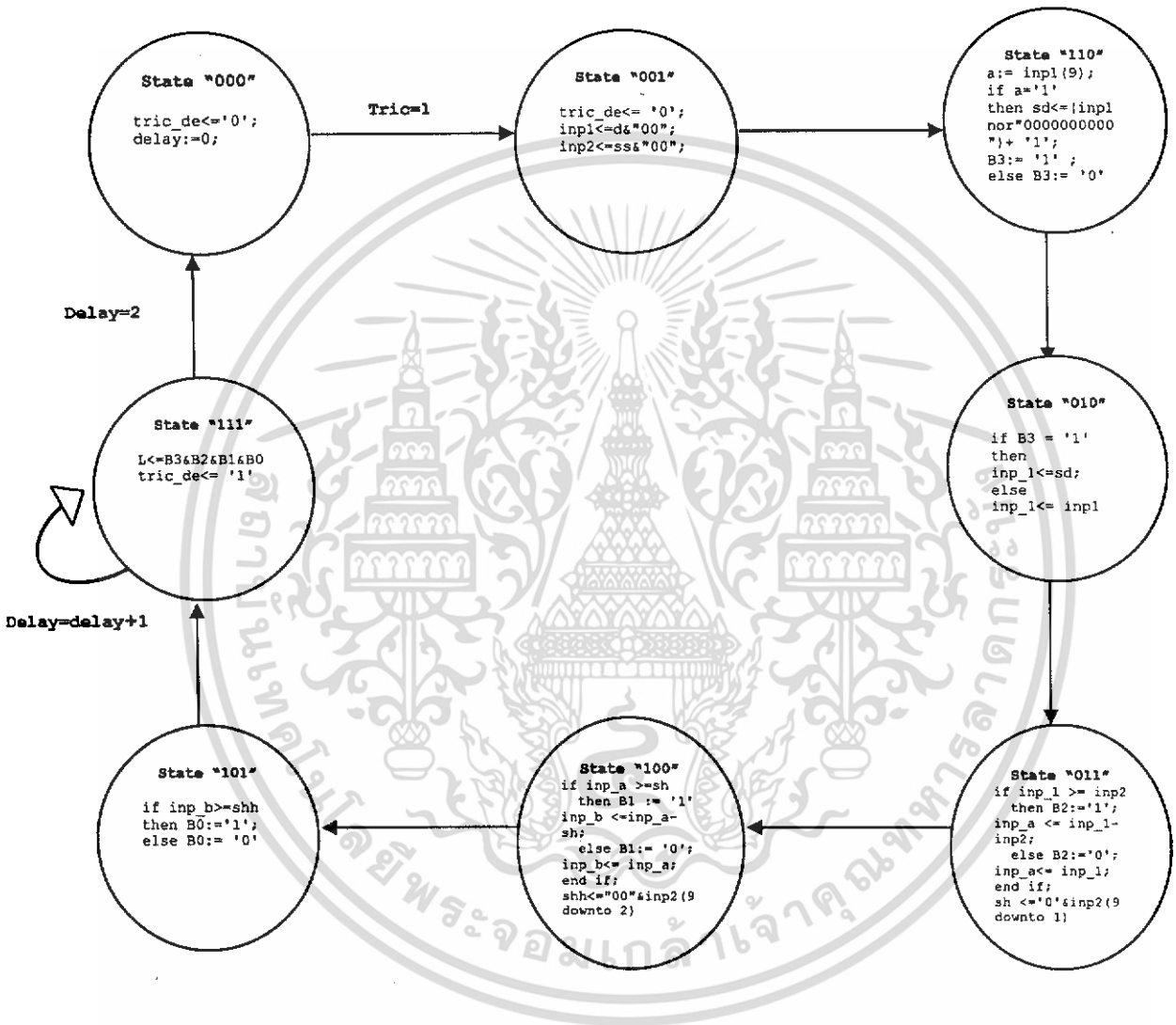
```
library ieee;
use ieee.std_logic_1164.ALL;
uieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity recover is
port( xpp  : in std_logic_vector(7 downto 0);
             dd : in std_logic_vector(7 downto 0);
             tric_re,clk : in  std_logic;
             tric_ser : out std_logic;
             recov : out std_logic_vector(7 downto 0));
end recover;
Architecture rtl of recover is
                signal d_1: std_logic_vector(7 downto 0);
                signal x_p: std_logic_vector(7 downto 0);
           begin
                process(xpp,dd,tric_re,clk)
                variable state : std_logic_vector(1 downto
0):="00";
                variable delay : integer range 0 to 20;
           begin
                if clk'event and clk = '1' then
                case state is
                when "00" =>
                     tric_ser <= '0';
                if  tric_re = '1' then
                     d_1 <= dd;
                     x_p <= xpp;
                     state := "01";
                end if;
                when "01" =>
                   recov <= d_1+x_p;
                     if  delay   = 7 then
                          state := "10";
                     else
                          delay := delay+1 ;
                     end if;
                when "10" =>
                     tric_ser <= '1';
                     state :="00";
                when others =>
                state  := "00";
           end case;
           end if;
           end process;
end rtl;
```

State diagram

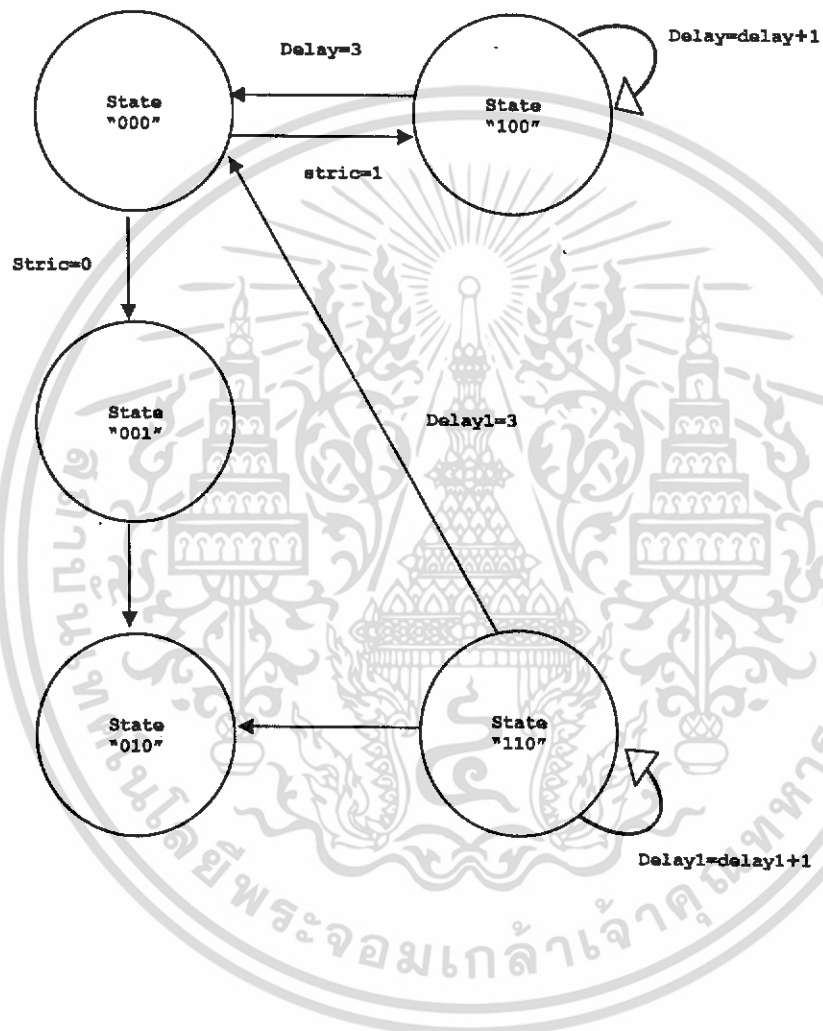
State diagram ของวงจรผลต่างและค่าสัญญาณก่อนหน้า



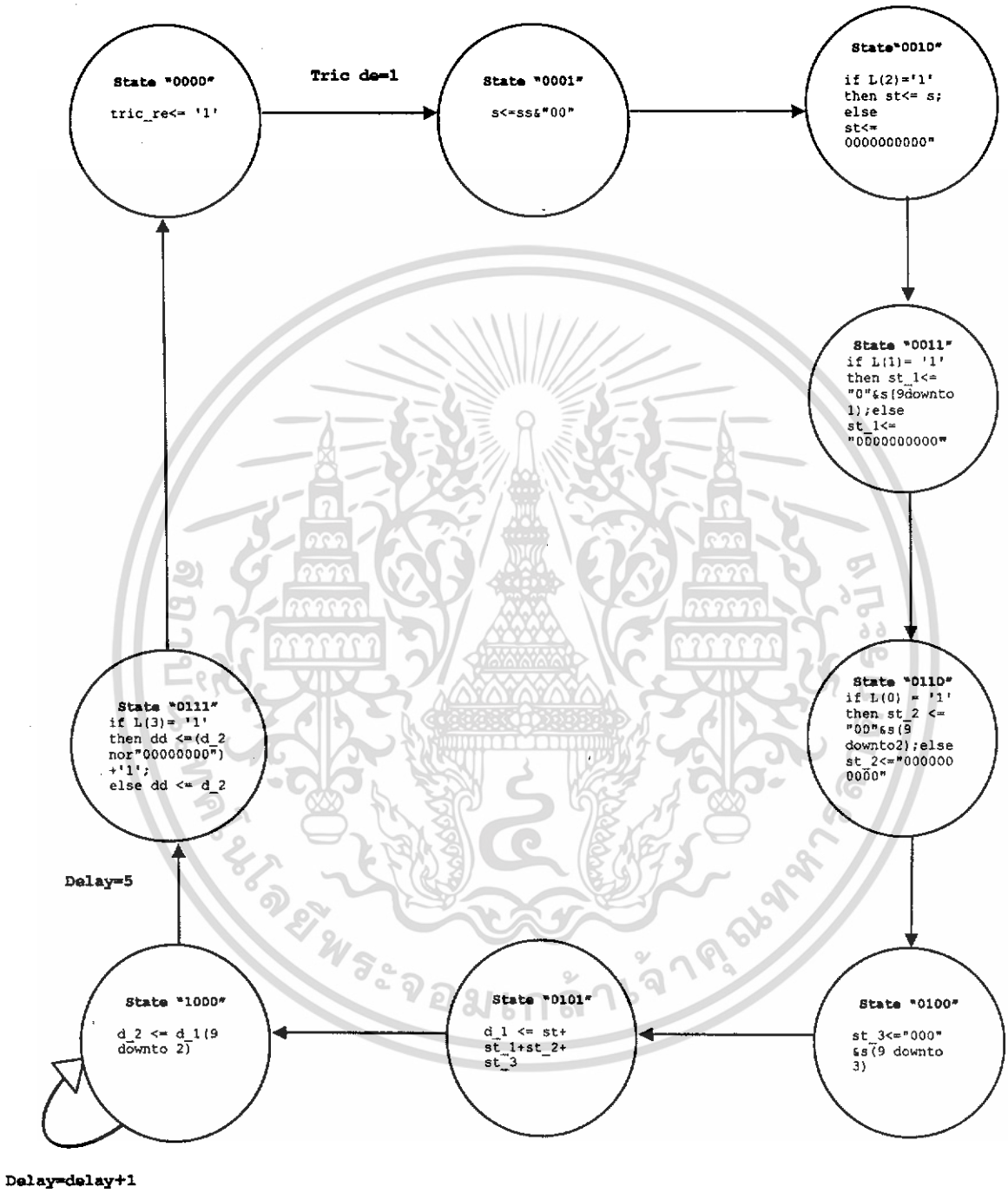
State diagram ของวงจรเข้ารหัส



State diagram ของวงจรจัดระดับสัญญาณ



State diagram ของวงจรถอดรหัส



สารบัญรูปภาพ

หน้า

หน้า

หน้า

สารบัญตาราง