การเข้ารหัสสัญญาณเสียงแบบ Real-time สำหรับมาตรฐาน ITU-T G.723.1 บน DSP chip TMS320C6713

REAL-TIME IMPLEMENTATION OF ITU-T G.723.1 STANDARD VOICE CODEC ON DSP CHIP TMS320C6713

นายธนิต นพพันธ์

นายนริศร พงษ์พานิช

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การเข้ารหัสสัญญาณเสียงแบบ Real-time สำหรับมาตรฐาน ITU-T G.723.1 บน DSP chip TMS320C6713

REAL-TIME IMPLEMENTATION OF ITU-T G.723.1 STANDARD VOICE CODEC ON DSP CHIP TMS320C6713

โดย

นายธนิต นพพันธ์

นายนริศร พงษ์พานิช

อาจารย์ที่ปรึกษา

ผศ.ดร.อรฉัตร จิตต์โสภักตร์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชา วิศวกรรมคอมพิวเตอร์

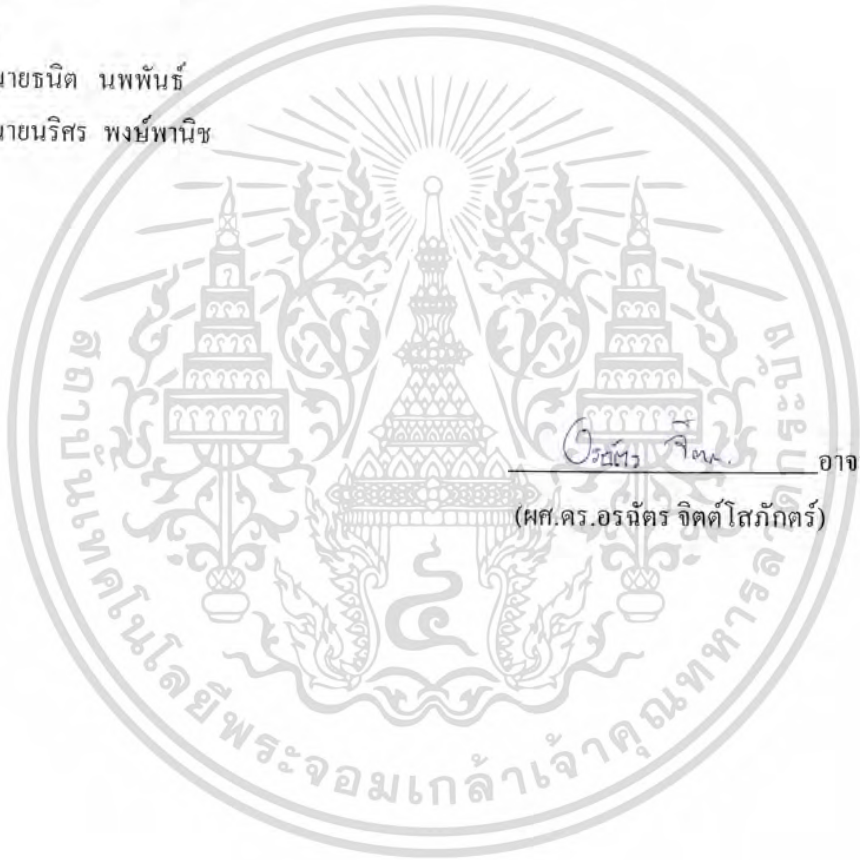
คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การเข้ารหัสสัญญาณเสียงแบบ Real-time สำหรับมาตรฐาน ITU-T G.723.1 บน DSP chip TMS320C6713

REAL-TIME IMPLEMENTATION OF ITU-T G.723.1 STANDARD VOICE CODEC ON DSP CHIP TMS320C6713

คณะผู้จัดทำ

1. นายธนิต นพพันธ์
2. นายนริศร พงษ์พานิช



อรฉัตร จิตต์โสภัณฑ์ อาจารย์ที่ปรึกษา

(ผศ.ดร.อรฉัตร จิตต์โสภัณฑ์)

การเข้ารหัสสัญญาณเสียงแบบ Real-time สำหรับมาตรฐาน ITU-T G.723.1 บน DSP chip TMS320C6713

นายธนิต นพพันธ์ 44010203
นายนริศร พงษ์พานิช 44010243
ผศ.ดร.อรฉัตร จิตต์โสภักตร์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

ในการส่งข้อมูลภาพและเสียงผ่านเครือข่ายในปัจจุบัน นิยมใช้การบีบอัดข้อมูลเสียงเพื่อลดอัตราการส่งข้อมูล วิธีการบีบอัดข้อมูลเสียงโดยใช้มาตรฐาน ITU G.723.1 เป็นวิธีที่แพร่หลายเพราะเหมาะสมกับเสียงพูดของมนุษย์

วิธีการบีบอัดข้อมูลเสียงโดยใช้มาตรฐาน ITU G.723.1 นั้น มีความซับซ้อนและการคำนวณมากมาย จึงใช้เวลาในการประมวลผลมาก เมื่อนำไปใช้กับระบบที่เป็น Real-time จะไม่สามารถทำงานได้ทันเวลา เพราะว่าการบีบอัดแต่ละช่วงนั้นต้องกระทำในช่วงเวลาที่จำกัด

มาตรฐาน G.723.1 จึงถูกนำมาประยุกต์ใช้กับหน่วยประมวลผลทางเสียงซึ่งมีการประมวลผลที่รวดเร็ว และมีสถาปัตยกรรมเหมาะสมกับการทำงานโดยเลือกใช้ DSP Chip TMS320C6713 นอกจากนี้เพื่อลดความซับซ้อนในการประมวลผล จึงปรับแต่งมาตรฐานเพื่อให้ใช้เวลาในการเข้ารหัสสัญญาณเสียงน้อยลงและปรับปรุงโปรแกรม G.723.1 ให้รองรับการทำงานแบบหลายช่องสัญญาณได้อย่างมีประสิทธิภาพ

Real-time Implementation of ITU-T G.723.1 Standard Voice Codec on DSP chip TMS320C6713

Thanit Noppan
Narisorn Pongpanich
Asst.Prof.Dr.Orachat Chitsobhuk Advisor

Abstract

Transmitting voice along the network in this modern-day usually uses voice compression to reduce data to be sent in transition line. G.723.1 voice compression algorithm is widely used because it is suitable for human voice.

G.723.1 algorithm requires high computational complexity. It uses large amount of time to compute. As a result, processor must be chosen carefully in order to implement this algorithm in real-time system. TMS320C6713 DSP Chip is an alternative solution appropriate for signal processing because of its powerful and high computational speed. Besides choosing right processor, G.723.1 algorithm optimization should be performed in order to further decrease computation load.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้เป็นอย่างดีด้วยคำแนะนำ คำปรึกษา และคอยดูแลจากหลายๆฝ่ายด้วยกัน โดยเฉพาะอาจารย์ที่ปรึกษาที่ให้โอกาสข้าพเจ้าได้ทำปริญญานิพนธ์ฉบับนี้ คอยให้ความเอาใจใส่ดูแล แนะนำและให้ความช่วยเหลือเสมอมา คือ ผศ.ดร.อรฉัตร จิตต์โสภักตร์ รวมทั้งกลุ่มบุคคลนักวิจัยของศูนย์เทคโนโลยีและคอมพิวเตอร์แห่งชาติ(เนคเทค) ที่ได้ให้คำปรึกษาด้านหลักการ และการพัฒนาโปรแกรม

นอกจากนี้ ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้จัดเตรียมสถานที่และสิ่งอำนวยความสะดวก เพื่อให้การวิจัยและพัฒนาโปรแกรมเป็นไปด้วยความสะดวกและรวดเร็วรวมทั้งยังมีอินเทอร์เน็ตคอยให้บริการสำหรับค้นคว้าหาข้อมูลต่างๆ ซึ่งท้ายที่สุดแล้วก็ได้ประกอบเป็นส่วนหนึ่งของโครงงานนี้

และสุดท้ายขอขอบคุณบุคคลที่สำคัญที่สุดในชีวิตที่ทำให้ข้าพเจ้ามีวันนี้ นั่นคือ บิดา มารดา และบุคคลในครอบครัว อันเป็นที่เคารพรักซึ่งได้เลี้ยงดู คอยสั่งสอนข้าพเจ้ามาเป็นอย่างดี พร้อมให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ ความรักเสมอมา ข้าพเจ้าขอกราบขอบพระคุณมา ณ ที่นี้ด้วย

ธนิต นพพันธ์

นริศร พงษ์พานิช

สารบัญ

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 หลักการและเหตุผล

เนื่องจากวิวัฒนาการที่ก้าวหน้าอย่างต่อเนื่องจากอดีตจนถึงปัจจุบัน วิวัฒนาการการสื่อสารทางโทรศัพท์นั้นได้รับการพัฒนามาเรื่อยๆอย่างต่อเนื่องเพื่อให้ตอบสนองความต้องการมากที่สุด เมื่อพูดถึงการสื่อสารทางโทรศัพท์นั้น ระบบบันทึกเสียงโทรศัพท์ถือเป็นพื้นฐานของโทรศัพท์เลยก็ว่าได้

จากเหตุผลข้างต้นจึงมีความต้องการพัฒนาการใช้งานแบนด์วิดท์ในการส่งข้อมูลอย่างมีประสิทธิภาพและมีการรักษาความลับข้อมูลมากขึ้น โดยเฉพาะอย่างยิ่งในระบบโทรศัพท์เคลื่อนที่และระบบส่งสัญญาณผ่านดาวเทียม นอกจากนี้อัตราแนวโน้มที่เพิ่มสูงขึ้นของการผสมผสานการใช้เสียงในโปรแกรมประยุกต์ที่ทำงานบนเครื่องคอมพิวเตอร์ทั้งแบบตั้งโต๊ะ และแบบพกพา ซึ่งมักพบเห็นได้ในระบบสื่อสารแบบมัลติมิเดีย โปรแกรมเหล่านี้ส่วนใหญ่ต้องการสัญญาณเสียงที่อยู่ในรูปของสัญญาณดิจิตอลเพื่อให้สามารถประมวลผล จัดเก็บ และส่งต่อได้อย่างสะดวก ภายใต้การควบคุมของซอฟต์แวร์ อย่างไรก็ตามแม้ว่าสัญญาณเสียงดิจิตอลมีความยืดหยุ่นและเหมาะสมกับการเข้ารหัส แต่มีข้อเสียตรงที่ปริมาณข้อมูลมากซึ่งส่งผลให้ต้องใช้แบนด์วิดท์ในการส่งและจัดเก็บในปริมาณที่สูงมากด้วย

เทคโนโลยีการบีบอัดเสียง (Speech coding / Speech compression) จึงก้าวเข้ามามีบทบาทในการแก้ไขจุดบกพร่องเหล่านี้ เพื่อใช้ในการแทนสัญญาณเสียงให้อยู่ในรูปสัญญาณดิจิตอลที่มีขนาดเล็ก ทำให้มีการใช้งานแบนด์วิดท์ในการส่งข้อมูลได้อย่างมีประสิทธิภาพ การบีบอัดสัญญาณเสียงพูดนี้เกี่ยวกับการสุ่มและการแทนเสียงสัญญาณ (Quantization signal) เนื่องจากการสุ่มสัญญาณถูกจำกัดด้วยหลักการของ Nyguist (ถ้าไม่ต้องการให้สัญญาณผิดเพี้ยน ความถี่ที่สุ่มหรือ sampling ต้องมากกว่าหรือเท่ากับสองเท่าของความถี่สูงสุดของสัญญาณที่จะ sampling) ด้วยเหตุนี้จึงไม่สามารถลดค่าความถี่ในการสุ่มลงได้ ดังนั้นการค้นคว้านี้จึงมุ่งเน้นไปที่การแทนรูปแบบเสียงสัญญาณ (Quantization)

1.2 วัตถุประสงค์

1.2.1 เพื่อศึกษาหลักการการเบื้องต้นของการกำเนิดเสียง-การเข้ารหัสสัญญาณเสียง

1.2.2 เพื่อศึกษาหลักการการเข้ารหัสสัญญาณเสียง สำหรับการติดต่อสื่อสารแบบมัลติมิเดีย ที่อัตราการส่ง 5.3 กิโลบิตต่อวินาที และ 6.3 กิโลบิตต่อวินาที ตามมาตรฐาน G.723.1

1.2.3 เพื่อศึกษาคู่มือการใช้งานและหลักการของ TMS320 DSP/BIOS

1.2.4 เพื่อศึกษาการใช้บอร์ด DSP TMS3206713 DSK

1.2.5 พัฒนามาตรฐาน G.723.1 เพื่อให้สามารถตอบสนองต่อการใช้งานและมีประสิทธิภาพในการประมวลผลมากขึ้น

1.2.6 ทำให้มาตรฐาน G.723.1 มีคุณสมบัติการทำงานแบบ Reentrant คือสามารถจัดสรรทรัพยากรหรือหน่วยความจำ(memory) เพื่อรองรับการทำงานแบบหลายช่องสัญญาณ(Multichannel)

1.2.7 ทำให้มาตรฐาน G.723.1 เป็นไปตาม Algorithm Standard เพื่อรองรับการนำไปใช้ร่วมกับมาตรฐานอื่นๆ

1.2.8 ทำให้มาตรฐาน G.723.1 สามารถรองรับการทำงานแบบเรียลไทม์หลายช่องสัญญาณ (Real - time Multichannel) ได้อย่างน้อย 4 ช่องสัญญาณ

1.2.9 ทำให้มาตรฐาน G.723.1 ง่ายต่อการเรียกใช้งานของผู้ใช้โดยการทำเป็น Library

1.3 ผลที่คาดว่าจะได้รับ

1.3.1 ได้รับความรู้ความเข้าใจเกี่ยวกับหลักการการเข้ารหัสสัญญาณเสียงตามมาตรฐานG.723.1

1.3.2 ได้รับความรู้ความเข้าใจเกี่ยวกับหลักการและการทำงานของ TMS320 DSP/BIOS

1.3.3 สามารถใช้ชุด TMS320C6713 DSK (บอร์ด DSP) ได้อย่างถูกต้องและมีประสิทธิภาพ

1.3.4 สามารถนำหลักการของ TMS320 DSP/BIOS มาใช้ร่วมกับการเข้ารหัสสัญญาณเสียงตามมาตรฐาน G.723.1 และสามารถนำมาพัฒนาต่อเพื่อให้สามารถตอบสนองต่อการใช้งานและมีศักยภาพในการประมวลผลมากขึ้น

1.3.5 มาตรฐาน G.723.1 สามารถทำงานแบบ Reentrant ได้

1.3.6 มาตรฐาน G.723.1 เป็นไปตาม Algorithm Standard

1.3.7 มาตรฐาน G.723.1 สามารถรองรับการทำงานแบบ Real-time Multichannel ได้

1.3.8 ทำให้มาตรฐาน G.723.1 เป็น Library เพื่อสะดวกต่อการเรียกใช้งาน

1.4 ขอบเขตของการพัฒนา

สำหรับโครงงานที่จัดทำขึ้นนั้นจะใช้ภาษาซี และพัฒนาด้วย TMS320 DSP/BIOS

1.4.1 การทำงานของโปรแกรม G.723.1 เดิม

1.4.1.2 สามารถส่งสัญญาณเสียงออกได้เพียง 1 ช่องสัญญาณ

1.4.1.3 สามารถทำงานได้บนเครื่องคอมพิวเตอร์เท่านั้น

1.4.2 ขอบเขตของโปรแกรม G.723.1 ที่ทำการพัฒนา

1.4.2.1 พัฒนาโปรแกรมให้ทำงานกับบอร์ด DSP ได้อย่างมีประสิทธิภาพ

1.4.2.2 สามารถทำงานประมวลผลบน TMS320C6713 board (บอร์ด DSP) ได้อย่างมีประสิทธิภาพ

1.4.2.3 สามารถประมวลผลสัญญาณเสียงได้แบบหลายช่องสัญญาณ

1.4.2.4 สามารถรองรับการทำงานร่วมกับมาตรฐานอื่นๆ ได้อย่างมีประสิทธิภาพ

1.4.2.5 สามารถรองรับการทำงานแบบ Real-time Multichannel อย่างน้อย 4 channels

1.4.2.6 ทำโปรแกรม G.723.1 ให้เป็น Library

1.5 วิธีการดำเนินงาน

1.5.1 ช่วงที่ 1 (ภาคการศึกษาที่ 1) กำหนดขอบเขต-เป้าหมายของโครงงาน ศึกษาหลักการการเข้าสัญญาณเสียงบนมาตรฐาน G.723.1

1.5.1.1 ศึกษาหลักการการทำงานของ Algorithm G.723.1

1.5.1.2 ทำการทดลองของ Algorithm บนเครื่องคอมพิวเตอร์ และตรวจสอบความถูกต้องของผลที่ได้จากการประผลโปรแกรม

1.5.1.3 ศึกษาโปรแกรมของ Algorithm G.723.1

1.5.2 ช่วงที่ 2 (ภาคการศึกษาที่ 2) ศึกษาหลักการของ TMS320 DSP/BIOS และนำมาใช้ร่วมกับการเข้ารหัสสัญญาณเสียงตามมาตรฐาน G.723.1 เพื่อให้เป็นไปตามขอบเขตของโครงงานที่ทำการพัฒนา ได้แก่ สามารถทำงานร่วมกับ TMS320C6713 board (บอร์ด DSP) ได้อย่างมีประสิทธิภาพและสามารถส่งสัญญาณเสียงออกได้อย่างน้อย 4 ช่องสัญญาณแบบ Real-time

1.5.2.1 ศึกษาการทำงานของบอร์ด DSP และการใช้งาน

1.5.2.2 แก้ไขโปรแกรมให้สามารถทำงานบนบอร์ด DSP ได้

1.5.2.3 ตรวจความถูกต้องของผลที่ได้จากการประมวลผลโปรแกรม

1.5.2.4 ใช้ Complier Optimization เพื่อปรับปรุงความเร็วในการ Compile

1.5.2.5 ตรวจความถูกต้องของผลที่ได้จากการประมวลผลโปรแกรม

1.5.2.6 แก้ไขโปรแกรมให้มีคุณสมบัติ Reentrant

1.5.2.7 ตรวจความถูกต้องของผลที่ได้จากการประมวลผลโปรแกรม

1.5.2.8 แก้ไขโปรแกรมให้เป็นไปตาม Algorithm Standard

1.5.2.9 ตรวจความถูกต้องของผลที่ได้จากการประมวลผลโปรแกรม

1.5.2.10 ทำการ Optimize โปรแกรม G.723.1 เพื่อรองรับการทำงานแบบ Real-time Multichannel

1.5.2.11 ตรวจความถูกต้องของผลที่ได้จากการประมวลผลโปรแกรม

1.5.2.12 ทำโปรแกรมให้เป็น Library เพื่อให้ง่ายต่อการเรียกใช้งาน

1.5.2.13 ตรวจความถูกต้องของผลที่ได้จากการประมวลผลโปรแกรม

บทที่ 2
ความรู้ทั่วไปในการเข้ารหัสสัญญาณเสียง

2.1 เสียง

คนเราเปล่งเสียงพูดด้วยอวัยวะที่ใช้ในการออกเสียง ออกเสียงตามภาษาของตน แม้ว่าคนที่อยู่ในสังคมเดียวกันจะใช้ภาษาเดียวกันแต่ถ้าพิจารณาเสียงที่เปล่งออกมาแล้ว แต่ละครั้งก็อาจจะสังเกตลักษณะที่ต่างกันได้ เราจึงสามารถจำเสียง จำวิธีพูดของคนที่เราคุ้นเคยได้ เสียงพูดจะอธิบายด้วยหลักเกณฑ์ทางวิทยาศาสตร์ แม้ว่าในภาษาหนึ่งๆจะมีเสียงต่างกันมากน้อยบ้าง แต่เสียงก็สามารถนำมาพิจารณาและให้คุณลักษณะการออกเสียงรวมถึงตำแหน่งที่เกิดเสียงได้ คำอธิบายนี้จะทำให้เข้าใจลักษณะเสียงทุกเสียง

2.1.1 อวัยวะที่ใช้ในการออกเสียง

อวัยวะที่ใช้ในการออกเสียงมีอยู่หลายส่วน แต่ละส่วนสามารถทำให้เกิดเสียงต่างกันออกไปได้ อวัยวะเหล่านี้มีปากและส่วนต่างๆในปาก ช่องคอ กล่องเสียง ช่องว่างในปาก และช่องว่างในจมูก อวัยวะที่ใช้ในการเปล่งเสียงพูดแบ่งออกเป็น 2 ประเภท คือ

2.1.1.1 อวัยวะที่ใช้ในการทำอาการ(Articulator) คืออวัยวะที่เคลื่อนไหว เพื่อผลักลมไปยังส่วนต่างๆ อวัยวะที่สำคัญคือ ลิ้น ซึ่งเป็นส่วนที่เคลื่อนไหวมากที่สุด

2.1.1.2 อวัยวะซึ่งเป็นตำแหน่งซึ่งเกิดเสียงต่างๆ (Point of Articulator) คือ ตำแหน่งที่เกิดเสียงต่างๆ เช่น ริมฝีปาก ฟัน เพดาน เป็นต้น

อวัยวะส่วนที่มีหน้าที่ออกเสียงโดยตรงมีดังนี้ ริมฝีปาก ฟัน ปุ่มเหงือก เพดานแข็ง เพดานอ่อน ลิ้นไก่ ลิ้น แผ่นเนื้อ ปาก หลอดลม ช่องคอ เส้นเสียง ช่องจมูก

2.1.2 การเกิดของเสียง

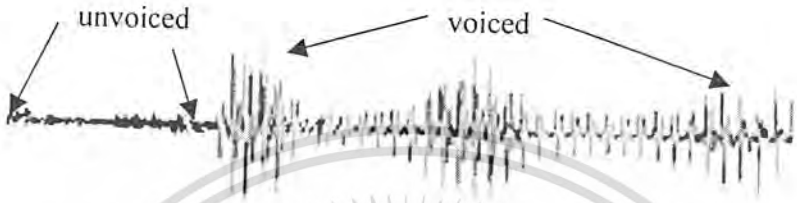
การเกิดของเสียงแบ่งเป็น 2 ขั้นตอน คือ

2.1.2.1 จุดเริ่มต้น เป็นขั้นตอนที่ลมเริ่มถูกขับออกจากปอด ผ่านเข้าไปสู่ขั้นตอนที่ 2

2.1.2.2 การดัดแปลงลมที่อวัยวะเส้นเสียง ที่ใช้ในขั้นตอนนี้คือส่วนที่ต่อจากปอดขึ้นมาถึงกล่องเสียง และที่กล่องเสียง เส้นเสียงจะทำหน้าที่ปิดเปิดทำให้เกิดเสียง 2 ชนิด คือ

1) Voiced เกิดจากเส้นเสียงปิดกั้นลมไว้ ลมที่ผ่านมาจะเพิ่มแรงดันมากขึ้นจนเส้นเสียงปิดเปิดสลับกัน ทำให้เกิดเสียงก้องขึ้นมา ซึ่งความถี่ในการปิด เปิด เส้นเสียงเรียกว่า "ความถี่มูลฐาน" มีลักษณะเป็นรายคาบ (Periodic) และมีค่าคาบของสัญญาณเสียง (Pitch Period) เป็นส่วนประกอบสำคัญ

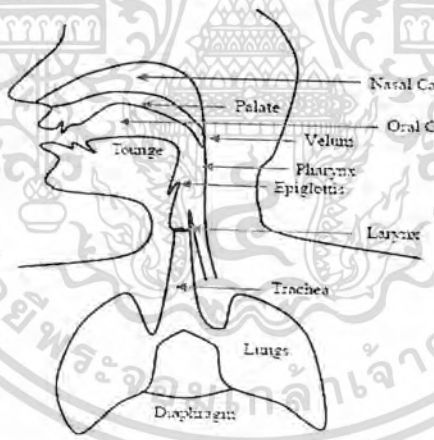
2) Unvoiced เสียงชนิดนี้เส้นเสียงจะเปิดตลอดเวลาที่ลมผ่าน ลมจึงผ่านออกมาได้สะดวกทำให้เกิดเสียงไม่ก้องขึ้น มีลักษณะไม่เป็นรายคาบ (Non Periodic) แต่จะมีลักษณะเป็นสัญญาณของเสียงรบกวน (Noise)



รูปที่ 2-1 แสดงองค์ประกอบของเสียง

2.1.3 การเปลี่ยนแปลงลักษณะเส้นเสียง

อวัยวะที่ใช้คือส่วนที่ต่อจากกล่องเสียงจนถึงริมฝีปาก โดยลมที่ผ่านออกจากกล่องเสียงจะทำให้เกิดเสียงในลักษณะต่างๆ ซึ่งเกิดจากการเปลี่ยนแปลงจากอวัยวะต่างๆ



รูปที่ 2-2 แสดงการกำเนิดเสียงและอวัยวะในการกำเนิดเสียง

ในการสร้างสัญญาณเสียงนั้นจะเลียนแบบรูปแบบทางกายภาพของมนุษย์ คือ ลม เปรียบได้กับสัญญาณกระตุ้น(Excitation Signal) จะเป็นตัวกระตุ้นให้เกิดสัญญาณเสียง

สัญญาณ Excitation ซึ่งถูกจำลองโดยใช้รูปแบบของฟิลเตอร์(Filter) โดยมีค่า Pitch เป็นความถี่ของเสียงที่ชัดที่สุด หลังจากนั้นนำสัญญาณกระตุ้นที่ได้มาปรับแต่งให้เหมือนกับเสียงที่ออกจากช่องปาก ในที่สุดจะได้สัญญาณเสียงที่สังเคราะห์ออกมา

2.2 การเข้ารหัสสัญญาณเสียง

การเข้ารหัสสัญญาณเสียงเกี่ยวกับการสุ่มสัญญาณข้อมูล (Sampling) และการควอนไทซ์แอมพลิจูด(Amplitude Quantization)

อัตราการสุ่มข้อมูลของเสียงพูดมักจะคงที่ประมาณสองเท่าของขนาดแถบความถี่ของสัญญาณเสียงแบบอนาล็อก ดังนั้นการควอนไทซ์จะเป็นวิธีการสำคัญในการกำหนดรูปแบบการเข้ารหัสสัญญาณเสียงพูด ซึ่งแบ่งได้เป็น 2 แบบใหญ่ๆ คือ

การควอนไทซ์โดยตรงหรือแบบนอนพาราเมตริก(Nonparametric) คือการนำสัญญาณไบนารี(Binary)มาแสดงสัญญาณเสียงโดยตรง

การควอนไทซ์แบบพาราเมตริก (Parametric) คือ การนำสัญญาณ Binary มาแสดงแทนโมเดลของเสียงหรือ/และพารามิเตอร์ทางความถี่ของเสียง

โดยวิธีการควอนไทซ์ที่สำคัญจะ กล่าวถึงในหัวข้อถัดไป เสียงพูดโดยปกติของมนุษย์มีความถี่ไม่เกิน 4 กิโลเฮิร์ต (kHz) และจะถูกสุ่มข้อมูลที่อัตรา 8 กิโลเฮิร์ต การเข้ารหัสเสียงโดยการควอนไทซ์โดยตรงที่ง่ายที่สุดคือ พัลส์โค้ดมอดูเลชัน(Pulse-Code Modulation : PCM) เสียงพูดที่เข้ารหัสที่อัตรา 64 กิโลบิตต่อวินาทีที่ใช้ลอการึทึมพีซีเอ็ม (Logarithm PCM) ถูกเรียกว่าการเข้ารหัสเสียงแบบไม่บีบอัดข้อมูล(Non-Compressed) ใช้เป็นการเข้ารหัสแบบอ้างอิงเทียบกับการเข้ารหัสแบบอื่นๆ การเข้ารหัสเสียงอาจเข้ารหัสได้พูดได้ตามอัตราการส่งข้อมูลดังนี้

1) การเข้ารหัสที่อัตราข้อมูลสูง ใช้อัตราข้อมูลมากกว่า 16 กิโลบิตต่อวินาที (kbit/s)

2) การเข้ารหัสที่อัตราข้อมูลขนาดกลาง ใช้อัตราข้อมูลระหว่าง 8-16 กิโลบิตต่อวินาที (kbit/s)

3) การเข้ารหัสที่อัตราข้อมูลต่ำ ใช้อัตราข้อมูลระหว่าง 2.4-8 กิโลบิตต่อวินาที (kbit/s)

4) การเข้ารหัสที่อัตราข้อมูลต่ำมาก ใช้อัตราข้อมูลน้อยกว่า 2.4 กิโลบิตต่อวินาที (kbit/s)

การเข้ารหัสเสียงพูดที่อัตราตั้งแต่ขนาดกลางลงมานั้นต้องใช้กระบวนการวิเคราะห์-สังเคราะห์(Analysis-Synthesis) ในขั้นของการวิเคราะห์จะหาชุดของพารามิเตอร์ที่ใช้แทนสัญญาณเสียงที่ถูกเข้ารหัสได้อย่างมีประสิทธิภาพ และในขั้นของการสังเคราะห์ค่าพารามิเตอร์เหล่านี้จะถูกถอดรหัสและสร้างเสียงพูดกลับมา การวิเคราะห์อาจเป็นไปได้ทั้งวงปิด (Close Loop) และแบบวงเปิด (Open Loop) ในแบบวงปิดค่าพารามิเตอร์จะถูกค้นหาจากความแตกต่างระหว่างเสียงต้นฉบับกับเสียงที่ถูกสร้างขึ้นมา นั่นคือในส่วนวงจรปิดจะต้องมีส่วนสังเคราะห์อยู่ภายใน กระบวนการแบบหลังนี้เรียกว่าการวิเคราะห์จากการสังเคราะห์(Analysis by Synthesis)

การเข้ารหัสเสียงพูดแบบพารามเตริกอาจเรียกอีกอย่างหนึ่งว่า "การเข้ารหัสเสียงตามลักษณะของเสียงพูดหรือโวโค้ดเดอร์(Speech-Specific or Voice Coder : Vocoder)" เป็นการเข้ารหัสที่เน้นในเรื่องคุณภาพของการรับฟังของเสียงพูดโดยไม่จำเป็นต้องได้สัญญาณที่เหมือนเดิมทุกประการ โวโค้ดเดอร์สามารถทำงาน

ที่อัตราข้อมูลต่ำมากๆโดยให้คุณภาพของเสียงที่ระดับเสียงสังเคราะห์ โดยที่ในอัตราข้อมูลที่สูงขึ้นก็จะให้คุณภาพเสียงที่ดีขึ้น

การเข้ารหัสสัญญาณเสียงแบ่งได้เป็น 2 แบบใหญ่ๆ คือ การเข้ารหัสเสียงตามรูปคลื่น(Waveform Coding) และการเข้ารหัสเสียงตามลักษณะของเสียงพูดหรือโวโค้ดเดอร์ การเข้ารหัสเสียงตามรูปคลื่น เช่น PCM , DPCM , ADPCM ส่วนในโครงงานนี้จะใช้การเข้ารหัสเสียงตามลักษณะของเสียงพูดหรือโวโค้ดเดอร์ ได้แก่ LPC-10 และ CELP

2.2.1 สัมประสิทธิ์เชิงเส้น 10 ค่า (10th Linear Prediction Coefficient : LPC-10)

เป็นการเข้ารหัสโดยใช้เทคนิคของแอลพีซีโดยการส่งลำดับของตัวกระตุ้น ซึ่งเสียงประเภท Voiced จะใช้ตัวกระตุ้นแบบขบวนของพัลส์ที่มีช่วงห่างระหว่างพัลส์เท่ากับคาบของเสียง (Pitch-pulse Excitation) แต่ในส่วนของเสียงประเภท unvoiced จะใช้สัญญาณรบกวนหรือ Noise แทน

การกระทำแบบนี้เป็นการกำหนดค่าลำดับของตัวกระตุ้นที่หยาบทำให้คุณภาพของเสียงที่ได้อยู่ในระดับเสียงสังเคราะห์ มีการคำนวณค่าสัมประสิทธิ์ของการทำนายจำนวน 10 ค่า

การส่งข้อมูลระหว่างตัวเข้ารหัสและถอดรหัสของแอลพีซี-10 นี้จะส่งค่าสัมประสิทธิ์ของการทำนายและลำดับของตัวกระตุ้นไป ถ้าเป็นเสียงประเภท Voiced จะส่งเฉพาะค่าแอมพลิจูดของพัลส์และคาบของเสียง ถ้าเป็นเสียงประเภท unvoiced ก็จะส่งเฉพาะพลังงานไปให้ทางด้านรับสร้างสัญญาณรบกวนเอาเอง ทำให้ข้อมูลที่ต้องใช้มีคุณภาพต่ำ

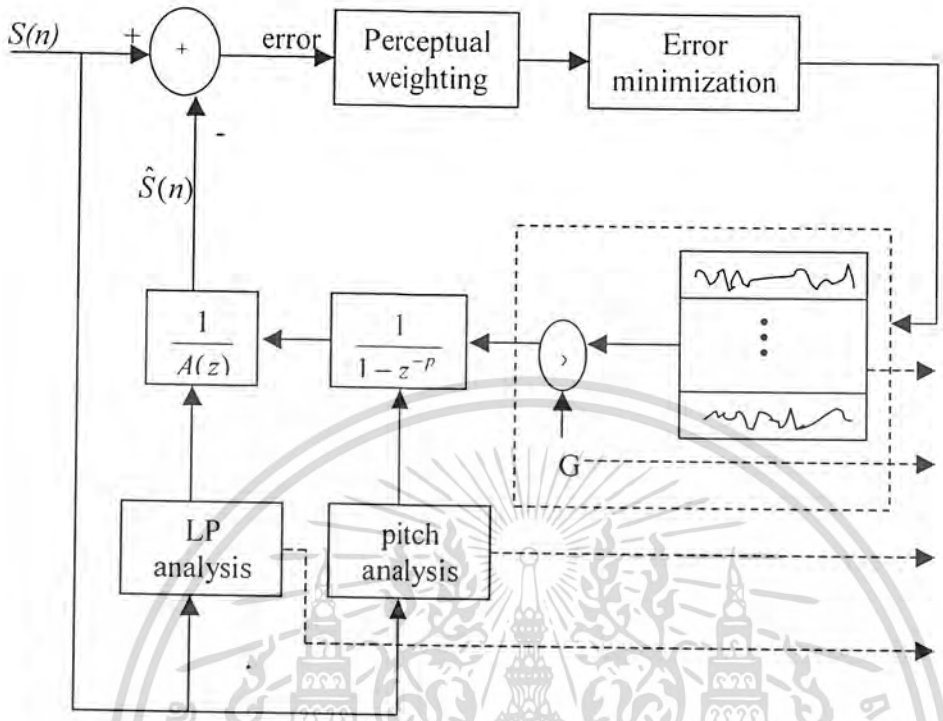
2.2.2 โค้ดเอกไซต์ลีเนียร์พรีดิกทีฟ (Code-Excite Linear Prediction : CELP)

เป็นการเข้ารหัสเสียงแบบแอลพีซีที่ใช้การเข้ารหัสเวคเตอร์ e(n) ด้วยรหัสที่มีเก็บอยู่ในชุดรหัสด้วยจำนวนที่จำกัด โดยการทำงานจะทำงานกับรหัสเสียงที่เป็นเวกเตอร์ให้คุณภาพของเสียงที่ดีโดยใช้อัตราข้อมูลที่ต่ำ

CELP เป็นการเข้ารหัสที่มีการวิเคราะห์ โดยด้านเข้ารหัสจะมีการสังเคราะห์เสียงตัวปัจจุบันขึ้นมาจากข้อมูลที่มีอยู่แล้วนำไปเปรียบเทียบกับเสียงจริงที่มีเข้ามา ความแตกต่างที่ได้จะเป็นสัญญาณความผิดพลาด (Error Signal) ที่ผ่านการให้น้ำหนักคุณภาพของการรับฟังแล้ว ซึ่งจะถูกนำไปผ่านการควอนไทซ์แบบเวกเตอร์เพื่อหาดัชนี (Index) ของลำดับของตัวกระตุ้นที่เหมาะสมที่สุดแล้วส่ง Index นั้นไปยังด้านรับ

นอกจาก Index แล้วยังต้องมีการส่งข้อมูลข้างเคียง (Side Information) ไปตามช่องสัญญาณด้วย ข้อมูลเหล่านี้ได้แก่สัมประสิทธิ์ของตัวทำนายและค่าคาบของเสียง

ดังนั้น CELP เป็นหลักการในการเข้ารหัสสัญญาณเสียง โดยใช้ค่าของสัญญาณความผิดพลาดในเข้ารหัสสัญญาณนำมาป้อนกลับเข้าไปในระบบเดิม เพื่อลดค่าความผิดพลาดในการเข้ารหัสสัญญาณเสียงในช่วงต่อไป



รูปที่ 2-3 แสดง Block Diagram ของ CELP

บทที่ 3
การเข้ารหัสสัญญาณเสียง สำหรับมาตรฐาน G.723.1

3.1 หลักการทั่วไป

3.1.1 สโคป (Scope)

มาตรฐาน G.723.1 เกี่ยวกับการบีบอัดสัญญาณ (Compress)เสียงพูดหรือสัญญาณออดิโออื่นๆ ซึ่งเป็นส่วนประกอบของมัลติมิเดีย ที่อัตราการส่ง(Bit Rate) ต่ำมากๆ

3.1.2 อัตราการส่ง (Bit Rate)

มี 2 Bit Rate คือ 5.3 กิโลบิตต่อวินาที (kbit/s) และ 6.3 กิโลบิตต่อวินาที (kbit/s) Bit Rate ที่สูงกว่าจะมีคุณภาพที่ดีกว่า ส่วน Bit Rate ที่ต่ำกว่าจะมีคุณภาพดีเช่นกันและยังมีความยืดหยุ่นด้วย มาตรฐานนี้สามารถที่จะสับเปลี่ยนระหว่าง 2 Bit Rate นี้ได้ ที่ทุกๆ 30 มิลลิวินาทีเฟรม(ms frame)

3.1.3 สัญญาณอินพุท (Input)

โปรแกรมนี้ถูกปรับปรุงให้ดีขึ้น (Optimize) เพื่อที่จะแทนที่เสียงให้มีคุณภาพโดยไม่ต้องมีความยุ่งยากซับซ้อนมาก เพลงและออดิโอที่ไม่ถือว่าเป็นเสียงพูดสามารถใช้การเข้ารหัสตามมาตรฐานนี้ได้เช่นกัน

3.1.4 ดีเลย์ (Delay)

อัลกอลิทึม (Algorithm) นี้เข้ารหัส (Encoder) สัญญาณ ใน 30 มิลลิวินาที (ms) และมีส่วนหัว (Look Ahead)ขนาด 7.5 ms ดังนั้น Algorithm นี้จะใช้ Delay ทั้งหมด 37.5 ms นอกจากนี้ยังมี Delay อื่นๆอีกที่ใช้ในการเข้ารหัสสัญญาณเสียงตามมาตรฐาน ได้แก่

- ใช้ในการคำนวณการเข้ารหัส (Encoderr) และถอดรหัส (Decoder)
- การส่งข้อมูลบนสาย (Communication Link)
- Multiplexing Delay ในกรณีที่มีการรวมข้อมูลเสียงเข้ากับข้อมูลชนิดอื่น

3.2 หลักการการเข้ารหัสเสียง (Encoderr Principles)

3.2.1 หลักการทั่วไป

มาตรฐานนี้เริ่มแรกจะกรองสัญญาณโทรศัพท์ออกจากสัญญาณอานาล็อก(Analog) แล้วสุ่มสัญญาณ(Sampling) ด้วยขนาด 8000 เฮิร์ต(Hz) หลังจากนั้นแปลงเป็นขนาด 16 bit Linear PCM แล้วจึงทำการ Encoder ส่วนการ Decode จะทำเหมือนกับส่วน Encoder แต่กลับขั้นตอนกัน และต้องได้สัญญาณที่เหมือนเดิม

มาตรฐานนี้อยู่บนพื้นฐานของการประมาณสัญญาณเชิงเส้น (Linear Prediction Analysis-by-Synthesis Coding) เพื่อลดสัญญาณที่ผิดเพี้ยน(Error) โดย Coder นี้จะแบ่งออกเป็นเฟรม (frame) frame ละ 240 แซมเปิล(samples) ซึ่งจะเท่ากับ 30 ms ที่ 8 กิโลเฮิร์ต(kHz)

แต่ละ frame จะนำองค์ประกอบคงที่ (DC Component) ออก และแบ่งออกเป็นเฟรมย่อยๆ(subframe) subframe ละ 60 samples ทุกๆ subframe จะทำการประมาณสัญญาณเชิงเส้น 10 ค่า (10^{th} Order Linear Prediction)

ตัวกรองสัญญาณ (Filter) ถูกสร้างขึ้นโดยใช้สัญญาณเสียงที่ยังไม่ได้ประมวลผล (Process) และใน subframe สุดท้ายนั้น สัญญาณจะถูกแทน(Quantize) โดยวิธี Predictive Split Vector Quantizer(PSVQ) ส่วน Unquantize LPC Coefficients จะใช้สร้าง Short-term Perceptual Weighting Filter สำหรับใช้ในทุกๆ frame

สำหรับทุกๆ 2 subframe (120 samples) จะใช้ค่าน้ำหนักของสัญญาณเสียง (Weighted Speech Signal) คำนวณค่า Open Loop Pitch Period (L_{oi}) และค่าคาบความถี่ของสัญญาณ (Pitch Period) จะถูกค้นหาในช่วง sample ที่ 18 ถึง sample ที่ 142

ตั้งแต่นี้ไปสัญญาณเสียงจะถูก Process โดยจะแบ่งการ Process ตาม subframe

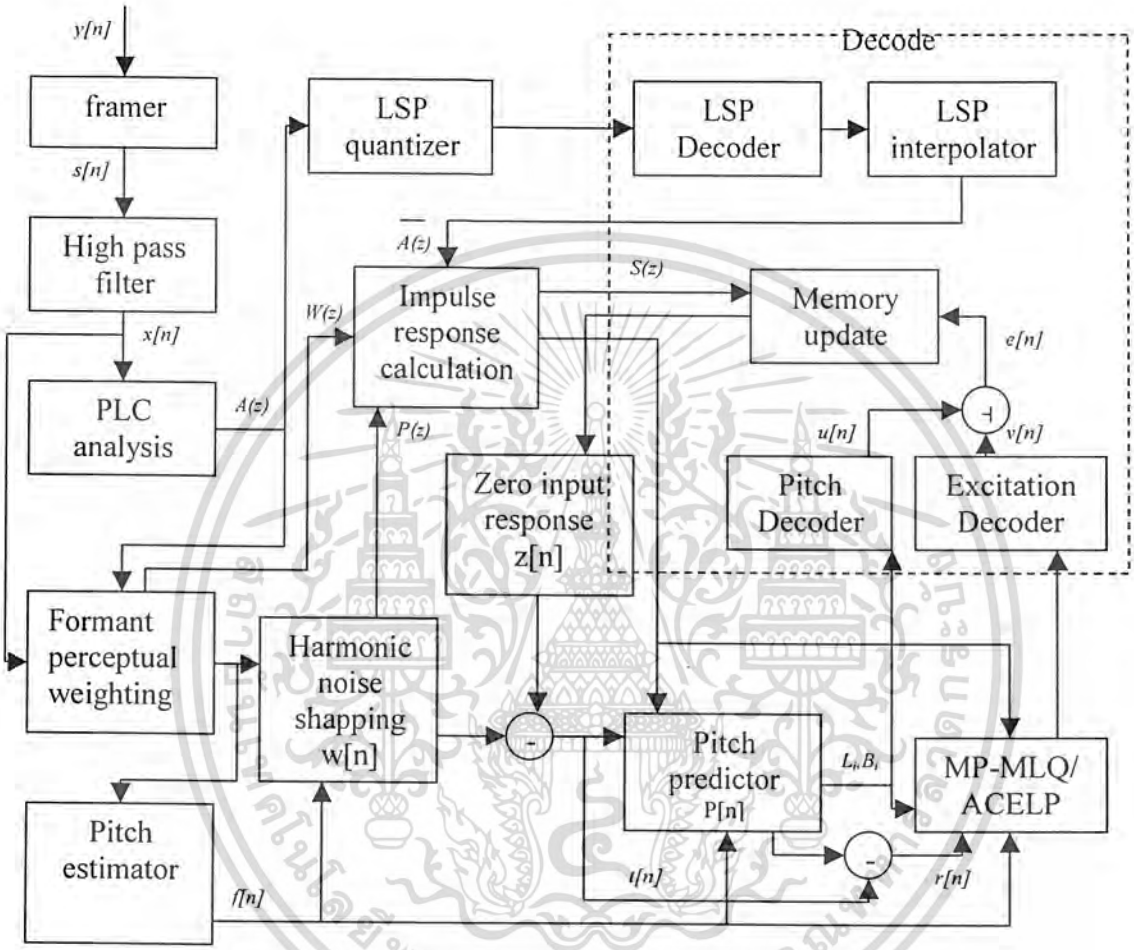
การประมาณค่า Pitch Period ก่อนหน้านี้ สามารถที่จะใช้สร้าง Harmonic Loop Shaping Filter ได้ การรวมกันของ LPC Synthesis Filter , The Formant Perceptual Filter และ The Harmonic Loop Shaping Filter จะใช้สร้าง Impulse Response เพื่อใช้ในการคำนวณต่อไป

การประมาณค่า Pitch Period Estimate (L_{oi}) และค่า Impulse Response จะเป็นการคำนวณ Close Loop Pitch Predictor โดยใช้ 5^{th} Order Pitch Predictor และค่า Pitch Period จะคำนวณจากค่าที่ใกล้เคียงกับค่า Open Loop Pitch Estimate ซึ่งจะทำให้ได้ค่าที่ละเอียดมากขึ้น

องค์ประกอบของ Pitch Predictor หรือ Pitch Contribution จะถูกลบออกจากค่าเริ่มต้นของสัญญาณเป้าหมาย (Target Vector) เพื่อส่งไปยังส่วน Decoder ต่อไป

ขั้นตอนสุดท้ายส่วนประกอบที่ไม่เป็นคาบเวลา (Non-Periodic) จะถูกประมาณ สำหรับบิตเรทสูง 6.3 kbit/s จะใช้วิธี Multi-Pulse Maximum LikeliHood Quantization (MP-MLQ) ส่วนบิตเรทต่ำ

5.3 kbit/s จะใช้วิธี Algebraic-Code-Excitation (ACELP) ส่วนรายละเอียดของวิธีต่างๆนั้นจะกล่าวในหัวข้อถัดไป



รูปที่ 3-1 แสดง Block diagram ของ การบีบอัดสัญญาณเสียง

3.2.2 การแบ่งเฟรม (Framer)

โปรแกรม G.723.1 นี้จะเก็บสัญญาณเสียง(Speech sample) ที่ตามกันมาไว้ใน Buffer(*y[n]*) แล้วส่งออกทีละ 240 samples (*s[n]*) สำหรับการคำนวณค่า Pitch ของแต่ละ frame จะถูกแบ่งออกเป็น 2 ส่วน ส่วนละ 120 samples และถูกแบ่งออกเป็น 2 ส่วนอีกครั้ง

ดังนั้น ใน 1 frame ที่มีขนาด 240 samples จะถูกแบ่งออกเป็น 4 subframe subframe ละ 60 samples

3.2.3 การกรององค์ประกอบคงที่ (High Pass Filter)

นำสัญญาณองค์ประกอบคงที่(DC Component) ออกจากสัญญาณอินพุท (Input) (*s[n]*) โดยใช้ฟังก์ชันตามสมการ (3.1) เป็นตัวกรองสัญญาณ(Filter)

$$H(z) = \frac{1 - z^{-1}}{1 - \frac{127}{128} z^{-1}} \tag{3.1}$$

เอาท์พุท(Output) ของ Filter นี้ คือ $x[n]_{n = 0..239}$

3.2.4 การวิเคราะห์ค่าสัมประสิทธิ์เชิงเส้น (LPC Analysis)

ขั้นตอนนี้ *x[n]* จะถูกทำ LPC analysis โดยใช้วิธีการประมาณค่าสัญญาณเชิงเส้น 10 ค่า(10^{th} order Linear predictive :LP) สำหรับแต่ละ subframe จะมีวินโดว์ (Window) ขนาด 180 samples อยู่ตรงกลาง เรียกว่าแฮมมิ่งวินโดว์(Hamming Window) ซึ่งจะถูกประยุกต์ใช้ในแต่ละ samples

ค่าสัมประสิทธิ์อัตสัมพันธ์(Autocorrelation Coefficient) จำนวน 11 ค่า จะถูกคำนวณจากสัญญาณวินโดว์ร่วมกับองค์ประกอบความสัมพันธ์ (Factor Correction) ค่าสัมประสิทธิ์อัตสัมพันธ์ของสัญญาณที่ไม่เป็นคาบเวลา(White Loop) มีค่าเท่ากับ 1024/1025 ซึ่งจะถูกนำมาใช้ในสูตร *R*[0] = *R*[0](1+1/1024) ส่วนค่า Autocorrelation Coefficient อีก 10 ค่า จะถูกคูณด้วยค่าในตารางสัมประสิทธิ์ทวินามของวินโดว์ (Binomial Window Coefficients Table)

ค่าประมาณสัมประสิทธิ์เชิงเส้น (Linear Prediction Coefficient หรือ LPC) จะถูกคำนวณโดยการใช้ กฎของเลอร์วินสันแอนด์เดอร์บิน (Levinson-Durbin Recursion) สำหรับทุกๆสัญญาณที่เข้ามาจะมี LPC ทั้งหมด 4 ชุด สำหรับ 1 ชุดจะนำไว้ใช้กับทุก ๆ subframe ค่า LPC เหล่านี้จะใช้ในการสร้าง Short Term Perceptual Weighting Filter ต่อไป

ตัวสร้างสัญญาณจากค่าสัมประสิทธิ์เชิงเส้น (LPC Synthesis Filter) สามารถอธิบายได้ด้วยสมการ (3.2)

$$A_i(z) = \frac{1}{1 - \sum_{j=1}^{10} a_{ij} z^{-j}}, 0 \le i \le 3 \tag{3.2}$$

โดย i คือ ลำดับที่ของ subframe ตั้งแต่ subframe ที่ 0 ถึง 3

สรุปได้ว่าขั้นตอนนี้คือ การนำ Hamming Windows มาคูณกับสัญญาณในแต่ละ subframe รวมไปถึง subframe ก่อนหน้าและหลังด้วย และคำนวณค่าของ autocorrelation หลังจากนั้นจะคำนวณค่าของ LPC โดยคำนวณจากวิธีการของ Levinson – Durbin ซึ่งเป็นการแทนค่าของสัญญาณ ผลลัพธ์สุดท้ายจะได้ค่า LPC 10 ค่า

3.2.5 การแทนสัญญาณคู่สเปกตรัมเชิงเส้น (LSP Quantizer)

ขั้นตอนแรกจะขยายแบนด์วิดท์ (Bandwidth) ของสัญญาณด้วยค่าความถี่ 7.5 Hz ค่า $A_3(z)$ ของ LP Filter จะถูกแทนสัญญาณ(Quantized) โดยใช้ Predictive Split Vector Quantizer หรือ PSVQ การ Quantize มีขั้นตอนดังนี้

- LP Coefficient ($\{a_i\}_{i=1..10}$) จะถูกเปลี่ยนเป็นค่าสัมประสิทธิ์คู่สเปกตรัมเชิงเส้น หรือ LSP Coefficients ($\{p'_i\}_{i=1..10}$) โดยใช้วิธีวงกลมหนึ่งหน่วย(Unit Circle) และ จะอินเทอร์โพเลทกัน (Interpolation) สำหรับค่าครอสซิ่งที่เป็นศูนย์(Zero Crossings)
- องค์ประกอบคงที่(DC Component : p_{DC}) จะถูกลบออกจาก LSP Coefficient (p') และ p จะแทนค่า LSP ที่ลบ DC Component ออกไปแล้ว
- First Order Fixed Predictor (b = 12/32) จะใช้กับ LSP Vector ที่ถูก Decode ก่อนหน้านั้น ($\widetilde{p}_{n-1}$) เพื่อที่จะเก็บ Predicted LSP Vector($\overline{p}_n$) ที่ถูกลบด้วย DC ออกไปและเก็บเวคเตอร์ความผิดพลาด (Error Vector) ส่วนที่เหลือ(e_n) ของ frame n นั้น

$$p_n^T = [p_{1,n}\ p_{2,n} \cdots p_{10,n}] \tag{3.3.1}$$

$$\overline{p}_n^T = [\overline{p}_{1,n}\ \overline{p}_{2,n} \cdots \overline{p}_{10,n}] \tag{3.3.2}$$

$$\overline{p}_n = b[\widetilde{p}_{n-1} - p_{DC}] \tag{3.3.3}$$

$$e_n = p_n - \overline{p}_n \tag{3.3.4}$$

- LSP Vector ที่ยังไม่ได้ถูก Quantized (p'_n) , LSP Vector ที่ถูก Quantized (p_n) และ LSP Error Vector (e_n) จะถูกแบ่ง ออกเป็น Subvector ขนาด 3 , 3 , 4 ตามลำดับ Subvector แต่ละส่วนลำดับที่ m จะถูก Quantized โดยใช้ 8 bit Codebook. และ Index i ของ Codebook ที่เหมาะสมสำหรับ Subvector นั้นที่ทำให้เกิด Error Criterion($E_{l,m}$) น้อยที่สุดจะถูกเลือกเพื่อใช้ส่งออกไป

$$p'^{T}_{m} = [p'1+3m \;\; p'2+3m\Lambda \;\; p'Km+3m], \quad Km = \begin{Bmatrix} 3, m=0 \\ 3, m=1 \\ 4, m=2 \end{Bmatrix} \tag{3.4.1}$$

$$\tilde{p}^{T}_{l,m} = [\tilde{p}_{1,l,m} \;\; \tilde{p}_{2,l,m} \mathrm{K} \;\; \tilde{p}_{K_m,l,m}], \quad \begin{matrix} 0 \le m \le 2 \\ 1 \le l \le 256 \end{matrix} \tag{3.4.2}$$

$$p' = p + p_{DC} \tag{3.4.3}$$

$$\tilde{p}_{l,m} = \bar{p}_m + p_{DC_m} + e_{l,m}, \quad \begin{matrix} 0 \le m \le 2 \\ 1 \le l \le 256 \end{matrix} \tag{3.4.4}$$

$$E_{l,m} = (p'_m - \tilde{p}_{l,m})^T W_m (p'_m - \tilde{p}_{l,m}), \quad \begin{matrix} 0 \le m \le 2 \\ 1 \le l \le 256 \end{matrix} \tag{3.4.5}$$

โดยที่ $e_{l,m}$ คือ ลำดับที่ l ของ Split Residual ลำดับที่ m ของ LSP Codebook และ W_n คือ Diagonal Weighting Matrix ที่คาดคะเนจาก Unquantized Coefficients Vector (p') ที่ อธิบายโดยสมการ(3.5)

$$w_{j,j} = \frac{1}{\min\{p'_j - p'_{j-1}, p'_{j+1} - p'_j\}}, 2 \le j \le 9$$

$$w_{1,1} = \frac{1}{p'_2 - p'_1} \tag{3.5}$$

$$w_{10,10} = \frac{1}{p'_{10} - p'_9}$$

- ตัว index ที่ถูกเลือกจะส่งออกไปยังช่องสัญญาณ

จุดประสงค์ของขั้นตอนนี้คือ เนื่องจากสัมประสิทธิ์ LPC ที่คำนวณได้ 10 ค่านั้นเป็นเลขทศนิยม ต้องใช้จำนวนบิต (bit) ในการจัดเก็บ และ/หรือ จำนวน bit ในการส่งข้อมูลมาก จึงต้องทำการ Quantize สัญญาณ เพื่อที่จะได้ส่งค่าอินเด็กซ์ (index) ที่ได้จากการ Quantize ไปแทนการส่งค่าสัมประสิทธิ์ LPC ที่เป็นตัวเลขทศนิยม ซึ่งจะทำให้ลดจำนวน bit ในการส่งข้อมูลได้มาก อย่างไรก็ดีเนื่องจากการ Quantize ค่าสัมประสิทธิ์

LPC โดยตรงนั้นอาจทำให้ Filter ไม่เสถียรได้ง่าย จึงต้องแปลงจากค่าสัมประสิทธิ์ LPC ไปเป็นค่า Linear Spectral Pair หรือ LSP ก่อนทำการ Quantize

3.2.6 การถอดรหัสสัญญาณคู่สเปกตรัมเชิงเส้น (LSP Decoder)

การถอดรัสของ LSP Coefficient กระทำดังนี้

- subVector 3 อันดับแรก $\{e_{m,n}\}_{m=0..2}$ จะถูกถอดรหัสให้อยู่ในรูปของ 10^{th}order Vector($\widetilde{e}_n$)
- Predicted Vector $\overline{p}_n$ จะถูกเพิ่มเข้าไปใน Vector ที่ถูกถอดรหัสแล้ว รวมทั้ง $\widetilde{e}_n$ และ p_{DC} Vector ด้วย เพื่อที่จะทำให้อยู่ในรูปแบบของ LSP Vector $\widetilde{p}_n$ ที่ถูกถอดรัสแล้ว
- LSP Vector ที่ถูกถอดรัสแล้วจะตรวจสอบความเสถียร (Stability Check) เพื่อให้แน่ใจว่า Vector นั้นเมื่อนำไปสร้าง Filter แล้วจะทำให้ Filter มีความเสถียร ซึ่งจะต้องเป็นไปตามเงื่อนไขต่อไปนี้ ตามสมการ (3.6)

$$\widetilde{p}_{j+1,n} - \widetilde{p}_{j,n} \geq \Delta_{\min}, 1 \leq j \leq 9 \tag{3.6}$$

$\triangle_{min}$ จะเท่ากับ 31.25 Hz ถ้า Stability Check ไม่เป็นตามสมการ(3.6) สำหรับค่า $\widetilde{p}_i$ และ $\widetilde{p}_{i+1}$ จะใช้การคำนวณตามสมการดังนี้

$$\widetilde{p}_{avg} = (\widetilde{p}_j + \widetilde{p}_{j+1})/2 \tag{3.7.1}$$

$$\widetilde{p}_j = \widetilde{p}_{avg} - \Delta_{\min}/2 \tag{3.7.2}$$

$$\widetilde{p}_{j+1} = \widetilde{p}_{avg} + \Delta_{\min}/2 \tag{3.7.3}$$

การคำนวณตามสมการ(3.7.1) – (3.7.3) จะถูกคำนวณจนค่าที่ได้เป็นไปตามสมการ(3.6) แต่ถ้ายังไม่เป็นไปตามสมการ(3.6) ภายใน 10 ครั้ง ค่า LSP Vector ก่อนหน้านี้จะถูกใช้แทนค่า LSP ปัจจุบัน ดังนั้นขั้นตอนนี้เป็นขั้นตอนที่ LSP ที่ถูกแทนเสียงสัญญาณ หรือ Quantize จากฝั่ง Encoder จะถูก Decoder หลังจากนั้นจะมีการตรวจสอบว่า LSP Vector ที่ถูก Decode นั้นเป็นไปตามเงื่อนไขที่กำหนดหรือไม่ ถ้าไม่เป็นไปตามที่กำหนดก็จะใช้ LSP Vector ปัจจุบัน และ LSP Vector ถัดไปมาปรับปรุงเพื่อให้เป็นไปตามเงื่อนไขที่กำหนด เพื่อให้วงจร Filter นั้นเสถียร

3.2.7 การอินเทอร์โพเลทค่าสัญญาณคู่สเปกตรัมเชิงเส้น (LSP Interpolation)

การ Interpolation จะคำนวณระหว่าง LSP Vector ที่ถูก Decode แล้ว ($\widetilde{p}_n$) กับ LSP Vector ก่อนหน้านี้ ($\widetilde{p}_{n-1}$) สำหรับแต่ละ subframe ค่า LSP Vector ที่ถูก Interpolate แล้วจำนวน 4 ค่า $\{\widetilde{p}_i\}_{i=0..3}$ จะถูกเปลี่ยนไปเป็น LPC Vector $\{\widetilde{a}_i\}_{i=0..3}$

$$\widetilde{p}_{ni} = \begin{Bmatrix} 0.75\widetilde{p}_{n-1} + 0.25\widetilde{p}_n, i = 0 \\ 0.5\widetilde{p}_{n-1} + 0.5\widetilde{p}_n, i = 1 \\ 0.25\widetilde{p}_{n-1} + 0.75\widetilde{p}_n, i = 2 \\ \widetilde{p}_n, i = 3 \end{Bmatrix} \tag{3.8}$$

$$a_i^T = [\widetilde{a}_{i1}\widetilde{a}_{i2}\mathrm{K}\,\widetilde{a}_{i10}]^T, \quad 0 \le i \le 3 \tag{3.9}$$

LPC Synthesis Filter ที่ถูก Quantized แล้ว $\widetilde{A}_i(Z)$ จะใช้สำหรับการสร้างสัญญาณเสียงในส่วนของการ Decoder ที่อธิบายด้วยสมการ(3.10)

$$\widetilde{A}_i(z) = \frac{1}{1 - \sum_{j=1}^{10} \widetilde{a}_{ij} z^{-j}}, \quad 0 \le i \le 3 \tag{3.10}$$

ดังนั้นขั้นตอนนี้จะนำค่า LSP มาแปลงกลับ(Convert) ให้เป็นค่าสัมประสิทธิ์ LPC ดังนั้นผลลัพธ์ที่ได้จะเป็นค่าสัมประสิทธิ์ LPC

3.2.8 การปรับค่าน้ำหนักของการรับสัมผัส (Formant Perceptual Weighting Filter)

สำหรับแต่ละ subframe ส่วนของ Perceptual Weighting Filter จะถูกสร้างขึ้นโดยการใช้ LPC Coefficient ที่ผ่านการ Quantized แล้ว $\{a_{ij}\}_{j=1...10}$ โดย Filter นี้ต้องอาศัยทรานซ์เฟอร์ฟังก์ชัน (Transfer function) ตามสมการ(3.11)

$$W_i(z) = \frac{1 - \sum_{j=1}^{10} a_{ij} z^{-j} \gamma_1^{\,j}}{1 - \sum_{i=1}^{10} a_{ij} z^{-j} \gamma_2^{\,j}}, \quad 0 \le i \le 3 \tag{3.11}$$

โดย $\gamma_1 = 0.9$ และ $\gamma_2 = 0.5$ frame เสียงที่เข้ามา $\{x[n]\}_{n=0..239}$ จะถูกแบ่งเป็น 4 subframe แต่ละ subframe จะถูกกรองสัญญาณ(Filter) โดยใช้ $W_i(z)$ Filter และ output ของสัญญาณเสียงที่ถูก weight หรือปรับค่าน้ำหนักแล้ว $\{f[n]\}_{n=0..239}$ จะถูกเก็บไว้

จุดประสงค์ขั้นตอนนี้เป็นการปรับปรุงสัญญาณให้มีคุณสมบัติคล้ายกับเสียงที่หูมนุษย์ได้ยิน หรือลักษณะการรับรู้ของหูคน โดยการเพิ่มคุณภาพเสียงนั้นจะลดค่าของยอดของสัญญาณ (Peak) ในทาง Frequency-Domain และเพิ่มเข้าในส่วนของท้องคลื่นแทน

3.2.9 การประมาณค่าคาบของความถี่สูงสุดแบบหยาบ (Pitch Estimation)

สำหรับทุกๆ frame จะมีการคำนวณค่า Pitch 2 ครั้ง (Pitch แรกสำหรับ 2 subframe แรก และ Pitch ที่ 2 สำหรับ 2 subframe สุดท้าย) การประมาณค่าความถี่สูงสุดแบบหยาบ (Open Loop Pitch estimate) จะถูกคำนวณโดยใช้ Perceptually Weighted Signal $f[n]$ และเกณฑ์อัตสัมพันธ์(Cross-Correlation Criterion :$C_{ol}(j)$) ค่าที่ได้จากสมการ(3.12) จะถูกนำมาใช้ในการประมาณค่า Pitch Period

$$C_{OL} = \frac{\left(\sum_{n=0}^{119} f[n]\cdot f[n-j]\right)}{\sum_{n=0}^{119} f[n-j]\cdot f[n-j]}, \quad 18 \le j \le 142 \tag{3.12}$$

ค่า j ที่ทำให้ค่า C_{ol} มากที่สุดจะถูกเลือกมาจากการประมาณค่า Open Loop Pitch Estimation สำหรับ 2 subframe ที่เหมาะสม ในการหา index ที่เหมาะสมบางครั้งจะได้ค่า Pitch Period ที่เล็กเกินไป จึงพยายามที่จะหลีกเลี่ยงการเลือกพิตช์มัลติเปิล (Pitch multiples)

ค่า $C_{ol}(j)$ สูงสุดจะถูกค้นหา ตั้งแต่ j=18 สำหรับค่า $C_{ol}(j)$ ที่พบทุกๆครั้งจะนำมาเปรียบเทียบกับค่า $C_{ol}(j)$ ที่สามารถหาได้สูงสุดก่อนหน้านี้ $C_{ol}(j')$ ถ้าหากว่าค่า j และ j' ต่างกันน้อยกว่า 18 และ $C_{ol}(j) > C_{ol}(j')$ ค่าใหม่ที่หาได้มากที่สุดจะถูกเลือก

หากค่า j และ j' ต่างกันมากกว่าหรือเท่ากับ 18 ค่าใหม่ที่หาได้มากที่สุดจะถูกเลือกเมื่อ $C_{ol}(j') > C_{ol}(j)$ อยู่ 1.25 dB

จุดประสงค์ของขั้นตอนนี้คือ หลังจากคำนวณส่วนของ LPC Filter ที่ใช้ในการสร้างสัญญาณเสียงแล้ว จะพิจารณาส่วนของสัญญาณอินพุท(input) ที่ใช้ป้อนให้กับ LPC Filter เพื่อใช้สร้างสัญญาณเสียงขึ้นมา โดยสัญญาณ input ที่ใช้ป้อนเข้า LPC Filter จะแยกเป็น 2 ส่วน คือ

3.2.9.1 input ที่ใช้แทนเสียงประเภท Voiced Sound มีคาบของสัญญาณที่แน่นอน

3.2.9.2 input ที่ใช้แทนเสียงประเภท Unvoiced Sound มีคาบของสัญญาณไม่แน่นอนหรือมีลักษณะสัญญาณเป็นแบบ Random

สำหรับ input ที่มีลักษณะเป็นคาบเวลาจะแทนค่าคาบเวลา (Pitch Period) และค่าอัตราขยาย(Pitch Gain) ซึ่งการคำนวณคาบเวลาในขั้นตอนนี้จะเป็นแบบไม่ละเอียด เรียกว่าการคำนวณค่า Open Loop Pitch

3.2.10 การโปรเซสของแต่ละสับเฟรม (Subframe Processing)

ตั้งแต่จุดนี้เป็นต้นไป ทุกๆการคำนวณระหว่าง Blocks จะถูกคำนวณในระดับ subframe

3.2.11 การปรับแต่งสัญญาณฮาร์โมนิก (Harmonic Loop Shaping)

เป็นการเพิ่มคุณภาพของเสียงโดยการเลื่อนสัญญาณ เปรียบเทียบได้กับการจูนช่องโทรทัศน์ที่จะชัดเป็นบางช่วง แต่ก็ยังไม่ชัดถึงที่สุด ต้องปรับไปอีกเล็กน้อยจึงได้จะภาพชัดสมบูรณ์

การปรับปรุงคุณภาพของเสียงที่ถูก Encoderd ส่วนของ Harmonic Loop Shaping จะถูกสร้างขึ้นโดยสมการ (3.13)

$$P_i = 1 - \beta z^{-L} \tag{3.13}$$

Optimal Lag L หรือค่า Delay สำหรับ Filter นี้คือค่า L ที่มีค่า correlation สูงสุด $C_{pw}(j)$ โดยจะคำนวณจากค่า $N(j)$ ที่เป็นบวกเท่านั้น ก่อนที่จะยกกำลัง 2 ตามสมการ

$$N(j) = \sum_{n=0}^{59} f[n] \cdot f[n-j] \tag{3.14.1}$$

$$C_{PII'}(j) = \frac{(N(j))^2}{\sum_{n=0}^{59} f[n].f[n-j]}, L_1 \le j \le L_2 \tag{3.14.2}$$

โดยที่ $L_1 = L_{OL}-3$ และ $L_2 = L_{OL}+3$ ซึ่งค่าที่มากที่สุดจะแทนด้วย C_L ส่วนค่าอัตราขยายของฟิลเตอร์(Filter Gain) ที่เหมาะสม G_{opt} จะคำนวณจากสมการ (3.15)

$$G_{opt} = \frac{\left(\sum_{n=0}^{59} f[n]f[n-L]\right)}{\sum_{n=0}^{59} f[n-L]f[n-L]} \tag{3.15}$$

ซึ่ง G_{opt} จะมีค่าอยู่ในช่วง 0...1 และพลังงาน E ของ Weight Wpeech Wignal $\{f[n]\}_{n=0..59}$ จะเป็นไปตามสมการ(3.16)

$$E = \sum_{n=0}^{59} f^2[n] \tag{3.16}$$

ค่าอัตราขยายของ Harmonic Loop Shaping Filter $P(z)$ คือค่าสัมประสิทธิ์ β สามารถอธิบายโดยสมการ (3.17)

$$\beta = \begin{Bmatrix} 0.3125G_{opt}, & if\ -10\log_{10}\left(1-\frac{C_L}{E}\right) \geq 2.0 \\ 0.0, & otherwise \end{Bmatrix} \tag{3.17}$$

หลังจากที่หาค่า Harmonic Loop Filter Coefficient แล้ว จะใช้ค่า Formant Perceptually Weighted Speech $f[n]$ ในการสร้าง Filter ซึ่งจะใช้ $P(z)$ ในการเก็บ Target Vector $w[n]$ ตามสมการ(3.18)

$$w[n] = f[n] - \beta f[n-L], \quad 0 \leq n \leq 59 \tag{3.18}$$

3.2.12 การคำนวณผลตอบสนองต่ออิมพัลส์ (Impulse Response Calculator)

สร้าง Filter จากการนำ Quantized LPC Synthesis Filter, Formant Perceptual Weighting Filter และ Harmonic Loop Shaping Filter มาคำนวณร่วมกัน แล้วนำ Unit Impulse Sequence เป็น input ผ่านเข้าไปใน Filter output ที่ได้เรียกว่า Impulse Response

สำหรับการรวบรวม Close Loop Analysis Filter $S_i(z)$ จะใช้ตามสมการ(3.19)

$$Si(z) = Ai(z) \cdot Wi(z) \cdot Pi(z), \quad 0 \leq i \leq 3 \tag{3.19}$$

ซึ่งจะอ้างอิงถึงสมการ (3.10) ,(3.11) และ (3.13) Impulse Response ของ Filter นี้จะถูกคำนวณและแทนด้วย $\{h_i[n]\}_{n=0..59,\ i=0..3}$

3.2.13 การตอบสนองต่ออินพุทที่เป็นศูนย์ (Zero Input Response and Ringing Subtraction)

เป็น output ของ Filter เมื่อ input มีค่าเป็นศูนย์(0)หมด แล้วค่าที่ได้จะนำมาลบออกจากค่าของ sample ที่ได้จากการเพิ่มคุณภาพของเสียงโดยการเลื่อนสัญญาณ (Harmonic Loop Shaping)

zero input Response จะแทนด้วย $\{Z[n]\}_{n=0..59}$ การทำ Ringing subtraction $\{W[n]\}_{n=0..59}$ จะลบ Zero Input Response ออกจากสัญญาณฮาร์โมนิก (Harmonic Speech Vector) ดังนั้นผลที่ได้คือ $t[n] = W[n]-Z[n]$

3.2.14 การประมาณค่าพิตช์ (Pitch Predictor)

เป็นการคำนวณค่าของโคลสลูปพิต (Closed Loop Pitch) โดยจะหาค่า Pitch อย่างละเอียด ซึ่งจะอาศัยค่าของ Pitch ที่ได้คำนวณมาแล้วอย่างคร่าวๆ ในขั้นตอน Pitch Estimation การหา Closed Loop Pitch นี้ จะมีค่าที่แตกต่างจากค่าเดิมโดยแยกการหาสำหรับ Bit Rate สูง (6.3 kbit/s) และ Bit Rate ต่ำ (5.3 kbit/s)

องค์ประกอบ Pitch Prediction ถูกคำนวณเหมือนกับองค์ประกอบของอะแดพทีฟโค้ดบุค (Adaptive Codebook) โดยจะใช้การประมาณค่าพิตช์ลำดับที่ห้า (5^{th} Order Pitch Predictor) ดูสมการ (3.41.2)

สำหรับ subframe ที่ 0 และ 2 ค่า Closed Loop Pitch Lag หรือ Delay ซึ่งจะถูกเลือกจากโอเพนลูปพิตช์แลก(Open Loop Pitch Lag) ที่เหมาะสมในช่วง ±1 และจะถูกเข้ารหัสโดยใช้ 7 bits

สำหรับ subframe ที่ 1 และ 3 ค่า Closed Loop Pitch Lag จะถูกเข้ารหัสแตกต่างกัน โดยใช้ 2 bits หรืออาจจะแตกต่างจาก subframe Lag ก่อนหน้านี้เพียง ±1 0 หรือ +2 เท่านั้น ค่า Pitch Lag ที่ถูก Decode และ Quantize แล้ว จะแทนด้วย L_i

อัตราขยายของการประมาณค่าพิตช์ (Pitch Predictor Gain) คือสัญญาณที่ถูก Quantize โดยใช้ 2 Codebook คือ 85 entries หรือ 170 entries สำหรับ Bit Rate สูง และ 170 entries สำหรับ Bit Rate สูง

สำหรับ Bit Rate สูง ถ้า L_0 น้อยกว่า 85 สำหรับ subframe 0 และ 1 หรือ ถ้า L_2 น้อยกว่า 58 สำหรับ subframe 2 และ 3 แล้ว Codebook ที่ 85 entry จะถูกใช้เป็น Pitch Gain Quantization ส่วนอื่นๆจะใช้ Codebook ที่ 170 entry เป็น Pitch Gain สำหรับ Gain Predictor $p\{n\}_{n=0..59}$ จะถูกลบออกจาก Target Vector $\{t[n]\}_{n=0..59}$ เพื่อคำนวณสัญญาณความผิดพลาด (Residual Signal) ตามสมการ(3.20)

$$r[n] = t[n] - p[n] \tag{3.20}$$

การเข้ารหัสสัญญาณกระตุ้น (Excitation Encoder) เป็นการจำลองสัญญาณที่เป็น Unvoiced ที่มีลักษณะไม่เป็นคาบ ซึ่งสัญญาณใน frame นั้นจะถูกแทนด้วยตำแหน่งต่างๆด้วยพัลส์ (Pulse)จำนวนหนึ่ง โดยจะประกอบด้วยค่า Pulse Sign (บวกหรือเป็นลบ) และ อัตราขยายของ Pulse โดยจะให้ ค่าของ Mean Square Error น้อยที่สุด สามารถแบ่งตามอัตราการส่งได้ 2 แบบ คือ MP-MLQ สำหรับ Bit Rate สูง และ ACELP สำหรับ Bit Rate ต่ำ

3.2.15 การเข้ารหัสสัญญาณกระตุ้นที่บิตเรทสูง (High Rate Excitation :MP-MLQ)

Residual Signal $\{r[n]\}_{n=0..59}$จะเป็นสัญญาณเสียงใหม่ที่จะถูกส่งไปยังขั้นตอนการทำ MP-MLQ เพื่อที่จะควอนไทซ์สัญญาณนี้ กระบวนการควอนไทซ์เป็นการประมาณ Target Vector *r[n]* ที่แทนด้วย $r'[n]$ ตามสมการ(3.21)

$$r'[n] = \sum_{j=0}^{n} h[j] \cdot v[n-j], \quad 0 \le n \le 59 \tag{3.21}$$

โดยที่ *v[n]* คือ การ Excitataion Filter *S(z)* ที่ถูกรวมเข้ากับ Impulse Response *h[n]* ซึ่งอธิบายโดย สมการ(3.22)

$$v[n] = G\sum_{k=0}^{M-1} \alpha_k d[n-m_k], \quad 0 \le n \le 59 \tag{3.22}$$

โดยที่ *G* คือ องค์ประกอบของอัตราขยาย (Gain Factor) $\delta[n]$ ของไดแรคฟังก์ชัน (Dirac function) ส่วน $\{\alpha_k\}_{k=0..M-1}$ และ $\{m_k\}_{k=0..M-1}$ คือเครื่องหมาย ± 1

ตำแหน่งของ Dirac function ที่เกี่ยวข้องกับ *M* คือจำนวนของ Pulse จะมีค่าเป็น 6 สำหรับ subframe คู่ และมีค่าเป็น 5 สำหรับ subframe คี่ มีข้อกำหนดสำหรับตำแหน่ง Pulse จะเป็นตำแหน่งคี่ได้ทั้งหมด และ/หรือตำแหน่งคู่ได้ทั้งหมด ซึ่งสามารถบ่งชี้โดย Grid Bit

ปัญหาคือการประมาณค่าของ *G*, $\{\alpha_k\}_{k=0..M-1}$, $\{m_k\}_{k=0..M-1}$ ที่จะต้องมีค่าความผิดเพี้ยน (Error) ของสัญญาณน้อยที่สุด *err[n]* ที่สามารถอธิบายได้โดยสมการ(3.23)

$$err[n] = r[n] - r'[n] = r[n] - G\sum_{k=0}^{M-1} \alpha_k h[n-m_k] \tag{3.23}$$

การประมาณค่าพารามิเตอร์ (Parameter) และกระบวนการควอนไทซ์จะใช้พื้นฐานของการสร้างจากการวิเคราะห์(Analysis-by-Synthesis) ซึ่ง G_{max} จะถูกประมาณและถูกควอนไทซ์

ในขั้นตอนแรกครอสคอรีเลชันฟังก์ชัน (Cross-Correlation Function) *d[j]* ระหว่าง Impulse Response *h[n]* กับ Target Vector *r[n]* Vector ใหม่ จะถูกคำนวณตามสมการ(3.24)

$$d[j] = \sum_{n=j}^{59} r[n] \cdot h[n-j], \quad 0 \le j \le 59$$

(3.24)

การประมาณค่า G_{max} จะใช้สมการ

$$G_{\max} = \frac{\max\left\{|d[j]|\right\}_{j=0\ldots59}}{\sum_{n=0}^{59} h[n] \cdot h[n]} \qquad (3.25)$$

หลังจากประมาณค่า Gain G_{max} จากการ Quantize โดย Logarithmic Quantizer แล้วขนาดของ Gain Quantizer จะถูกใช้พร้อมกันสำหรับสอง Bit Rate

นอกจาก Quantized Value $\widetilde{G}_{\max}$ แล้ว ยังมี Gain Value ที่ถูกเลือกภายในช่วง [$\widetilde{G}_{\max}$ – 3.2 , $\widetilde{G}_{\max}$ + 6.4] สำหรับ Gain Value ทั้งหมดนี้ เครื่องหมายและตำแหน่งของ Pulses จะออพติไมซ์เรียงกันตามลำดับ ซึ่งจะทำซ้ำสำหรับตำแหน่ง Grid คู่ และ Grid คี่

สุดท้าย การรวมกันของพารามิเตอร์ที่ถูกควอนไทซ์แล้วที่ทำให้เกิดความผิดเพี้ยน *err[n]* น้อยสุดจะถูกเลือก ตำแหน่งที่ดีที่สุดของ Pulse Position และ Pulse Gain จะถูกส่งออกไปพร้อมกัน

เพื่อที่จะพัฒนาคุณภาพของเสียงด้วยค่า Pitch Period ที่สั้น ขั้นตอนต่อไปนี้จะถูกใช้ ถ้า L_0 น้อยกว่า 58 สำหรับ subframe 0 และ 1 หรือ L_2 น้อยกว่า 58 สำหรับ subframe 2 และ 3 แล้ว สายของ Dirac function กับคาบของ Pitch index L_0 หรือ L_2 จะถูกใช้ แต่ละตำแหน่ง m_k จะใช้ แทน Single Dirac Function

การเลือกระหว่างสาย Dirac function หรือ Dirac Function เดี่ยวจะพิจารณาจาก Residual Signal ที่ได้จากการคำนวณ Mean Square Error ค่า index ที่ทำให้ Mean Square Error น้อยที่สุดจะถูกใช้ต่อไป

3.2.16 การเข้ารหัสสัญญาณกระตุ้นที่บิตเรทต่ำ (Low Rate Excitation :ACELP)

Codebook ขนาด 17 bit นี้จะถูกใช้สำหรับ Fixed Codebook Excitation *v[n]* แต่ละ Fixed Codebook ส่วนใหญ่ประกอบไปด้วยพัลส์ที่ไม่เป็นศูนย์ (Non-Zero Pulse) ซึ่งทั้ง 4 Pulse สามารถที่จะสันนิษฐานเครื่องหมายและตำแหน่งของพัลส์ได้จากตารางที่ 3-1

ตำแหน่งของ Pulse ทั้งหมดสามารถที่จะเลื่อนไปพร้อมกันได้ทีละ 1 โดยอาศัย bit พิเศษ 1 bit ซึ่ง 2 ตำแหน่งสุดท้ายจะตกจากขอบของ subframe ซึ่งจะบ่งชี้ได้ว่า Pulse นั้นไม่ได้ปรากฏขึ้นมา

sign	positions
±1	0,8,16,24,32,40,48,56
±1	2,10,18,26,34,42,50,58
±1	4,12,20,28,36,44,52,(60)
±1	6,14,22,30,38,46,54,(62)

ตารางที่ 3-1 แสดง ACELP Excitation Codebook

ตำแหน่งของ Pulse ทั้งหมด จะถูก Encode ด้วย 3 bits และเครื่องหมายของ Pulse ทั้งหมด จะถูก Encoderd ใน 1 bit ซึ่งจะใช้ 16 bits สำหรับ 4 Pulses นอกจากนี้จะมีบิตพิเศษใช้สำหรับ Encoder ดังนั้นจะใช้ทั้งหมด 17 bit Codebook

Codebook จะถูกหาโดยการทำ Mean Square Error ระหว่าง Weighted Speech Signal *r[n]* กับ weighted Synthesis Speech โดยสมการ(3.26)

$$E_{\xi} = \left\| r - GHv_{\xi} \right\|^2 \tag{3.26}$$

r คือเวคเตอร์เป้าหมาย (Target Vector) ซึ่งประกอบไปด้วย Weighted Speech หลังจากที่ลบด้วย Zero-Input Response ของ Weighted Synthesis Filter กับ Pitch Contribution

G คือ Codebook Gain $v\xi$ คือ Algebraic Codeword ที่ ξ และ *H* คือ ค่าเมตริกซ์ 3 เหลี่ยมล่าง (Lower Triangular Toeplitz Convolution Matrix) ด้วยไดโกนอล (dDiagonal) *h(0)* และ Lower Diagonal *h(1),…,h(L-1)* โดย *h(n)* คือ Impulse Response ของ Weighted Synthesis Filter $S_i(z)$

โค้ดเวิร์ด (Codeword) ที่เหมาะสมคือ Codeword ที่สามารถ Maximizes การคำนวณตามสมการ(3.27)

$$\tau_\xi = \frac{C_\xi^{\ 2}}{\varepsilon_\xi} = \frac{\left(d^T v_\xi\right)^2}{v_\xi^{\ T} \Phi v_\xi} \tag{3.27}$$

โดย $d = H^T r$ คือ Correlation ระหว่าง Target Vector Signal $r[n]$ กับ Impulse Response $h(n)$ และ $\Phi = H^T H$ คือ Covariance Matrix ของ Impulse Response

เวคเตอร์ d และ matrix Φ คือค่าที่คำนวณก่อนที่จะหา Codebook สามารถหาได้ตามสมการ(3.28) และ (3.29)

$$d(j) = \sum_{n=j}^{59} r[n] \cdot h[n-j], \quad 0 \le j \le 59 \tag{3.28}$$

$$\Phi(i,j) = \sum_{n=j}^{59} h[n-i] \cdot h[n-j], \quad \begin{matrix} j \ge i \\ 0 \le i \le 59 \end{matrix} \tag{3.29}$$

การหา Codebook นั้นสามารถค้นหาหาได้เร็วมากเพราะ Excitation Vector $v\xi$ นั้นประกอบไปด้วย 4 Non-Zero Pulse เท่านั้น กระบวนการหานั้นทำบนพื้นฐานของ 4 Nested Loops ซึ่งมีตำแหน่ง Pulse ที่ตรงกัน ที่ตำแหน่งนั้น Pulse ลูกใหม่จะถูกเพิ่มเข้ามาซึ่งค่า C ในสมการ (3.27) สามารถหาได้จากสมการ(3.30)

$$C = \alpha_0 d[m_0] + \alpha_1 d[m_1] + \alpha_2 d[m_2] + \alpha_3 d[m_3] \tag{3.30}$$

m_k คือตำแหน่งของ Pulse ที่ k และ Φ_k คือเครื่องหมาย ± 1 ค่าพลังงานของ Pulse ตำแหน่งคู่ ε ในสมการ (3.27) สามารถหาได้จากสมการ(3.31)

$$\begin{aligned} \varepsilon = \; & \Phi(m_0, m_0) \\ & + \Phi(m_1, m_1) + 2\alpha_0\alpha_1\Phi(m_0, m_1) \\ & + \Phi(m_2, m_2) + 2[\alpha_0\alpha_2\Phi(m_0, m_2) + \alpha_1\alpha_2\Phi(m_1, m_2)] \\ & + \Phi(m_3, m_3) + 2[\alpha_0\alpha_3\Phi(m_0, m_3) + \alpha_1\alpha_3\Phi(m_1, m_3) + \alpha_2\alpha_3\Phi(m_2, m_3)] \end{aligned} \tag{3.31}$$

สำหรับ ตำแหน่ง Pulse คี่ พลังงานในสมการ (3.27) จะถูกประมาณเหมือนกับ Pulse ตำแหน่งคู่ที่ถูกเลื่อนไปแล้ว การที่จะทำให้การ Search ง่ายขึ้นนั้นทำได้โดยการเปลี่ยนฟังก์ชัน $d\,[j]$ และ $\Phi(m_1, m_2)$

การปรับเปลี่ยนจะถูกกระทำโดยขั้นตอนดังนี้ ขั้นแรก สัญญาณ *s[j]* จะถูกกำหนด แล้วสัญญาณ *d'[j]* จะถูกสร้าง ตามสมการ(3.32)

$$\begin{aligned} s[2j] = s[2j+1] &= sign(d[2j]) \quad if \left|d[2j]\right| > \left|d[2j+1]\right| \\ s[2j] = s[2j+1] &= sign(d[2j+1]) \quad otherwise \end{aligned} \tag{3.32}$$

และสัญญาณ *d'* ซึ่งถูกกำหนดโดย *d'[j] = d[j]s[j]* ขั้นตอนที่สอง Matrix Φ จะถูกแก้ไขโดยการรวมกับสัญญาณข้อมูล จะได้ $\Phi'_{(i,j)} = s[i]s[j]\Phi_{(i,j)}$ ซึ่งค่า Correlation ในสมการ(3.30) คำนวณโดย

$$C = d'[m_0] + d'[m_1] + d'[m_2] + d'[m_3] \tag{3.33}$$

และพลังงานในสมการ (3.31)ถูกคำนวณ โดย

$$\begin{aligned} \varepsilon = \; & \Phi'(m_0, m_0) \\ & + \Phi'(m_1, m_1) + 2\Phi'(m_0, m_1) \\ & + \Phi'(m_2, m_2) + 2[\Phi'(m_0, m_2) + \Phi'(m_1, m_2)] \\ & + \Phi'(m_3, m_3) + 2[\Phi'(m_0, m_3) + \Phi'(m_1, m_3) + \Phi'(m_2, m_3)] \end{aligned} \tag{3.34}$$

การค้นหาโดยเน้นจุดสนใจจะถูกใช้เพื่อค้นหาครั้งต่อไป ซึ่งจะต้องมีการคำนวณค่าเทรสโฮลด์ (threshold) ก่อนที่จะเข้าสู่ Loop สุดท้าย และจะเข้า Loop นี้ก็ต่อเมื่อมีค่าเกิน Threshold ที่จำกัดไว้ ทำให้เปอร์เซนต์ที่จะต้องทำ Codebook search น้อยลง

ซึ่งการคำนวณ Threshold นั้นคิดจากค่า Correlation *C* ซึ่งเป็นค่า Correlation ที่มากที่สุดโดยสมบูรณ์ (Absolute Correlation) และค่าเฉลี่ย Correlation ที่เหมาะสมสำหรับ 3 Pulse แรก max_3 และ av_3 จะถูกพบก่อนการทำ Codebook Search ซึ่ง Threshold ถูกคำนวนโดย สมการ(3.35)

$$thr_3 = av_3 + (\max_3 - av_3)/2 \tag{3.35}$$

Loop ที่ 4 จะถูกนำมาคำนวณก็ต่อเมื่อค่า Absolute Correlation เกิน thr_3 จำนวนครั้งที่เข้า Loop สุดท้าย (สำหรับ 4 subframe) จะไม่อนุญาตให้เกิน 600 ครั้ง (ค่าเฉลี่ย Worst Case ของแต่ละ subframe จะเท่ากับ 150 ครั้ง ซึ่งจะพิจารณาโดยการหาแค่ 150 x 8 ลำดับของ Codebook และไม่คิดช่วง 3 Loop แรก)

คุณสมบัติพิเศษ Codebook คือ ถ้า Pitch Delay น้อยกว่า 60 แล้ว Pitch Contribution จะขึ้นอยู่กับ index ของ $PGInd_i$ ของ LTP Pitch Predictor Gain Vector ที่ถูกเพิ่มเข้าไปในโค้ด

หลังจาก Optimum Algebraic Code *v[n]* ถูกพิจารณาแล้ว *v[n]* จะแก้ไขโดย $v[n] \leftarrow v[n] + b(PGInd_i)v[n - L_i - e(PGInd_i)]$ ซึ่งค่า $\beta(PGInd_i)$ และ $\varepsilon(PGInd_i)$ จะถูกใส่ในตาราง และ L_i จะเป็น Pitch Period ที่เป็นจำนวนเต็ม

ขั้นตอนสุดท้าย หลังจากได้ลำดับ *v[n]* ที่ Quantizing Gain *G* แล้ว ค่า Gain นี้จะถูก Quantize แบบเดียวกับ High Rate Excitation ซึ่งทำได้โดยการเลือกจาก Gain Quantization Table และเลือกจาก index $MGInd_i$ ซึ่งทำให้ค่าของ $|G - \bar{G}_j|, 0 \le j \le 23$ น้อยลง

3.2.17 การถอดรหัสสัญญาณกระตุ้น (Excitation Decoder)

จะกระทำได้ตามขั้นตอนนี้

- หาค่า G_{max} มากที่สุด โดยใช้สมการ(3.36)

$$MGInd_i = GInd_i - PGInd_i \cdot GSize \tag{3.36}$$

ซึ่ง *Gsize* = 24 คือขนาดของตาราง $\tilde{G}$ และ $PGInd_i$ จะถูกกำหนดในหัวข้อ 3.2.18

- ตำแหน่งของ Pulse ที่ถูก Decode จะใช้ทั้งหมด $\binom{30}{M}$ รูปแบบ ซึ่ง *M* คือ 6 หรือ 5 สำหรับบิตเรทสูง และการทำถอดรหัสโดยตรงสำหรับบิตเรทต่ำโดยตำแหน่งของ index จะถูกคำนวณ
- ตำแหน่งของ Grid ได้มาจาก Grid Bit
- เครื่องหมายของ Pulse ได้รับมาจาก Sign Bits
- สำหรับบิตเรทสูงการ Decode สำหรับขบวน Pulse จะกระทำก็ต่อเมื่อ L_i น้อยกว่า 58
- เวกเตอร์ *v[n]* จะถูกสร้างใหม่จากพารามิเตอร์ที่ Decode แล้ว
- สุดท้าย Pitch Contribution *u[n]* และตำแหน่งของ Pulse Contribution *v[n]* จะถูกรวมเข้าด้วยกันในรูปแบบของ Excitation Vector *e[n]*

3.2.18 การถอดรหัสพิตช์ (Decoding of the Pitch Information)

การ Decode ของ Pitch Information จะถูกกระทำ โดยขั้นตอนดังนี้

- เริ่มแรก Lag หรือ Delay ของ Pitch Predictor สำหรับ subframe คู่จะถูก Decode โดยสมการ (3.37)

$$L_i = PInd_i + 18, \quad i = 0,2 \tag{3.37}$$

- สำหรับ Lag หรือ Delay ของ Pitch Predictor สำหรับ subfram คี่จะถูก Decode ตามสมการ(3.38)

$$L_i = L_{i-1} + \Delta_i, \quad i = 1,3 \tag{3.38}$$

- Gain Vector ของ Pitch Predictor ใน subframe ที่ i นั้น จะถูกกำหนดจาก Gain index $GInd_i$ สำหรับ Bit Rate ที่ต่ำ index นี้จะประกอบไปด้วยข้อมูลเกี่ยวกับ Pitch Predi ctor , Gain Vector และ index ของ Gain ในลำดับของ Pulse

ในกรณีนี้ Pitch Gain index $PGInd_i$ คำนวณได้จากสมการ(3.39)

$$PGInd_i = \lfloor Gind_i / GSize \rfloor, \quad i = 0...3 \tag{3.39}$$

โดยที่ $\lfloor x \rfloor$ คือค่าจำนวนเต็มที่มากสุดที่ $\leq x$ สำหรับบิตเรทสูงในกรณีที่ L_i มากกว่า 58 index นี้ก็จะได้รับมาเช่นเดียวกับในสมการ(3.39) นอกนั้น $PGInd_i$ จะเท่ากับ 170 entries Gain Vector Codebook

ถ้า index มีค่าเท่ากับ 85 entries และประกอบไปด้วย Train Impulse ในกรณีนี้ Pitch index จะหาจากสมการ

$$PGInd_i = \lfloor Gind_i \ \& \ 0x7FF / GSize \rfloor, \quad i = 0...3 \tag{3.40}$$

Lag และ Gain Vector ของ Pitch Predictor จะถูก Decode จาก index เหล่านี้และถูกใช้ประโยชน์สำหรับ Pitch Contribution $u[n]$ extraction ตามสมการ (3.41.1) และ (3.41.2)

$$\begin{aligned} e'[0] &= e[-L_i - 2] \\ e'[1] &= e[-L_i - 1] \\ e'[n] &= e[(n \bmod L_i) - L_i], \quad 2 \leq n \leq 63 \end{aligned} \tag{3.41.1}$$

$$u[n] = \sum_{j=0}^{j=4} \beta_{ij} e'[n+j], \quad 0 \leq n \leq 59 \tag{3.41.2}$$

3.2.19 การอัพเดทหน่วยความจำ (Memory Update)

หน้าที่สุดท้ายของ subframe ที่ i ก่อนที่จะทำ subframe ต่อไป คือ การ Update Memory ของ Synthesis Filter $\widetilde{A}_i(Z)$ และ Formant Perceptual Weighting Filter $W_i(z)$ และ Harmonic Loop Shaping Filter $P_i(z)$

ก่อนที่จะ Encoder ใน subframe ต่อไปจะต้อง Update Memory ได้แก่ค่า Synthesis Filter , Formant Perceptual Weighting Filter และ Harmonic Loop shaping Filter สำหรับใช้ใน subframe ถัดไปที่จะใช้ในส่วนของการคำนวณ Zero Input Response

3.2.20 การจัดสรรตำแหน่งบิต (Bit Allocation)

ข้อแตกต่างที่สำคัญระหว่างบิตเรทต่ำและบิตเรทสูงนี้คือ ตำแหน่งของ Pulse และการเข้ารหัสแอมพลิจูด เช่นเดียวกันที่บิตเรทต่ำ 170 Codebook entries จะถูกใช้สำหรับ Gain Vector ของการประมาณในระยะยาว (Long Term Predictor) ดูตารางที่ 3-2, 3-3, 3-4

Parameters code	Subframe0	Subframe1	Subframe2	Subframe3	Total
LPC indices					24
Adaptive Codebook Lags	7	2	7	2	18
All the Gains combined	12	12	12	12	48
Pulse positions	20	18	20	18	73(note)
Pulse signs	6	5	6	5	22
Grid index	1	1	1	1	4
Total:					189

ตารางที่ 3-2 แสดง bit allocation ของบิตเรท 6.3 kbit/s

Parameters code	Subframe0	Subframe1	Subframe2	Subframe3	Total
LPC indices					24
Adaptive Codebook Lags	7	2	7	2	18
All the Gains combined	12	12	12	12	48
Pulse positions	12	12	12	12	48
Pulse signs	4	4	4	4	16
Grid index	1	1	1	1	4
Total:					158

ตารางที่ 3-3 แสดง bit allocation ของบิตเรท 5.3 kbit/s

Name	Transmitted parameters	High rate # bits	Low rate # bits
LPC	LSP VQ index	24	24
ACL0	Adaptive Codebook Lag	7	7
ACL1	Differential Adaptive Codebook Lag	2	2
ACL2	Adaptive Codebook Lag	7	7
ACL3	Differential Adaptive Codebook Lag	2	2
GAIN0	Combination of adaptive and fixed Gains	12	12
GAIN1	Combination of adaptive and fixed Gains	12	12
GAIN2	Combination of adaptive and fixed Gains	12	12
GAIN3	Combination of adaptive and fixed Gains	12	12
POS0	Pulse positions index	20 (note)	12
POS1	Pulse positions index	18 (note)	12
POS2	Pulse positions index	20 (note)	12
POS3	Pulse positions index	18 (note)	12
PSIG0	Pulse sign index	6	4
PSIG1	Pulse sign index	5	4
PSIG2	Pulse sign index	6	4

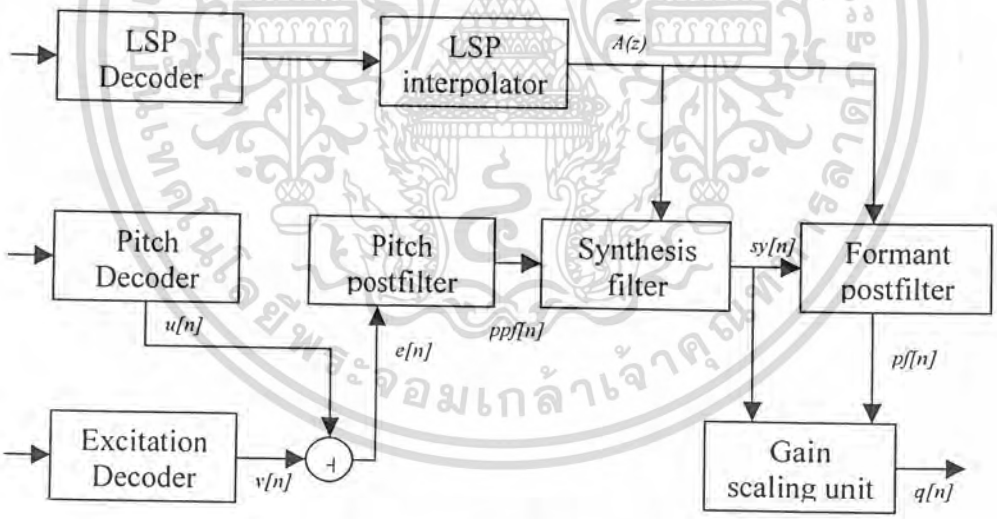
PSIG3	Pulse sign index	5	4
GRID0	Grid index	1	1
GRID1	Grid index	1	1
GRID2	Grid index	1	1
GRID3	Grid index	1	1
Note -			

ตารางที่ 3.4 แสดง พารามิเตอร์ที่ถูกส่งออกไป

3.2.21 การกำหนดค่าเริ่มต้นของการเข้ารหัส (Coder Initialization)

ค่า Static Coder จะถูกกำหนดให้เป็นศูนย์ ยกเว้น LSP Vector ก่อนหน้านี้ที่จะถูกกำหนด ด้วยค่า p_{DC}

3.3 หลักการการถอดรหัสเสียง (Decoder Principles)



รูปที่ 3-2 แสดง Block diagram ของ speech Decoder

3.3.1 หลักการทั่วไป

การ Decoder จะกระทำแบบ frame ต่อ frame ในขั้นแรก LPC ที่ถูก Quantize แล้วจะถูก Decode ต่อมาตัวถอดรหัสจะสร้าง LPC Synthesis Filter สำหรับทุกๆ subframe ทั้ง Adaptive

Codebook Excitation และ Fixed Codebook Excitation จะถูก Decode และเป็น input ให้แก่ Synthesis Filter เพื่อใช้สร้างสัญญาณเสียง

ในส่วน Adaptive Postlilter จะประกอบด้วยฟอร์มแมนท์ (Formant) และฟอร์เวิร์ด-แบ็คเวิร์ดพิตช์โพสต์ฟิลเตอร์ (Forward-Backward Pitch Postfilter) สัญญาณ Excitation จะเป็น input ให้แก่ Postfilter และ out put ที่ออกจาก Postfilter จะเป็น input ให้แก่ Synthesis Filter และ output ที่ได้จะเป็น input ให้แก่ฟอร์มแมนท์โพสท์ฟิลเตอร์ (Formant Postfilter)

ในส่วนของการเพิ่มอัตราขยาย (Gain Scaling Unit) จะรักษาระดับพลังงานของ input ที่เข้ามาที่ Gain Scaling Unit ให้เท่ากับระดับพลังงานของ input ที่เข้ามาที่ Formant Filter

3.3.2 การถอดรหัสสัญญาณคู่สเปกตรัมเชิงเส้น (LSP Decoder)

เหมือนกับหัวข้อ 3.2.6

LSP ที่ถูก Quantize จากฝั่ง Encoder จะถูก Decode หลังจากนั้นจะมีการตรวจสอบว่า LSP Vector ที่ถูก Decode นั้นเป็นไปตามเงื่อนไขที่กำหนดหรือไม่ ถ้าไม่เป็นไปตามที่กำหนดก็จะใช้ LSP Vector ปัจจุบัน และ LSP Vector ถัดไปมาปรับปรุงเพื่อให้เป็นไปตามเงื่อนไขที่กำหนด เพื่อให้วงจร Filter ที่สร้างขึ้นนั้นมีความเสถียร

3.3.3 การอินเทอร์โพเลทค่าสัญญาณคู่สเปกตรัมเชิงเส้น (LSP Interpolator)

เหมือนกับหัวข้อ 3.2.7

จะนำ ค่า LSP ที่ Decode แล้วมา Convert กลับให้เป็น LPC Coefficient ดังนั้น output ที่ได้จะเป็นค่า LPC Coefficient 10 ค่า

3.3.4 การถอดรหัสพิตช์ (Decoding of the Pitch Information)

เหมือนกับหัวข้อ 3.2.18

สำหรับทุกๆ subframe Adaptive Codebook ที่ประกอบด้วยค่า Pitch Delay และ Pitch Gain จะถูก Decode สำหรับค่า Delay นั้นจะแยกคำนวณตาม subframe คู่หรือคี่ ในส่วนของ Gain จะแยกคำนวณตามอัตราการส่งว่าเป็น Bit Rate ต่ำ (5.3 kbit/s) หรือ Bit Rate สูง (6.3 kbit/s) หลังจากนั้นจะนำค่า Delay และ Gain ที่คำนวณได้มาสร้างเป็น Pitch Contribution เพื่อใช้ในการหา Excitation Vector ต่อไป

3.3.5 การถอดรหัสสัญญาณกระตุ้น (Excitation Decoder)

เหมือนกับหัวข้อ 3.2.17

fix Codebook สำหรับทุกๆ subframe ที่ประกอบด้วยค่าพารามิเตอร์(parameter) ต่างๆ ได้แก่ Gain , Pulse position , Grid Position และ Pulse sign สำหรับ rate 5.3 kbit/s และ 6.3 kbit/s ค่าทั้งหมดเหล่านี้จะถูกนำมา Decode ดังนั้น output ที่ได้จะเป็น Pulse Contribution ที่ถูกสร้างขึ้นมาใหม่หลังจากที่ถูก Decode แล้ว

3.3.6 พิตช์โพสฟิลเตอร์ (Pitch Postfilter)

จะใช้ค่า output ที่ได้จาก Pitch Decoder (ค่า Pitch Contrib ution) และ Ex citation D ecoder (Pulse Contribution) มาคำนวณหา Excitation Vector

Pitch Postfilter จะใช้สำหรับปรับปรุงคุณภาพของ Synthesis Signal ซึ่งจะกระทำอย่างนี้ทุกๆ subframe และปรับปรุง subframe เหล่านั้น

subframe ที่ต้องการใช้จะต้องเป็น Signal Excitation Frame ที่สมบูรณ์ $\{e[n]\}_{n=0..239}$ ที่จะถูกสร้างขึ้นและบันทึกไว้ การปรับปรุงคุณภาพจะทำโดยการเพิ่ม SNR ที่เป็นผลคูณของ Pitch Period Signal ที่ถูก Postfilter แล้ว $\{ppf[n]\}_{n=0.59}$ ซึ่งจะถูกเก็บไว้จาก Excitation Signal ที่ถูก Decode แล้ว$\{e[n]\}_{n=0..239}$ ที่สามารถแสดงได้ด้วยสมการ(3.42)

$$\begin{aligned} ppf[n] &= g_p \cdot \{e[n] + g_{lfp}(w_f \cdot g_f \cdot e[n+M_f] + w_b \cdot g_b \cdot e[n-M_b])\} \\ &= g_p \cdot ppf'[n] \end{aligned} \tag{3.42}$$

โดยที่ $e[n]$ คือ Excitation Signal ที่ถูก Decode แล้ว การคำนวณ Gain g_p, g_f , g_b และ Delay M_f , M_b จะอยู่บนพื้นฐานของ Forward and Backward Analysis ส่วน Weights w_f , w_b อาจจะต้องใช้ค่า (0,0),(0,1),(1,0) ซึ่ง Delay จะถูกเลือกโดย Cross-Correlation ที่มีค่าสูงสุด โดยที่ค่า Cross-Correlation สำหรับ Forward Pitch Lag ถูกกำหนดโดยสมการ(3.43.1)

$$C_f = \sum_{n=0}^{59} e[n]e[n+M_f], \quad M_1 \le M_f \le M_2 \tag{3.43.1}$$

และ Cross-Correlation สำหรับ Backward Pitch Lag ถูกกำหนดโดยสมการ(3.43.2)

$$C_b = \sum_{n=0}^{59} e[n]e[n+M_b], \quad M_1 \le M_b \le M_2 \tag{3.43.2}$$

โดยที่ $M_1=L_1-3$ และ $M_2=L_1+3$ และ $\{L_i\}_{i=0,2}$ ที่ได้มาจาก Pitch Lag สำหรับ subframe ที่ 1 และ 3 L_0 จะถูกใช้สำหรับ 2 subframe แรก ส่วน L_2 จะถูกใช้สำหรับ 2 subframe สุดท้าย ถ้า 1 ใน Maximum Correlations มีค่าเป็นลบ หรือบาง n ในช่วง 0..59 ไม่มีค่า Sample $e[n+M_i]$ แล้ว Corresponding และ Delay จะถูกกำหนดให้เป็นศูนย์ ซึ่งมีกรณีที่เป็นไปได้ 4 กรณี คือ

0) ค่าที่ได้สูงสุด C_f และ C_b เป็นค่าลบและไม่มีค่า Pitch Postfilter Weight ที่ต้องการคำนวณ

1) ถ้าเฉพาะ Forward Maximum เท่านั้นที่มีค่าเป็นบวก ดังนั้นค่านี้จะถูกเลือก

2) ถ้าเฉพาะ Backward Maximum เท่านั้นที่มีค่าเป็นบวก ดังนั้นค่านี้จะถูกเลือก

3) ถ้าทั้ง 2 ค่ามีค่าเป็นบวก ตัวที่ทำให้ค่า Contribution มีค่ามากกว่าจะเป็นตัวที่ถูกเลือก

สำหรับ กรณี1) ,2) ,3) จะเป็นค่าพลังงานของสัญญาณที่ได้รับ (Relevant Signal Energy:T_{en} ,D_f and/or D_b) สำหรับ Pitch Lag ที่ดีที่สุด(M_f หรือ M_b) จะถูกคำนวณตามสมการ

$$D_f = \sum_{n=0}^{59} e[n+M_f]e[n+M_f] \tag{3.44.1}$$

$$D_b = \sum_{n=0}^{59} e[n+M_b]e[n+M_b] \tag{3.44.2}$$

$$T_{en} = \sum_{n=0}^{59} e[n]e[n] \tag{3.44.3}$$

Forward Energy คำนวณโดยสมการ(3.45.1)

$$E_f = \sum_{n=0}^{59} (e[n]-g_f e[n+M_f])^2 \tag{3.45.1}$$

Backward Energy คำนวณโดยสมการ(3.45.2)

$$E_b = \sum_{n=0}^{59} (e[n] - g_b e[n + M_b])^2 \quad (3.45.2)$$

E จะถูกทำให้มีค่าน้อย โดยใช้ค่า C^2/D ในกรณี 3) การเลือกค่ะหว่าง Forward and Backward จะเลือกโดยใช้ค่า C^2_f/D_f หรือ C^2_b/D_b ที่มีค่ามากกว่า

ค่า prediction Gain จะมีค่าเท่ากับ $-10 \log (1- (C^2/DT_{en}))$ ถ้าหากว่า Gain นี้น้อยกว่า 1.25 dB แล้ว Contribution จะถูกประมาณว่ามีค่าเล็กน้อยเท่านั้น และจะไม่มีค่า Pitch Postfilter ให้ใช้ ถ้าค่า Pitch Postfilter ถูกใช้ ค่าที่เหมาะสมที่สุดจะหาจากสมการ(3.46)

$$g = \frac{C}{D} \quad (3.46)$$

ค่า Gain ที่ดีจะถูกคูณด้วย Weighting Factor γ_{ltp} ที่มีค่าเท่ากับ 0.1875 สำหรับ Bit Rate สูงและ 0.25 สำหรับ Bit Rate ต่ำ สุดท้าย Scaling g_p จะถูกคำนวณโดยสมการ(3.47)

$$g_p = \sqrt{\frac{\sum_{n=0}^{59} e^2[n]}{\sum_{n=0}^{59} (ppf'[n])^2}} \quad (3.47)$$

ในสมการ(3.47) ถ้าตัวหารน้อยกว่าตัวเศษ ค่า Gain จะถูก set เป็น 1

3.3.7 การสร้างสัญญาณเสียง (LPC Analysis Filter)

นำค่า LPC Coefficient ที่ได้จาก LSP interpolator มาสร้างเป็น LPC Filter และนำสัญญาณ Excitation Signal ที่ผ่านการทำ Pitch Postfilter มาแล้วมาป้อนให้กับ LPC Filter เพื่อสร้างเป็นสัญญาณเสียงขึ้นมา รวมทั้งมีการ Update Memory ด้วย โดย พารามิเตอร์ของ frame ก่อนหน้านี้ 1 frame ที่ถูกนำมาคำนวณแล้วจะอัพเดทให้เป็นของเก่าแทน

10^{th} LPC Analysis Filter $\widetilde{A}_i(Z)$ ถูกใช้เพื่อสร้างสัญญาณเสียง *sy[n]* จาก *ppf[n]* ตามสมการ(3.48)

$$sy[n] = ppf[n] + \sum_{j=1}^{10} \widetilde{a}_{ij} sy[n-j] \tag{48}$$

3.3.8 การปรับปรุงคุณภาพสัญญาณเสียงในช่วงความถี่สูง (Formant Postfilter)

นำค่าสัญญาณเสียงที่ได้จาก Synthesis Filter มาคำนวณหาพลังงานเพื่อใช้ปรับปรุงคุณภาพสัญญาณเสียงในช่วงความถี่สูงให้เสียงมีความไพเราะมากขึ้น

ARMA จะถูกใช้และ Transfer Function ของ short term Postfilter จะถูกใช้โดยสมการ

$$k = \frac{\sum_{n=1}^{59} sy[n]sy[n-1]}{\sum_{n=0}^{59} sy[n]sy[n]} \tag{3.49.1}$$

$$k_1 = \frac{3}{4}k_{1old} + \frac{1}{4}k \tag{3.49.2}$$

$$F(z) = \frac{1 - \sum_{i=1}^{10} \widetilde{a}_i \gamma'^{i}_{1} z^{-1}}{1 - \sum_{i=1}^{10} \widetilde{a}_i \gamma'^{i}_{2} z^{-1}} (1 - 0.25 k_1 z^{-1}) \tag{3.49.3}$$

โดยที่ λ_1 = 0.65 และ λ_2 =0.75 k เป็น Autocorrelation Coefficient ที่ถูกประมาณจากสัญญาณเสียง *sy[n]* ที่ถูกสร้างแล้ว และ k_{1old} คือ ค่าของ k_1 จาก subframe ก่อนหน้านี้ที่ผ่าน Postfilter แล้ว

pf[n] จะเป็น output ของ Formant Postfilter จาก input Signal *sy[n]* ที่เข้ามา

3.3.9 การเพิ่มอัตราขยาย (Gain Scaling Unit)

เป็นการขยายสัญญาณเสียงด้วยการคูณด้วยค่า Gain ที่ได้มาจากการคำนวณสัญญาณเสียงที่สร้างมาจาก Synthesis Filter กับสัญญาณเสียงที่ได้จาก Formant Postfilter

Gain scaling unit นี้รับเข้ามา 2 Input Vector คือ Speech Vector ที่ถูกสร้าง $\{sy[n]\}_{n=0..59}$ และ Output Vector ที่ผ่าน Postfilter แล้ว $\{pf[n]\}_{n=0..59}$

เริ่มต้นอัตราแอมพลิจูด (Amplitude Ratio) g_s จะคำนวณโดย สมการ(3.50)

$$g_s = \sqrt{\frac{\sum_{n=0}^{59} sy^2[n]}{\sum_{n=0}^{59} pf^2[n]}} \tag{3.50}$$

ถ้าหากตัวส่วนเป็นศูนย์(0) ค่า g_s จะถูก set เป็น 1

Output Vector *q[n]* ที่ได้จะถูกเก็บไว้โดย Scaling Signal ที่ผ่าน Postfilter แล้วหลังจากนั้น *pf[n]* และ Gain g_s จะถูก Update โดยสมการ(3.51),(3.52)

$$g[n] = (1-\alpha)g[n-1] + \alpha g_s \tag{3.51}$$

$$q[n] = pf[n] \cdot g[n] \cdot (1+\alpha) \tag{3.52}$$

3.3.10 การควบคุมการอินเทอร์โพเลทเฟรม (Frame Interpolation Handling)

อัลกอริทึมนี้ถูกออกแบบมาให้มีความเหมาะสมสำหรับที่จะบ่งชี้ frame ที่จะต้องถูกลบทิ้ง ซึ่งจะอยู่ในส่วนของการ Decoder

อย่างไรก็ตามวิธีการลบ frame ต้องทำด้วยความรวดเร็วโดยอาศัยบิตสตรีม (BitStream) เพราะว่าอัลกอริทึมนี้ออกแบบมาสำหรับเบิร์สเออเรอร์ (Burst Error) เนื่องจากไม่มีเครื่องมือใดที่จะสามารถสุ่มหาบิตเออเรอร์(Bit Error) ได้

ถ้าหากมี frame ที่ต้องลบ frame ส่วนของ Decoder จะเปลี่ยนจากการทำงานปกติมาเป็นการทำงานสำหรับการปกปิด frame ที่ต้องถูกลบ

ขั้นตอน Frame interpolation จะถูกกระทำอย่างอิสระสำหรับ LSP Coefficient และ Residual Signal

- LSP Interpolation

การ Decode ของ LSP Coefficient ใน Mode Frame Interpolation จะทำตามขั้นตอนดังนี้

1) set Vector $\tilde{e}_n$ เท่ากับศูนย์(0)

2) Vector ที่ถูก Predict แล้ว $\bar{p}_n$ จะถูกเพิ่มเข้าไปใน Vector $\tilde{e}_n$ และ Vector p_{DC} เพื่อให้อยู่ในรูป LSP ที่ถูก Decode แล้ว ($\tilde{p}_n$) เพื่อใช้สร้าง $\bar{p}_n$ ค่า b_e จะเท่ากับ 23/32

ตั้งแต่จุดนี้เป็นต้นไป การถอดรหัส LSP จะเหมือนกับหัวข้อ 3.2.6 ยกเว้น $\Delta_{min} = 62.5$ Hz ซึ่งจะดีกว่าใช่ค่า 31.25 Hz

- Residual interpolation

Residual Interpolation จะถูกกระทำใน 2 วิธีที่ต่างกัน ขึ้นอยู่กับ frame ที่ดีก่อนหน้านี้ก่อนที่จะมีการลบ frame

การแบ่งระดับชั้น (Classifier) ซึ่งจะใช้พื้นฐานของฟังก์ชันที่ให้ค่า Cross–Correlation มากที่สุด โดย 120 samples สุดท้ายเฟรมจะถูกคิดค่า Cross-Correlate ด้วย $L_2 \pm 3$ index ที่ชี้ถึงค่า Correlation สูงสุดจะถูกนำไปเป็น Interpolation index แล้ว Prediction Gain ของ Vector ที่ดีที่สุดก็จะถูกทดสอบ ถ้า Gain นี้มากกว่า 0.58 dB เฟรมนั้นก็จะถูกประกาศให้เป็น Voiced ส่วนเฟรมที่เหลือจะถูกประกาศให้เป็น Unvoiced

Classifier จะ Return ศูนย์สำหรับกรณีที่เป็น Unvoiced และจะถูกประมาณค่า Pitch ในกรณีที่เป็น Voiced ถ้าเฟรมถูกประกาศให้เป็น Voiced ค่า Gain เฉลี่ยที่บ่งชี้ถึง subframe ที่ 2 และ 3 จะถูกเก็บไว้

ถ้าเฟรมปัจจุบันถูกบ่งชี้ว่าต้องลบทิ้ง ถ้าหากว่า frame ก่อนหน้านี้ถูกประกาศเป็น Unvoiced ดังนั้น เฟรมปัจจุบันจะถูกสร้างโดยใช้ตัวสร้างสัญญาณยูนิฟอร์มแบบสุ่ม (Uniform random number generator) ซึ่ง output ที่ได้จะถูกนำมาขยายให้มีขนาดเท่ากับ Gain ที่คำนวณได้ก่อนหน้านี้

ในกรณี Voiced ส่วนของเฟรมปัจจุบันจะถูกสร้างขึ้นอีกครั้งด้วยสัญญาณกระตุ้นแบบคาบ (Periodic Excitation) ที่มีค่าคาบเท่ากับค่าที่ประมาณโดย Classifier

ถ้าหากว่าเฟรมถูกลบต่อเนื่องกัน 3 เฟรมแล้ว Vector ที่ถูกสร้างขึ้นใหม่จะถูกทำให้มีขนาดน้อยลง สำหรับแต่ละเฟรมซึ่งหลังจากที่ 3 เฟรมถูก Interpolate ไปแล้ว จะทำ output ให้เงียบขึ้น (Amplitude ลดลง) อย่างสมบูรณ์

3.3.11 การกำหนดค่าเริ่มต้นของการถอดรหัส (Decoder Initialization)

ค่า static coder ทุกค่าควรจะถูกกำหนดค่าเริ่มต้นเป็นศูนย์(0) แต่มีข้อยกเว้นดังนี้

- LSP Vector ก่อนหน้านี้ควรถูกกำหนดค่าเริ่มต้นเป็น LSP DC Vector : p_{DC}
- Postfilter Gain : *g[-1]* ควรถูกกำหนดค่าเริ่มต้นเป็น 1

3.4 ขบวนบิตที่ถูกส่งออกไป (Bitstream Packing)

Note - แต่ละ bit ที่ถูกส่ง ส่วนของ parameter จะถูกเรียกว่า PAR(x)_By ซึ่ง PAR เป็นชื่อของ parameter และ x เป็น subframe index PARx_By...PARx_Bz ใช้ค่าในช่วงของการ Transmit Bit ตั้งแต่ bit y ถึง bit z ส่วนบิตที่ไม่ถูกใช้เรียกว่า UB (value=0)

RATEFLAG_B0 เป็นตัวบอกว่า high rate(0) หรือ low rate (1) ที่จะถูกใช้ใน Current Frame

VADFLAG_B0 บอกว่า Active speech(0) หรือ Non-active speech(1)

ผลรวมของ RATEFLAG_B0 กับ VADFLAG_B0 จะถูก set เป็น 1 เพื่อเก็บไว้ใช้ในอนาคต

Transmitted octets	PARx By,...
1	LPC_B5...LPC_B0, VADFLAG_B0, RATEFLAG_B0
2	LPC_B13...LPC_B6
3	LPC_B21...LPC_B14
4	ACL0_B5...ACL0_B0, LPC_B23, LPC_B22
5	ACL2_B4...ACL2_B0, ACL1_B1, ACL1_B0, ACL0_B6
6	GAIN0_B3...GAIN0_B0, ACL3_B1, ACL3_B0, ACL2_B6, ACL2_B5
7	GAIN0_B11...GAIN0_B4
8	GAIN1_B7...GAIN1_B0
9	GAIN2_B3...GAIN2_B0, GAIN1_B11...GAIN1_B8
10	GAIN2_B11...GAIN2_B4
11	GAIN3_B7...GAIN3_B0
12	GRID3_B0, GRID2_B0, GRID1_B0, GRID0_B0, GAIN3_B11...GAIN3_B8
13	MSBPOS_B6...MSBPOS_B0, UB
14	POS0_B1, POS0_B0, MSBPOS_B12...MSBPOS_B7
15	POS0_B9...POS0_B12
16	POS1_B2, POS1_B0 , POS0_B15...POS0_B10
17	POS1_B10... POS1_B3
18	POS2_B3... POS2_B0, POS1_B13... POS1_B11
19	POS2_B11... POS2_B4
20	POS3_B3... POS3_B0, POS2_B15... POS2_B12
21	POS3_B11... POS3_B4
22	PSIG0_B5...PSIG0_B0, POS3_B13, POS3_B12
23	PSIG2_B2...PSIG2_B0, PSIG1_B4...PSIG1_B0
24	PSIG3_B4...PSIG3_B0, PSIG2_B5...PSIG2_B3

ตารางที่ 3-5 แสดง Octet bits Packing for the Bit Rate สูง

Transmitted octets	PARx By,...
1	LPC_B5...LPC_B0, VADFLAG_B0, RATEFLAG_B0
2	LPC_B13...LPC_B6
3	LPC_B21...LPC_B14
4	ACL0_B5...ACL0_B0, LPC_B23, LPC_B22
5	ACL2_B4...ACL2_B0, ACL1_B1, ACL1_B0, ACL0_B6
6	GAIN0_B3...GAIN0_B0, ACL3_B1, ACL3_B0, ACL2_B6, ACL2_B5
7	GAIN0_B11...GAIN0_B4
8	GAIN1_B7...GAIN1_B0
9	GAIN2_B3...GAIN2_B0, GAIN1_B11...GAIN1_B8
10	GAIN2_B11...GAIN2_B4
11	GAIN3_B7...GAIN3_B0
12	GRID3_B0, GRID2_B0, GRID1_B0, GRID0_B0, GAIN3_B11...GAIN3_B8
13	POS0_B7...POS0_B0
14	POS1_B3...POS1_B0, POS0_B11...POS0_B8
15	POS1_B11...POS1_B4
16	POS2_B7...POS2_B0
17	POS3_B3... POS3_B0, POS2_B11...POS2_B8
18	POS3_B11... POS3_B4
19	PSIG1_B3... PSIG1_B0, PSIG0_B3... PSIG0_B0
20	PSIG3_B3... PSIG3_B0, PSIG2_B3... PSIG2_B0

ตารางที่ 3-6 *แสดง Octet bits Packing for the Bit Rate ต่ำ*

บทที่ 4

Reentrant and Algorithm Standard

การปรับปรุงโปรแกรมให้สามารถรองรับการทำงานแบบหลายช่องสัญญาณ (Multichannel) ได้นั้น จะต้องมีการจัดการในส่วนของทรัพยากรหรือหน่วยความจำให้มีความเหมาะสมเรียกว่าการทำ Reentrant ในขณะเดียวกันหากต้องการให้อัลกอริทึมสามารถทำงานร่วมกับอัลกอริทึมอื่นๆได้อย่างมีประสิทธิภาพ และมีความสามารถพื้นฐานเหมือนกันภายใต้การประมวลผลบน DSP จำเป็นต้องทำให้อัลกอริทึมหรือโปรแกรมเหล่านั้นให้เป็นมาตรฐานเดียวกันเรียกว่า Algorithm Standard

4.1 Reentrant

Reentrant เป็นคุณสมบัติของที่จะยอมให้หลายโปรแกรมหรือชุดคำสั่งสามารถทำงานพร้อมกันได้ ซึ่งเป็นประโยชน์มากสำหรับ โปรแกรมที่ทำ Multichannel ซึ่งควรมีคุณสมบัติดังนี้

- ข้อมูลที่จะแก้ไขนั้นอยู่ในวัตถุเฉพาะ (Instance Object)
- สำหรับตัวแปรที่ไม่จำกัดการใช้ (Global Variable) และตัวแปรคงค่าไว้ (Static -Variable) จะใช้ในการอ่านค่าอย่างเดียว

```
CODCNGDEF CodCng;
CODSTATDEF  CodStat ;
void Init(void) {
 int I;
 CodCng.CurGain = (FLOAT)0.0;
 CodCng.Acf[i] = (FLOAT)0.0;
 CodCng.SidLpc[i] = (FLOAT)0.0;
 CodStat.PrevLsp[i] = LspDcTable[i];
 CodStat.Err[i] = Err0;
 CodCng.PastFtyp = 1;
 CodCng.RandSeed = 12345;
 return;
}
```

รูปที่ 4-1 แสดงโปรแกรมที่ไม่เป็น Reentrant

จากรูปที่ 4-1 จะเห็นว่าเมื่อเรียกใช้ Function init การใช้ตัวแปรประเภท Global Variable CodCng และ CodStat ทำให้การเรียกใช้ function init จากแต่ละช่องสัญญาณหรือ Channel นั้นไม่แบ่งแยกกันชัดเจน เพราะต้องใช้ Global Variable ร่วมกัน

การแก้ไขสามารถทำได้โดยการรวม Global Variable นั้น (หรือ Static Variable) เอาไว้ใน Instance Object เมื่อ function ต้องการใช้ Global Variable นั้น ให้นำ Instance Object เข้ามาใน function ด้วย ดังรูปที่ 4.2

```
typedef struct {
    CODCNGDEF CodCng;
    CODSTATDEF  CodStat ;
} Inst_Obj;

void Init(Inst_Obj *Obj) {
  int I;
  Obj->CodCng.CurGain = (FLOAT)0.0;
  Obj->CodCng.Acf[i] = (FLOAT)0.0;
  Obj->CodCng.SidLpc[i] = (FLOAT)0.0;
  Obj->CodStat.PrevLsp[i] = LspDcTable[i];
  Obj->CodStat.Err[i] = Err0;
  Obj->CodCng.PastFtyp = 1;
  Obj->CodCng.RandSeed = 12345;
 return;
}
```

รูปที่ 4-2 แสดงโปรแกรมที่เป็น Reentrant

จากรูปที่ 4-2 โปรแกรมที่นี้ถูกแก้ไขให้มีคุณสมบัติ Reentrant เรียบร้อยแล้ว จะเห็นว่า Global Variable ได้ถูกรวมเอาไว้ใน Instance Object การทำเช่นนี้จะมีผลดีในกรณีที่มีหลายช่องสัญญาณ เพราะทำให้สามารถสร้าง Instance Object ให้แต่ละช่องสัญญาณได้เลย เมื่อเรียกใช้ Function init จะใช้ Instance Object ที่ตัวเองสร้างมาใช้ในการประมวลผล

4.2 Algorithm Standard

ในปัจจุบัน การพัฒนาโปรแกรมแต่ละโปรแกรมอาจใช้อัลกอริึม (Algorithm) หลากหลายในการทำงานที่แตกต่างกัน เพราะฉะนั้นควรมีการวางโครงร่างพื้นฐานที่ทำให้แต่ละอัลกอริึมนั้นทำงานได้ในสภาพแวดล้อมเดียวกันและทำให้ผู้ใช้สามารถที่จะสลับสับเปลี่ยนแต่ละอัลกอริึมได้โดยไม่ต้องทำการเรียบเรียงโปรแกรมใหม่

ประโยชน์ของ Algorithm Standard

- ผู้พัฒนา Algorithm มีแบบแผนในการเขียน ไม่ต้องทำความเข้าใจกับส่วนการจัดการหน่วยความจำที่ซับซ้อน
- ผู้พัฒนาโปรแกรม มีรูปแบบการติดต่อหรือเรียกใช้ Algorithm ต่างๆ ที่ถูกพัฒนาจากหลายผู้ผลิตอย่างเดียวกัน และสามารถทำการเปรียบเทียบการทำงานของแต่ละ อัลกอริึมได้อย่างง่ายดาย

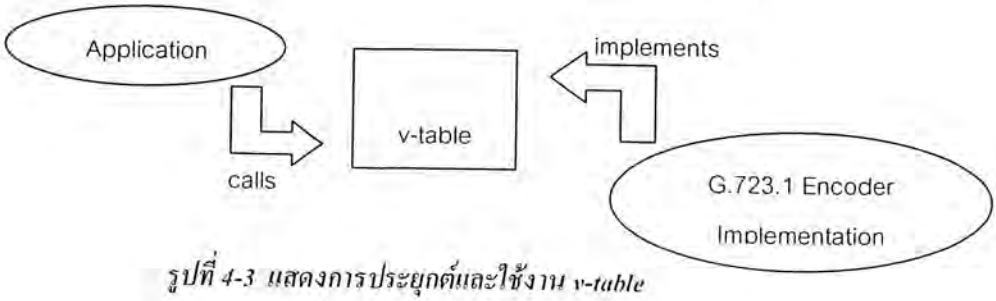
Algorithm Standard เน้นเฉพาะการติดต่อระหว่างอัลกอริึมและระบบทั้งหมด มากกว่าการพัฒนาประสิทธิภาพในการทำงานของอัลกอริึมนั้น โดยจะเน้นไปที่การจัดการทรัพยากร เช่น การใช้หน่วยความจำ การจัดการรูปแบบส่งข้อมูลเข้าและออก ซึ่งมีประโยชน์เมื่อมีหลายอัลกอริึมทำงานร่วมกัน

Algorithm Standard จะกำหนดให้แต่ละอัลกอริึมไม่สามารถใช้ทรัพยากรของระบบโดยตรง แต่ให้จัดการผ่านส่วนที่ Algorithm Standard กำหนดไว้ให้ (Application Program Interface: API)

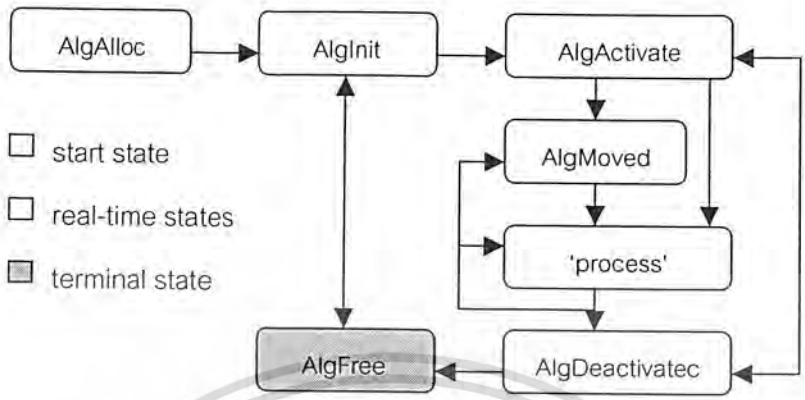
4.2.1 DSP Algorithm Standard API

ผู้พัฒนา Algorithm ต้องใช้งาน IALG ซึ่งเป็นรูปแบบพื้นฐานการติดต่อที่กำหนดโดย Algorithm Standard นอกจากนี้เพื่อให้แต่ละอัลกอริึมทำงานได้ ผู้พัฒนาต้องทำการขยายรูปแบบการติดต่อจาก IALG และสร้างเป็นตารางของ Function Pointer (v-table) โดยผู้ใช้อัลกอริึมนั้นจะจัดการอัลกอริึมโดยเรียกผ่านทาง v-table นี้ ดังรูปที่ 4-3

Function ที่กำหนดใน v-table นั้นมีหลายชนิด บางชนิดเป็นแค่ส่วนเพิ่มเติมเพื่อให้ Algorithm ทำงานได้หลากหลาย เช่น algAlloc(), algInit(), algActivate(), algMoved, algDeactivate(), algFree() เป็นต้นดังรูปที่ 4-4



รูปที่ 4-3 แสดงการประยุกต์และใช้งาน v-table



รูปที่ 4-4 แสดงลำดับการทำงาน Function ใน v-table

Function	Description
algActivate()	เป็น function ที่ทำหน้าที่เตรียมหน่วยความจำชั่วคราวที่จำเป็นในการคำนวณสำหรับ Algorithm นั้นๆ
algAlloc()	ทำหน้าที่จองหน่วยความจำสำหรับ Algorithm นั้นๆ
algDeactivate()	ทำการบันทึกค่าของตัวแปรจากหน่วยความจำที่ใช้ในการคำนวณ ไปไว้ในหน่วยความจำสำหรับเก็บค่า
algFree()	ทำหน้าที่คืนหน่วยความจำที่จองไว้สำหรับ Algorithm
algInit()	ทำการจัดเตรียมค่าที่จำเป็นสำหรับ Algorithm ในการประมวลผล
algMoved()	ทำการอัพเดทค่าหน่วยความจำในกรณีที่ Algorithm นั้นถูกเคลื่อนย้ายโดยผู้ใช้

ตารางที่ 4-1 แสดงตัวอย่างฟังก์ชันที่กำหนดใน v-table

4.2.2 การออกแบบโปรแกรม G.723.1 เพื่อทำเป็น Algorithm Standard

การปรับปรุงโปรแกรมให้เป็น Algorithm standard จำเป็นต้องสร้าง Instance Object สำหรับเก็บค่าตัวแปรต่างๆที่ใช้ในการคำนวณและบันทึกค่าไว้ โดยแบ่งเป็นส่วนของการเข้ารหัส (G723ENC) และการถอดรหัส (G723DEC)

การออกแบบโครงสร้างของหน่วยความจำสำหรับเก็บค่าของตัวแปร ได้แบ่งหน่วยความจำออกเป็นส่วนๆ (memTab) โดยแต่ละ memTab นั้นจะมีคุณสมบัติของตัวเอง เช่น ขนาด (size), ตำแหน่ง (base), ชนิดของหน่วยความจำ (space), ลักษณะของการใช้งาน (atrrs) เป็นต้น คุณสมบัติที่สำคัญของ memTab มีดังนี้

4.2.2.1 ชนิดของหน่วยความจำ (space) แยกประเภทตามการใช้งานได้เป็น

- IALG_DARAM เป็นหน่วยความจำภายใน มีลักษณะทำงานได้เร็ว แต่มีขนาดเล็ก ส่วนมากจะใช้เพื่อเก็บค่าชั่วคราวเพื่อใช้คำนวณ
- IALG_EXTERNAL เป็นหน่วยความจำภายนอก มีลักษณะทำงานได้ช้ากว่า DARAM แต่มีขนาดใหญ่ จะใช้เก็บค่าที่ต้องการคงไว้ในกรณีที่ Algorithm ไม่ได้ประมวลผล

4.2.2.2 ลักษณะของการใช้งาน (atrrs) แยกประเภทตามการใช้งานได้เป็น

- IALG_SCRATCH เป็นหน่วยความจำชั่วคราว ในกรณีที่มีตัวแปรอื่นมาเขียนค่าใหม่ทับค่าในตำแหน่งเดิม หน่วยความจำชนิดนี้จะไม่มีการเก็บค่าไว้
- IALG_PERSIST เป็นหน่วยความจำไว้ใช้เก็บค่าหรือสถานะของ Algorithm

ในโครงสร้างหลักของโปรแกรม G.723.1 สามารถแบ่งได้ 2 โครงสร้างหลัก คือ G723ENC และ G723DEC ซึ่งเป็นโครงสร้างสำหรับใช้ในการเข้ารหัสและถอดรหัสสัญญาณเสียงตามลำดับ

G723ENC ชนิดของ Instance Object ของ Algorithm ในการเข้ารหัสมีโครงสร้างดังนี้

```
typedef struct G723ENC_NECTEC_Obj {
    IALG_Obj            alg;  /* MUST be first field of all G723ENC objs */
    XDAS_UInt32         framesizeIn0;
    XDAS_UInt32         framesizeOut0;
    XDAS_Bool           useHp;
    XDAS_Bool           useVx;
    XDAS_Bool           rate63;
    CODSTATDEF          *w_codstat;
    VADSTATDEF          *w_vadstat;
    CODCNGDEF           *w_codcng;
    CODSTATDEF          *h_codstat;
    VADSTATDEF          *h_vadstat;
    CODCNGDEF           *h_codcng;
    int                 extra;
} G723ENC_NECTEC_Obj;
```

ส่วนประกอบของ Instance Object		รายละเอียด
IALG_Obj	alg	เป็นส่วนของการติดต่อกับ Algorithm Standard API
XDAS_UInt32	framesizeIn0	เป็นตัวแปรที่บอกขนาดของข้อมูลที่ Algorithm รับเข้ามาคำนวณ
XDAS_UInt32	framesizeOut0	เป็นตัวแปรที่บอกขนาดของข้อมูลที่ Algorithm นำออกเมื่อคำนวณเสร็จ
XDAS_Bool	useHp	เป็นตัวแปรที่บอกว่า Algorithm จะใช้คุณสมบัติ High Pass Filter ในการทำงานหรือไม่
XDAS_Bool	rate63	เป็นตัวแปรที่บอกว่า Algorithm ทำการเข้ารหัสสัญญาณเสียงที่ rate 6.3 Kbit/s ถ้าตัวแปรนี้มีค่าเป็น FALSE จะเข้ารหัสสัญญาณเสียงที่ rate 5.3 Kbit/s
CODSTATDEF	*w_codstat	เป็นตัวแปรที่ใช้เก็บค่าในการทำงาน โดยจะเก็บค่าที่เกี่ยวกับการเข้ารหัสของ Algorithm
VADSTATDEF	*w_vadstat	เป็นตัวแปรที่ใช้ในการทำงานของ Algorithm เข้ารหัสสัญญาณเสียงในส่วนของ Voice Activation Detection
CODCNGDEF	*w_codcng	เป็นตัวแปรที่ใช้ในการทำงานของ Algorithm เข้ารหัสสัญญาณเสียงในส่วนของ Comfort Noise Generation

ตารางที่ 4-2 แสดงรายละเอียดสำหรับองค์ประกอบของ Instance Object ในส่วนการเข้ารหัส

โดยตัวแปรที่นำหน้าด้วย w จะเป็นส่วนที่ใช้ในการคำนวณค่า (Work Buffer) ส่วนตัวแปรที่นำหน้าด้วย h เป็นส่วนที่ต้องการเก็บค่าไว้ระหว่างที่ Algorithm ไม่ได้ทำงาน (History Buffer)

การออกแบบหน่วยความจำมีตัวอย่างดังโค้ดต่อไปดังนี้

```
memTab[0].size = sizeof(G723ENC_NECTEC_Obj);
memTab[0].space = IALG_DARAM0;
memTab[0].attrs = IALG_PERSIST;
/* Request memory for w_codstat array */
memTab[W_CODSTAT].size = (W_CODSTAT_SIZE) * sizeof(CODSTATDEF);
memTab[W_CODSTAT].space = IALG_DARAM0;
memTab[W_CODSTAT].attrs = IALG_SCRATCH;
/* Request memory for w_vadstat array */
memTab[W_VADSTAT].size = (W_VADSTAT_SIZE) * sizeof(VADSTATDEF);
```

```
memTab[W_VADSTAT].space = IALG_DARAM0;
memTab[W_VADSTAT].attrs = IALG_SCRATCH;
/* Request memory for w_codcng array */
memTab[W_CODCNG].size = (W_CODCNG_SIZE) * sizeof(CODCNGDEF);
memTab[W_CODCNG].space = IALG_DARAM0;
memTab[W_CODCNG].attrs = IALG_SCRATCH;
/* Request memory for h_codstat array */
memTab[H_CODSTAT].size = (H_CODSTAT_SIZE) * sizeof(CODSTATDEF);
memTab[H_CODSTAT].space = IALG_EXTERNAL;
memTab[H_CODSTAT].attrs = IALG_PERSIST;
/* Request memory for h_vadstat array */
memTab[H_VADSTAT].size = (H_VADSTAT_SIZE) * sizeof(VADSTATDEF);
memTab[H_VADSTAT].space = IALG_EXTERNAL;
memTab[H_VADSTAT].attrs = IALG_PERSIST;
/* Request memory for h_codcng array */
memTab[H_CODCNG].size = (H_CODCNG_SIZE) * sizeof(CODCNGDEF);
memTab[H_CODCNG].space = IALG_EXTERNAL;
memTab[H_CODCNG].attrs = IALG_PERSIST;
```

ขนาดของโครงสร้างหน่วยความจำที่ใช้ในการเข้ารหัส

memTab[ ] ที่	หน่วยความจำ	ขนาด (Byte)
0	memTab [0]	48
1	memTab [W_CODSTAT]	3412
2	memTab [W_VADSTAT]	64
3	memTab [W_CODCNG]	332
4	memTab [H_CODSTAT]	3412
5	memTab [H_VADSTAT]	64
6	memTab [H_CODCNG]	332

ตารางที่ 4-3 แสดงขนาดของโครงสร้างหน่วยความจำในการเข้ารหัส

G723DEC ชนิดของ Instance Object ของ Algorithm ในการถอดรหัสมีเป็นโครงสร้างดังนี้

```
typedef struct G723DEC_NECTEC_Obj {
IALG_Obj      alg;  /* MUST be first field of all G723DEC objs */
XDAS_UInt32          framesizeIn0;
XDAS_UInt32          framesizeOut0;
XDAS_Bool            usePf;
XDAS_Bool            rate63;
DECSTATDEF           *w_DecStat;
DECCNGDEF            *w_DecCng;
DECSTATDEF           *h_DecStat;
DECCNGDEF            *h_DecCng;
} G723DEC_NECTEC_Obj;
```

ส่วนประกอบของ Instance Object	รายละเอียด
IALG_Obj alg	เป็นส่วนของการติดต่อกับ Algorithm Standard API
XDAS_UInt32 framesizeIn0	เป็นตัวแปรที่บอกขนาดของข้อมูลที่ Algorithm รับเข้ามาคำนวณ
XDAS_UInt32 framesizeOut0	เป็นตัวแปรที่บอกขนาดของข้อมูลที่ Algorithm นำออกเมื่อคำนวณเสร็จ
XDAS_Bool usePf useHp	เป็นตัวแปรที่บอกว่า Algorithm จะใช้คุณสมบัติ Post Filter ในการทำงานหรือไม่
XDAS_Bool rate63	เป็นตัวแปรที่บอกว่า Algorithm ทำการเข้ารหัสสัญญาณเสียงที่ rate 6.3 Kbit/s ถ้าตัวแปรนี้มีค่าเป็น FALSE จะเข้ารหัสสัญญาณเสียงที่ rate 5.3 Kbit/s
DECSTATDEF *w_DecStat	เป็นตัวแปรที่ใช้เก็บค่าในการทำงาน โดยจะเก็บค่าที่เกี่ยวกับการถอดรหัสของ Algorithm
DECCNGDEF *w_DecCng	เป็นตัวแปรที่ใช้เก็บค่าในการทำงาน โดยจะเก็บค่าที่เกี่ยวกับการถอดรหัสของ Comfort Noise Generator

ตารางที่ 4-4 แสดงรายละเอียดสำหรับองค์ประกอบของ Instance Object ในส่วนการถอดรหัส

โดยตัวแปรที่นำหน้าด้วย w จะเป็นส่วนที่ใช้ในการคำนวณค่า (Work Buffer) ส่วนตัวแปรที่นำหน้าด้วย h เป็นส่วนที่ต้องการเก็บค่าไว้ระหว่างที่ Algorithm ไม่ได้ทำงาน (History Buffer)

การออกแบบหน่วยความจำมีรายละเอียดดังนี้

```
memTab[0].size = sizeof(G723DEC_NECTEC_Obj);
memTab[0].space = IALG_DARAM0;
memTab[0].attrs = IALG_PERSIST;
/* Request memory for w_DecStat array */
memTab[W_DECSTAT].size = (W_DECSTAT_SIZE) * sizeof(DECSTATDEF);
memTab[W_DECSTAT].space = IALG_DARAM0;
memTab[W_DECSTAT].attrs = IALG_SCRATCH;
/* Request memory for w_DecCng array */
memTab[W_DECCNG].size = (W_DECCNG_SIZE) * sizeof(DECCNGDEF);
memTab[W_DECCNG].space = IALG_DARAM0;
memTab[W_DECCNG].attrs = IALG_SCRATCH;
/* Request memory for h_DecStat array */
memTab[H_DECSTAT].size = (H_DECSTAT_SIZE) * sizeof(DECSTATDEF);
memTab[H_DECSTAT].space = IALG_EXTERNAL;
memTab[H_DECSTAT].attrs = IALG_PERSIST;
/* Request memory for h_DecCng array */
memTab[H_DECCNG].size = (H_DECCNG_SIZE) * sizeof(DECCNGDEF);
memTab[H_DECCNG].space = IALG_EXTERNAL;
memTab[H_DECCNG].attrs = IALG_PERSIST;
```

ขนาดของโครงสร้างหน่วยความจำที่ใช้ในการถอดรหัส

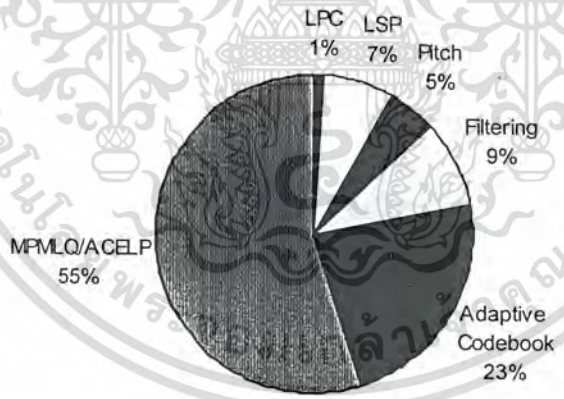
memTab[ ] ที่	หน่วยความจำ	ขนาด (Byte)
0	memTab [0]	32
1	memTab[W_DECSTAT].	784
2	memTab[W_DECCNG].	56
3	memTab[H_DECSTAT].	784
4	memTab[H_DECCNG].	56

ตารางที่ 4-5 แสดงขนาดของโครงสร้างหน่วยความจำในถอดรเข้ารหัส

บทที่ 5
การปรับปรุงโปรแกรมสำหรับมาตรฐาน G.723.1
(G.723.1 Optimization)

เนื่องจากการทำงานของโปรแกรมการเข้ารหัสสัญญาณเสียงแบบ Real-time สำหรับมาตรฐาน ITU-T G.723.1 บน DSP chip TMS320C6713 การทำงานจะเป็นแบบ Real-time เมื่อการประมวลผลเสร็จสิ้นภายใน 30 ms ซึ่งเวลาดังกล่าวนี้สำหรับการประมวลผลเพียง 1 Channel เท่านั้น การที่จะให้อัลกอรึทึมที่มีอยู่สามารถประมวลผลการเข้ารหัสสัญญาณเสียงให้ได้ 4 Channel แบบ Real-time นั้นจำเป็นที่จะต้องมีการปรับปรุงในส่วนของโปรแกรมเพื่อให้สามารถประมวลผลได้ ดังนั้นต้องปรับปรุงโปรแกรมให้สามารถประมวลผลการเข้ารหัสสัญญาณเสียงเสร็จสิ้นภายใน 7.5 ms ต่อ 1 Channel

จากรูปที่ 5-1จะเห็นว่าส่วนของอัลกอรึทึมหรือโปรแกรมที่ต้องปรับปรุงคือส่วน Excitation Encode MPMLQ/ACELP ซึ่งเป็นส่วนที่ใช้ภาระการประมวลผลมากที่สุดถึง 55 %



รูปที่ 5-1 แสดงภาระการคำนวณในแต่ละส่วนของ G.723.1

5.1 การปรับปรุงโปรแกรมสำหรับอัตราส่ง 5.3 kbit/s (ACELP Optimization)

การปรับปรุงโปรแกรมสำหรับอัตราส่ง 5.3 kbit/s หรือ ACELP Optimization นั้นจะใช้วิธีการ Candidate Scheme for Fast ACELP Search ซึ่งจะใช้ Pilot Function ในการทำนาย Candidate Pulse Position

วิธีการ ACELP Codebook Search ตามมาตรฐาน G.723.1 นั้นในส่วนของการหา Pulse Position ที่เหมาะสมทั้ง 4 Pulse นั้นจะคำนวณโดย ในแต่ละ Pulse จะพิจารณาว่า Position ใดจากทั้งหมด 8 Position เมื่อนำมาคำนวณแล้วให้ค่าพลังงานมากที่สุด จะนำค่านั้นส่งออกไป โดยตำแหน่งของ Pulse เป็นดังตารางที่ 5-1

sign	Positions
±1	0, 8, 16, 24, 32, 40, 48, 56
±1	2, 10, 18, 26, 34, 42, 50, 58
±1	4, 12, 20, 28, 36, 44, 52, (60)
±1	6, 14, 22, 30, 38, 46, 54, (62)

ตารางที่ 5-1 แสดง ACELP Excitation Codebook

วิธีการของ Candidate Scheme for Fast ACELP Search ซึ่งจะใช้ Pilot Function ในการทำนายก่อนว่าในแต่ละ Pulse นั้น Position ที่เหมาะสมน่าจะตกอยู่ที่พื้นที่(Region) ใดจาก 8 Region แล้วค่อยนำ Pulse Position ใน Region นั้นๆ (อาจจะมากกว่า 1 Region) มาคำนวณว่า Position ใดให้ค่าพลังงานมากที่สุด ซึ่งจะส่งค่านั้นออกไป

ฟังก์ชันสำหรับการคำนวณพลังงาน $E(k)$ เพื่อใช้เปรียบเทียบว่าควรจะส่ง Pulse Position ตำแหน่งใดออกไปสามารถหาได้จาก Pilot Function ดังสมการ (5.1)

$$E(k) = p_k \sum_{n \in R_k} |r(n)|, \qquad k = 0,1,2,\ldots,K-1 \tag{5.1}$$

เมื่อ k คือลำดับของ Region ที่พิจารณา $r(n)$ คือ target signal K คือจำนวน Region ที่จะพิจารณาทั้งหมด และ $p_k = sign\left(\sum_{n \in R_k} r(n)\right)$

ค่าพลังงาน $E(k)$ สามารถคำนวณให้มีความแม่นยำได้มากขึ้นโดยการปรับปรุง Pilot Function ซึ่งจะเรียกว่า condensing function ตามสมการ (5.2)

$$\widetilde{E}(k)=\begin{cases} E(k)+\beta E\left(k+\widetilde{T}\right) & k=0,\ldots,\left(K-1-\widetilde{T}\right) \\ E(k) & k=\left(K-\widetilde{T}\right),\ldots,\left(\widetilde{T}-1\right) \\ \left(1-\beta p_k P_{k-\widetilde{T}}\right)E(k) & k=\widetilde{T},\ldots,(K-1) \end{cases} \quad (5.2)$$

เมื่อ $\widetilde{T}=\lfloor T/8 \rfloor$,T คือ pitch period β คือ Gain ของ Pitch Period

หลังจากที่ลดการคำนวณในส่วน ACELP Codebook Search ด้วยวิธี Candidate Scheme for Fast ACELP Search พบว่าสามารถลดภาระการคำนวณในโปรแกรมลงได้ประมาณ 9.54 %

5.2 การปรับปรุงโปรแกรมสำหรับอัตราการส่ง 6.3 kbit/s (MP-MLQ Optimization)

การปรับปรุงโปรแกรมสำหรับอัตราการส่ง 6.3 kbit/s หรือ MP-MLQ Optimization นั้นจะใช้วิธีการปรับปรุงโปรแกรมทั้งหมด 3 วิธี คือ Odd-Indexed Autocorrelation Removed , Instruction and Hardware Acceleration for MP-MLQ และ The Access Rate of Candidate for Gain

5.2.1 Odd-Indexed Autocorrelation Removed

วิธีการ Odd-Indexed Autocorrelation Removed สามารถนำมาใช้ได้เนื่องจากค่าอัตสัมพันธ์สำหรับอินเด็กซ์คู่ (Even-Indexed Autocorrelation) เท่านั้นที่จำเป็นสำหรับการประมวลผลของโปรแกรม G.723.1

เนื่องจากเหตุผลที่กล่าวมาข้างต้นทำให้ค่าอัตสัมพันธ์สำหรับอินเด็กซ์คี่ หรือ Odd-Indexed Autocorrelation ไม่ต้องนำมาคำนวณร่วมด้วย ดังนั้นจึงไม่มีความจำเป็นที่จะต้องคำนวณหาค่า Odd-Indexed Autocorrelation

การปรับปรุงส่วนของ coding นั้นสามารถดูได้จากโปรแกรม ดังรูปที่ 5-2

```
/* Compute Imr AutoCorr function */
        for ( i=0; i < SubFrLen; i++ )
    ImrCorr[i] = DotProd(&Imr[i],Imr,SubFrLen-i);
```

```
/* Compute Imr AutoCorr function */
        for ( i=0; i < SubFrLen; i+=2 ) /* odd-indexed autocorrelation was removed */
    ImrCorr[i] = DotProd(&Imr[i],Imr,SubFrLen-i);
```

รูปที่ 5-2 แสดงการปรับปรุงโค้ดสำหรับ Odd-Indexed Autocorrelation was Removed

หลังจากที่ลดการคำนวณในส่วน Odd-Indexed Autocorrelation Removed แล้วพบว่าสามารถลดภาระการคำนวณในโปรแกรมลงได้ประมาณ 1.6 %

5.2.2 Instruction and Hardware Acceleration for MP-MLQ

การปรับปรุงโปรแกรมโดยใช้หลักการของ Instruction and Hardware Acceleration for MP-MLQ จะเป็นการปรับปรุงวิธีการเขียนโค้ดซึ่งจะทำให้จำนวน Loop ในการคำนวณลดลง

จากรูปที่ 5-3 จะเห็นว่าในกรณีที่เลวร้ายที่สุด (Worst Case) จะมีการประมวลผลของโปรแกรมในส่วนนี้ทั้งหมดเท่ากับ $64 \times \sum_{l=0}^{59} \sum_{j=0}^{j=l} 1 = 64 \times \frac{59 \times 60}{2} = 113280$ ครั้ง ซึ่งจำนวน 113280 ครั้งนี้จะมีการคำนวณในส่วนของ Acc0 += OccPos[j] * Imr[l-j] เพียง 21120 ครั้งเท่านั้นที่กระทำการคูณกับ OccPos[j] ที่ไม่เท่ากับศูนย์ นั่นหมายความว่าการประมวลกว่า 90000 ครั้งที่เหลือไม่มีประโยชน์เลย

```
/* Compute error vector */
        for (j=0; j < SubFrLen; j++)
           OccPos[j] = (FLOAT)0.0;
        for (j=0; j < Np; j++)
           OccPos[Temp.Ploc[j]] = Temp.Pamp[j];
        for (l=SubFrLen-1; l >= 0; l--)    {
           Acc0 = (FLOAT)0.0;
           for (j=0; j <= l; j++)
              Acc0 += OccPos[j] * Imr[l-j];
           OccPos[l] = Acc0;
        }
```

รูปที่ 5-3 แสดงการคำนวณ Error Vector ตามมาตรฐาน G.723.1

ในตอนแรกของโปรแกรมส่วนนี้นั้น OccPos[j] จะถูกเซตให้มีค่าเป็นศูนย์หมด หลังจากนั้นจะมีเพียง 5 หรือ 6 array (Temp.Ploc[j]) เท่านั้นซึ่งจะมีค่าของตำแหน่งของ Index อยู่ระหว่าง 0 ถึง 59 และเนื่องจาก Temp.Ploc[j] เป็นค่าของแอมพลิจูดซึ่งจะไม่เป็นศูนย์ นอกจากนี้ยังมี 6 elements สำหรับ subframe คู่ และ 5 elements สำหรับ subframe คี่ ใน 60 entry ของ OccPos ที่ไม่เป็นศูนย์

การแก้ไขโค้ดในส่วนนี้เป็น ดังรูปที่ 5-4 หลังจากที่ลดการคำนวณในส่วน Error Vector ด้วยวิธี Instruction and Hardware Acceleration for MP-MLQ พบว่าสามารถลดภาระการคำนวณในโปรแกรมลงได้ประมาณ 14 %

```
/* Instruction and hardware acceleration for MP-MLQ in G.723.1 */
        for (I=SubFrLen-1; I >= 0; I--) {
                Acc0 = (FLOAT)0.0;
                for (j=0; j < Np; j++) {
                        if ( I-Temp.Ploc[j] >=0 )
                                Acc0 += OccPos[j]*Imr[I-j];
                }
                OccPos[I] = Acc0;
        }
```

รูปที่ 5-4 แสดงการปรับปรุงโค้ดสำหรับการคำนวณ Error Vector ด้วยวิธี Instruction and Hardware Acceleration for MP-MLQ in G.723.1

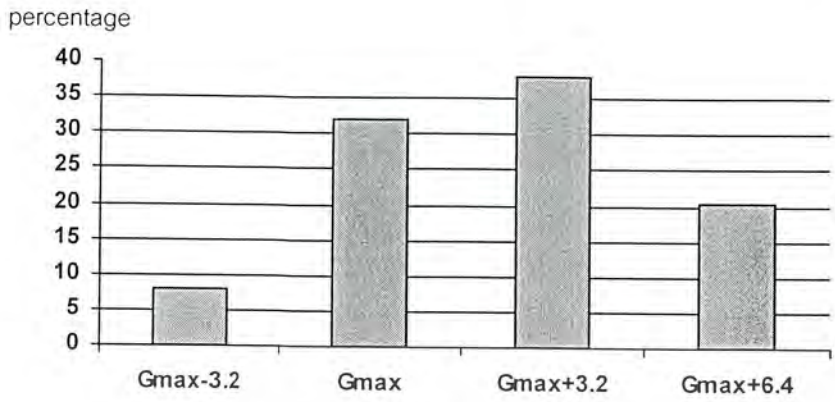
5.2.3 The Access Rate of Candidate for Gain

จากอัลกอริทึมสำหรับมาตรฐาน G.723.1 ได้กล่าวถึงวิธีการ Quantize ค่า Estimated gain ($G_{\max}$) ในส่วนของการหาค่า MP-MLQ Codebook Search ดังนี้

หลังจากประมาณค่า Gain G_{max} จากการ Quantize โดย Logarithmic Quantizer ซึ่งขนาดของ Gain Quantizer จะใช้ด้วยกันสำหรับทั้งสอง bit rate คือ Quantize 24 step และ Step ละ 3.2 เดซิเบล(dB)

นอกจาก Quantized Value $\widetilde{G}_{\max}$ แล้ว ยังมี Gain Value ที่ถูกเลือกภายในช่วง [$\widetilde{G}_{\max} - 3.2$, $\widetilde{G}_{\max}$ + 6.4] สำหรับ Gain Value ทั้งหมดนี้ เครื่องหมายและตำแหน่งของ Pulse จะออพติไมซ์เรียงกันตามลำดับ ซึ่งจะทำซ้ำสำหรับตำแหน่ง Grid คู่ และ Grid คี่ สุดท้ายการรวมกันของพารามิเตอร์ที่ถูกควอนไทซ์แล้วที่ทำให้เกิดความผิดเพี้ยน *err[n]* น้อยสุดจะถูกเลือก ผลรวมที่ดีที่สุดของ Pulse Position และ Pulse gain จะถูกออกส่งไป

แต่ในกระบวนการ The Access Rate of Candidate for Gain นั้นพบว่าค่า $\widetilde{G}_{\max}$ ส่วนมากจะถูกพบหรือถูกเลือกในช่วง [$\widetilde{G}_{\max}$, $\widetilde{G}_{\max}$ + 3.2] ดังในรูปที่ 5-5



รูปที่ 5-5 แสดง The Access Rate of Candidate for Gain

หลังจากที่ลดการคำนวณในส่วน MP-MLQ Codebook Search ด้วยวิธี The Access Rate of Candidate for Gain พบว่าสามารถลดภาระการคำนวณในโปรแกรมลงได้ประมาณ 18 %

บทที่ 6
การทดลอง

การทดลองที่ 6.1 การทดสอบโปรแกรม G.723.1 บนเครื่องคอมพิวเตอร์

จุดประสงค์ เพื่อทดสอบว่าโปรแกรม G.723.1 ที่มีนั้น เมื่อนำมาทดลองทำงานบนเครื่องคอมพิวเตอร์ จะสามารถทำงานได้ถูกต้องเพียงใด

ขั้นตอนการทดลอง

1. นำไฟล์เสียงที่เป็นไฟล์ประเภท pcm แล้ววัดว่ามี Size เท่าไร
2. นำไฟล์เสียงนั้นมาเข้ารหัส(Encode) แล้ววัดว่าหลังจากที่ Encode แล้ว ได้ไฟล์ที่มี Size เท่าไร
3. นำไฟล์เสียงที่ Encode เสร็จจากข้อ 2. มาถอดรหัส(Decode) แล้วดูว่าหลังจากที่ Decode แล้ว ได้ไฟล์ที่มี Size เท่าไร
4. เปรียบเทียบ Size ของไฟล์ระหว่าง ไฟล์ต้นฉบับ , ไฟล์ที่ Encode เสร็จ และ ไฟล์ที่ Decode เสร็จแล้ว
5. ทดลองฟังและเปรียบเทียบเสียง ระหว่าง เสียงต้นฉบับ และเสียงที่ผ่านการประมวลผลตามโปรแกรม G.723.1 (Encode และ Decode กลับ: Voice Codec)แล้ว
6. เปรียบเทียบรูปสัญญาณคลื่นเสียงที่ได้ระหว่าง เสียงต้นฉบับ และเสียงที่ผ่านการประมวลผลตามโปรแกรม G.723.1 โดยใช้โปรแกรม Adobe Audition
7. ทำการทดลองเช่นเดียวกันทั้ง rate 5.3 kbit/s และ 6.3 kbit/s

ผลการทดลอง

ไฟล์สัญญาณเสียงที่นำมาเป็น Input สำหรับทดสอบมีขนาด 138 KB

ไฟล์สัญญาณเสียงที่ผ่านการเข้ารหัสสัญญาณเสียง(Encode และ Decode กลับ) มีขนาดดังนี้

Rate (kbit/s)	Encode (KB)	Decode (KB)
5.3	5.08	138
6.3	6.09	138

ตารางที่ 6-1 แสดงขนาดของไฟล์เสียงเมื่อทำการเข้ารหัสและถอดรหัสบนเครื่องคอมพิวเตอร์

เมื่อทดลองฟังเสียงที่ได้หลังจากการทำ Voice Codec เทียบกับเสียงต้นฉบับ พบว่าหูมนุษย์สามารถรับฟังเสียงรู้เรื่อง ชัดเจน ไม่แตกต่างจากเสียงต้นฉบับมากนัก และ รูปสัญญาณเสียงที่ได้จะมีความแตกต่างกันเล็กน้อยระหว่าง เสียงต้นฉบับ และ เสียงที่ผ่านการทำ Voice Codec

รูปที่ 6-1 แสดงลักษณะสัญญาณเสียงต้นฉบับ

รูปที่ 6-2 แสดงลักษณะสัญญาณเสียงที่ผ่านการเข้ารหัสสัญญาณเสียง(rate 5.3 kbit/s) ตามมาตรฐาน G.723.1

รูปที่ 6-3 แสดงลักษณะสัญญาณเสียงที่ผ่านการเข้ารหัสสัญญาณเสียง(rate 6.3 kbit/s)ตามมาตรฐาน G.723.1

สรุปผลการทดลอง

โปรแกรม G.723.1 ที่มีนั้น เมื่อนำมาทดลองทำ Voice Codec จะสามารถทำงานได้ โดยหูมนุษย์สามารถรับเสียงฟังรู้เรื่อง ชัดเจน ไม่แตกต่างจากเสียงต้นฉบับมากนัก แสดงว่าโปรแกรม G.723.1 ที่มีนั้นมีประสิทธิภาพในการประมวลผลบนเครื่องคอมพิวเตอร์สามารถนำไปประยุกต์ใช้ต่อบนบอร์ด DSP ได้

การทดลองที่ 6.2 การทดสอบโปรแกรม G.723.1 บนบอร์ด DSP

จุดประสงค์ เพื่อทดสอบว่าโปรแกรม G.723.1 ที่มีนั้น เมื่อนำมาทดลองทำงานบนบอร์ด DSP จะสามารถทำงานได้ถูกต้องเพียงใดเมื่อเทียบกับการทำงานบนเครื่องคอมพิวเตอร์

ขั้นตอนการทดลอง

1. แก้ไขโปรแกรมให้สามารถนำมาทดลองใช้บนบอร์ด DSP ได้

2. นำไฟล์เสียงต้นฉบับนั้นมาเข้ารหัส(Encode) แล้วดูว่าหลังจากที่ Encode แล้ว ได้ไฟล์ที่มี Size เท่าไร

3. นำไฟล์เสียงที่ Encode เสร็จจากข้อ 2. มาถอดรหัส(Decode) แล้วดูว่าหลังจากที่ Decode แล้ว ได้ไฟล์ที่มี Size เท่าไร

4. เปรียบเทียบขนาดของไฟล์ระหว่าง ไฟล์ต้นฉบับ , ไฟล์ที่ Encode เสร็จ และ ไฟล์ที่ Decode เสร็จแล้ว

5. ทดลองฟังและเปรียบเทียบเสียงระหว่าง ไฟล์เสียงที่ได้จากการประมวลผลของโปรแกรม G.723.1 (Encode และ Decode กลับ) บนเครื่องคอมพิวเตอร์ กับ ไฟล์เสียงที่ได้จากการประมวลผลของโปรแกรม G.723.1 บน DSP board

6. ทำการทดลองเช่นเดียวกันทั้ง rate 5.3 kbit/s และ 6.3 kbit/s

ผลการทดลอง

ไฟล์สัญญาณเสียงที่นำมาเป็น Input สำหรับทดสอบมีขนาด 138 KB

ไฟล์สัญญาณเสียงที่ผ่านการประมวลผลของโปรแกรม G.723.1 (Encode และ Decode กลับ) มีขนาดดังนี้

Rate (kbit/s)	Encode (KB)	Decode (KB)
5.3	5.08	138
6.3	6.09	138

ตารางที่ 6-2 แสดงขนาดของไฟล์เสียงเมื่อทำการเข้ารหัสและถอดรหัส

เมื่อทดลองฟังเสียงที่ได้หลังจากการประมวลผลของโปรแกรม G.723.1 บนบอร์ด DSP เปรียบเทียบกับเสียงที่ได้หลังจากการประมวลผลของโปรแกรม G.723.1 บนเครื่องคอมพิวเตอร์พบว่าเสียงที่ได้ยินไม่แตกต่างกันมากนัก

สรุปผลการทดลอง

โปรแกรม G.723.1 ที่มีนั้น เมื่อนำมาทดลองใช้งานให้ทำงานบน DSP board สามารถทำงานได้อย่างมีประสิทธิภาพ และสามารถนำไปประยุกต์ใช้งานต่อได้

การทดลองที่ 6.3 การออพติไมซ์คอมไพเลอร์

จุดประสงค์ นำโปรแกรม G.723.1 ที่สามารถทำงานบนบอร์ด DSP ได้มาปรับปรุงการทำงานโดยใช้การออพติไมซ์ (Optimize) แบบต่างๆ ที่มีอยู่ใน Compiler ของโปรแกรม Code Composer ที่ใช้กับ DSP Board

ขั้นตอนการทดลอง

1. เลือกประเภทการ Optimize แบบต่าง ๆ ที่มีอยู่ใน Compiler ของโปรแกรม Code Composer ที่ใช้กับ DSP Board

2. วัดค่าต่างๆในการทำงานของโปรแกรมในแต่ละรอบ ได้แก่เวลาที่ใช้มากสุด (Max) เวลาเฉลี่ย (Average) และ จำนวน Frame เสียงที่ใช้คำนวณ แล้วเปรียบเทียบค่าที่ได้ว่าการ Optimize แบบใดให้ผลดีที่สุด

3. ทำการทดลองเช่นเดียวกัน ทั้งการตั้งค่าหน่วยความจำ (Set Memory) แบบ SDRAM และ IRAM (rate 5.3 kbit/s และ 6.3 kbit/s

ผลการทดลอง

Speed level	Optimize level	Rate 5.3 kbit/s				Rate 6.3 kbit/s			
		Max (ms)	Average	Frames	MIPs	Max (ms)	Average	Frames	MIPs
Speed more	Function(-o2)	7.85	6.12	422	1.7662	10.78	6.87	422	2.4255
Speed more	File(-o3)	8.10	6.10	422	1.8225	10.62	6.85	422	2.3895
Speed most	Function(-o2)	7.76	5.72	422	1.7460	10.20	6.81	422	2.2950
Speed most	File(-o3)	8.05	6.23	422	1.8112	10.45	6.82	422	2.3512

ตารางที่ 6-3 แสดงผลที่ได้จากการวัดค่าต่างๆ โดย set memory เป็นแบบ SDRAM

Speed level	Optimize level	Rate 5.3 kbit/s				Rate 6.3 kbit/s			
		Max (ms)	Average	Frames	MIPs	Max (ms)	Average	Frames	MIPs
Speed more	Function(-o2)	7.26	5.87	422	1.6335	9.69	6.93	422	2.1802
Speed more	File(-o3)	7.24	5.80	422	1.6290	9.75	6.84	422	2.1937
Speed most	Function(-o2)	7.06	5.80	422	1.5885	9.53	6.62	422	2.1442
Speed most	File(-o3)	7.28	5.86	422	1.6380	9.59	6.72	422	2.1577

ตารางที่ 6-4 แสดงผลที่ได้จากการวัดค่าต่างๆ โดย set memory เป็นแบบ IRAM

สรุปผลการทดลอง

การ Optimize Compiler ของโปรแกรม Code Composer ที่ใช้กับบอร์ด DSP การเลือก Speed Level แบบ Speed most และ เลือก Optimize level แบบ Function(-02) จะทำให้ได้ค่าเวลาที่ใช้ในการประมวลผลของแต่ละเฟรม คือ ค่า Max และ ค่า Average ดีที่สุด(น้อยที่สุด)

หมายเหตุ

Speed more critical เป็นการ Optimize โดยจะเน้นไปที่ความเร็วของโปรแกรมที่ทำงานบน DSP Board

Speed most critical เป็นการ Optimize โดยจะให้มีความเร็วสูงสุด ของโปรแกรมที่ทำงานบน DSP Board

Function level (-02) เป็นการ Optimize โดยคิด scope ภายใน function เท่านั้น เช่น Local Variable , การ Pass Parameter

File level (-03) เป็นการ Optimize โดยคิด Scope ทั้ง File เช่น Global Variable

การทดลองที่ 6.4 เรียกใช้งานโปรแกรมหลังจากทำ Reentrant

จุดประสงค์ เพื่อทดสอบความถูกต้องการทำงานของโปรแกรม G.723.1 หลังจากที่ทำเป็น Reentrant เพื่อรองรับการทำงานแบบ Multichannel

ขั้นตอนการทดลอง

1. เตรียมไฟล์สัญญาณเสียง 2 ไฟล์
2. ทดสอบการประมวลผลสัญญาณเสียงของโปรแกรมทีละ 1 สัญญาณเสียง
3. แก้ไขโปรแกรมให้สามารถอ่านเฟรมของสัญญาณเสียงทีละเฟรม สลับกันทั้ง 2 ไฟล์ (เฟรมที่ 1 ของไฟล์ที่ 1 , เฟรมที่ 1 ของไฟล์ที่ 2 , เฟรมที่ 2 ของไฟล์ที่ 1 , เฟรมที่ 2 ของไฟล์ที่ 2 , เฟรมที่ *n* ของไฟล์ที่ 1 , เฟรมที่ *n* ของไฟล์ที่ 2)
4. แก้ไขโปรแกรมต่อจากข้อ 3. ให้โปรแกรมมีคุณสมบัติ Reentrant
5. ทดสอบโปรแกรมในข้อ 4. โดยให้ประมวลผลสัญญาณเสียงพร้อมกันทั้ง 2 สัญญาณเสียง
6. นำไฟล์สัญญาณเสียงที่ได้ในข้อ 2. และ 5. มาเปรียบเทียบกัน (ไฟล์ที่ 1 ในข้อ 2. เปรียบเทียบกับไฟล์ที่ 1 ในข้อ 5. และ ไฟล์ที่ 2 ในข้อ 2. เปรียบเทียบกับไฟล์ที่ 2 ในข้อ 5.) โดยใช้คำสั่ง file compare (fc /b)

ผลการทดลอง

เมื่อทดลองฟังเสียงเพื่อเปรียบเทียบสัญญาณเสียงที่ได้จากการทำงานของโปรแกรมระหว่างโปรแกรม G.723.1 ก่อนที่จะทำ Reentrant กับ โปรแกรม G.723.1 หลังจากที่ทำเป็น Reentrant แล้ว พบว่า หูมนุษย์ไม่สามารถแยกความแตกต่างของเสียงที่ได้ยินได้

เมื่อทดสอบด้วยการนำไฟล์เสียงที่ได้มาเปรียบเทียบกันโดยใช้คำสั่ง file compare ผลที่ได้คือ ไม่มีความแตกต่างของข้อมูลระหว่างไฟล์สัญญาณเสียงที่นำมาเปรียบเทียบกัน

สรุปผลการทดลอง

โปรแกรม G.723.1 ที่ทำเป็น Reentrant แล้วสามารถทำงานได้ถูกต้อง แสดงว่าโปรแกรม G.723.1 ที่ปรับปรุงแล้วสามารถรองรับการทำงานแบบ Multichannel

การทดลองที่ 6.5 เรียกใช้งานโปรแกรมหลังจากทำ Algorithm Standard

จุดประสงค์ เพื่อทดสอบว่าการทำงานของโปรแกรม G.723.1(หลังจากที่ทำ Reentrant แล้ว) เมื่อทำเป็น Algorithm Standard จะให้ผลเหมือนกัน หรือ ต่างกันอย่างไร เมื่อเปรียบเทียบกับการเรียกใช้งานโปรแกรม G.723.1 ก่อนที่จะทำเป็น Algorithm Standard

ขั้นตอนการทดลอง

1. ทดลองฟังเสียงที่ได้จากการทำงานของโปรแกรม G.723.1 ก่อนที่จะทำเป็น Algorithm Standard
2. ทดลองฟังเสียงที่ได้จากการทำงานของโปรแกรมG.723.1 หลังจากที่ทำเป็น Algorithm Standard
3. นำไฟล์เสียงจากข้อ 1. และ 2. มาเปรียบเทียบกัน โดยใช้คำสั่ง file compare (fc /b)

ผลการทดลอง

เมื่อทดลองฟังเสียงเพื่อเปรียบเทียบการทำงานของโปรแกรม ระหว่างโปรแกรม G.723.1 ก่อนที่จะทำเป็น Algorithm Standard กับ โปรแกรม G.723.1 หลังจากที่ทำเป็น Algorithm Standard แล้ว พบว่า หูมนุษย์ไม่สามารถแยกความแตกต่างของเสียงที่ได้ยินได้

เมื่อทดสอบด้วยการนำไฟล์เสียงที่ได้ทั้ง 2 ไฟล์มาเปรียบเทียบกันโดยใช้คำสั่ง file compare ผลที่ได้คือ ไม่มีความแตกต่างของข้อมูลระหว่างไฟล์ทั้ง 2

สรุปผลการทดลอง

โปรแกรม G.723.1 ที่ทำเป็น Algorithm Standard แล้วสามารถทำงานได้ถูกต้อง แสดงว่าโปรแกรม G.723.1 ที่ปรับปรุงแล้วสามารถรองรับการทำงานร่วมกับอัลกอริทึมอื่นๆได้

การทดลองที่ 6.6 การออพติไมซ์โปรแกรม G.723.1

จุดประสงค์ เพื่อปรับปรุงหรือออพติไมซ์ (Optimize)โปรแกรม G.723.1 ให้สามารถประมวลผลแบบ Real-time ได้สำหรับ 4 ช่องสัญญาณ (7.5 ms / channel)

ขั้นตอนการทดลอง

1. ใช้ไฟล์สัญญาณเสียง 3 ไฟล์ โดยมีลักษณะ คือ สัญญาณเสียงพูดปกติ สัญญาณเสียงพูดที่มีเสียงสูง แหลม และสัญญาณเสียงเพลง

2. เซต Memory ให้เป็น SDRAM

3. วัดเวลาที่ใช้ในการทำงานของโปรแกรม G.723.1 ในการประมวลผลแต่ไฟล์สัญญาณเสียง ก่อนที่จะทำการออพติไมซ์โปรแกรม ทั้ง rate 5.3 kbit/s และ 6.3 kbit/s

3. Optimize โปรแกรม G.723.1 สำหรับ rate 5.3 kbit/s ในส่วน ACELP Codebook Search ด้วยวิธี Candidate Scheme for Fast ACELP Search และวัดเวลาที่ใช้ในการทำงานของโปรแกรมในการประมวลผลแต่ละไฟล์สัญญาณเสียงหลังจากที่ Optimize โปรแกรมแล้ว

4. Optimize โปรแกรม G.723.1 สำหรับ rate 6.3 kbit/s ในส่วน MP-MLQ codebook Search ด้วยวิธี Odd-Indexed Autocorrelation Removed , Instruction and Hardware Acceleration for MP-MLQ และ The Access Rate of Candidate for Gain หลังจากนั้นวัดเวลาที่ใช้ในการทำงานของโปรแกรมในการประมวลผลแต่ละไฟล์สัญญาณเสียงหลังจากที่ Optimize โปรแกรมในแต่ละวิธีและคำนวณค่า *RMS* หลังจากที่ Optimize เสร็จทุกวิธีในแต่ละ rate โดย $RMS = \sqrt{\frac{\sum (x_1 - x_2)^2}{N}}$ โดย x_1, x_2 คือค่าแอมพลิจูดในแต่ละ sample ของสัญญาณเสียง input และสัญญาณเสียงที่ได้จากการประมวลผล ตามลำดับ ส่วน *N* คือ จำนวน sample ทั้งหมด

5. วิเคราะห์สัญญาณเสียงที่ได้ในทาง Frequency Domain และ Time Domain ระหว่างสัญญาณเสียงที่ได้ก่อนการ Optimize โปรแกรมกับหลัง Optimize โปรแกรม ทั้ง rate 5.3 kbit/s และ rate 6.3 kbit/s

ผลการทดลอง

เมื่อทดลองฟังเสียงโดยการเปรียบเทียบระหว่างสัญญาณเสียงที่ได้ยินก่อนการ Optimize โปรแกรมกับเสียงที่ได้ยินหลังจากการประมวลผลบนโปรแกรมที่ Optimize แล้ว พบว่าหูมนุษย์สามารถแยกความแตกต่างของเสียงที่ได้ยินเพียงเล็กน้อยเท่านั้น ยกเว้นเสียงเพลงจะได้ยินถึงความแตกต่างอย่างชัดเจน โดยค่าต่างๆที่วัดได้เป็นดังตารางที่ 6-5 , 6-6 , 6-7 , 6-8 , 6-9 และ 6-10

Optimization	Max(ms)	Averages	Frame	MIPs	RMS
ก่อนทำการ optimize	**7.76**	7.52	422	1.7460	-
หลังทำการ optimize ด้วยวิธี Candidate Scheme for Fast ACELP Search	**7.02**	6.46	422	1.5795	3065.73

ตารางที่ 6-5 แสดงผลที่ได้จากการวัดค่าต่างๆ ก่อนและหลังจากออพติไมซ์โปรแกรม G.723.1 ที่อัตราการส่ง 5.3 kbit/s สำหรับสัญญาณเสียงพูดปกติ

Optimization	Max(ms)	Averages	Frame	MIPs	RMS
ก่อนทำการ optimize	**7.80**	7.72	326	1.7550	-
หลังทำการ optimize ด้วยวิธี Candidate Scheme for Fast ACELP Search	**7.07**	6.54	326	1.5907	3112.23

ตารางที่ 6-6 แสดงผลที่ได้จากการวัดค่าต่างๆ ก่อนและหลังจากออพติไมซ์โปรแกรม G.723.1 ที่อัตราการส่ง 5.3 kbit/s สำหรับสัญญาณเสียงพูดที่มีลักษณะเสียงสูงแหลม

Optimization	Max(ms)	Averages	Frame	MIPs	RMS
ก่อนทำการ optimize	**7.87**	7.70	548	1.7705	-
หลังทำการ optimize ด้วยวิธี Candidate Scheme for Fast ACELP Search	**7.15**	6.46	548	1.6087	3726.19

ตารางที่ 6-7 แสดงผลที่ได้จากการวัดค่าต่างๆ ก่อนและหลังจากออพติไมซ์โปรแกรม G.723.1 ที่อัตราการส่ง 5.3 kbit/s สำหรับสัญญาณเสียงเพลง

Optimization		Max(ms)	Averages	Frame	MIPs	RMS
ก่อนทำการ optimize		**10.20**	9.38	422	2.2950	-
หลัง optimize ด้วยวิธีต่างๆ	Odd-Indexed Autocorrelation - Removed	10.03	9.25	422	2.2566	
	Instruction and Hardware - Acceleration for MP-MLQ	8.67	8.17	422	1.9508	
	The Access Rate of Candidate - for Gain	**7.10**	6.84	422	1.5975	3734.33

ตารางที่ 6-8 แสดงผลที่ได้จากการวัดค่าต่างๆ ก่อนและหลังจากออพติไมซ์โปรแกรม G.723.1 ที่อัตราการส่ง 6.3 kbit/s สำหรับสัญญาณเสียงพูดปกติ

Optimization	Max(ms)	Averages	Frame	MIPs	RMS
ก่อนทำการ optimize	**10.22**	9.40	326	2.2995	-
หลังทำการ optimize ด้วย 3 วิธี	**7.13**	6.87	326	1.6042	3759.26

ตารางที่ 6-9 แสดงผลที่ได้จากการวัดค่าต่างๆ ก่อนและหลังจากออพติไมซ์โปรแกรม G.723.1 ที่อัตราการส่ง 6.3 kbit/s สำหรับสัญญาณเสียงที่มีลักษณะเสียงสูงแหลม

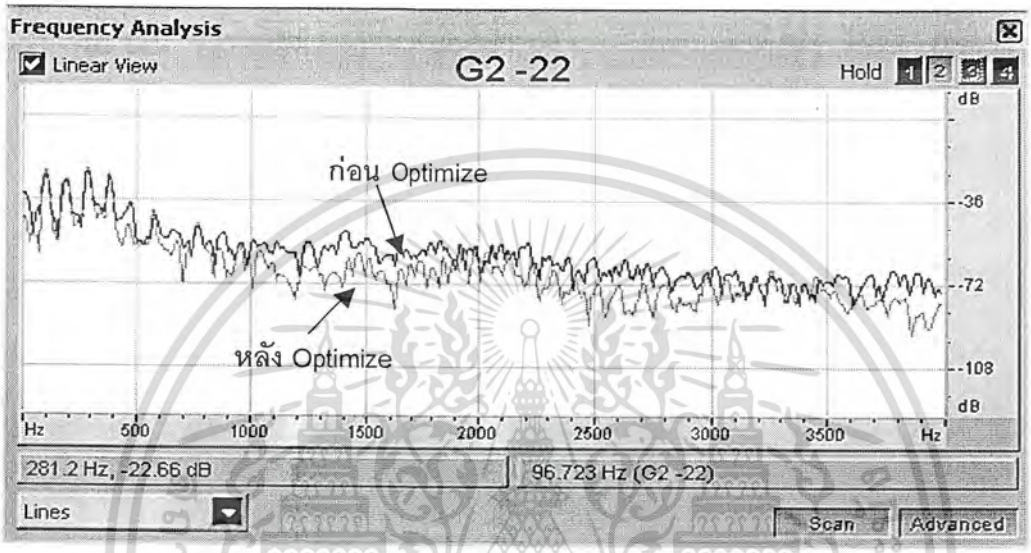
Optimization	Max(ms)	Averages	Frame	MIPs	RMS
ก่อนทำการ optimize	**10.31**	9.75	548	2.3197	-
หลังทำการ optimize ด้วย 3 วิธี	**7.19**	7.62	548	1.6178	3982.36

ตารางที่ 6-10 แสดงผลที่ได้จากการวัดค่าต่างๆ ก่อนและหลังจากออพติไมซ์โปรแกรม G.723.1 ที่อัตราการส่ง 6.3 kbit/s สำหรับสัญญาณเสียงเพลง

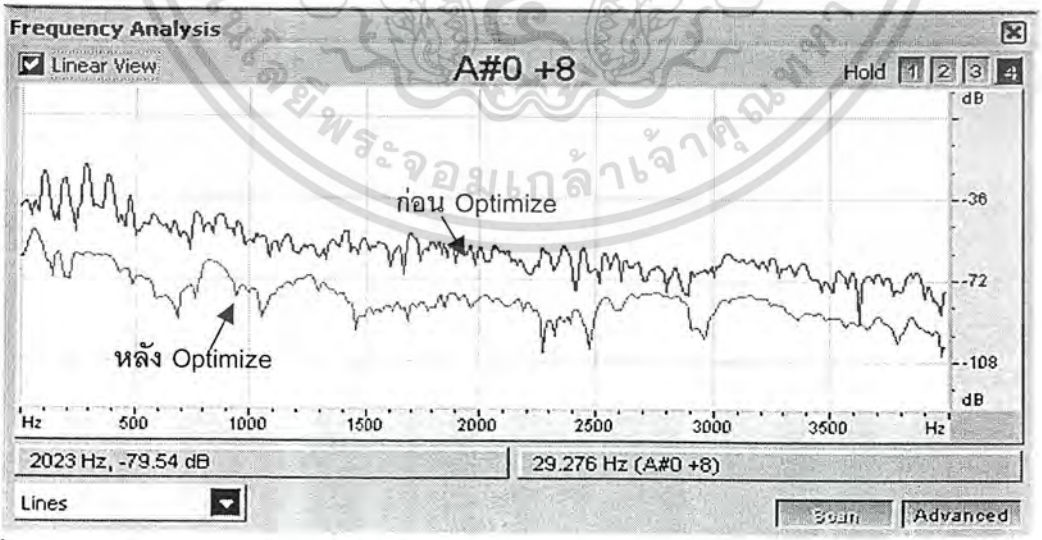
ที่อัตราการส่ง 5.3 kbit/s และ 6.3 kbit/s เมื่อวิเคราะห์สัญญาณเสียงในทาง Frequency Domain ระหว่างผลที่ได้ก่อนการ Optimize และหลัง Optimize โปรแกรม พบว่า ที่ความถี่เดียวกันค่าของแอมพลิจูด

จะต่างกัน โดยค่าแอมพลิจูดของสัญญาณเสียงที่ผ่านการประมวลผลของโปรแกรมที่ Optimize จะต่ำกว่าแอมพลิจูดของสัญญาณเสียงที่ผ่านการประมวลผลของโปรแกรมที่ยังไม่ได้ Optimize ดังรูปที่ 6-4 , 6-5

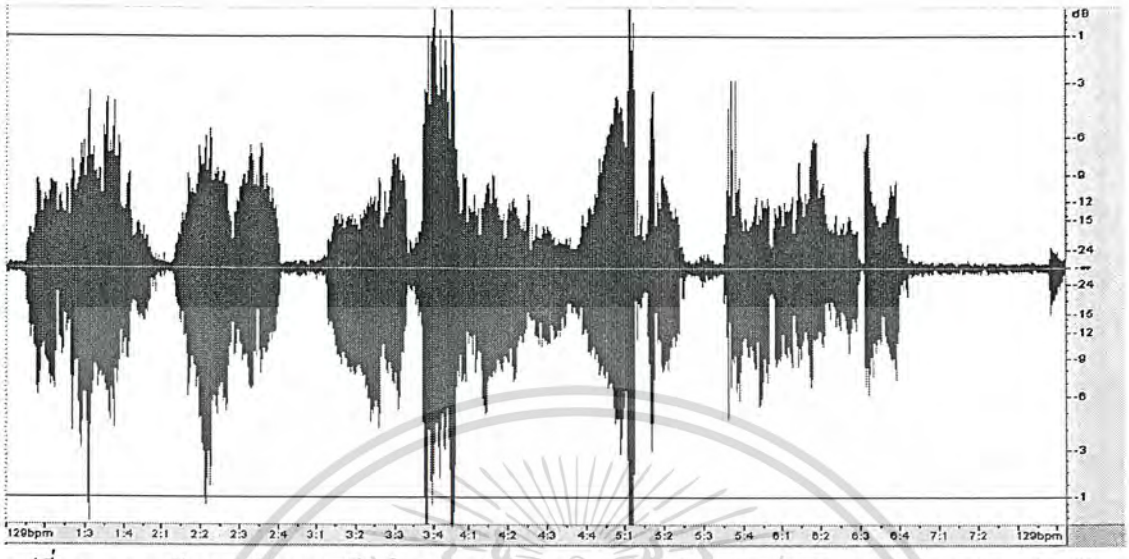
ที่อัตราการส่ง 5.3 kbit/s และ 6.3 kbit/s เมื่อวิเคราะห์สัญญาณเสียงสำหรับสัญญาณเสียงพูดปกติในทาง Time Domain พบว่า ลักษณะรูปกราฟจะต่างกัน ดังรูปที่ 6-6 , 6-7, 6-8 และ 6-9



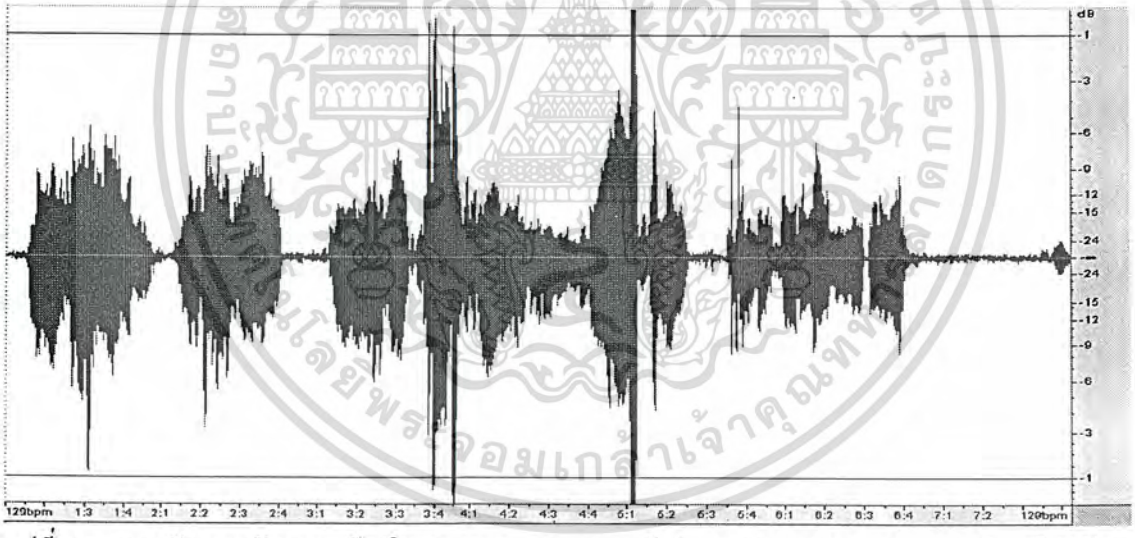
รูปที่ 6-4 แสดงลักษณะของสัญญาณเสียงในทาง Frequency Domain ระหว่างผลที่ได้ก่อนการ Optimize และหลัง Optimize โปรแกรม (rate 5.3 kbit/s) สำหรับสัญญาณเสียงพูดปกติ



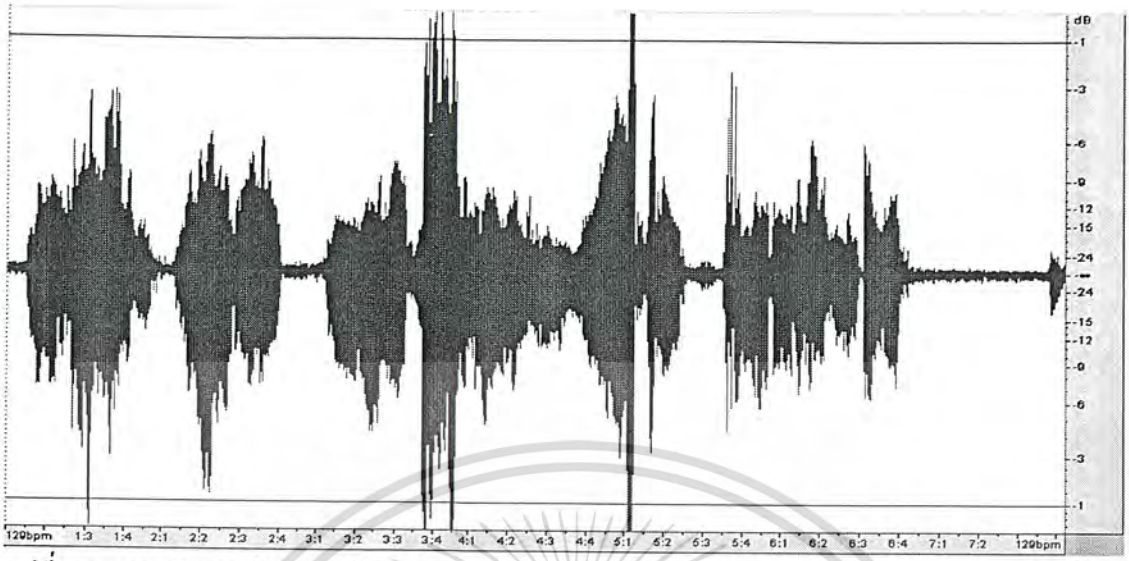
รูปที่ 6-5 แสดงลักษณะของสัญญาณเสียงในทาง Frequency Domain ระหว่างผลที่ได้ก่อนการ Optimize และหลัง Optimize โปรแกรม (rate 6.3 kbit/s) สำหรับสัญญาณเสียงพูดปกติ



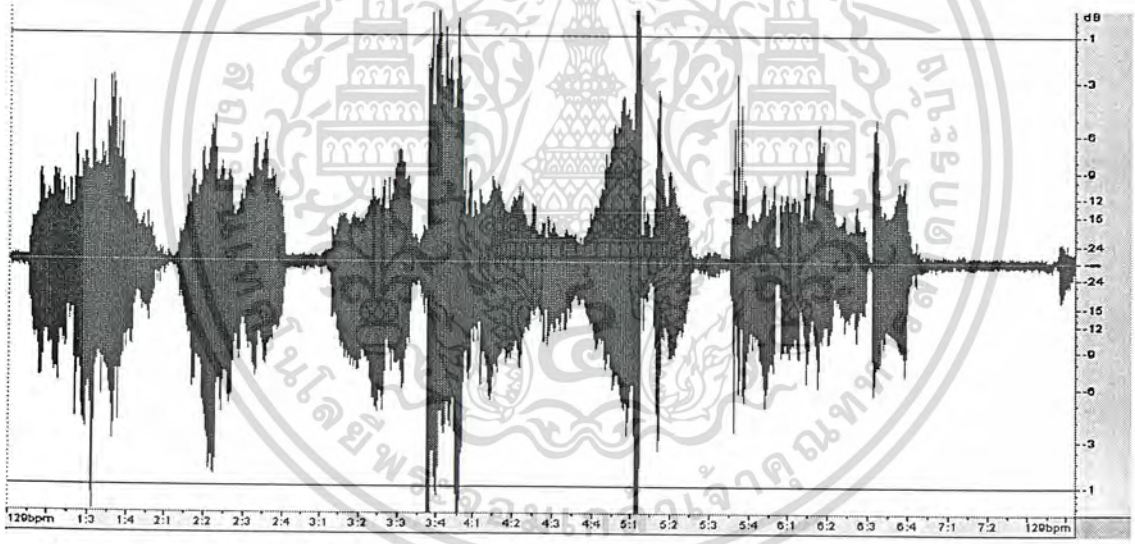
รูปที่ 6-6 แสดงลักษณะสัญญาณเสียงในทาง Time Domain ก่อนทำการ Optimize (rate 5.3 kbit/s) สำหรับสัญญาณเสียงพูดปกติ



รูปที่ 6-7 แสดงลักษณะสัญญาณเสียงในทาง Time Domain หลังทำการ Optimize (rate 5.3 kbit/s) สำหรับสัญญาณเสียงพูดปกติ



รูปที่ 6-8 แสดงลักษณะสัญญาณเสียงในทาง Time Domain ก่อนทำการ Optimize (rate 6.3 kbit/s) สำหรับสัญญาณเสียงพูดปกติ



รูปที่ 6-9 แสดงลักษณะสัญญาณเสียงในทาง Time Domain หลังทำการ Optimize (rate 6.3 kbit/s)สำหรับสัญญาณเสียงพูดปกติ

สรุปผลการทดลอง

หลังจากการ Optimize โปรแกรม G.723.1 ทั้งใน rate 5.3 kbit/s และ 6.3 kbit/s แล้วจะสามารถประมวลผลการทำงานให้เป็นแบบ Real-time ได้สำหรับ 4 ช่องสัญญาณ คือ น้อยกว่า 7.5 ms ต่อ 1 channel โดยการเปรียบเทียบระหว่างสัญญาณเสียงที่ได้ยินก่อนการ Optimize โปรแกรมกับเสียงที่ได้ยินหลังจากการ Optimize โปรแกรมแล้วพบว่าหูมนุษย์สามารถแยกความแตกต่างของเสียงที่ได้ยินเพียงเล็กน้อยเท่านั้น

เสียงทั้ง 3 ลักษณะที่นำมาทดสอบ เสียงพูดปกติจะให้คุณภาพเสียงดีที่สุด รองลงมาคือเสียงพูดที่มีลักษณะสูงแหลม และแย่ที่สุดคือเสียงเพลง แสดงว่าอัลกอริทึม G.723.1 ไม่เหมาะสำหรับการเข้ารหัสสัญญาณเสียงเพลง

การทดลองที่ 6.7 คุณภาพของเสียงเมื่อถูกประมวลผลหลายครั้ง

จุดประสงค์ เพื่อทดสอบว่าการทำงานของโปรแกรม G.723.1 หลังจากที่ O ptimize โปรแกรมแล้ว นำมาประมวลผลสัญญาณเสียงโดยรับ Input จากสัญญาณเสียงที่ได้จากการประมวลก่อนหน้านี้ เมื่อทำการทดสอบหลายๆครั้งจะให้ผลเป็นอย่างไร

ขั้นตอนการทดลอง

1. ประมวลผลสัญญาณเสียงพูดปกติโดยโปรแกรม G.723.1 หลังจากที่โปรแกรมทำการ Optimize แล้ว

2. นำไฟล์เสียงที่ได้จากข้อ 1. มาประมวลผลแบบเดิม

3. ทำการทดลองเช่นเดิมเป็นจำนวน 10 รอบ

4. เปรียบเทียบคุณภาพของเสียง โดยการฟังเสียงระหว่างเสียงที่ได้จากการถูกประมวลครั้งแรกกับเสียงที่ได้จากกการประมวลผลครั้งที่ 10

5. เปรียบเทียบขนาดของไฟล์และค่าผิดเพี้ยนของสัญญาณ (Root Mean Square Error : RMS)ระหว่างสัญญาณเสียง input ที่เข้ามาครั้งแรกกับสัญญาณเสียงที่ได้จากการประมวลผลของโปรแกรม

ในแต่ละรอบ $RMS = \sqrt{\frac{\sum (x_1 - x_2)^2}{N}}$ โดย x_1, x_2 คือค่าแอมพลิจูดในแต่ละ sample ของสัญญาณเสียง input และสัญญาณเสียงที่ได้จากการประมวลผล ตามลำดับ ส่วน N คือ จำนวน sample ทั้งหมด

6. วิเคราะห์สัญญาณเสียงที่ได้ในทาง Frequency Domain ระหว่างลักษณะของสัญญาณเสียง input, สัญญาณเสียงที่ได้จากการประมวลผลครั้งที่ 1 ของโปรแกรมที่ Optimize แล้ว และสัญญาณเสียงที่ได้จากการประมวลผลครั้งที่ 10 ของโปรแกรมที่ Optimize แล้ว

7. ทำการทดลองเช่นเดียวกันทั้งอัตราการส่งที่ 5.3 kbit/s และ 6.3 kbit/s

ผลการทดลอง

สำหรับทั้งอัตราการส่งที่ 5.3 kbit/s และ 6.3 kbit/s เมื่อทดลองฟังเสียงที่ได้จากการถูกประมวลครั้งแรกกับเสียงที่ได้จากกการประมวลผลครั้งที่ 10 พบว่าหูมนุษย์สามารถแยกความแตกต่างของเสียงที่ได้ยินได้ แต่เสียงที่ได้ยินนั้นสามารถฟังได้ชัดเจน รู้เรื่อง โดยที่อัตราการส่ง 5.3 kbit/s ความดังของเสียงจะเบาลง และที่ 6.3 kbit/s ความดังของเสียงจะเท่าเดิม

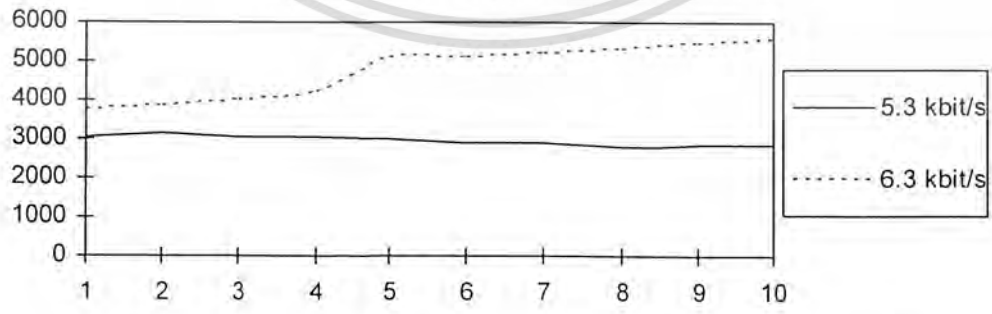
ขนาดของไฟล์เสียงที่ได้มีขนาดเท่าเดิมไม่ว่าจะผ่านการประมวลผลโปรแกรมกี่ครั้ง ดังตารางที่ 6-7

สำหรับค่า RMS ที่วัดได้ ที่อัตราการส่ง 5.3 kbit/s ค่า RMS จะเพิ่มขึ้นในครั้งที่ 2 และหลังจากครั้งที่ 2 เป็นต้นไปค่า RMS จะลดลงเรื่อยๆ ส่วนที่อัตราการส่ง 6.3 kbit/s ค่า RMS จะเพิ่มขึ้นเรื่อยๆ ดังตารางที่ 6-7

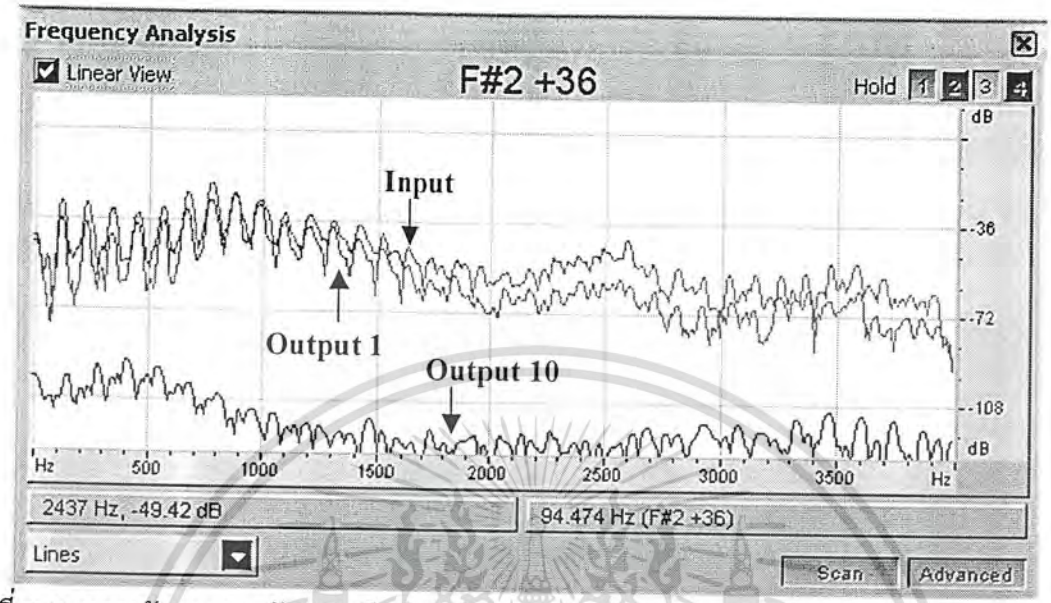
เมื่อวิเคราะห์สัญญาณเสียงที่ได้ในทาง Frequency Domain ระหว่างลักษณะของสัญญาณเสียง input, สัญญาณเสียงที่ได้จากการประมวลผลครั้งที่ 1 ของโปรแกรมที่ Optimize แล้ว และสัญญาณเสียงที่ได้จากการประมวลผลครั้งที่ 10 ของโปรแกรมที่ Optimize แล้ว พบว่าที่ความถี่เดียวกันค่าแอมพลิจูดของสัญญาณเสียงจะลดลงเรื่อยๆ ดังรูปที่ 6-11 ,6-12

ครั้งที่	ขนาดของไฟล์ (kB)		RMS	
	Rate 5.3 kbit/s	Rate 6.3 kbit/s	Rate 5.3 kbit/s	Rate 6.3 kbit/s
1	116	116	3065.73	3734.33
2	116	116	3135.24	3874.44
3	116	116	3036.98	3987.72
4	116	116	3030.60	4216.10
5	116	116	3998.59	5106.69
6	116	116	2902.13	5098.21
7	116	116	2885.10	5191.50
8	116	116	2850.83	5305.86
9	116	116	2859.02	5428.09
10	116	116	2833.15	5540.46

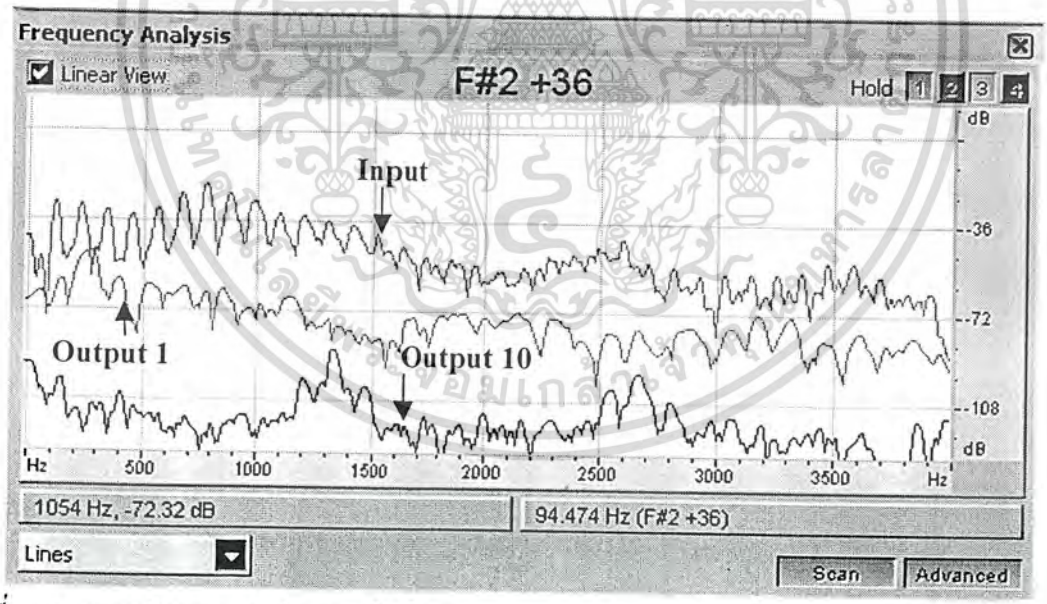
ตารางที่ 6-11 แสดงขนาดของไฟล์และค่า RMS ของการประมวลผลสัญญาณเสียงหลายๆครั้ง



รูปที่ 6-10 แสดงกราฟค่า RMS ที่วัดได้จากการประมวลผลสัญญาณเสียงหลายๆครั้ง



รูปที่ 6-11 แสดงลักษณะของสัญญาณเสียงในทาง Frequency Domain ระหว่างสัญญาณเสียง input , ผลที่ได้จากการประมวลผล ครั้งที่ 1 และครั้งที่ 10 หลัง Optimize โปรแกรม (rate 5.3 kbit/s)



รูปที่ 6-12 แสดงลักษณะของสัญญาณเสียงในทาง Frequency Domain ระหว่างสัญญาณเสียง input , ผลที่ได้จากการประมวลผล ครั้งที่ 1 และครั้งที่ 10 หลัง Optimize โปรแกรม (rate 6.3 kbit/s)

สรุปผลการทดลอง

1. ขนาดของไฟล์สัญญาณเสียงที่ได้ยังคงเท่าเดิมไม่ว่าจะประมวลผลกี่ครั้ง
2. หูมนุษย์สามารถแยกความแตกต่างของเสียงที่ได้ยินได้ แต่เสียงที่ได้ยินยังฟังรู้เรื่อง ชัดเจน
3. ที่อัตราการส่ง 5.3 kbit/s ค่าความผิดเพี้ยนของสัญญาณจะลดลงเรื่อยๆ ในขณะที่เสียงเบาลงเรื่อยๆ เช่นกัน แสดงว่าคุณภาพของเสียงแย่ลง และค่า RMS บางครั้งไม่สามารถนำมาวัดคุณภาพของเสียงได้ เนื่องจากขั้นตอนของการหาค่า RMS มาจากค่าเฉลี่ย
4. ที่อัตราการส่ง 6.3 kbit/s ค่าความผิดเพี้ยนของสัญญาณจะเพิ่มขึ้นเรื่อยๆ แสดงว่าคุณภาพของเสียงแย่ลง

การทดลองที่ 6.8 เรียกใช้งานโปรแกรมจาก Library

จุดประสงค์ เพื่อทดสอบว่าการทำงานของโปรแกรม G.723.1 หลังจากที่ทำเป็น Library แล้ว จะให้ผลเหมือนกัน หรือ ต่างกันอย่างไร เมื่อเปรียบเทียบกับการเรียกใช้งานโปรแกรม G.723.1 ก่อนที่จะทำเป็น Library

ขั้นตอนการทดลอง

1. ทดลองฟังเสียงที่ได้จากการทำงานของโปรแกรม G.723.1 ก่อนที่จะทำเป็น Library
2. ทดลองฟังเสียงที่ได้จากการทำงานของโปรแกรม G.723.1 หลังจากที่ทำเป็น Library แล้ว
3. นำไฟล์เสียงจากข้อ 1. และ 2. มาเปรียบเทียบกันโดยใช้คำสั่ง file compare (fc /b)

ผลการทดลอง

เมื่อทดลองฟังเสียงเพื่อเปรียบเทียบการทำงานของโปรแกรม ระหว่างโปรแกรม G.723.1 ก่อนที่จะทำเป็น Library กับ โปรแกรม G.723.1 หลังจากที่ทำเป็น Library แล้ว พบว่า หูมนุษย์ไม่สามารถแยกความแตกต่างของเสียงที่ได้ยินได้

เมื่อทดสอบด้วยการนำไฟล์เสียงที่ได้ทั้ง 2 ไฟล์มาเปรียบเทียบกันโดยใช้คำสั่ง file compare ผลที่ได้คือ ไม่มีความแตกต่างของข้อมูลระหว่างไฟล์ทั้ง 2

สรุปผลการทดลอง

โปรแกรม G.723.1 ที่ทำเป็น Library แล้วสามารถทำงานได้ถูกต้อง

บทที่ 7
บทสรุปและวิจารณ์

7.1 สรุปผลที่ได้จากการทำโครงงาน

จากการทำโครงงานที่ผ่านมาทำให้มีความเข้าใจหลายๆด้าน ซึ่งเกี่ยวกับหลักการการเข้ารหัสสัญญาณเสียงสำหรับมาตรฐาน G.723.1 ไม่ว่าจะเป็นขั้นตอนต่างๆในการเข้ารหัส(Encode) หรือ ถอดรหัส(Decode) การใช้สมการแทนหลักการในขั้นตอนต่างๆ จุดประสงค์ในการทำแต่ละขั้นตอน การใช้ DSP/BIOS TMS320C6713 เป็นต้น

เนื่องจากหน่วยประมวลผล DSP/BIOS สามารถประมวลผลได้รวดเร็วและมีประสิทธิภาพมากกว่าการประมวลบนเครื่องคอมพิวเตอร์ ดังนั้นโครงงานนี้จึงได้ประยุกต์ให้มาตรฐาน G.723.1 สามารถประมวลผลบน DSP/BIOS ได้

สำหรับมาตรฐาน G.723.1 เดิมที่มีอยู่นั้นสามารถรองรับประมวลผลเพียง 1 ช่องสัญญาณเท่านั้น และยังไม่สามารถนำไปประยุกต์ใช้ร่วมกับมาตรฐานอื่นๆได้ โครงงานนี้จึงพัฒนาต่อให้มาตรฐานสามารถรองรับการประมวลผลแบบหลายช่องสัญญาณ (อย่างน้อย 4 ช่องสัญญาณ) ซึ่งจะต้องปรับปรุงมาตรฐานในเรื่องของการจัดสรรทรัพยากรให้เพียงพอ และการประมวลผลแบบหลายช่องสัญญาณนี้จะต้องเป็นการประมวลผลแบบทันเวลาหรือเรียลไทม์ด้วย โดยประยุกต์จากวิธีการต่างๆที่มีผู้ศึกษาไว้

เมื่อมาตรฐาน G.723.1 สามารถประมวลผลแบบเรียลไทม์หลายช่องสัญญาณได้แล้ว โครงงานนี้ได้พัฒนาต่อให้มาตรฐาน G.723.1 สามารถทำงานร่วมกับมาตรฐานอื่นๆที่มีความซับซ้อนได้อย่างมีประสิทธิภาพ และสะดวกต่อการเรียกใช้งานโดยการทำเป็นไลบรารี (Library)

การพัฒนาที่กล่าวมาทั้งหมดนี้ เนื่องจากการปรับปรุงมาตรฐานให้สามารถประมวลผลแบบเรียลไทม์หลายช่องสัญญาณ จะทำให้สัญญาณเสียงที่ผ่านการเข้ารหัสและถอดรหัสแล้วมีเสียงสัญญาณผิดแปลกไปจากเดิมอยู่บ้าง แต่หูของมนุษย์ก็เพียงแค่ฟังออกว่ามีความแตกต่างกันอยู่เล็กน้อยเท่านั้น และเสียงที่ได้ยินมีความชัดเจน ฟังรู้เรื่อง

7.2 ปัญหาที่พบในการทำโครงงานและวิธีการแก้ปัญหา

7.2.1 ใช้ระยะเวลานานในการศึกษาอัลกอริทึมและโปรแกรม G.723.1

7.2.2 ขาดความรู้ความเข้าใจในการเขียนโปรแกรมให้เป็น Algorithm Standard ทำให้ต้องใช้ระยะเวลาในการศึกษาและแก้ไขโปรแกรมนาน

7.2.3 บอร์ด DSP ไม่เสถียรทำให้บางครั้งบอร์ดไม่สามารถทำงานได้ วิธีแก้ไขคือ ต้องรีเซตบอร์ดใหม่

7.2.4 บางครั้งการประมวลผลบนบอร์ด DSP ใช้เวลานานผิดปกติ แสดงว่ามีปัญหาในการเขียนโปรแกรม วิธีแก้คือ ให้หยุดการประมวลผล และแก้ไขโปรแกมให้ถูกต้องแล้วจึงนำโปรแกรมที่แก้ไขแล้วไปประมวลผลบนบอร์ด DSP อีกครั้ง

7.2.5 การใช้เครื่องมือของบอร์ด DSP ในการแสดง Statistic View หากต้องการให้แสดงผลแบบ Real-time จะต้องตั้งค่าการใช้หน่วยความจำของ DSP/BIOS เป็น IRAM

7.2.6 การจองพื้นที่หน่วยความจำของโครงสร้างของ Object นั้นมีขนาดผิดพลาด เพราะว่าโครงสร้างหลักนั้นบรรจุโครงสร้างย่อยเป็นชนิดของ Structure Pointer ไว้ภายในด้วย การแก้ไข คือ ทำการจองพื้นที่หน่วยความจำของโครงสร้างย่อยนั้นเพิ่ม แล้วให้โครงสร้างย่อยชนิด Structure Pointer ที่อยู่ในโครงสร้างหลักชี้ไปที่หน่วยความจำที่จองเพิ่มขึ้นมา

7.2.7 ในการทำโปรเจคให้เป็น Library นั้น ได้ขนาดของ Library ไม่ใกล้เคียงกับขนาดของแฟ้ม Obj ทั้งหมดที่ใช้งาน ทำให้ไม่สามารถใช้ Library นั้นได้ การแก้ไขคือ เพิ่มรายการของแฟ้ม Obj ที่ต้องการเพิ่มเข้าไปในกระบวนการทำ Library ไว้ในแฟ้ม G723DEC.cmd หรือ G723ENC.cmd ด้วย

7.3 แนวทางการพัฒนา

เนื่องจากมาตรฐาน G.723.1 เป็นมาตรฐานเกี่ยวกับการเข้ารหัสสัญญาณเสียง ดังนั้นแนวทางการพัฒนาส่วนใหญ่จึงเป็นไปตามความต้องการ ความพึงพอใจของผู้ใช้

7.3.1 พัฒนามาตรฐาน G.723.1 ให้สามารถประมวลผลแล้วได้สัญญาณเสียงในระดับที่ผู้ใช้พึงพอใจมากที่สุด

7.3.2 พัฒนามาตรฐาน G.723.1 ให้สามารถรองรับการทำงานแบบเรียลไทม์หลายช่องสัญญาณมากกว่า 4 ช่องสัญญาณ ได้

7.3.3 นำมาตรฐาน G.723.1 นี้มาพัฒนาร่วมกับมาตรฐานอื่นๆเพื่อให้รองรับการทำงานหลายด้านมากขึ้น

7.3.4 นำมาตรฐาน G.723.1 ไปพัฒนาบนหน่วยประมวลอื่น หากหน่วยประมวลนั้นสามารถประมวลได้ดีกว่าหรือใกล้เคียงกับ DSP/BIOS TMS320C6713

บรรณานุกรม

[1] Blonstein, Steve. *The TMS320 DSP Algorithm Standard*, SPRA581C, 2002.

[2] Chen, Fu-Kun and Yang, Jar-Ferr. 2001."Candidate Scheme for MP-MLQ Searching in G.723.1." 368-371. Third IEEE Signal Processing Workshop on Signal Processing Advances in Wireless Communications. Taiwan: IEEE.

[3] Fu, Xiangdong and Zhaohong, Zhang. *A Multichannel/Algorithm Implementation on the TMS320C6000 DSP*, SPRA556B, 2000.

[4] Chen,F.K, Yang,J.F and Lan,Y.L. *Candidate Scheme For Fast ACELP Search*, 2002.

[5] ITU-T Recommendation G.723.1, *Dual Rate Speech Coder for Multimedia Communnications Transmitting at 5.3 and 6.3 kbit/s*, March 1996.

[6] Lee, Sang - Min, Park, Sangil and Jang, Youngbeom. "Cost-Effective Implementation of ITU-T G.723.1 On a DSP Chip." 31-34.

[7] Negrescu, Cristian. "Optimization Algorithm for the MP-MLQ Excitation in G723.1 Encoder." 1003-1006 : IEEE.

[8] Olausson, Mikael and Liu, Dake. "Instruction And Hardware Acceleration For MP-MLQ In G.723.1." 235-239.

[9] Thomas, J and Dillon, Jr. *G.723.1 Dual-Rate Speech Coder: Multichannel TMS320C62x Implementation*, SPRA552B, 2000.

[10] Texas Instrument, *TMS320 DSP Algorithm Standard API Reference*, SPRU360C, 2002.

[11] Texas Instrument, *TMS320 DSP Algorithm Standard Rules and Guidelines*, SPRU352E, 2002.

[12] Texas Instrument, *Code Composer Studio Getting Started Guide*, SPRU509C, 2001.

[13] Texas Instrument, *TMS320 DSP/BIOS User's Guide*, SPRU423B, 2002.

[14] Texas Instrument, *TMS320C6000 Technical Brief*, SPRU197D, 1999.

[15] Torud, Stig. *A Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers*, SPRA579C, 2002.