การเข้ารหัสเสียงโดยวิธีเอดีพีซีเอ็ม

Speech Compression Using ADPCM

โดย

นายธีรพงษ์ เจนวิริยะกุล

นางสาวนรินทร์ทิพย์ เชาว์ทอง

นางสาวชญาดา ชั้นบุญใส

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การเข้ารหัสเสียงโดยวิธีเอดีพีซีเอ็ม

Speech Compression Using ADPCM

โดย

นายธีรพงษ์ เจนวิริยะกุล 44010220

นางสาวนรินทร์ทิพย์ เชาว์ทอง 44010242

นางสาวชญาดา ชั้นบุญใส 44010248

อาจารย์ที่ปรึกษา

รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี

รศ.ดร.ปัญญา ฐิติมัชฌิมา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การเข้ารหัสเสียงโดยวิธีเอดีพีซีเอ็ม**

Speech Compression Using ADPCM

ผู้จัดทำ

1. นายธีรพงษ์ เจนวิริยะกุล 44010220
2. นางสาวนรินทร์ทิพย์ เชาว์ทอง 44010242
3. นางสาวชญาดา ชั้นบุญใส 44010248

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ยุทธพงษ์ รังสรรค์เสรี)

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ปัญญา ฐิติมัชฌิมา)

การเข้ารหัสเสียงโดยวิธี ADPCM

Speech Compression Using ADPCM

โดย นายธีรพงษ์ เจนวิริยะกุล 44010220

นางสาวนรินทร์ทิพย์ เชาว์ทอง 44010242

นางสาวชญาดา ชั้นบุญใส 44010248

อาจารย์ที่ปรึกษา รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี

รศ.ดร.ปัญญา ฐิติมัชฌิมา

บทคัดย่อ

โครงงานนี้มีวัตถุประสงค์เพื่อทำการศึกษาและสร้างตัวเข้ารหัสสัญญาณเสียงโดยวิธี Adaptive Differential Pulse Code Modulation (ADPCM) ซึ่งใช้การจัดระดับกับค่าผลต่างระหว่างแซมเปิ้ลที่อยู่ติดกัน ทำให้สามารถเข้ารหัสได้โดยใช้จำนวนบิตที่ลดลง ซึ่งในที่นี้จะใช้การเขียนโปรแกรมในการเข้ารหัสและสร้างตัวเข้ารหัสเพื่อทดสอบว่าโปรแกรมนั้นสามารถใช้งานได้จริง

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. We use program to encode and construct the coder to test this program

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

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

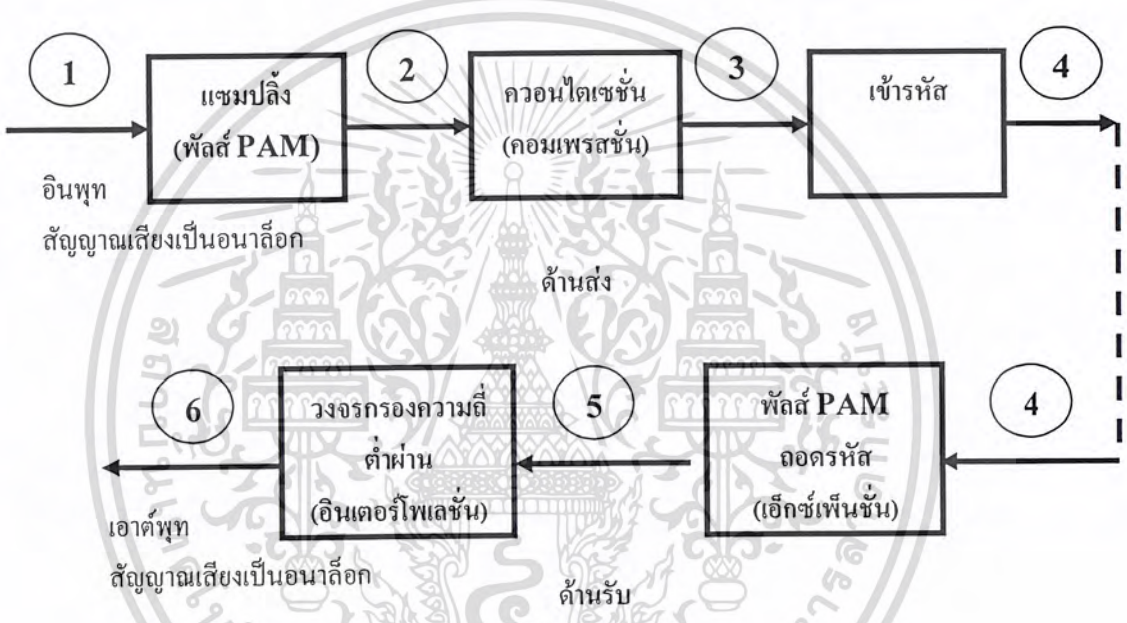
วิธีที่ใช้กันอย่างแพร่หลายก็คือ พัลส์โค้ดมอดูเลชั่น (Pulse Code Modulation: PCM) ซึ่งถูกนำไปใช้อย่างกว้างขวางโดยเฉพาะกับสัญญาณเสียง แต่ยังมีการเข้ารหัสอีกแบบหนึ่งที่ทำการส่งด้วยรหัสที่น้อยกว่า 8 บิตคือการเข้ารหัสสัญญาณแบบอแด็พทีฟว์ดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น (Adaptive Pulse Code Modulation: ADPCM)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

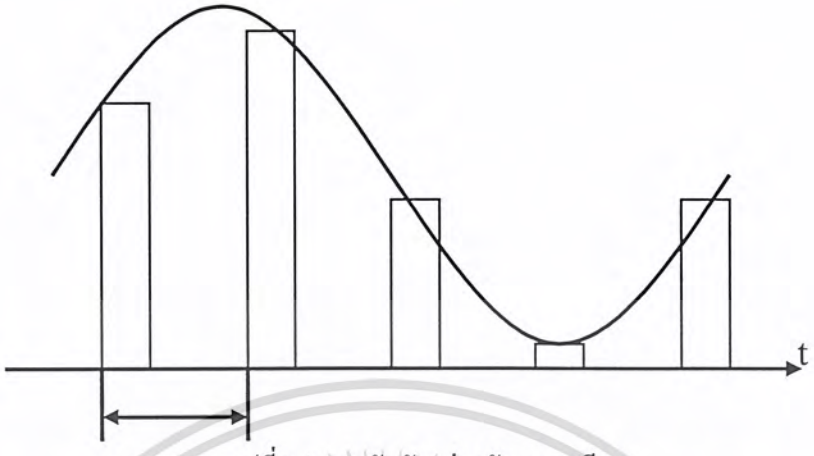


รูปที่ 2.1 กระบวนการเข้ารหัสและถอดรหัสของระบบพีซีเอ็ม

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

ในการส่งโดยระบบพีซีเอ็มนั้นต้องส่งด้วยจำนวนบิต 8 บิต แต่ยังมีการเข้ารหัสอีกแบบหนึ่งที่ทำการส่งด้วยรหัสน้อยกว่า 8 บิตคือ การเข้ารหัสเสียงแบบอแด๊พทิฟว์ดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น ซึ่งมีข้อได้เปรียบกว่าระบบพีซีเอ็มหลายประการ ดังจะกล่าวต่อไป

2.2 การชักตัวอย่าง (Sampling)

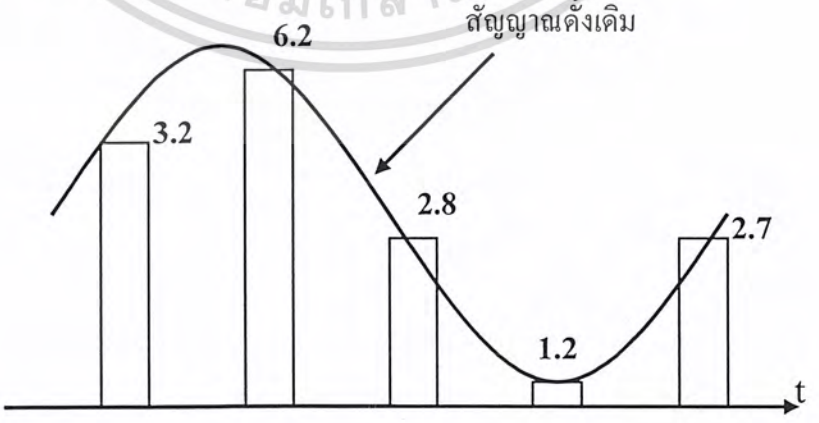


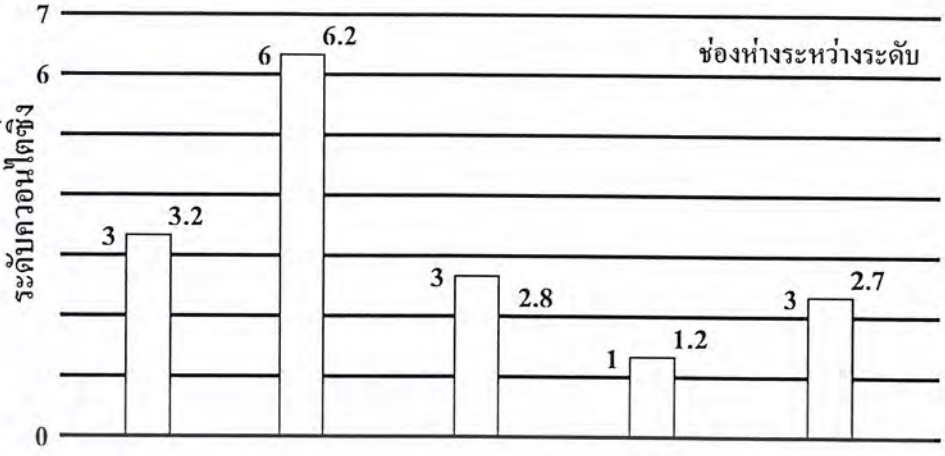
รูปที่ 2.2 การชักตัวอย่างสัญญาณเสียง

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

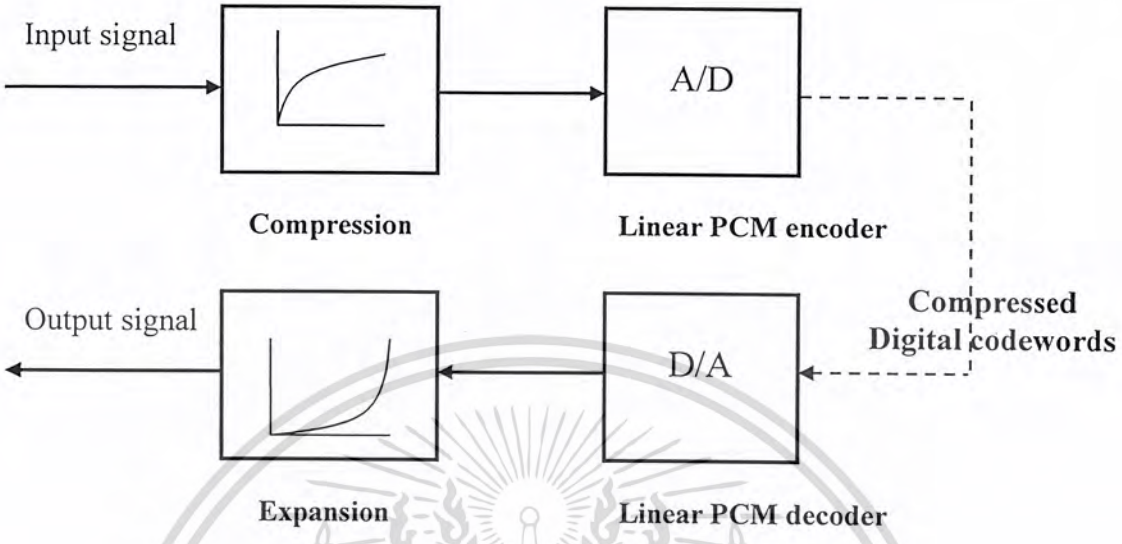
จากรูปที่ 2.3 แอมปลิจูดของการชักตัวอย่างทุกตัวของพีเอเอ็มจะถูกจัดให้เป็นระดับซึ่งเรียกว่า ระดับการควอนไตซ์ (quantize level) โดยมีระยะห่างระหว่างระดับข้างเคียง เรียกว่า ควอนไตซ์ซิ่งอินเทอร์วัล (quantizing interval) เท่ากัน กรณีเรียกว่าเป็นการจัดแบบยูนิฟอร์ม (uniform quantizing) ขนาดของแซมเปิ้ลทุกตัวจะแสดงด้วยค่าระดับการควอนไตซ์ที่ใกล้เคียงที่สุด 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 แต่เราสามารถทำให้มันลดลงได้โดยการควอนไตซ์ซิ่งอินเทอร์วัลหรือการเพิ่มจำนวนระดับนั่นเอง แต่เมื่อจำนวนระดับเพิ่มขึ้นแล้วจำนวนบิทที่ใช้ก็เพิ่มขึ้นด้วย จึงจำเป็นต้องใช้ความเร็วในการส่งสัญญาณดิจิตอลสูงขึ้น ตามปกติควอนไตซ์ซิ่งนอยส์จะเกิดขึ้นสม่ำเสมอในทุกอินเทอร์วัลโดยไม่เกี่ยวข้องกับแอมปลิจูดของสัญญาณเดิม ในกรณีที่สัญญาณมีระดับสูง คุณภาพของการเข้ารหัสของสัญญาณเสียงจะดีกว่าของสัญญาณซึ่งมีระดับต่ำ ดังนั้นจึงจำเป็นต้องพิจารณาควอนไตซ์ซิ่งนอยส์ในบริเวณที่สัญญาณมีระดับต่ำ การจัดระดับแบบนอนยูนิฟอร์ม (Non-uniform quantizing) คือบริเวณที่สัญญาณมีแอมปลิจูดต่ำจะควอนไตซ์ซิ่งอินเทอร์วัลแคบๆ

และในทางตรงกันข้ามบริเวณที่สัญญาณมีแอมปลิจูดสูงจะใช้ควอนไตซ์ซึ่งอินเทอร์วัลกว้างๆ ซึ่งการทำให้เป็นแบบนอนยูนิฟอร์มนั้นจะใช้หลักการคอมแพนดิ้งเข้าช่วย

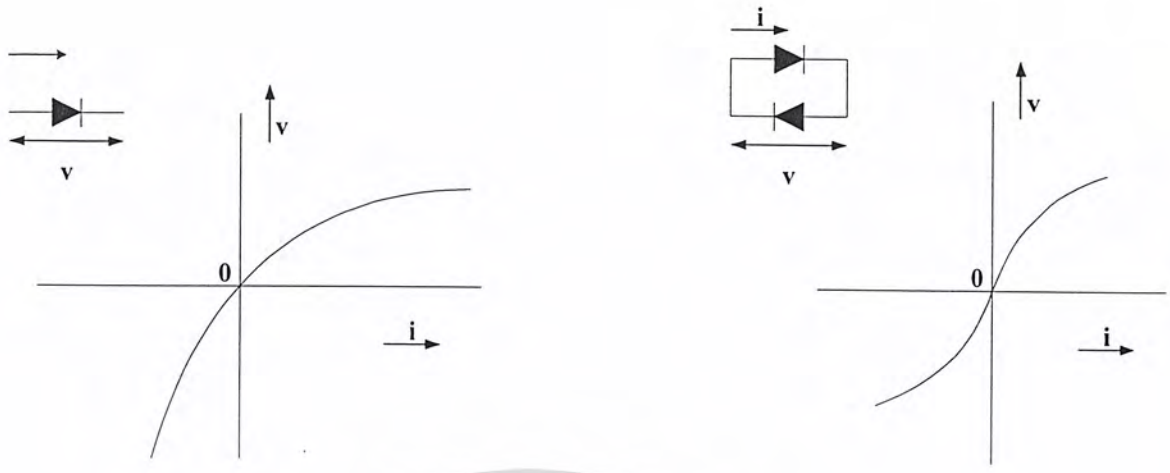


รูปที่ 2.4 การแทรกขั้นตอนของการอัดสัญญาณและการยืดสัญญาณลงในระบบพีซีเอ็ม

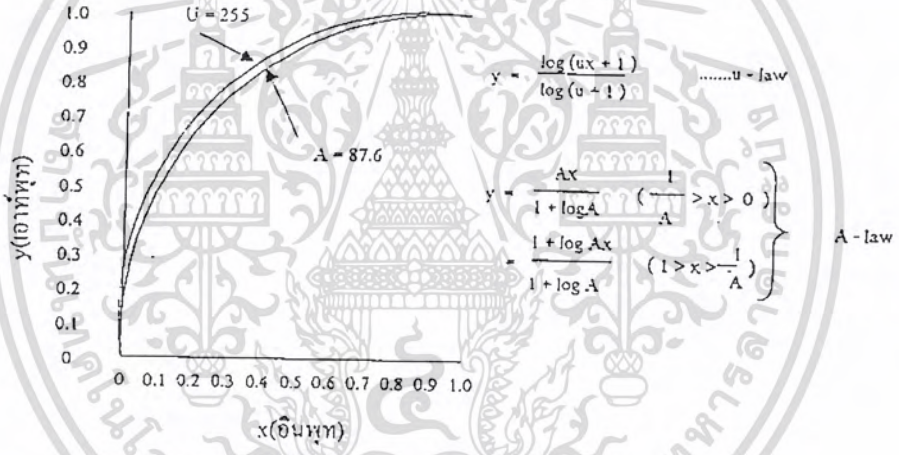
การทำคอมแพนดิ้งเมื่อนำมาใช้ในระบบพีซีเอ็ม จะทำให้ประสิทธิภาพในการใช้จำนวนบิทในการเข้ารหัสสูงขึ้น กรณีที่จัดระดับแบบยูนิฟอร์มนั้นจะใช้ประมาณ 2,000 ระดับ จึงจะสามารถรักษาระดับคุณภาพของเสียงให้ดี ในการเข้ารหัสต้องใช้ถึง 11 บิตต่อแซมเปิ้ล 1 แต่ถ้าใช้แบบนอนยูนิฟอร์มแล้วจะใช้เพียง 7 บิต ซึ่งมีระดับเพียง 128 ระดับเท่านั้น ก็เพียงพอที่จะทำให้อัตราส่วนของสัญญาณต่อควอนไตซ์ซึ่งนอยส์ใกล้เคียงกับการจัดระดับแบบยูนิฟอร์ม CCITT กำหนดให้ใช้ 8 บิตต่อแซมเปิ้ล 1 ตัว และระดับการควอนไตซ์เท่ากับ 256 ระดับ ก็จะเป็นการรับรองว่าเสียงพูดจะมีคุณภาพดี

ลักษณะของการคอมเพรสเซอร์นั้นจะเป็นแบบลอการิทึม รูปแบบโดยทั่วไปจะใช้คุณสมบัติของ V-I ของไดโอด ตามรูปที่ 2.5 กรณีที่ใช้คอมเพรสเซอร์จะมีกระแส I เป็นอินพุท โวลเตจ V เป็นเอาต์พุท สำหรับกรณีที่ใช้เป็นเอกซ์แพนเดอร์จะมีโวลเตจเป็นอินพุท และกระแสเป็นเอาต์พุท

คุณลักษณะของคอมเพรสเซอร์ที่ใช้สำหรับประกอบการเข้ารหัสสัญญาณเสียงในปัจจุบันคือ μ-law ซึ่งใช้ในทวีปอเมริกาเหนือและญี่ปุ่นเป็นหลัก โดยที่ค่า μ ที่ใช้คือ 255 สำหรับในทวีปยุโรปและประเทศไทยใช้ A-law ซึ่งโดยหลักการแล้วจะเหมือนกับกฎ μ-law แต่จะแตกต่างกันรายละเอียด 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 จะใช้วงจรคอมเพรสเซอร์ตามรูปที่ 2.5 แต่กรณีที่ μ=255 และ A=87.6 จะใช้วงจรคอมเพรสเซอร์ที่มีคุณลักษณะเป็นเส้นตรง โดยแยกเป็นส่วนๆเรียกว่า เซกเมนต์ (Segment)

2.5 การเข้ารหัสแบบดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น (Differential pulse code modulation: DPCM)

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

2.5.1อัลกอริทึมเบื้องต้น (The Basic Algorithm)

ในการทำนายอย่างง่ายเราจะทำนายอย่างง่าย เราจะทำนายว่า ตัวอย่างถัดไปจะเท่ากับตัวอย่างปัจจุบัน

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

X_{n-1} คือตัวอย่างที่ n-1

แม้ว่าในการเข้าโค้ดของผลต่างจะใช้จำนวนบิตน้อยกว่าเดิม (PCM) แต่เรากล่าวมาได้ว่ามันสามารถครอบคลุมได้ในเรื่องของการถอดรหัสที่ยอมรับได้ของ ลำดับสัญญาณจากค่าแตกต่างที่ถูกควอนไตซ์ (Quantize) แล้ว

ถ้าเราสนใจกับการบีบอัดข้อมูลแบบไม่มีการสูญเสียข้อมูล (Lossless Compression) เราจะพบว่าถ้าทำการเข้าโค้ดความแตกต่างระหว่างตัวอย่าง เราจะสามารถคลอบคลุมลำดับเดิมได้อย่างไม่สูญเสีย ตัวอย่างเช่น ลำดับต่อไปนี้

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) สมมุติว่าเรามีตัว ควอนไตซ์เจ็ดระดับ ด้วยค่าเอาท์พุท -6,-4,-2, 0,2,4,6 จะได้ลำดับที่ถูกควอนไทซ์ 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) ในการพิสูจน์นี้สามารถพิสูจน์ได้ด้วยสมการ พิจารณาลำดับตัวอย่าง $\{X_n\}$ ลำดับผลต่าง $\{d_n\}$ ซึ่งเกิดจาก $X_n - X_{n-1}$ ซึ่งลำดับผลต่างนี้ถูกควอนไทซ์ได้ $\{\hat{d}_n\}$

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

โดย q_n คือ ค่าความผิดพลาดที่เกิดจากการควอนไทซ์ (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 จะได้

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

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

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

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

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

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

$$= X_2 + q_1 + q_2 \tag{2.9}$$

เมื่อกระบวนการนี้มีต่อเรื่อยๆ ไปจนถึงลำดับที่ n จะได้ว่า

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

จะพบว่าเมื่อลำดับมากขึ้นค่าความผิดพลาดจะมากขึ้น ก็เนื่องจาก $\sum_{k=1}^{n} q_k$

จะมีค่าสะสมมากขึ้นเรื่อยๆ ปัญหานี้จะแก้ไขโดยการแทน $d_n = X_n - \hat{X}_{n-1}$ ด้วย $\hat{X}_0 = X_0$ จะได้

$$d_1 = X_1 - X_0 \tag{2.11}$$

$$\hat{d}_1 = Q[d_1] = d_1 + q_1 \tag{2.12}$$

$$\hat{X}_1 = X_0 + \hat{d} = X_0 + d_1 + q_1 = X_1 + q_1 \tag{2.13}$$

$$d_2 = X_2 - \hat{X}_1 \tag{2.14}$$

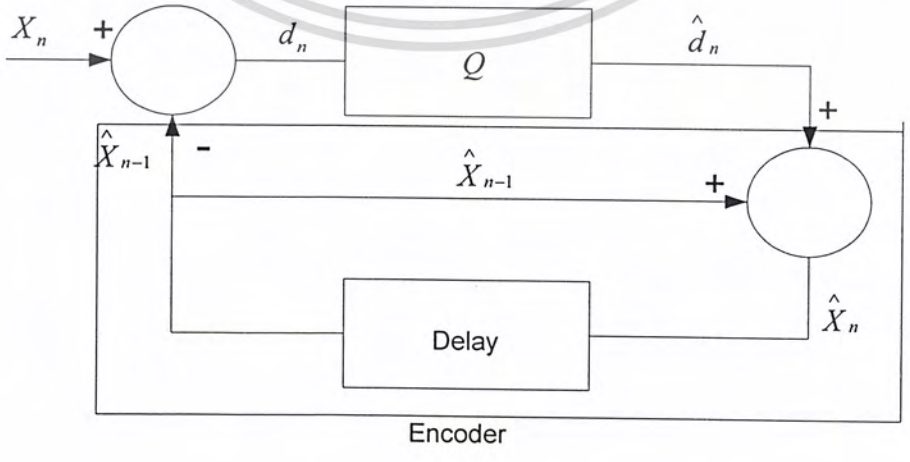
$$\hat{d}_2 = Q[d_2] = d_2 + q_2 \tag{2.15}$$

$$\hat{X}_2 = \hat{X}_1 + \hat{d}_2 = \hat{X}_1 + d_2 + q_2 \tag{2.16}$$

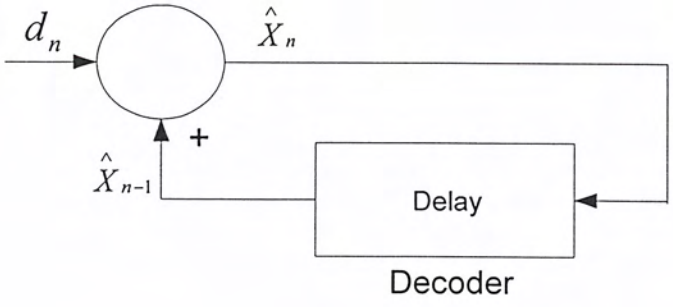
$$= X_2 + q_2 \tag{2.17}$$

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

ซึ่งค่าผิดพลาดจากการควอนไทซ์จะไม่เพิ่มขึ้นเมื่อลำดับเพิ่มขึ้น ซึ่งเทคนิคที่กล่าวมาสามารถเขียนเป็นระบบการเข้ารหัส และถอดรหัสได้ดังนี้

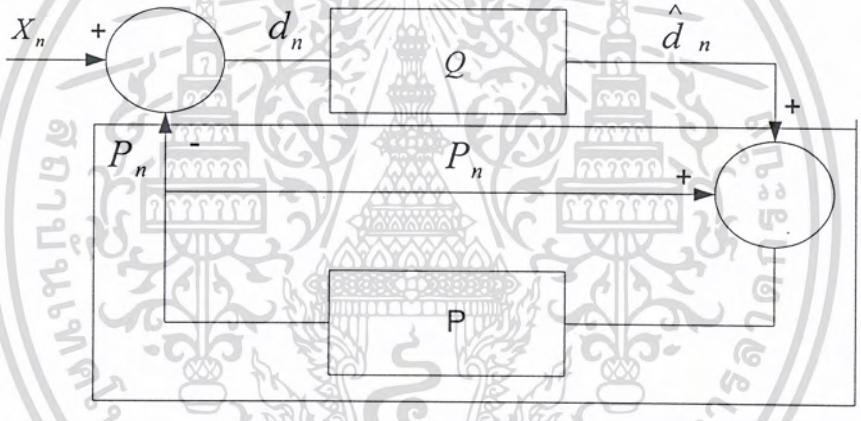


รูป 2.7 แสดงระบบการเข้ารหัสแบบผลต่าง

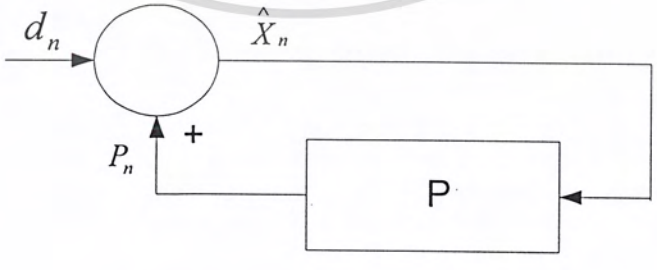


รูปที่ 2.8 แสดงระบบการถอดรหัสแบบผลต่าง

จากรูปที่ 2.7 และ 2.8 เป็นตัวอย่างของระบบเข้ารหัสและถอดรหัสเมื่อทำนายว่า $X_n = X_{n-1}$ ซึ่งเป็นรูปแบบหนึ่งของระบบในรูป 2.9



รูป 2.9 แสดงระบบการเข้ารหัสแบบผลต่าง



รูปที่ 2.10 แสดงระบบการถอดรหัสแบบผลต่าง

จากรูปที่ 2.9 และ 2.10 นำมาเขียนเป็นสมการได้ดังนี้

$$\hat{d}_n = Q[d_n] = d_n + q_n \quad (2.19)$$

$$d_n = X_n - p_n \quad (2.20)$$

$$\hat{X}_n = \hat{d}_n + p_n = d_n + q_n + p_n = X_n + q_n \quad (2.21)$$

โดย p_n คือค่าที่ทำนายได้โดย

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

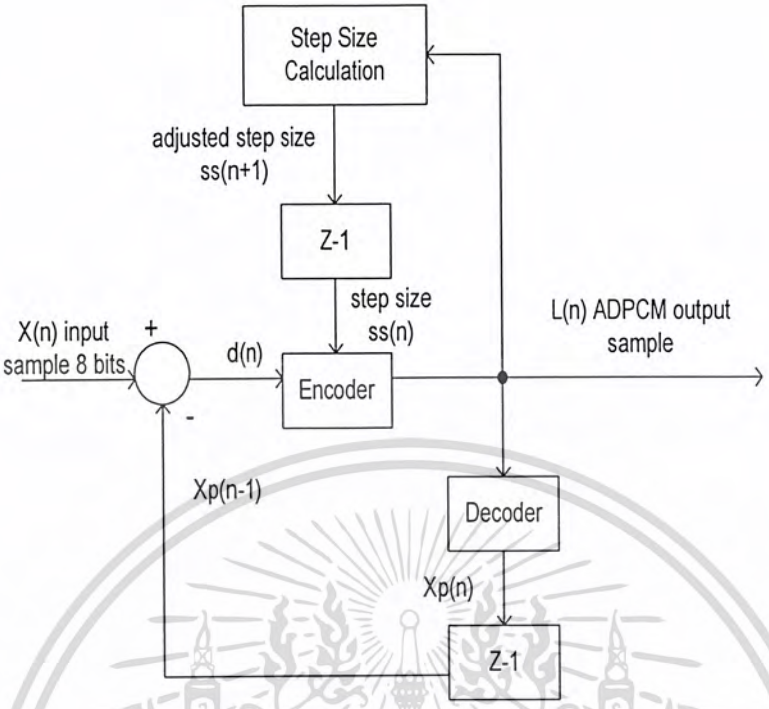
2.6 การเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น (Adaptive Differential Pulse Code Modulation: ADPCM)

ในการใช้งานพีซีเอ็ม มีอัตราการส่งมาตรฐานคือ 64 Kbit/s ซึ่งต้องการช่องสัญญาณในการส่งที่มีแบนด์วิทธ์กว้างในการนำไปใช้งาน เช่นงานเกี่ยวกับความปลอดภัยของสัญญาณเสียงในการส่งในสายส่งบนช่องสัญญาณวิทยุซึ่งมีความจุต่ำ การประยุกต์ใช้งานแบบนี้เป็นที่ต้องการสำหรับการเข้ารหัสสัญญาณเสียงที่มีอัตราการส่งต่ำ จากการศึกษาได้พบว่าการเข้ารหัสสัญญาณแบบดีพีซีเอ็ม เป็นวิธีการบีบอัดสัญญาณวิธีหนึ่งที่ถูกนำมาใช้ในการลดจำนวนบิตของการเข้ารหัสสัญญาณ แต่ความต้องการในการลดจำนวนบิตในการเข้ารหัสนั้นยังคงมีอยู่เรื่อยๆ จึงได้เกิดระบบการเข้ารหัสสัญญาณเสียงอีกแบบหนึ่งขึ้นมาคือ การเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น

หลักการของการเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น หรือ เอดีพีซีเอ็ม จะคล้ายคลึงกับระบบดีพีซีเอ็ม แต่มีการเพิ่มคุณสมบัติในการปรับขั้นของการควอนไตซ์ และวิธีการเปรียบเทียบสัญญาณเข้าไปในระบบดีพีซีเอ็ม ซึ่งทำให้ลดจำนวนบิทในการเข้ารหัสได้มากขึ้น มาตรฐานของระบบเอดีพีซีเอ็ม จะอธิบายถึงการคอมเพรซชิ่งและเอ็กแพนดิ้ง ค่าความยาวของแซมเปิ้ลจาก 8 บิต จะลดลงเหลือ 3,4 หรือ 5 บิต โดยมีอัตราการบีบอัดเป็น2.67,2 และ 1.6 ตามลำดับ และมีอัตราการส่งสัญญาณเท่ากับ 24,32 และ 40 Kbit/s ตามลำดับ เมื่อมีอัตราการแซมปลิ้งเท่ากับ 8 KHz โดยใช้ควอนไตซ์เซอร์เป็นแบบอแด็พทีฟควอนไตซ์เซอร์(adaptive quantizer) ซึ่งค่าระดับการควอนไตซ์จะเปลี่ยนตามค่าระดับของสัญญาณแบบนอนยูนิฟอร์มแล้ว ค่า สเต็ปไซต์ของระดับการควอนไตซ์ยังปรับตามค่าระดับสัญญาณผลต่างที่เข้ามา คือ เมื่อระดับของสัญญาณผลต่างมีค่าสูง ค่าสเต็ปไซต์ก็จะปรับเป็นค่าสูง แต่ถ้าระดับของผลต่างมีค่าต่ำ ค่าสเต็ปไซต์ก็จะปรับเป็นค่าต่ำ เพื่อให้จำนวนบิตที่ใช้ในการเข้ารหัสสัญญาณสามารถครอบคลุมค่าระดับของผลต่างของสัญญาณได้ทั้งหมด ส่วนการเปรียบเทียบสัญญาณจะใช้อแด็พทีฟพรีดิกเตอร์(adaptive predictor) เป็นตัวหาค่าประมาณของสัญญาณอินพุทเพื่อนำมาใช้เปรียบเทียบกับสัญญาณอินพุทที่เข้ามาตัวถัดไป

2.6.1หลักการของการเข้ารหัส เอดีพีซีเอ็ม

จากรูปที่ 2.11 เป็นบล็อกไดอะแกรมของเอดีพีซีเอ็ม ซึ่งสามารถแบ่งเป็นขั้นตอนการทำงานได้ดังนี้ คือ ทำการแปลงสัญญาณเสียงอินพุท 64 Kbit/s ให้เป็นสัญญาณแบบยูนิฟอร์มพีซีเอ็ม ซึ่งได้ค่าแอมปลิจูดของสัญญาณ แล้วใช้เป็นสัญญาณอินพุทในการเข้ารหัสสัญญาณ จากนั้นหาค่าผลต่างของสัญญาณ (difference signal) ซึ่งได้จากการลบระหว่างค่าคาดคะเนสัญญาณอินพุทปัจจุบันกับค่าสัญญาณอินพุทปัจจุบัน



รูปที่2.11 แสดงบล็อกไดอะแกรมการเข้ารหัส เอดีพีซีเอ็ม

เมื่อกำหนดให้

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

$Xp(n)$ = สัญญาณอินพุท (Input signal)

$Xp(n)$ = ค่าคาดคะเนสัญญาณอินพุทปัจจุบัน (Signal estimate)

จะได้ว่า

$$d(n) = X(n) - Xp(n) \tag{2.23}$$

ค่าสัญญาณผลต่างที่ได้จะถูกนำไปทำการจัดระดับการควอนไตซ์ตามค่าสเต็ปไซต์ปัจจุบัน [ss (n)] ซึ่งแทนลอจิกการเข้ารหัส และผลที่ได้คือ รหัสเอดีพีซีเอ็ม ซึ่งเป็นเอาท์พุท เพื่อนำไปทำการถอดรหัสให้ได้สัญญาณเสียงที่ใกล้เคียงกับสัญญาณเสียงเดิมกลับมา

ค่ารหัสเอดีพีซีเอ็มที่ได้จะถูกนำกลับไปใช้ในการคำนวณค่าสเต็ปไซต์ต่อไป : $ss(n+1)$ และอีกส่วนจะถูกป้อนเข้าไปยังอินเวอร์สอแด็พทีฟ ควอนไตซ์เซอร์ (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)

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

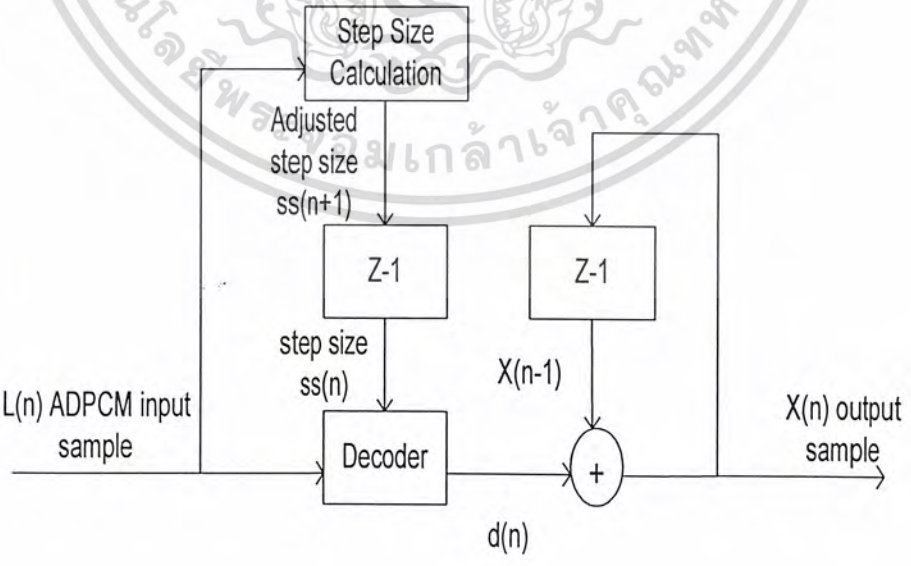
จะได้ว่า

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

ค่าคาดคะเนสัญญาณอินพุทที่ได้จะนำไปเปรียบเทียบกับสัญญาณอินพุทปัจจุบันที่เข้ามาใหม่ เพื่อหาค่าสัญญาณผลต่างตัวถัดไป แล้วนำไปสร้างเป็นรหัสเอดีพีซีเอ็มตัวถัดไป

2.6.2 การถอดรหัสเอดีพีซีเอ็ม

ในรูปที่ 2.12 จะแสดงในส่วนของบล็อกไดอะแกรมของกระบวนการถอดรหัสเอดีพีซีเอ็ม โดยโครงสร้างของกระบวนการถอดรหัสสัญญาณจะเหมือนกับส่วนป้อนกลับของสัญญาณที่กระบวนการเข้ารหัสสัญญาณ ซึ่งจะรับค่ารหัสเอดีพีซีเอ็ม และค่าสเตปไซต์ โดยเป็นการคำนวณผลของค่าความแตกต่างซ้ำไปอีกครั้ง และจะทำการสะสมค่าที่ประมาณได้ให้อยู่ในค่าของสัญญาณอินพุทที่กระบวนการเข้ารหัส(x)



รูปที่ 2.12 แสดงบล็อกไดอะแกรมการถอดรหัส เอดีพีซีเอ็ม

เมื่อกำหนดให้

$\hat{X}$ = สัญญาณที่ได้จากการถอดรหัสของเอดีพีซีเอ็ม

จะได้ว่า

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

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

$$MSE = \frac{1}{N}\sum_{i=1}^{N} [X - \hat{X}]^2 \tag{2.28}$$

เมื่อกำหนดให้

$\hat{X}$ = สัญญาณที่ได้จากการถอดรหัสของเอดีพีซีเอ็ม

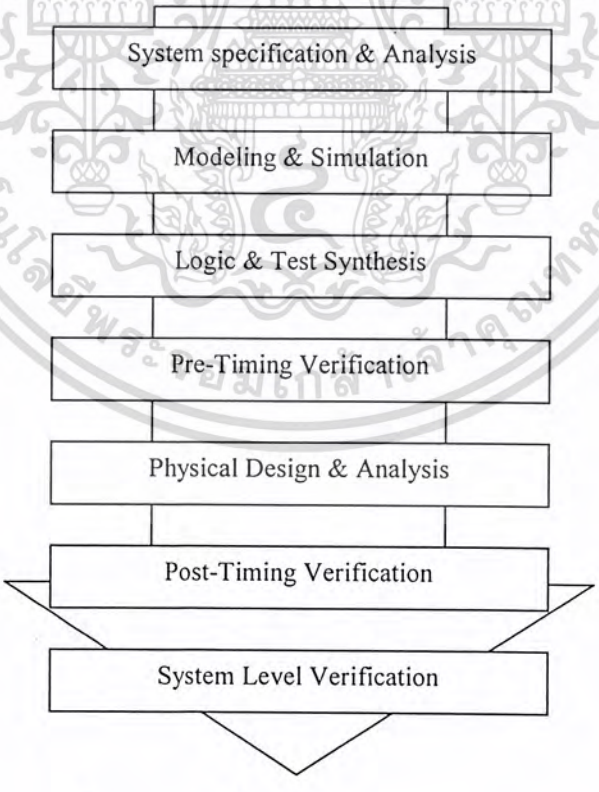
2.8 ภาษาวีเอชดีแอล (VHDL)

การออกแบบวงจรเชิงเลข (Digital Circuit) นั้นในปัจจุบันก้าวหน้าไปอย่างมาก โดยการใช้ภาษาบรรยายการทำงานของวงจร (Hardware Description Language : HDL) ซึ่งในภาษาที่ใช้สำหรับออกแบบฮาร์ดแวร์ 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 ตัวอย่างเช่นการใช้ภาษา VHDL มาทำการออกแบบวงจรกรองสัญญาณเชิงเลข (Digital Filter) ทำให้ลดความยุ่งยากในการนำเอาอุปกรณ์มาเชื่อมต่อให้เป็นวงจร รวมทั้งลดเวลาที่ใช้ในการออกแบบและทดสอบการทำงาน ซึ่งมีความแตกต่างเป็นอย่างมากเมื่อเปรียบเทียบกับการออกแบบในอดีตที่ผ่านมา คือผู้ออกแบบจะต้องนำเอาอุปกรณ์แต่ละตัวที่ทำการออกแบบไว้มาทำการต่อทดลองในวงจรจริงและทำการทดสอบวงจรเพื่อหาข้อผิดพลาดซึ่งต้องใช้เวลานานกับการแก้ปัญหาแต่ละอย่างที่เกิดขึ้น แต่ในการออกแบบด้วยภาษา VHDL ผู้ออกแบบเพียงแต่เขียนซอสโค้ด (Source Code) บรรยายการทำงานของวงจร หลังจากนั้นก็ทำการคอมไพล์ (Compile) แล้วจำลองการทำงาน (Simulate) ดูว่าฟังก์ชั่นการทำงานและไทม์มิ่ง (Timing) ตามที่ต้องการหรือไม่ จากนั้น

ก็ทำการนำซอสโค้ดที่ได้ไปทำการสังเคราะห์ด้วยโปรแกรมสังเคราะห์ (Synthesis Tool) สุดท้ายนำวงจรที่ได้จากการสังเคราะห์ไปทำการแม็ป (map) ลงไปยัง FPGA (Field Programmable Gate Array) เพื่อเป็นชิป (Chip) ต้นแบบสำหรับการนำไปทดสอบการทำงาน

2.8.1 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมเชิงเลขขนาดใหญ่ที่มีความซับซ้อน ผู้ออกแบบมักมองการออกแบบให้อยู่ในรูปของบล็อกไดอะแกรมก่อน จากนั้นจึงวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาติให้อธิบายการทำงานของในแต่ละบล็อก และวิเคราะห์การทำงาน แก้ไขปรับปรุงการทำงานจากผลที่วิเคราะห์ เพื่อให้ได้การทำงานตามที่ต้องการ โดยการออกแบบในลักษณะนี้เรียกว่าหลักการทำงานจากบนลงล่าง (Top-Down Design) : ซึ่งถ้าเปรียบเทียบกับการออกแบบจากล่างขึ้นบน(bottom-up design) จะเห็นได้ว่าการออกแบบจากล่างขึ้นบนจะใช้เวลาในการออกแบบมากกว่าเพราะเป็นการวาดรูปด้วยอุปกรณ์ต่างๆ (Schematic Capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบจำลองการทำงานตรวจสอบความถูกต้องซึ่งใช้เวลามาก และถ้าวงจรที่ต้องการออกแบบมีความซับซ้อนก็จะเป็นเรื่องที่ยากมากในการออกแบบลักษณะนี้ ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่างจึงเป็นวิธีการที่เหมาะสมสำหรับการออกแบบและพัฒนาวงจรที่มีความซับซ้อนมากขึ้นทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบ



รูปที่ 2.13 แสดงขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.13 แสดงให้เห็นถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในการปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย โดยขั้นตอนการออกแบบจากบนลงล่างมีรายละเอียดดังนี้

1. ขั้นตอนการสร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and concept) ในการแก้ปัญหา

2. ขั้นตอนการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงาน เพื่อตรวจสอบความถูกต้องของข้อกำหนด

3. ขั้นตอนการสังเคราะห์ ซึ่งจะต้องทำการกำหนดเทคโนโลยีที่จะมารองรับวงจรที่ออกแบบและระบบช่วยออกแบบจำทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กโทรนิกส์ หรือวงจรในระดับเกท (Gate level) และการเชื่อมต่อกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของเน็ทลิสต์ (Net list) ที่สามารถนำไปผลิตลงอุปกรณ์อื่นได้

4. หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือเน็ทลิสต์ ข้อมูลที่ได้นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงานในเรื่องของความถูกต้องของฟังก์ชั่นแล้ว ยังมีข้อมูลที่เกี่ยวกับเวลาด้วยซึ่งจากความจริงที่ว่า อุปกรณ์อิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของการเคลื่อนผ่าน (Propagation Delay Time) เสมอถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับนาโนวินาที แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชั่นต่างๆจำนวน 10,000 เกท ขึ้นไป เวลา ดังกล่าวนี้จะสะสมมากขึ้นจนอาจจะทำให้การทำงานของวงจรทั้งหมดผิดไปหรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาสูงๆได้

5. ขั้นตอนของการผลิตเป็นวงจรจริง (Technology and Device Mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของอุปกรณ์ FPGA หรือวงจรรวม ASIC

6. หลังจากที่ได้วงจรจริงมาแล้วยังต้องมีความจำเป็นที่จะต้องตรวจสอบการทำงานที่คำนึงถึงเวลาด้วยความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นๆให้เป็นระบบ เพราะในขั้นตอนนี้วงจรที่ออกแบบจะประกอบด้วยอินพุทและเอาท์พุทแพด (Pad) ซึ่งเป็นจุด ต่อสำหรับรับและส่งสัญญาณกับภายนอก

7. หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่นๆให้เป็นระบบแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นๆอีกครั้ง ซึ่งเป็นการทดสอบการทำงานจริงขั้นสุดท้าย

2.8.2 การออกแบบวงจรเชิงเลขด้วยอุปกรณ์ FPGA

อุปกรณ์ FPGA (Field Programmatic Gate Array) เป็นอุปกรณ์ที่ใช้ในการวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการออกแบบตามที่กำหนดไว้ ในการทำ FPGA ซึ่งเป็นวิธีการออกแบบ IC (Integrated Circuit) แบบ Semi custom อีกวิธีหนึ่ง เมื่อเทียบกับการทำ ASICs (Application Specific Integrated Circuits) แล้วนั้นจะมีทั้งข้อดีและข้อเสีย คือ การทำ FPGA จะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัดและการทำFPGA ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือผลิตในปริมาณต่ำ ส่วนข้อดีของการทำ FPGA ก็เหมาะสำหรับการผลิตต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำ FPGA ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (download) นั้นน้อยกว่าการทำ ASIC มากและการตรวจสอบหรือแก้ไขการออกแบบก็ทำได้สะดวก

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

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

2.8.3 การออกแบบโดยใช้ภาษาอธิบายการทำงานของฮาร์ดแวร์

ในการออกแบบวงจรเชิงเลขนั้นทำได้ดีโดยการวาดวงจร หรือใช้ภาษาอธิบายฮาร์ดแวร์ ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วย FPGA และ ASIC ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์ แต่ในกรณีที่ออกแบบโดยวิธีการวาดวงจรจะแตกต่างกันโดยที่การทำวิธีนี้จะต้องคำนึงถึงเทคโนโลยีที่ใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป จะเห็นว่าการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ทำได้สะดวกกว่า เพราะการทำด้วยวิธีนี้ไม่ต้องคำนึงถึงเทคโนโลยีที่จะใช้ (Technology Independence) และที่สำคัญการออกแบบด้วยวิธีนี้สามารถจะแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกเพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี

ในการเขียนโค้ด สิ่งที่ต้องคำนึงถึงคือเขียนอย่างไรจึงจะสามารถสังเคราะห์เป็นวงจรได้และให้คุณสมบัติของวงจรตามกำหนด เพราะลักษณะการเขียนโค้ดจะมีผลโดยตรงกับวงจรที่ได้ เนื่องจากในการสังเคราะห์วงจรนั้นซอร์ฟแวร์สังเคราะห์วงจร (Synthesis Tools) จะทำการสังเคราะห์ตามโค้ดที่เขียน ถ้าอธิบายการทำงานของวงจรเดียวกันแต่เขียนโค้ดในลักษณะที่ต่างกันเมื่อสังเคราะห์แล้วจะได้วงจรที่ต่างกัน และจากวงจรที่ต่างกันเมื่อนำไปทำต้นแบบด้วย FPGA หรือการทำ ASIC แล้วจะได้ไอซีที่มีคุณสมบัติต่างกันทั้งในด้านขนาดและความเร็ว (area and time) ส่วนการเขียนโค้ดลักษณะใดเพื่อให้ได้ผลลัพธ์ที่ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ในการออกแบบ

2.8.4 การจำลองการทำงานของวงจร (Simulation)

ขั้นตอนการนี้เป็นขั้นตอนที่สำคัญเพราะเป็นขั้นตอนที่ใช้ตรวจสอบฟังก์ชั่นการทำงานของวงจรว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหน เพื่อที่จะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สำหรับทำการจำลองการทำงานของวงจร เช่น V-System และ ModelSim ของบริษัท Model Technology

2.8.5 การสังเคราะห์วงจร

ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สังเคราะห์วงจร (Synthesis tools) ทำการสังเคราะห์โค้ดเพื่อให้ได้เป็นวงจรขึ้นมา แต่ตรวจสอบว่าซอร์ฟแวร์นั้นๆสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการใช้หรือไม่ โดยFPGA ที่นิยมใช้งานเช่นบริษัท Xilinx ตระกูล XC4000 และบริษัท Altera ตระกูล FLEX 10k ซอร์ฟแวร์สังเคราะห์วงจรที่นิยมใช้เช่นโปรแกรม Leonardo Spectrum ของบริษัท Exemplar Logic ซึ่งในขั้นตอนนี้ซอร์ฟแวร์สังเคราะห์วงจรจะแปลงโค๊ดและทำการออปติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่ใช้ นอกจากนี้ยังสามารถกำหนดข้อบังคับสำหรับวงจรได้เช่น ข้อบังคับในเรื่องของเวลา (time constraints) หรือข้อบังคับในเรื่องของพื้นที่ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน ออปติไมซ์เพื่อให้วงจรที่ได้เป็นไปตามกำหนด ส่วนสำคัญในการออปติไมซ์คือการเทียบ (Mapping) วงจรเข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างสถาปัตยกรรมภายในอุปกรณ์ FPGA ในกรณีของ Xilinx ตระกูลXC4000 และ Altera ตระกูล FLEX 10k จะเทียบโดยใช้วิธี LUT (look Up Table) เมื่อทำการสังเคราะห์วงจรเสร็จแล้วซอร์ฟแวร์สังเคราะห์วงจรก็จะมีการรายงานผลว่าวงจรที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีความหน่วง (delay) เท่าไร ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้างเป็นต้น

2.8.6 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ให้เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่แยกออกจากกันมีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จำได้เพื่อช่วยลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (Routing) ในขั้นตอนนี้จะใช้ซอร์ฟแวร์ทำ โดยซอร์ฟแวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate) , ฟลิปฟลอป (flipflop) ลงในทรัพยากรต่างๆที่มีอยู่ภายในอุปกรณ์ FPGA (CKBs, IOBs, BUFT และedge decoder) หลังจากทำขั้นตอนนี้เสร็จแล้วสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไหร่ ส่วนซอร์ฟแวร์ที่ใช้ในขั้นตอนนี้ขึ้นอยู่กับตัว FPGA ที่ใช้งาน แทน FPGA ของบริษัท Xilinx จะใช้ Xilinx Foundation Series 2.11 ซึ่งซอร์ฟแวร์ตัวนี้จะรวมเอาซอร์ฟแวร์ย่อยอื่นๆอีก เพื่อให้การทำ PPR (Partitioning, Placement and Routing) เป็นต้นไปอย่างต่อเนื่อง ส่วน FPGA ของบริษัท Altera จะใช้ Altera MAX+II

2.8.7 การวางอุปกรณ์(Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควรจะอยู่ในตำแหน่งใดในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่นส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทาง (route) ได้ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมีความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วทำให้ความหน่วงเพิ่มขึ้นหรือตัว router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด

2.8.8 การเชื่อมต่อสัญญาณ (Route)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA เช่น ระหว่าง CLBs หรือระหว่าง CLBs กับ IOBs ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอร์ฟแวร์ก็การเชื่อมต่อสัญญาณด้วยตัวเอง (Manual layout) ก็ได้ แต่ทางที่ดีควรใช้ซอร์ฟแวร์ทำทำได้ดีกว่าโดยทำการค้นหาเส้นทางหลายๆครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการกำหนดข้อบังคับทางเวลา (time constraints) จะช่วยให้ได้ผลที่ได้จากการเชื่อมต่อสัญญาณขึ้นได้

2.8.9 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่วงจรผ่านขั้นตอนต่างๆจนกระทั่งผ่านการทำ PPR แล้วนั้น ถึงตอนนี้สามารถที่จะดาวน์โหลด ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้ให้เป็นข้อมูลวงจร (Configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (Bit-stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามวงจรที่ออกแบบไว้

จากที่อธิบายมาทั้งหมดจะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นทำได้สะดวกกว่าการทำ ASIC มากเพราะใช้เวลาน้อยกว่ามาก ส่วนสำคัญที่ใช้ในการทำ FPGA คือซอร์ฟแวร์ที่ใช้ตั้งแต่การเขียนโค๊ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงในอุปกรณ์ FPGA ซึ่งซอร์ฟแวร์ที่ใช้ต้องเป็นซอร์ฟแวร์ที่ใช้ต้องทำงานต่อเนื่องกัน

บทที่ 3

หลักการคำนวณและการออกแบบการเข้ารหัสแบบเอดีพีซีเอ็ม

3.1 การเข้ารหัส

3.1.1 รับสัญญาณเสียงอินพุทเข้ามาเก็บไว้ในไฟล์

3.1.2. หาค่าผลต่างของสัญญาณอินพุทกับค่าคาดคะเนสัญญาณอินพุท: d (n)

3.1.3. นำค่าผลต่างที่ได้มาพิจารณาตามค่าสเต็ปไซด์: ss (n) โดยบีบอัดให้เหลือ 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:

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

3.1.4 การคำนวณค่าสเตปไซต์

จากทั้งกระบวนการเข้ารหัสและกระบวนถอดรหัสเอดีพีซีเอ็ม ในอัลกอริทึมของเอดีพีซีเอ็ม นั้นจะเป็นการปรับค่าของการควอนไตซ์สเตปไซต์ โดยอยู่บนพื้นฐานของค่าเอดีพีซีเอ็มก่อนหน้า ซึ่งค่าสเตปไซต์ในแซมเปิลถัดไปจะเป็น $[ss(n+1)]$

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))
1111 or 0111	+8
1110 0110	+6
1101 0101	+4
1100 0100	+2

L(n)	M(L(n))
1011 0011	-1
1010 0010	-1
1001 0001	-1
1000 0000	-1

ตารางที่ 3.1 แสดงค่าแมกนีจูดของเอดีพีซีเอ็มแบบ 4 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

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

ถ้า $B3 = 1$

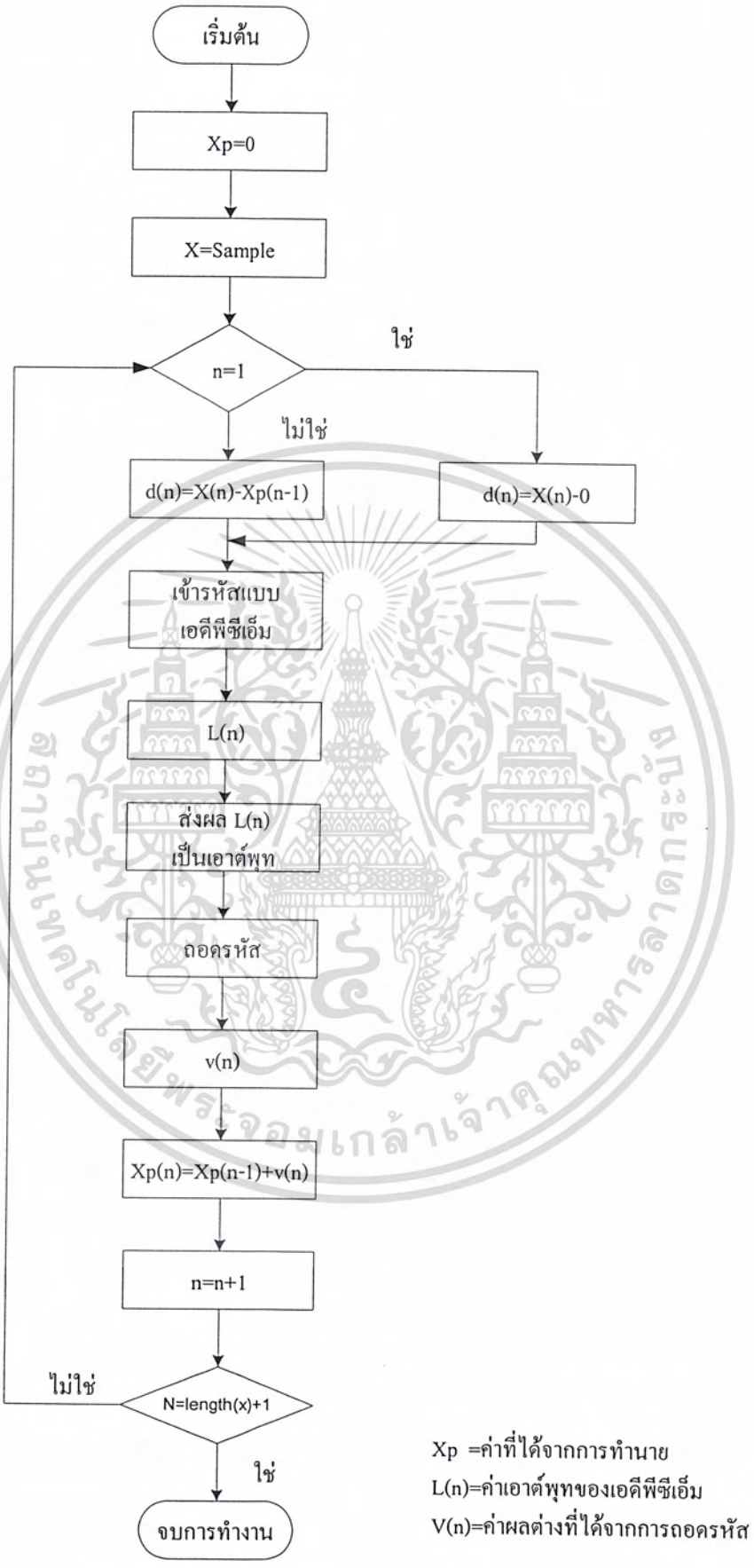
แล้ว $d(n) = d(n) * (-1)$

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

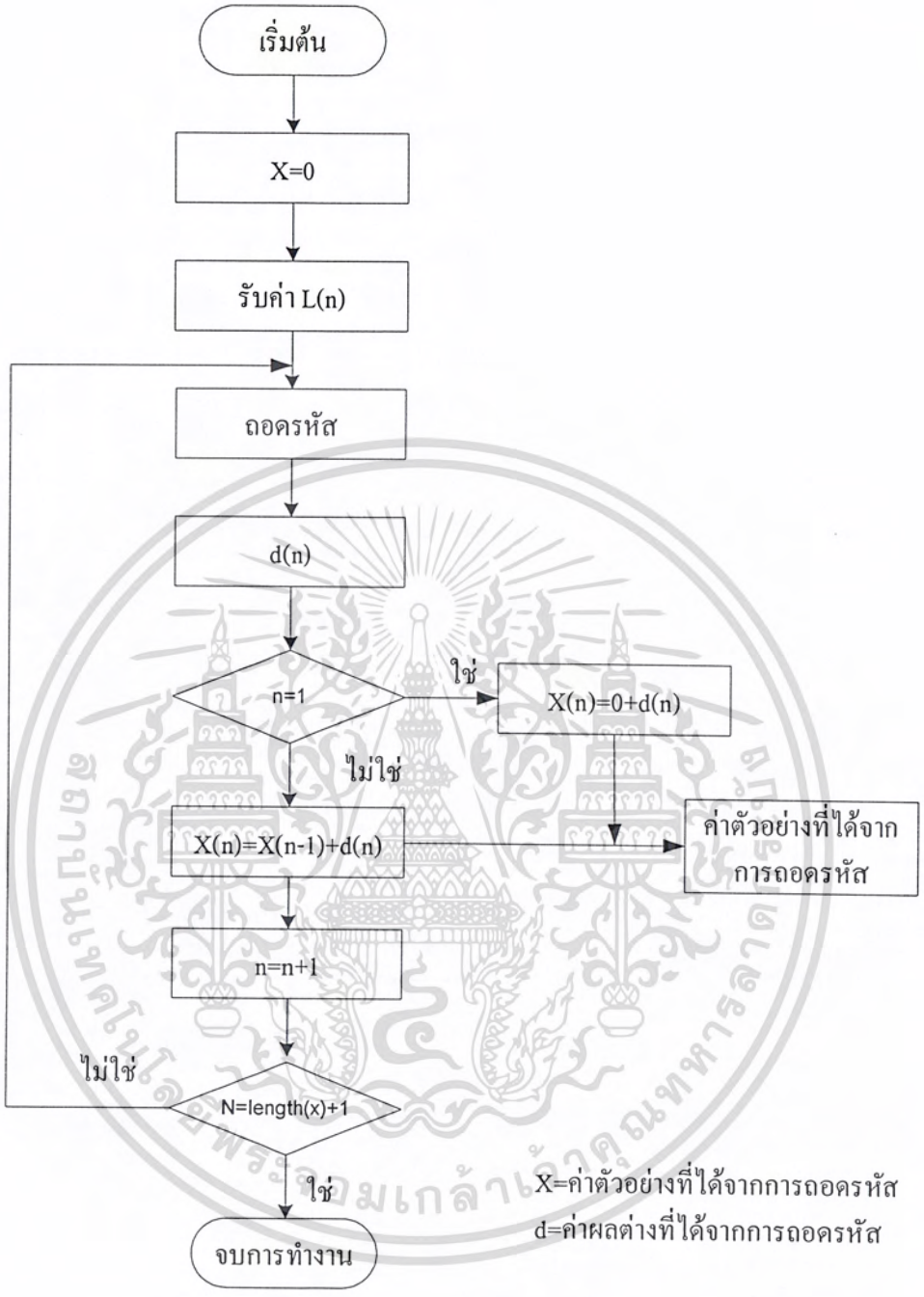
สำหรับวิธีการเอดีพีซีเอ็มนี้ เป็นวิธีการที่เหมาะสมสำหรับการปรับค่าในรูปแบบของคลื่นแบบไซนูซอยดอล แต่ไม่เหมาะสำหรับคลื่นในแบบอื่น

3.3 การแสดง Flow Chart

จากการคำนวณ นำค่าอัลกอริทึมมาแสดงกระบวนการทำงานการเข้ารหัสสัญญาณและการถอดรหัสสัญญาณแบบเอดีพีซีเอ็ม ดังแสดงในรูปที่ 3.1 และ รูปที่ 3.2



รูปที่3.1 แสดง Flowchart การเข้ารหัสเสียงแบบเอดีพีซีเอ็ม



รูปที่3.2 แสดง Flowchart การถอดรหัสข้อมูลเสียงแบบเอดีพีซีเอ็ม

3.4 การสร้างวงจรด้วยเอฟพีจีเอ

สำหรับวงจรสื่อสารข้อมูลแบบอนุกรมที่ออกแบบนี้จะมีคุณสมบัติดังนี้

Baud Rate = 9600 bits/sec

Data = 8 bits

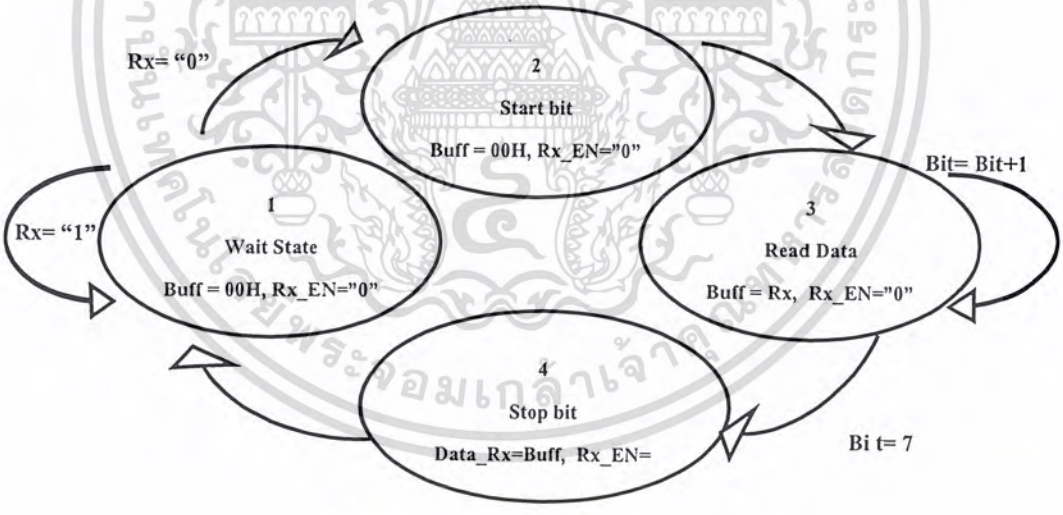
Start bit = 1 bit

Stop bit = 1 bit

3.4.1 ภาครับข้อมูลแบบอนุกรม

ในการออกแบบจะต้องมีสัญญาณ Clock เป็นสัญญาณอ้างอิง เพื่อกำหนดความเร็วในการรับข้อมูล ซึ่งจะต้องสอดคล้องกับภาคส่งด้วย (9600 Hz)

เมื่อสัญญาณ Rx มีค่าเป็นลอจิก "1" แสดงว่ายังไม่มีการส่งข้อมูลออกมา จะรอจนกว่าเป็นลอจิก "0" แสดงว่ามีการส่งข้อมูลออกมาแล้ว หลังจากนั้นจะทำการอ่านข้อมูลทีละบิตจนครบ 8 บิตและตรวจสอบสัญญาณ Rx ว่าเป็นลอจิก "1" หรือไม่ หากเป็นลอจิก "1" แสดงว่าสิ้นสุดการส่งข้อมูล และกำหนดให้สัญญาณ Data_Rx มีค่าเท่ากับสัญญาณที่รับมาได้ และกำหนดให้สัญญาณ Rx_EN มีค่าเป็นลอจิก "1" เพื่อเป็นสัญญาณกระตุ้นให้วงจรต่อไปนำข้อมูล Data_Rx ไปใช้งาน สามารถเขียนการทำงานเป็น State Diagram ได้ดังรูป



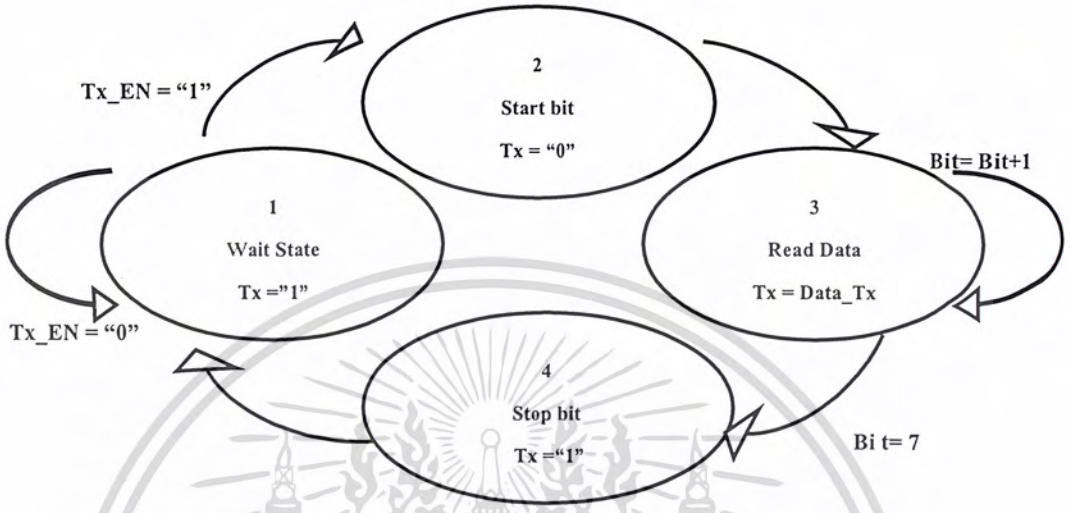
รูปที่ 3.3 State Diagram ของการรับข้อมูลแบบอนุกรม

3.4.2 ภาคส่งข้อมูลแบบอนุกรม

ในการส่งข้อมูลจะต้องมีสัญญาณ Clock เป็นสัญญาณอ้างอิง เพื่อกำหนดความเร็วในการส่งข้อมูล (Baud Rate) มีค่าเท่ากับ 9600 Hz

เมื่อสัญญาณ Tx_ENเป็น "0" แสดงว่ายังไม่ต้องการส่งข้อมูล จะทำให้สัญญาณ Tx เป็น "1" จนกว่าสัญญาณ Tx_EN มีค่าเป็นลอจิก "1" แสดงว่าต้องการส่งข้อมูลออกไป จะทำให้สัญญาณ Tx มีค่าเป็นลอจิก "0" เพื่อเป็นการบอกทางภาครับว่าจะเริ่มต้นส่งข้อมูลออกไป และจะทำการส่งข้อมูลออกไป

Data_Tx เป็นข้อมูลที่ต้องการส่งแบบอนุกรม โดยบิตที่ต่ำสุด (Bit 0) ออกไปก่อน จะส่งข้อมูลไปทีละบิต จนครบ 8 บิต จากนั้น Tx จึงมีสถานะเป็นลอจิก “1” เพื่อเป็นการบอกว่าสิ้นสุดการส่งข้อมูล สามารถเขียนเป็น State Diagram ได้ดังรูป



รูปที่ 3.4 State Diagram ของการส่งข้อมูลแบบอนุกรม

3.4.3 DIV 1000

เป็นโมดูลของวงจรหารความถี่ เนื่องจากภายในบอร์ดใช้แหล่งกำเนิดความถี่ 9.6 MHz แต่ความถี่ที่เราใช้เป็น Baud Rate ให้กับวงจรรับ-ส่ง มีค่าเท่ากับ 9600 Hz เพราะฉะนั้นเราต้องทำการหารความถี่ที่ได้จาก Oscillator ลงให้เหลือ 9600 Hz ซึ่ง

$$\text{ค่าของตัวหารจะเท่ากับ} = \frac{9.6MHz}{9600Hz} = 1000$$

บทที่ 4
การทดลองและผลการทดลอง

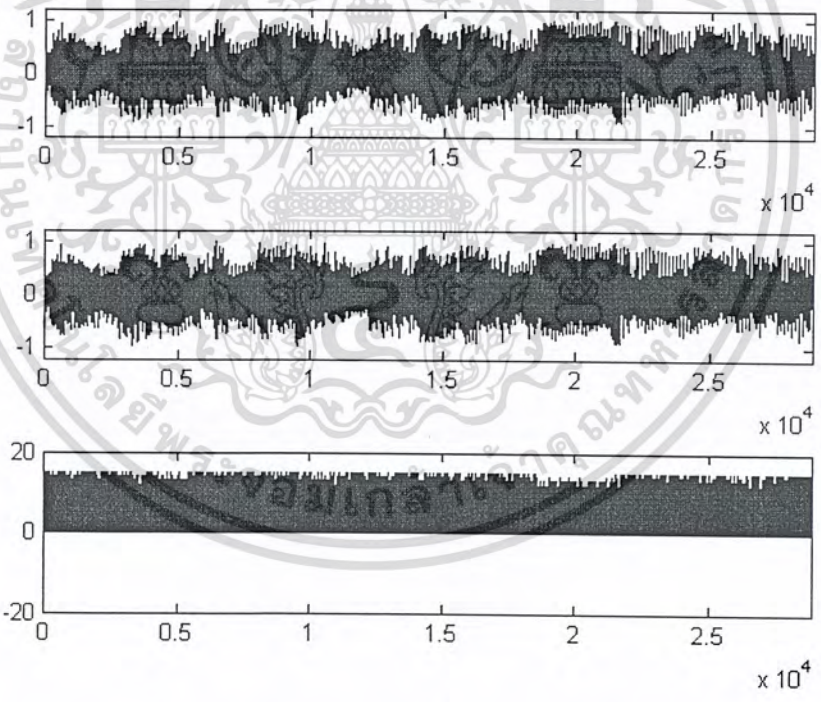
โครงงานนี้ได้ทำการศึกษาการเข้ารหัสสัญญาณเสียงโดยใช้หลักการเอดีพีซีเอ็ม ทำการบีบอัดข้อมูลเสียงจากความยาวแซมเปิ้ล 8 บิต ให้เหลือ 4 บิต แล้วทำการหาค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน (Signal to noise ratio: SNR) และจำลองการทำงานของการเข้ารหัสสัญญาณเสียงโดยใช้หลักการเอดีพีซีเอ็มด้วยภาษา VHDL

4.1 ส่วนของ MATLAB

4.1.1 การบีบอัดเสียงโดยใช้หลักการเอดีพีซีเอ็ม

โดยแบ่งให้สัญญาณเสียงอินพุทที่เข้ามามีลักษณะแตกต่างกันจะแบ่งเป็น 3 รูปบนสุดเป็นสัญญาณอินพุท รูปกลางเอาท์พุท รูปล่างสัญญาณที่เข้ารหัสซึ่งจะมีดังนี้

4.1.1.1 สัญญาณเสียงเพลงที่ร้องว่า "หากแม้มีใครที่เดินเข้ามา"



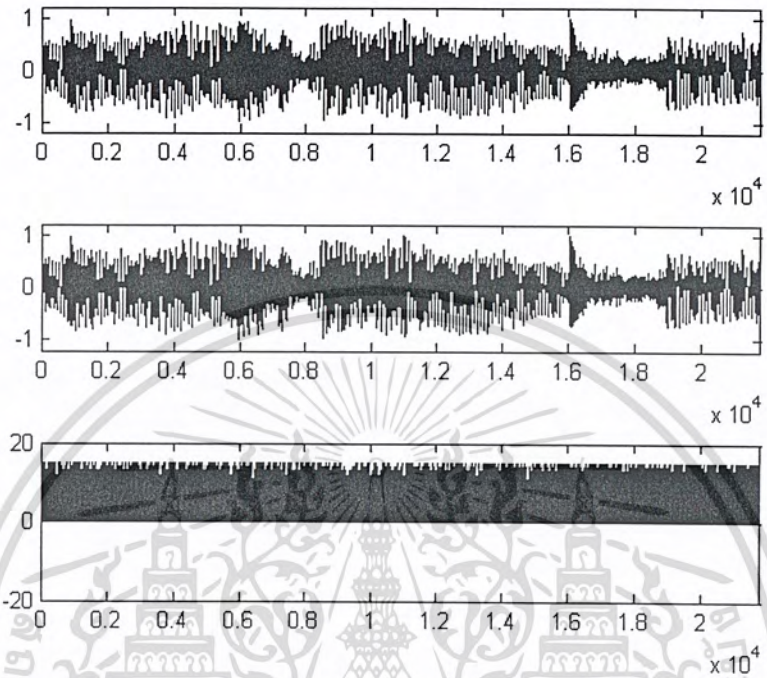
รูปที่ 4.1 แสดงการเปรียบเทียบสัญญาณอินพุทที่ก่อนทำการเข้ารหัสกับสัญญาณเอาท์พุทที่ได้จากการถอดรหัสและสัญญาณที่ได้จากการบีบอัดจากความยาวแซมเปิ้ล 8 บิตเหลือ 4 บิต

จากรูปที่ 4.1 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 23.6128$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

4.1.1.2 สัญญาณเสียงเพลงที่ร้องว่า “ไม่แน่ใจ”



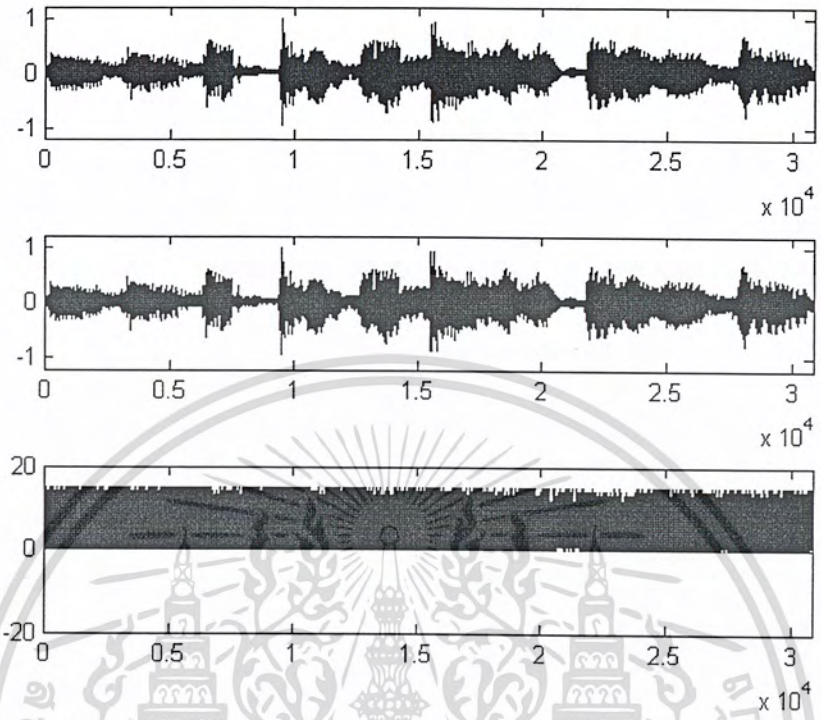
รูปที่ 4.2 แสดงการเปรียบเทียบสัญญาณอินพุทที่ก่อนทำการเข้ารหัสกับสัญญาณเอาท์พุทที่ได้จากการถอดรหัสและสัญญาณที่ได้จากการบีบอัดจากความยาวแซมเปิล 8 บิตเหลือ 4 บิต

จากรูปที่ 4.2 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 23.9374$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

4.1.1.3 สัญญาณเสียงเพลงที่ร้องว่า “ส่งกลิ่นอบอวนป่วนในใจให้เกิน”



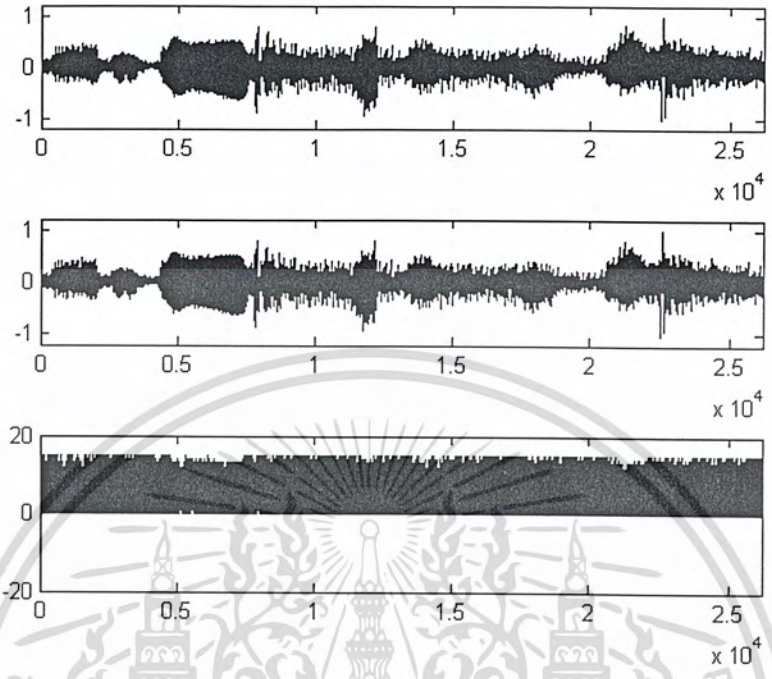
รูปที่ 4.3 แสดงการเปรียบเทียบสัญญาณอินพุทที่ก่อนทำการเข้ารหัสกับสัญญาณเอาท์พุทที่ได้จากการถอดรหัสและสัญญาณที่ได้จากการบีบอัดจากความยาวแซมเปิ้ล 8 บิตเหลือ 4 บิต

จากรูปที่ 4.3 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 22.4991$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

4.1.1.4 สัญญาณเสียงเพลงที่ร้องว่า “อยากให้เธอรู้ว่าทั้งหมดใจ”



รูปที่ 4.4 แสดงการเปรียบเทียบสัญญาณอินพุทที่ก่อนทำการเข้ารหัสกับสัญญาณเอาท์พุทที่ได้จากการถอดรหัสและสัญญาณที่ได้จากการบีบอัดจากความยาวแซมเปิล 8 บิตเหลือ 4 บิต

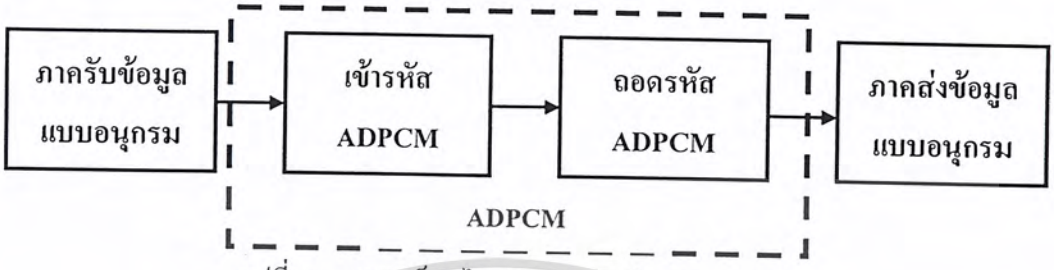
จากรูปที่ 4.4 ค่าอัตราส่วนของสัญญาณกับสัญญาณรบกวน มีค่าเท่ากับ

$$SNR = 22.4339$$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

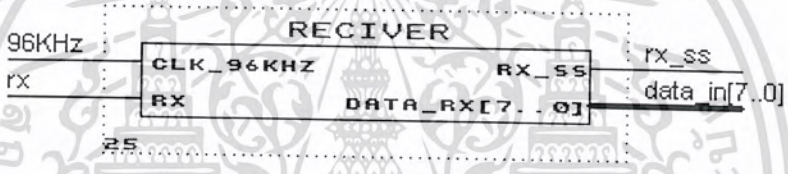
4.2 ส่วนการทำงานโดยใช้ภาษา VHDL

จากการออกแบบ สามารถทำการเขียนโปรแกรมการทำงานโดยใช้ภาษา VHDL ทำการคอมไพล์ (Compile) แล้วจำลองการทำงาน (Simulation) ของโปรแกรมแต่ละส่วนที่ได้เขียนขึ้น โดยแบ่งออกเป็นส่วนๆ ดังนี้



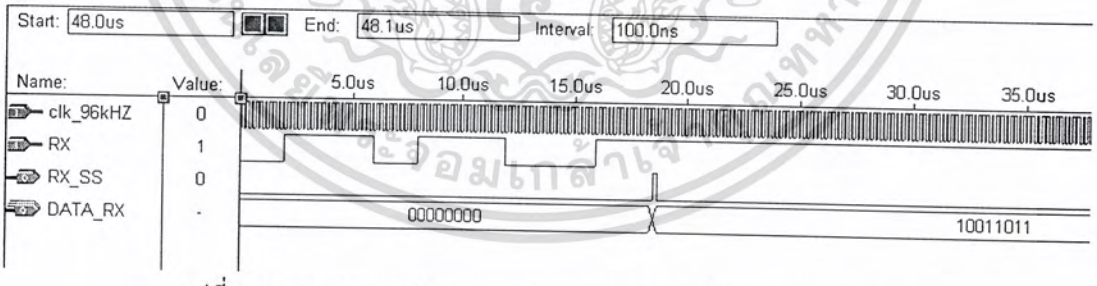
รูปที่ 4.5 แสดงบล็อกไดอะแกรมการทำงานของวงจร

4.2.1 ภาครับข้อมูลแบบอนุกรม



รูปที่ 4.6 แสดงวงจรของภาครับข้อมูลแบบอนุกรม

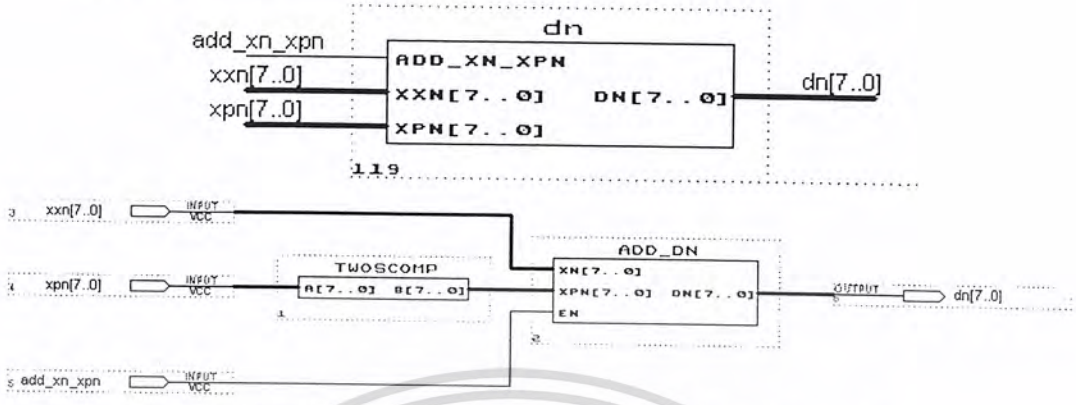
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงานได้ดังรูปที่ 4.15



รูปที่ 4.7 แสดงผลการจำลองการทำงานของภาครับข้อมูลแบบอนุกรม

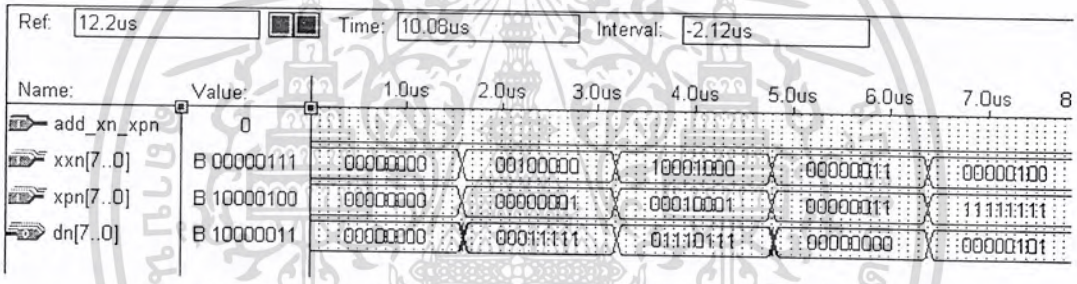
4.2.2 ส่วนการเข้ารหัส

4.2.2.1 วงจรผลต่างและค่าสัญญาณก่อนหน้า



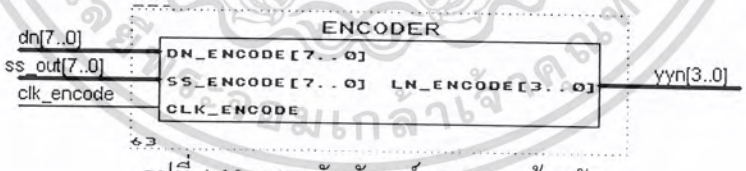
รูปที่ 4.8 แสดงสัญลักษณ์ของวงจรผลต่างและค่าสัญญาณก่อนหน้า

สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.9



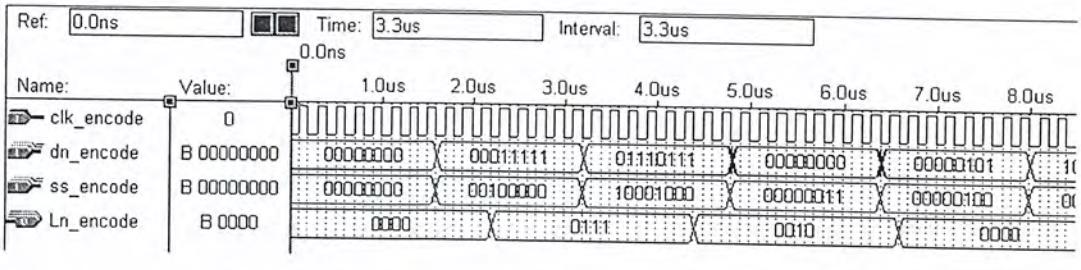
รูปที่ 4.9 แสดงผลการจำลองการทำงานของวงจรผลต่างและค่าสัญญาณก่อนหน้า

4.2.2.2 วงจรเข้ารหัส



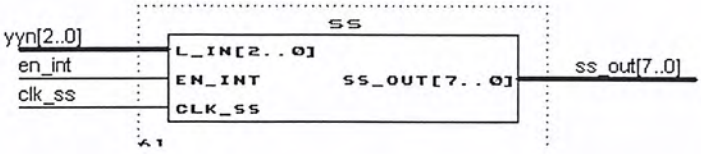
รูปที่ 4.10 แสดงสัญลักษณ์ของวงจรเข้ารหัส

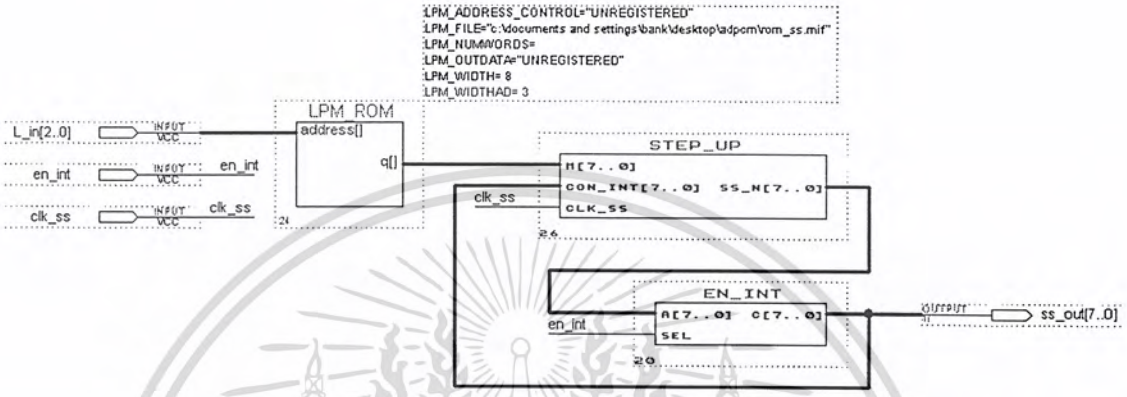
สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.11



รูปที่ 4.11 แสดงผลการจำลองการทำงานของวงจรเข้ารหัส

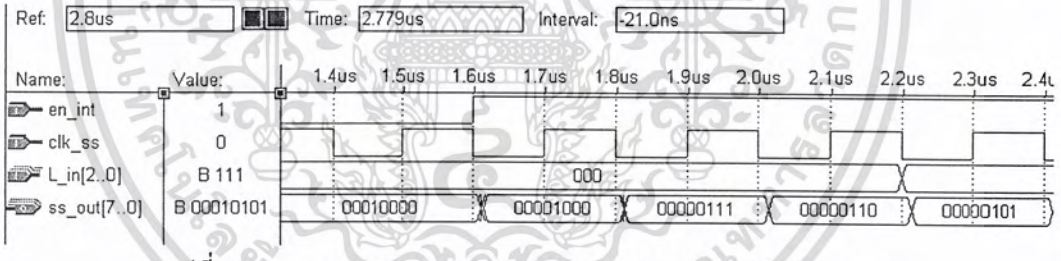
4.2.2.3 วงจรจัดระดับสัญญาณ





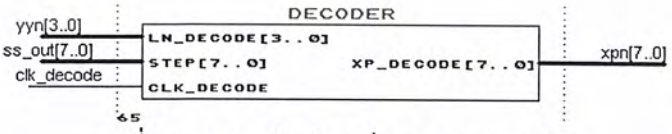
รูปที่ 4.12 แสดงสัญลักษณ์ของวงจรจัดระดับสัญญาณ

สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.13



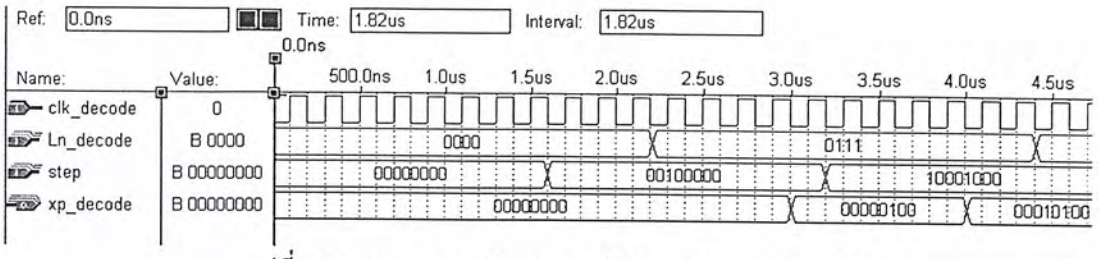
รูปที่ 4.13 แสดงผลการจำลองการทำงานของวงจรจัดระดับสัญญาณ

4.2.3 ส่วนการถอดรหัส



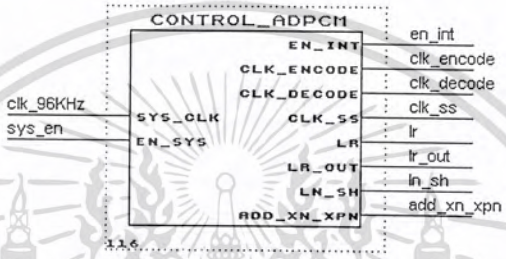
รูปที่ 4.14 แสดงสัญลักษณ์ของวงจรถอดรหัส

สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.15



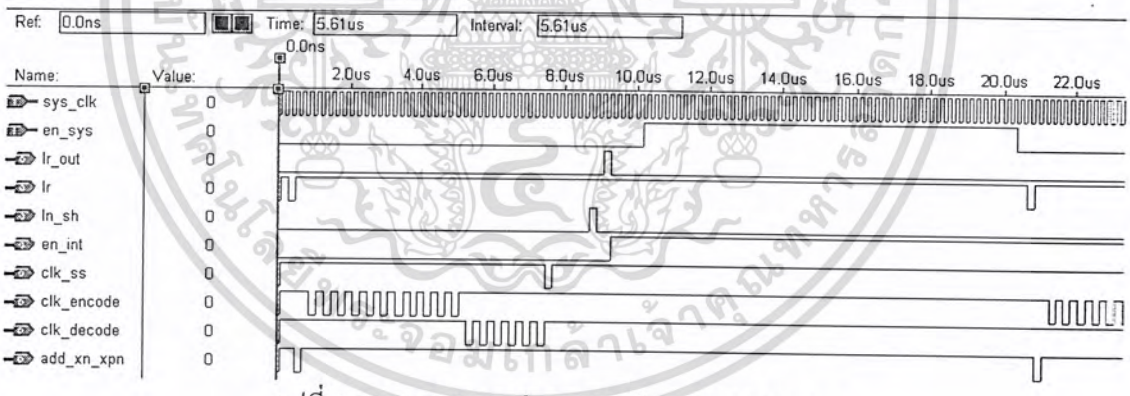
รูปที่ 4.15 แสดงผลการจำลองการทำงานของวงจรถอดรหัส

4.2.4 วงจรควบคุมการทำงานของส่วนเอดีพีซีเอ็ม



รูปที่ 4.16 แสดงสัญลักษณ์ของวงจรควบคุม

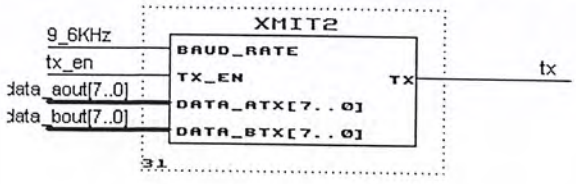
สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.17



รูปที่ 4.17 แสดงผลการจำลองการทำงานวงจรควบคุม

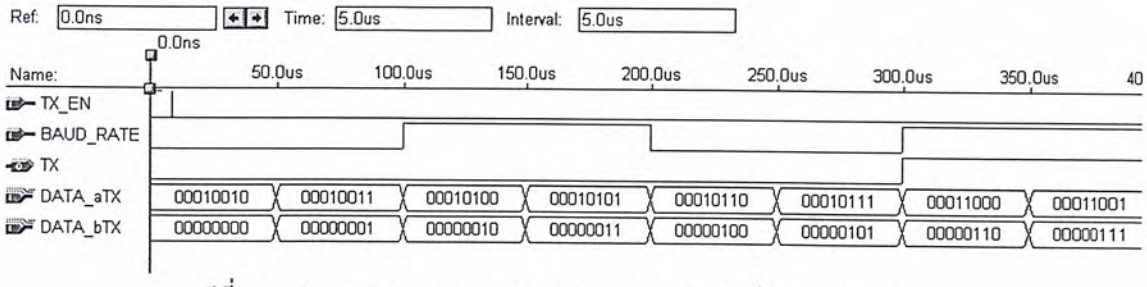
4.2.5 ภาคส่งข้อมูลแบบอนุกรม

ในส่วนนี้จะมีส่วนของ Tric_serial ควบคุมให้วงจรส่งข้อมูลทำงาน และใช้แรมในการเก็บข้อมูลก่อนส่งข้อมูลออกไป เช่นเดียวกับในภาครับข้อมูลที่ใช้แรมในการเก็บข้อมูลที่รับเข้ามา



รูปที่ 4.18 แสดงสัญลักษณ์ของภาคส่งข้อมูล

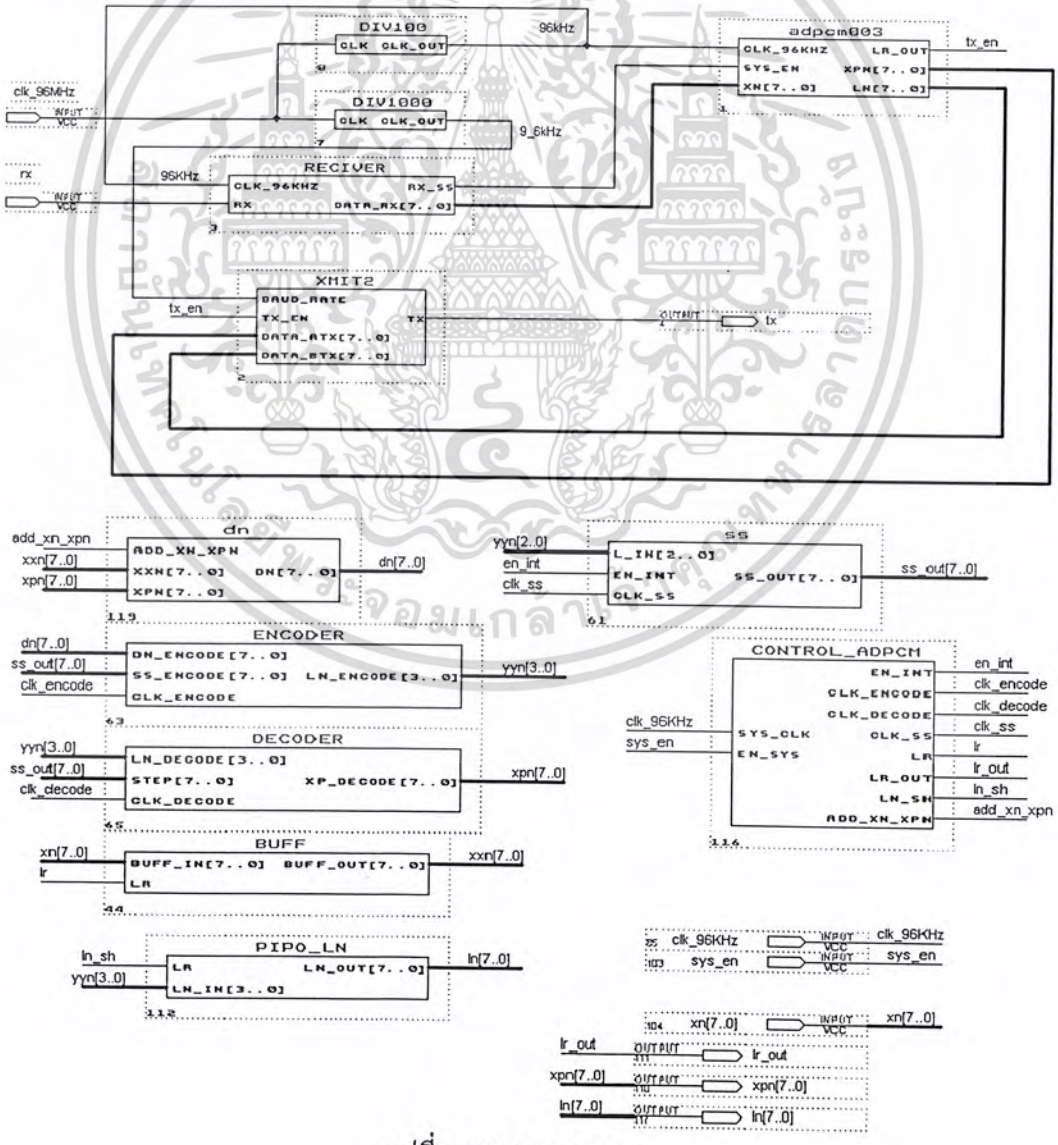
สามารถทำการจำลองการทำงานได้ดังรูปที่ 4.19



รูปที่ 4.19 แสดงผลการจำลองการทำงานของวงจรส่งข้อมูลแบบอนุกรม

4.2.6 วงจรรวม

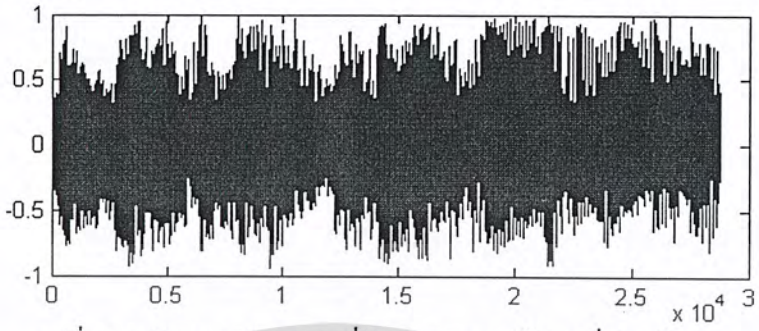
นำวงจรในแต่ละส่วนการทำงานมาต่อรวมกัน เป็นวงจรรวมดังรูปที่ 4.20



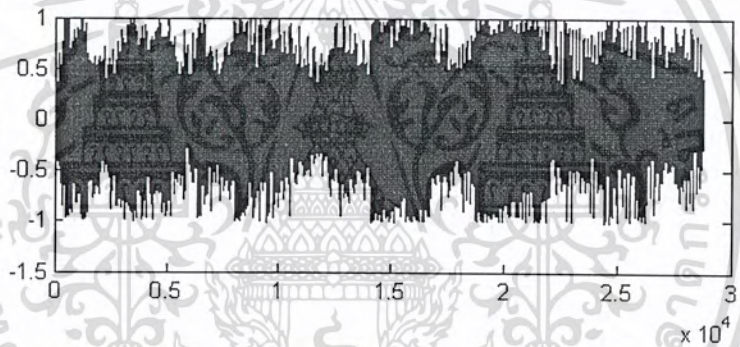
รูปที่ 4.20 แสดงวงจรรวม

จากวงจรในรูปที่ 4.20 ได้ทำการโปรแกรมลงบนบอร์ดจำลอง ACEX1K-EP1K50TC44-3 โดยรับข้อมูลจากคอมพิวเตอร์ (Computer) มาประมวลผล ได้ผลการทดลอง โดยป้อนสัญญาณเสียงดังนี้

4.2.6.1 เมื่ออินพุทคือ สัญญาณเสียงเพลงที่ร้องว่า "หากแม้มีใครที่เดินเข้ามา"



รูปที่ 4.21 สัญญาณเสียงเพลงที่ร้องว่า "หากแม้มีใครที่เดินเข้ามา"

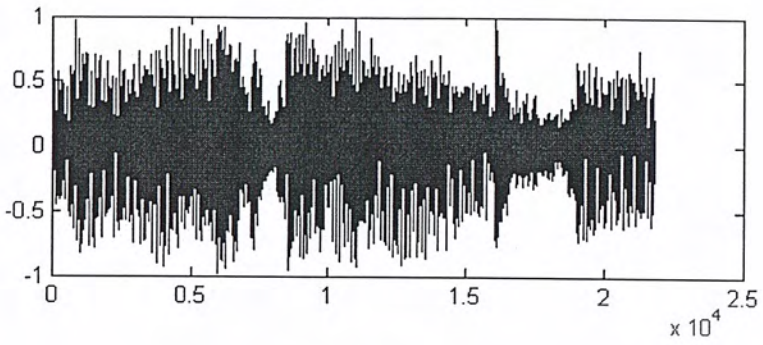


รูปที่ 4.22 สัญญาณเสียงเพลงที่ร้องว่า "หากแม้มีใครที่เดินเข้ามา" หลังทำการถอดรหัส

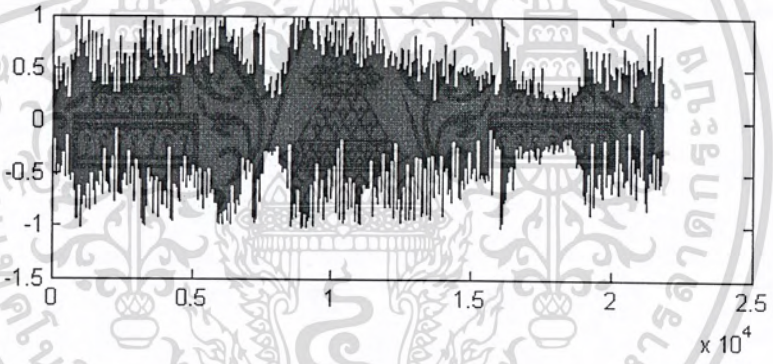
$SNR = 3.00643$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

4.2.6.2 เมื่ออินพุทคือ สัญญาณเสียงเพลงที่ร้องว่า “ไม่แน่ใจ”



รูปที่ 4.23 สัญญาณเสียงเพลงที่ร้องว่า “ไม่แน่ใจ”

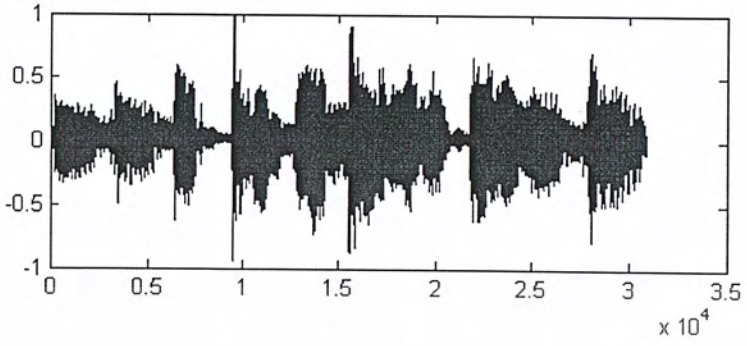


รูปที่ 4.24 สัญญาณเสียงเพลงที่ร้องว่า “ไม่แน่ใจ”
หลังทำการถอดรหัส

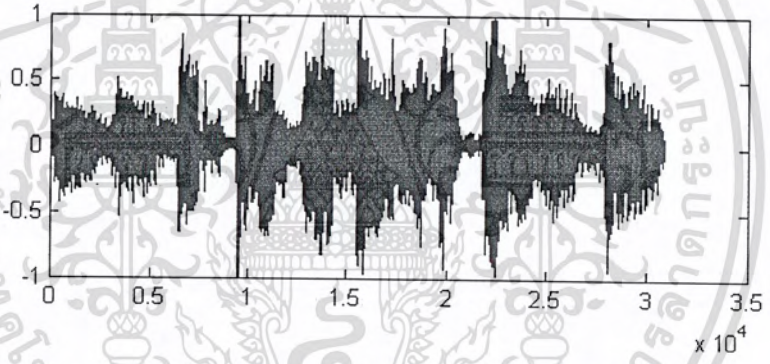
$SNR = 5.51878$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

4.2.6.3 เมื่ออินพุทคือ สัญญาณเสียงเพลงที่ร้องว่า “ส่งกลิ่นอบอวนป่วนในหัวใจให้เกิน”



รูปที่ 4.25 สัญญาณเสียงเพลงที่ร้องว่า “ส่งกลิ่นอบอวนป่วนในหัวใจให้เกิน”

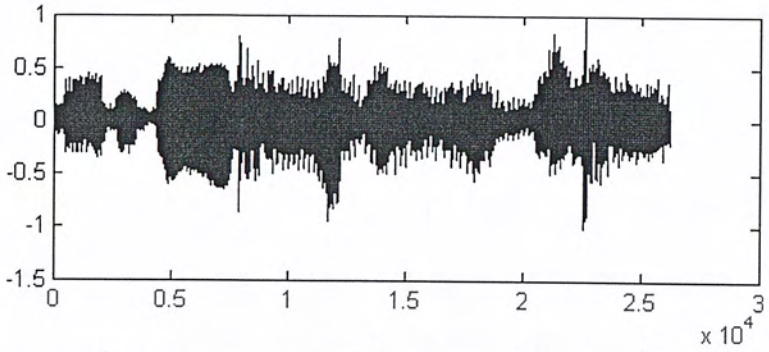


สัญญาณเสียงเพลงที่ร้องว่า “ส่งกลิ่นอบอวนป่วนในหัวใจให้เกิน”
หลังทำการถอดรหัส

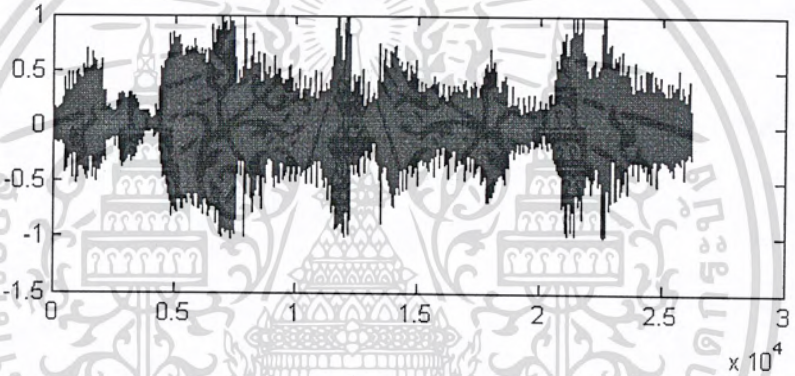
$SNR = 4.14506$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

4.2.6.4 เมื่ออินพุทคือ สัญญาณเสียงเพลงที่ร้องว่า“อยากให้เธอรู้ว่าทั้งหมดใจ”



รูปที่ 4.27 สัญญาณเสียงเพลงที่ร้องว่า“อยากให้เธอรู้ว่าทั้งหมดใจ”



สัญญาณเสียงเพลงที่ร้องว่า“อยากให้เธอรู้ว่าทั้งหมดใจ”

หลังทำการถอดรหัส

$SNR = 2.99952$

ค่าอัตราส่วนของการบีบอัด มีค่าเท่ากับ 2.00

บทที่ 5

สรุปผลการทดลอง

จากที่ได้ทำการทดลองการเข้ารหัสแบบเอดีพีซีเอ็ม คุณภาพของสัญญาณเสียงที่ได้จากเอดีพีซีเอ็มมีคุณภาพดี ซึ่งเห็นได้จากค่าอัตราส่วนของสัญญาณต่อสัญญาณที่เกิดความผิดพลาด : SNR

โดยจากผลการทดลอง เมื่อเราบีบอัดข้อมูลจากความยาวแซมเปิ้ล 8 บิต ให้เหลือ 4 บิตจะเป็นดังตาราง

เสียงเพลงที่ร้องว่า	ค่า SNRของ ADPCM(MATLAB)	ค่า SNRของ ADPCM(FPGA)
“หากแม้มีใครที่เดินเข้ามา”	23.6128	3.00643
“ไม่แน่ใจ”	23.9374	5.51878
“ส่งกลิ่นอบอวนป่วนในหัวใจให้เกิน”	22.4991	4.14506
“อยากให้เธอรู้ว่าทั้งหมดใจ”	23.4339	2.99952

ตารางที่ 5.1 แสดงค่า SNR ที่ได้จากการบีบอัดสัญญาณเสียง

จากการจำลองผลการทดลองจากโปรแกรม MATLAB จะได้คุณภาพของสัญญาณเสียงที่ทำการถอดรหัสกลับมามีลักษณะคล้ายกับสัญญาณต้นแบบ

แต่ในทางปฏิบัติ ในการโปรแกรมลงบอร์ดเอฟพีจีเอนั้น คุณภาพของเสียงที่ได้จากการถอดรหัสค่อนข้างแตกต่างจากสัญญาณต้นแบบ เนื่องจากมีการลดทอนจากอุปกรณ์

ข้อได้เปรียบของระบบการเข้ารหัสสัญญาณแบบเอดีพีซีเอ็ม ที่เหนือกว่าระบบการเข้ารหัสสัญญาณแบบพีซีเอ็มและแบบดีพีซีเอ็ม คือ

1. สามารถลดจำนวนบิตที่ใช้ในการเข้ารหัสสัญญาณได้มากกว่า เนื่องจากใช้การเข้ารหัสจากค่าผลต่างของสัญญาณที่อยู่ข้างเคียงกันซึ่งมีค่าน้อย และมีการปรับขั้นระดับของการควอนไตซ์ตามขนาดของสัญญาณผลต่าง จึงสามารถลดจำนวนบิตที่ใช้เข้ารหัสลงได้มากกว่าระบบพีซีเอ็ม และดีพีซีเอ็ม ซึ่งเท่ากับว่าเป็นการลดขนาดของข้อมูลลงด้วย ทำให้ประหยัดหน่วยความจำที่ใช้ในการเก็บข้อมูลลงได้

2. มีคุณสมบัติในการใช้สายส่งสัญญาณได้อย่างมีประสิทธิภาพมากกว่าระบบพีซีเอ็ม และระบบดีพีซีเอ็ม เนื่องจากอัตราเร็วที่ใช้ในการส่งสัญญาณจะลดลงเมื่อขนาดของข้อมูลลดลง จึงสามารถเพิ่มจำนวนของสัญญาณที่จะส่งไปในสายส่งได้มากขึ้น

เอกสารอ้างอิง

[1] ถวิล กิ่งทอง, "เทคโนโลยีการส่งสัญญาณดิจิตอล", กรุงเทพฯ: ตำราชุดวิศวกรรมศาสตร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,เมษายน 2539

[2] CCITT Recommendation G.726, "CCITT 40, 32, 24, 16 kbps ADPCM",December 2000

[3] Kishan Shanoi, "Digital Signal Processing In Telecommunication" United State of America: Prentice-Hall International Editions,APRIL 1989

[4] มนัส สังวรศิลป์, วรรัตน์ ภัทรอมรกุล, "คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์", สำนักพิมพ์อินโฟเพรส พิมพ์ครั้งที่ 1,มีนาคม 2543

[5] ชำนาญ ปัญญาใส, วัชรากร หนูทอง, "ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล", บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน),มกราคม 2547

ภาคผนวก

Encode4

```
x=wavread('C:\Documents and Settings\BanK\Desktop\arm.wav');
x=x*128;
x=round(x);
    d=0;        L=0;
    v=0;       xp=0;
    ss=16;      M=0;
    B3=zeros(1,length(x));
    B2=zeros(1,length(x));
    B1=zeros(1,length(x));
    B0=zeros(1,length(x));
    n=1

    d(n)=x(n)-0;

    if d(n)<0
        B3(n)=1;
        z=d(n)
        d(n)=abs(z)
    end
     if d(n)>=ss(n)
        B2(n)=1;
        d(n)=d(n)-ss(n);
     end
      if d(n)>=ss(n)/2
         B1(n)=1;
         d(n)=d(n)-(ss(n)/2);
      end
      if d(n)>=ss(n)/4
         B0(n)=1;
      end

      L(n)=(2^3*B3(n))+(2^2*B2(n))+(2^1*B1(n))+B0(n);

      if L(n)==15|L(n)==7;
            M(n)=8;
        elseif L(n)==14|L(n)==6;
            M(n)=6;
        elseif L(n)==13|L(n)==5;
            M(n)=4;
        elseif L(n)==12|L(n)==4;
            M(n)=2;
        else
             M(n)=-1;
        end

v(n)=(ss(n)*B2(n))+((ss(n)/2)*B1(n))+((ss(n)/4)*B0(n))+(ss(n)/8);
        if B3(n)==1;
           v(n)=v(n)*(-1);
        end
        xp(n)=0+v(n);
        ss(n+1)=ss(n)+1.1*M(n);
        er=x(n)-xp(n);
        er_r(n)=abs(er);

 for n=2:length(x)
        d(n)=x(n)-xp(n-1);
       if d(n)<0
```

```
        B3(n)=1;
        z=d(n);
        d(n)=abs(z);
        end
        if d(n)>=ss(n)
            B2(n)=1;
           d(n)=d(n)-ss(n);
        end
         if d(n)>=ss(n)/2
            B1(n)=1;
            d(n)=d(n)-(ss(n)/2);
         end
          if d(n)>=ss(n)/4
            B0(n)=1;
          end
        L(n)=(2^3*B3(n))+(2^2*B2(n))+(2^1*B1(n))+B0(n);
        if L(n)==15|L(n)==7;
            M(n)=8;
        elseif L(n)==14|L(n)==6;
            M(n)=6;
        elseif L(n)==13|L(n)==5;
            M(n)=4;
        elseif L(n)==12|L(n)==4;
            M(n)=2;
        else
             M(n)=-1;
        end

v(n)=(ss(n)*B2(n))+((ss(n)/2)*B1(n))+((ss(n)/4)*B0(n))+(ss(n)/8);
        if B3(n)==1;
          v(n)=v(n)*(-1);
        end
        xp(n)=xp(n-1)+v(n);
        ss(n+1)=ss(n)+1.1*M(n);
        er=x(n)-xp(n);
        er_r(n)=abs(er);
        end
      close all
    sound(x,8000);pause
    sound(L(n),8000);pause
    sound(xp,8000);
    figure(1)

    xlabel('sample');
    ylabel('amplitude');
    subplot(3,1,1);
    plot(x,'k');
       axis([0 length(x) -150 150])
       hold
       subplot(3,1,2);
      plot(xp,'r');
       axis([0 length(x) -150 150])
       subplot(3,1,3);
       plot(L,'b');
       axis([0 length(x) -20 20 ])
```

Test_adpcm

```
clc;
clear all;


xn=wavread('C:\Documents and Settings\BanK\Desktop\heng.wav');
%x=0.8*(x/max(x));
x=xn;
p = round(length(x)/1000);

for n=1: p%length(x)
     if x(n)<0
         x(n)=x(n)+2;
     else
         x(n)=x(n);
     end;

end;

x=x*128;
x=round(x);

%k=[1:1:99];
%f=0.02;
%x=1+(0.4*sin(2*pi*k*f));
%x=x*128;
%x=round(x);
%xpn(1)=0;

s = serial('COM1');
set(s,'BaudRate',9600);
set(s,'Databits',8);
fopen(s);

m=0;
for n=1:p%length(x)

      fwrite(s,x(n));
      rev=fread(s,4);
      xpn(n)=rev(1);

      na=n/2;
      naa=fix(na);
      nb=naa-na;
      if nb == 0
         m=m+1;
         ln(m)=rev(2);
      end;
      dn(n)=rev(3);
      ss(n)=rev(4);


end;

fclose(s)
delete(s)
clear s
instrreset
```

```
for n=1:p%length(x)
    if xpn(n)>127
        xpna(n)=xpn(n)-256;
    else xpna(n)=xpn(n);
    end;

    if dn(n)>127
        dna(n)=dn(n)-256;
    else dna(n)=dn(n);
    end;

end;

xpnaA=xpna;

xpnaA=xpnaA*((2^(-7)));

 dnaA=dna;

dnaA=dnaA*((2^(-7)));

save c:\ln_ha.dat ln;
save c:\xn_ha.dat xn;
save c:\xpn_ha.dat xpn;
save c:\xpnaA_ha.dat xpnaA;
save c:\dn_ha.dat dn;
save c:\ss_ha.dat ss;
close all;


figure(1)
plot(xn)
axis([0 length(x) -1 1])
figure(2)
plot(xpn)
axis([0 length(x) -150 150])
figure(3)
plot(xpnaA)
 axis([0 length(x) -1 1])
figure(4)
plot(dnaA)
 axis([0 length(x) -1 1])
figure(5)
plot(ss)
 axis([0 length(x) -50 50])
```

Add_dn

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_unsigned.ALL;
Entity add_dn is
      port( xn,xpn  : in  std_logic_vector(7 downto 0);
            en      : in  std_logic;
            dn      : out std_logic_vector(7 downto 0));
End;
Architecture rtl of add_dn is
  begin
    Process(xn,xpn,en)
      begin
        if  en='0' then
            dn<=xn+xpn;
        end if;
    end process;
end;
```

Decoder

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;

 Entity decoder is
      port( Ln_decode             : in   std_logic_vector(3 downto
0);
              step                                : in
std_logic_vector(7 downto 0);
          clk_decode            : in   std_logic;
              xp_decode             : out  std_logic_vector(7 downto
0));

    end;

Architecture rtl  of decoder is
    Signal   state    :  integer range 0 to 7;
begin
Process(Ln_decode,step,clk_decode)
      Variable vn_decode_con    : std_logic_vector(7 downto 0);
      Variable step_1con : std_logic_vector(7 downto 0);
      Variable step_2con : std_logic_vector(7 downto 0);
      Variable step_3con : std_logic_vector(7 downto 0);
      Variable step_4con : std_logic_vector(7 downto 0);
      Variable Ln_decode_con            : std_logic_vector(3 downto
0);
      Variable xp_decode_delay_con     : std_logic_vector(7 downto
0);
      Variable xp_decode_con            : std_logic_vector(7 downto
0);

        BEGIN
          IF clk_decode'event and clk_decode ='0' then
            CASE state IS
                  When 0 =>       state<=1;
                              xp_decode_delay_con:=xp_decode_con;
                                          Ln_decode_con :=
Ln_decode;
                                          step_1con := step;
                                          step_2con := step;
                                          step_3con := step;
                                          step_4con := step;
                                          vn_decode_con    :=
(others=>'0');

                  WHEN 1 =>  if   Ln_decode_con(2) = '1' then
                                    step_1con := step_1con;
                       else
                                             step_1con
:=(others=>'0');
                                  end if;

                                   if   Ln_decode_con(1) = '1' then
                                 step_2con(6 downto 0):=step_2con(7
downto 1);
step_2con(7):=step_2con(7);
```

```
                        else
                                        step_2con
:=(others=>'0');
                        end if;

                             if   Ln_decode_con(0) = '1' then
                            step_3con(5 downto 0):=step_3con(7
downto 2);

step_3con(7):=step_3con(7);

step_3con(6):=step_3con(7);
                        else
                                        step_3con
:=(others=>'0');
                             end if;

                        step_4con(4 downto 0):=step_4con(7 downto
3);
                                  step_4con(7):=step_4con(7);
                                  step_4con(6):=step_4con(7);
                                  step_4con(5):=step_4con(7);

                        state<=2;

                   WHEN 2 =>  state<=3;

vn_decode_con:=step_4con+step_3con+step_2con+step_1con;

                   WHEN 3 =>  state<=4;
                        if   Ln_decode_con(3) = '1' then
                             vn_decode_con:=(vn_decode_con xor
"11111111");
                                  vn_decode_con:=vn_decode_con+1;
                        end if;

                   WHEN others=>  state<=0;
                             xp_decode_con    :=
xp_decode_delay_con+vn_decode_con;
                             xp_decode        <=
xp_decode_delay_con+vn_decode_con;
                  END CASE;
              END IF;
    END PROCESS ;
END;
```

Div1000

```
library ieee;
use ieee.std_logic_1164.all;
entity Div1000 is
port( Clk : in std_logic;
        Clk_out : out std_logic );
end ;

architecture rtl of Div1000 is
begin
process(Clk)
variable Clk_temp : std_logic := '0';
variable count : integer range 0 to 499 := 0;
begin
      if Clk'Event and Clk = '1' then
            if count < 499 then
                  count := count + 1;
                  Clk_temp := Clk_temp;
            else
                  count := 0;
                  Clk_temp := not(Clk_temp);
            end if;
            Clk_out <= Clk_temp;
      end if;
end process;
end rtl;
```

encoder

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;

 Entity encoder is
      port( dn_encode,ss_encode  : in   std_logic_vector(7 downto
0);
          clk_encode           : in   std_logic;
              Ln_encode            : out  std_logic_vector(3 downto
0));
    end;


Architecture rtl  of encoder is
    Signal   state    :  integer range 0 to 15;
    Signal   sta_con  :  integer range 0 to 1;

begin
Process(clk_encode,dn_encode,ss_encode)
      Variable dn_encode_con   :  std_logic_vector(7 downto 0);
      Variable ss_encode_con   :  std_logic_vector(7 downto 0);
      Variable Ln_encode_con   :  std_logic_vector(3 downto 0);

        BEGIN
         IF clk_encode'event and clk_encode ='0' then
            CASE state IS
                  When 0 =>       state<=1;
                                  dn_encode_con  := dn_encode;
                                  ss_encode_con  := ss_encode;
                                  Ln_encode_con  :=(others=>'0');

                  WHEN 1 =>  if   dn_encode_con(7) = '1' then
                             state<=2;
                                        sta_con<=0;
                             else state<=2;
                                        sta_con<=1;
                             end if;

                 WHEN 2 =>   state<=3;
                             if  sta_con=0 then
                                   dn_encode_con:=(dn_encode_con xor
"11111111");

      dn_encode_con:=dn_encode_con+1;
                                                  Ln_encode_con(3) :=
'1';
                        else
                              Ln_encode_con(3) := '0';
                        end if;


                  WHEN 3 =>  if   dn_encode_con >= ss_encode_con
then
                           state<=4;
                  ss_encode_con:=(ss_encode_con xor "11111111");
                                ss_encode_con:=ss_encode_con+1;
                                        sta_con<=0;
```

```
                    else state<=4;
                                        sta_con<=1;
                         end if;

                WHEN 4 =>  state<=5;
                    if  sta_con=0 then
                                    Ln_encode_con(2) := '1';

dn_encode_con:=dn_encode_con+ss_encode_con;
                                    else
                                Ln_encode_con(2) := '0';
                    end if;

                WHEN 5 =>  state<=6;
                    ss_encode_con(6 downto 0):=ss_encode_con(7
downto 1);

                WHEN 6 =>  if   dn_encode_con >= ss_encode_con
then
                         state<=7;
                                        sta_con<=0;
                         ss_encode_con:=(ss_encode_con xor
"11111111");
                                ss_encode_con:= ss_encode_con+1;
                    else state<=7;
                                        sta_con<=1;
                    end if;

                WHEN 7 =>  state<=8;
                    if  sta_con=0 then
                                    Ln_encode_con(1) := '1';

      dn_encode_con:=dn_encode_con+ss_encode_con;
                                    else
                                Ln_encode_con(1) := '0';
                    end if;


                WHEN 8 =>  state<=9;
              ss_encode_con(6 downto 0):=ss_encode_con(7 downto
1);

                WHEN 9 =>  if   dn_encode_con >= ss_encode_con
then
                         state<=10;
                                    Ln_encode_con(0) := '1';

                    else state<=10;
                         Ln_encode_con(0):='0';
                    end if;

               WHEN others=>  state<=0;
                           Ln_encode<=Ln_encode_con;
               END CASE;
          END IF;
    END PROCESS ;
END;
```

reciver

```
library ieee;
use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;

Entity reciver is
Port( clk_96kHZ        : in  std_logic;
        RX               : in  std_logic;
      RX_SS            : out std_logic;
        DATA_RX          : out std_logic_vector(7 downto 0));
End;

Architecture rtl of reciver is
      signal State_RX : integer range 0 to 7 := 0;
  Begin
      Process(clk_96kHZ,RX)
           variable RX_Data_Count : integer range 0 to 7;
           variable Buffer_RX     : std_logic_vector(7 downto 0);
        variable wait_count    : integer range 0 to 15;
           variable acc_var       : std_logic_vector(3 downto 0);

        Begin

       if clk_96kHZ'Event and clk_96kHZ = '1' then
                CASE State_RX is

                 WHEN 0 =>      RX_SS <= '0';
                                wait_count:=0;
                                acc_var:=(others=>'0');
Buffer_RX:=(others=>'0');
                                 if RX = '0' then
                                RX_Data_Count := 0;
                                State_RX <= 1;
                               else
                                RX_Data_Count := 0;
                                State_RX <= 0;
                               end if;

                WHEN 1 =>   if wait_count = 9 then
                                wait_count := 0;
                           State_Rx <= 2;
                                else
                           wait_count:=wait_count+1;

                                end if;

               WHEN 2 =>    if wait_count = 9 then
                           State_Rx <= 3;
                           wait_count := 1;
                           acc_var:=acc_var+RX;
                                else
                           wait_count:=wait_count+1;
                           acc_var:=acc_var+RX;
                                end if;

              WHEN 3 =>    if RX_Data_Count = 7 then
                                RX_SS <= '0';
```

```
                    State_Rx <= 4;
                if acc_var >= "101" then
                    Buffer_RX(RX_Data_Count) := '1';
                    acc_var:=(others=>'0');
                else
                    Buffer_RX(RX_Data_Count) := '0';
                    acc_var:=(others=>'0');
                end if;
                    RX_Data_Count:=RX_Data_Count+1;
                        else

                    State_Rx <= 2;
                if acc_var >= "101" then
                    Buffer_RX(RX_Data_Count) := '1';
                    acc_var:=(others=>'0');
                else
                    Buffer_RX(RX_Data_Count) := '0';
                    acc_var:=(others=>'0');
                end if;
                    RX_Data_Count:=RX_Data_Count+1;
                        end if;

            when 4 =>    RX_Data_Count := 0;
                                DATA_RX <=
Buffer_RX;
                                RX_SS <= '1';
                                State_RX <= 5;

            when others =>
                                RX_Data_Count := 0;
                                RX_SS <= '0';
                                State_RX <= 0;

            end case;

        end if;
    end process;
end;
```

Twocom

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity twoscomp is
Port( A    : in  std_logic_vector(7 downto 0);
      B    : out std_logic_vector(7 downto 0));
End;

Architecture rtl of twoscomp is
Begin

      Process (A)
            Variable s : std_logic_vector(7 downto 0);
                  Begin
                  s:=(A xor "11111111");
                B<=s+1;
    End Process;
End;
```

Buff

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity buff is
Port( buff_in   : in  std_logic_vector(7 downto 0);
      lr        : in  std_logic;
      buff_out  : out std_logic_vector(7 downto 0));
End;

Architecture rtl of buff is
  Begin
    Process(buff_in,lr)
       begin
         if lr'event and lr = '0'  then
             buff_out<=buff_in;
         end if;
       End process;
End;
```

Control_adpcm

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;

Entity control_adpcm is
Port (sys_clk,en_sys  : in  std_logic;
      en_int          : out std_logic;
        clk_encode      : out std_logic;
        clk_decode      : out std_logic;
        clk_ss          : out std_logic;
      lr,lr_out       : out std_logic;
      ln_sh           : out std_logic;
      add_xn_xpn      : out std_logic);
End ;

Architecture rtl  of control_adpcm is
    signal  int_con : std_logic;
    signal  state   : integer range 0 to 15;
    signal  count   : integer range 0 to 15;

  Begin
    PROCESS(sys_clk,en_sys)
        BEGIN

        if en_sys='1' then
          state<=0;
         elsif sys_clk'event and sys_clk='1' then
            CASE state IS
              WHEN 0 =>          state<=1;
                              en_int      <= int_con;
                          clk_encode  <= '1';
                          clk_decode  <= '1';
                            clk_ss      <= '1';
                               add_xn_xpn  <= '1';
                              lr          <= '1';
                                 lr_out      <= '0';
                             count       <=  0 ;
                                  ln_sh       <= '0';

                WHEN 1 =>          state<=2;
                                      en_int      <= int_con;
                                 clk_encode  <= '1';
                                 clk_decode  <= '1';
                                 clk_ss      <= '1';
                                lr          <= '0';
                                 add_xn_xpn  <= '1';

                WHEN 2 =>          state<=3;
                                      en_int      <= int_con;
                                      clk_encode  <= '1';
                                         clk_decode  <= '1';
                                 clk_ss      <= '1';
                                lr          <= '1';
                                 add_xn_xpn  <= '0';

                WHEN 3 =>          state<=4;
                                      en_int      <= int_con;
```

```
                    clk_encode  <= '1';
                    clk_decode  <= '1';
                       clk_ss       <= '1';
                lr                    <= '1';
                       add_xn_xpn   <= '1';


WHEN 4 =>       if count=10 then
                   state<=5;
                       en_int       <= int_con;
                       clk_encode   <= '0';
                          clk_decode   <= '1';
                      clk_ss       <= '1';
                          add_xn_xpn   <= '1';
                   count          <=  0 ;
                else
                   state<=3;
                       en_int       <= int_con;
                          clk_encode   <= '0';
                          clk_decode   <= '1';
                          clk_ss       <= '1';
                          add_xn_xpn   <= '1';
                   count          <=count+1;
                end if;

WHEN 5 =>          state<=6;
                          en_int       <= int_con;
                          clk_encode   <= '1';
                          clk_decode   <= '1';
                          clk_ss       <= '1';
                      add_xn_xpn   <= '1';

WHEN 6 =>       if count=5 then
                   state<=7;
                          en_int       <= int_con;
                          clk_encode   <= '1';
                          clk_decode   <= '0';
                          clk_ss       <= '1';
                          add_xn_xpn   <= '1';
                   count          <=0;
                else
                   state<=5;
                          en_int       <= int_con;
                          clk_encode   <= '1';
                          clk_decode   <= '0';
                          clk_ss       <= '1';
                          add_xn_xpn   <= '1';
                   count          <=count+1;
                end if;

WHEN 7 =>          state<=8;
                          en_int       <= int_con;
                          clk_encode   <= '1';
                          clk_decode   <= '1';
                          clk_ss       <= '0';
                          add_xn_xpn   <= '1';

WHEN 8 =>       if count=4 then
```

```
                    state<=9;
                            en_int      <= int_con;
                            clk_encode  <= '1';
                            clk_decode  <= '1';
                            clk_ss      <= '1';
                            lr_out      <= '0';
                            add_xn_xpn  <= '1';
                    count       <=  0;
                else
                    state<=8;
                          en_int      <= int_con;
                            clk_encode  <= '1';
                            clk_decode  <= '1';
                            clk_ss      <= '1';
                            lr_out      <= '0';
                            add_xn_xpn  <= '1';
                    count       <=count+1;
                end if;

WHEN 9 =>           state<=10;
                            int_con     <= '1';
                            clk_encode  <= '1';
                            clk_decode  <= '1';
                            clk_ss      <= '1';
                            lr_out      <= '0';
                            add_xn_xpn  <= '1';
                           ln_sh        <= '1';

WHEN 10 =>          state<=11;
                            int_con     <= '1';
                            clk_encode  <= '1';
                            clk_decode  <= '1';
                            clk_ss      <= '1';
                            lr_out      <= '0';
                            add_xn_xpn  <= '1';
                          ln_sh         <= '0';
                    count       <=  0;


WHEN 11 =>          state<=12;
                            int_con     <= '1';
                            clk_encode  <= '1';
                            clk_decode  <= '1';
                            clk_ss      <= '1';
                            lr_out      <= '1';
                            add_xn_xpn  <= '1';
                          ln_sh         <= '0';
                    count       <=  0;

WHEN others =>      state<=12;
                    en_int      <= int_con;
                            clk_encode  <= '1';
                            clk_decode  <= '1';
                            clk_ss      <= '1';
                            add_xn_xpn  <= '1';
                    lr          <= '1';
                            lr_out      <= '0';
                    count       <=  0 ;
```

```
                                        ln_sh       <= '0';

                    END CASE;
                END IF;
    END PROCESS ;
END;
```

Div100

```
architecture rtl of Div100 is
begin
process(Clk)
variable Clk_temp : std_logic := '0';
variable count : integer range 0 to 49 := 0;
begin
      if Clk'Event and Clk = '1' then
            if count < 49 then
                  count := count + 1;
                  Clk_temp := Clk_temp;
            else
                  count := 0;
                  Clk_temp := not(Clk_temp);
            end if;
            Clk_out <= Clk_temp;
      end if;
end process;
end rtl;
```

En_int

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;

Entity en_int is
Port( A     : in  std_logic_vector(7 downto 0);
      sel   : in  std_logic;
      C     : out std_logic_vector(7 downto 0));
End;

Architecture rtl of en_int is
  Begin
    Process(A,sel)
       begin
         if   sel = '1'  then
              C<=A;
         else
              C<="00010000";
         end if;
       End process;
  End;
```

Pipo_ln

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity pipo_ln is
  port( lr          : in  std_logic;
        ln_in       : in  std_logic_vector(3 downto 0);
        ln_out      : out std_logic_vector(7 downto 0));
  End;

Architecture rtl of pipo_ln is
begin
process (lr,ln_in)
   variable sh : std_logic_vector(7 downto 0);
  begin
   if lr'event and lr='1' then
         sh(7 downto 4):= sh(3 downto 0);
         sh(3 downto 0):= ln_in;
         ln_out<=sh;
   end if;
end process;
end;
```

Step_up

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_unsigned.ALL;
Entity step_up is
      port( M,con_int : in  std_logic_vector(7 downto 0);
            clk_ss    : in  std_logic;
            ss_n      : out std_logic_vector(7 downto 0));
End;
Architecture rtl of step_up is
         signal state : integer range 0 to 1;
begin
    Process(M,con_int,clk_ss)
             variable ss_con : std_logic_vector(7 downto 0);
       begin
         if clk_ss'event and clk_ss = '0'  then
                   CASE state IS
                        WHEN 0 =>              state<=1;

      ss_con:=con_int+M;
                                              ss_n<=ss_con;

                        WHEN OTHERS =>  ss_con:=ss_con+M;
                                              ss_n<=ss_con;
                   END CASE;
         end if;
    End process;
end;
```

Xmit2

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_unsigned.ALL;
Entity xmit2 is
      port( BAUD_RATE     : in std_logic;
            TX_EN           : in std_logic;
            DATA_aTX      : in std_logic_vector(7 downto 0);
            DATA_bTX      : in std_logic_vector(7 downto 0);
            DATA_cTX      : in std_logic_vector(7 downto 0);
            DATA_dTX        : in std_logic_vector(7 downto 0);
            TX              : out std_logic
        );
      end;

Architecture rtl of xmit2 is
type State_Type_TX is
(Idel1,Start1,TransData1,Stop1,Idel2,Start2,TransData2,Stop2,

Idel3,Start3,TransData3,Stop3,Idel4,Start4,TransData4,Stop4);
signal State_TX : State_Type_TX;
begin
    process(BAUD_RATE,TX_EN,DATA_aTX,DATA_bTX)
            variable TX_Data_Count : integer range 0 to 7 := 0;
            begin
        if TX_EN = '1' then
            State_Tx<=Start1;
            elsif BAUD_RATE'Event and BAUD_RATE = '1' then
                case State_TX is
                when Idel1 =>
                        TX <= '1';
                                    TX_Data_Count := 0;
                                    State_TX <= Idel1;

                when Start1 =>
                                    TX <= '0';
                                    TX_Data_Count := 0;
                                    State_TX <= TransData1;

            when TransData1 =>
                                    if TX_Data_Count = 7 then
                                       TX <= DATA_aTX(TX_Data_Count); --xpn
                                       State_TX <= Stop1;
                                    else
                                       TX <= DATA_aTX(TX_Data_Count);
                                       TX_Data_Count := TX_Data_Count + 1;
                                       State_TX <= TransData1;
                                    end if;
                when Stop1 =>
                                       TX <= '1';
                                       TX_Data_Count := 0;
                                       State_TX <= Idel2;
-------------------------------------------------------------------
-----
                when Idel2=>
                        TX <= '1';
                                    TX_Data_Count := 0;
                                    State_TX <= Start2;
```

```
                when Start2 =>
                            TX <= '0';
                            TX_Data_Count := 0;
                            State_TX <= TransData2;

            when TransData2 =>
                            if TX_Data_Count = 7 then
                               TX <= DATA_bTX(TX_Data_Count); --Ln
                               State_TX <= Stop2;
                            else
                               TX <= DATA_bTX(TX_Data_Count);
                               TX_Data_Count := TX_Data_Count + 1;
                               State_TX <= TransData2;
                            end if;
              when Stop2 =>
                               TX <= '1';
                               TX_Data_Count := 0;
                               State_TX <= Idel3;
-------------------------------------------------------------------
-----------
                when Idel3 =>
                               TX <= '1';
                               TX_Data_Count := 0;
                               State_TX <= Start3;

                when Start3 =>
                            TX <= '0';
                            TX_Data_Count := 0;
                            State_TX <= TransData3;

            when TransData3 =>
                            if TX_Data_Count = 7 then
                               TX <= DATA_cTX(TX_Data_Count); --dn
                               State_TX <= Stop3;
                            else
                               TX <= DATA_cTX(TX_Data_Count);
                               TX_Data_Count := TX_Data_Count + 1;
                               State_TX <= TransData3;
                            end if;
              when Stop3 =>
                               TX <= '1';
                               TX_Data_Count := 0;
                               State_TX <= Idel4;
-------------------------------------------------------------------
-----------
                when Idel4 =>
                            TX <= '1';
                                  TX_Data_Count := 0;
                                  State_TX <= start4;

                when Start4 =>
                            TX <= '0';
                            TX_Data_Count := 0;
                            State_TX <= TransData4;
            when TransData4 =>
```

```
                                    if TX_Data_Count = 7 then
                                       TX <= DATA_dTX(TX_Data_Count); --
ss_out
                                       State_TX <= Stop4;
                                    else
                                       TX <= DATA_dTX(TX_Data_Count);
                                       TX_Data_Count := TX_Data_Count + 1;
                                       State_TX <= TransData4;
                                    end if;
                        when Stop4 =>
                                       TX <= '1';
                                       TX_Data_Count := 0;
                                       State_TX <= Idel1;

                          when others =>
                                       TX_Data_Count := 0;
                                       TX <= '1';
                                       State_TX <= Idel1;

                          end case;
end if;
end process;
end;
```