การเข้ารหัสสัญญาณเสียง สำหรับมาตรฐาน ITU – T G. 729

Implementation of ITU-T G.729 Standard Voice Codec

โดย

นายกรัณย์ยศ ปิ่นจันทร์

นายภอบโชค สุวรรณโณ

นายกิตตินิตย์ ยัสสระ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

การเข้ารหัสสัญญาณเสียง สำหรับมาตรฐาน ITU – T G. 729

Implementation of ITU-T G.729 Standard Voice Codec

โดย

นายกรัณย์ยศ ปิ่นจันทร์ 45010010

นายกอบโชค สุวรรณโณ 45010033

นายกิตตินิตย์ ยัสสระ 45010052

อาจารย์ที่ปรึกษา

ผศ. อัครพล ตรีรัตน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การเข้ารหัสสัญญาณเสียง สำหรับมาตรฐาน ITU-T G.729**

Implementation of ITU-T G.729 Standard Voice Codec

ผู้จัดทำ นายกรัณย์ยศ ปิ่นจันทร์ 45010010

นายกอบโชค สุวรรณโณ 45010033

นายกิตตินิตย์ ยัสสระ 45010052

อาจารย์ที่ปรึกษา

(ผศ. อัครพล ตรีรัตน์)

การเข้ารหัสสัญญาณเสียง สำหรับมาตรฐาน ITU-T G.729

Implementation of ITU-T G.729 Standard Voice Codec

โดย นายกรัณย์ยศ ปิ่นจันทร์ 45010010

นายกอบโชค สุวรรณโณ 45010033

นายกิตตินิตย์ ยัสสระ 45010052

อาจารย์ที่ปรึกษา ผศ. อัครพล ตรีรัตน์

บทคัดย่อ

โครงงานนี้ได้ทำการพัฒนาการเข้ารหัสสัญญาณเสียง สำหรับมาตรฐาน ITU-T G. 729 (Implementation of ITU-T G.729 Standard Voice Codec) โดย ทำการพัฒนาไลบราลี สำหรับเข้ารหัสและถอดรหัสสัญญาณเสียงตามมาตรฐาน ITU-T G 729 บน DSP Chip ตระกูล TMS320C6000 ซึ่งงานวิจัยนี้สามารถนำไปประยุกต์ ใช้ได้กับโครงการVoice Over IP สำหรับ การส่งสัญญาณ เสียงผ่านเครือข่ายอินเตอร์เน็ต

Abstract

This thesis presents a development of an encoding speech signal as specified in ITU-T G.729 standard. It is useful for Voice Over IP project since its capacity can improve the performance of communication systems like the internet .However , we try to search for algorithm which can decrease the complexity of the system while enhance this standard. We have implemented the proposed technique on DSP chip TMS320C6713

สารบัญ

สารบัญรูป

รูป **หน้า**

สารบัญรูป

รูป หน้า

สารบัญรูป

รูป **หน้า**

สารบัญตาราง

ตาราง **หน้า**

บทที่ 1
บทนำ

1.1 หลักการและเหตุผล

เทคโนโลยีด้านการเข้ารหัสสัญญาณเสียง (Speech Coding / Speech Compression) ใช้สำหรับการแทนสัญญาณเสียงให้อยู่ในรูปของสัญญาณดิจิตอลที่มีขนาดเล็กทำให้มีการใช้งานแบนด์วิธ (Bandwidth) ในการส่งและจัดเก็บข้อมูลได้อย่างมีประสิทธิภาพ การเข้ารหัสสัญญาณเสียงนี้เกี่ยวข้องกับการสุ่มสัญญาณและควอนไทต์ (quantize) สัญญาณ เนื่องจากการสุ่มสัญญาณถูกจำกัดด้วยทฤษฎีของไนกวิสท์ (Nyguist) อยู่แล้วไม่สามารถลดค่าความถี่ในการสุ่มลงได้ ดังนั้นการค้นคว้าด้านบีบอัดสัญญาณเสียง (Speech Compression) จึงมุ่งเน้นไปที่การพัฒนาการแทนสัญญาณโดยใช้ขนาดที่เล็กลงแต่ยังคงให้คุณภาพเสียงที่ดี เราแบ่งการแทนสัญญาณหรือการควอนไทต์สัญญาณออกเป็น 2 ประเภทคือการควอนไทต์โดยตรง ซึ่งเป็นการแทนสัญญาณที่ถูกสุ่มโดยตรงและการควอนไทต์แบบพารามิเตอร์ (parameter) ซึ่งเป็นการแทนแบบจำลองที่ใช้สร้างเสียงนั้นขึ้นมา

ด้วยเหตุผลข้างต้น ผู้จัดทำโครงงานจึงมีแนวคิดที่จะศึกษาวิธีการเข้ารหัส / ถอดรหัส สัญญาณเสียงตามมาตรฐานของ ITU-T G.729 (1996) ซึ่งเป็นวิธีการเข้ารหัสแบบ Conjugate-Structure Algebraic-Code-Excited-Linear-Prediction (CS-ACELP) ซึ่งอยู่ในประเภทของการควอนไทต์โดยใช้พารามิเตอร์และอิมพลีเมนต์ (implement) บน ดี เอส พี บอร์ด (DSP Board) ของเท็กซัส อินสตรูเมนต์ (Texas Instrument) รุ่น TMS320C6713

1.2 วัตถุประสงค์

1. เพื่อศึกษากระบวนการประมวลผลสัญญาณดิจิตอล
2. เพื่อศึกษาวิธีการเข้ารหัส / ถอดรหัส สัญญาณเสียงตามมาตรฐานของ ITU-T G.729
3. เพื่อศึกษาวิธีการโปรแกรมระบบฝังตัวบน ดี เอส พี เพลตฟอร์ม (DSP Platform)
4. เพื่อศึกษาวิธีการโปรแกรมตามมาตรฐานของการพัฒนาอัลกอริทึมของ TI

1.3 ขอบเขตของการพัฒนา

สำหรับโครงงานที่จัดทำนั้นจะใช้ภาษา C ในการพัฒนาอัลกอริทึมตามมาตรฐานของ เท็กซัส อินสตรูเมนต์ โดยใช้เครื่องมือในการพัฒนาคือ Code Composer Studio Version 2.0 โดย สามารถเข้ารหัส / ถอดรหัสสัญญาณเสียงได้ตามมาตรฐานของ ITU-T G.729 บน ดี เอส พี บอร์ด TMS320C6713

1.4 ขั้นตอนการดำเนินงาน

เป็นการศึกษาข้อมูลที่จำเป็นในการทำโครงงานนี้ ประกอบด้วย

1. ศึกษาพื้นฐานของการประมวลผลสัญญาณดิจิตอล
2. ศึกษามาตรฐานของการเข้ารหัส / ถอดรหัส สัญญาณเสียงตามมาตรฐานของITU-T G.729
3. ศึกษามาตรฐานการเขียนโปรแกรมบน ดี เอส พี เพลตฟอร์ม ของ เท็กซัส อินสตรูเมนต์ และวิธีการงาน Code composer studio V 2.0

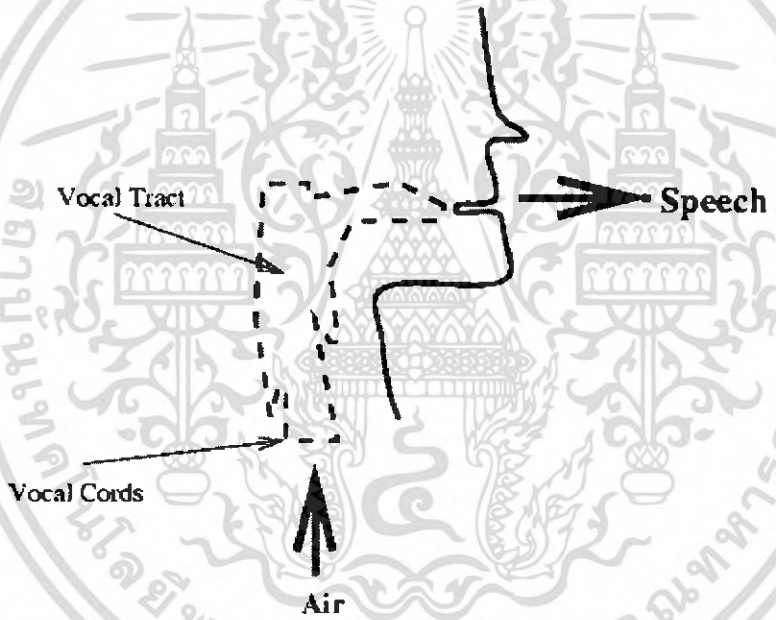
บทที่ 2
ทฤษฎีและหลักการ

ทฤษฎีการสังเคราะห์เสียง

2.1 บทนำ

การเข้ารหัสสัญญาณเสียงโดยวิธีควอนไทต์โดยใช้พารามิเตอร์นั้น เราพยายามหาแบบจำลองทางคณิตศาสตร์เพื่อแทนระบบของการเกิดเสียงของมนุษย์ พารามิเตอร์ที่นำมาใช้ในการสังเคราะห์นั้น แบ่งออกเป็นสองส่วนหลัก ๆ คือ สัญญาณกระตุ้น (Excitation signal) และวงจรกรอง (filter) ซึ่งเมื่อเปรียบเทียบกับการกำเนิดเสียงของมนุษย์นั้น สัญญาณกระตุ้น คือ ลมในช่องปอด และ วงจรกรอง คือ อวัยวะกำเนิดเสียง และเมื่อนำสัญญาณกระตุ้นผ่านวงจรกรอง ก็จะเกิดเป็นเสียงได้ในที่สุด

2.2 การสังเคราะห์เสียงเชิงกายภาพ (Physical Model)

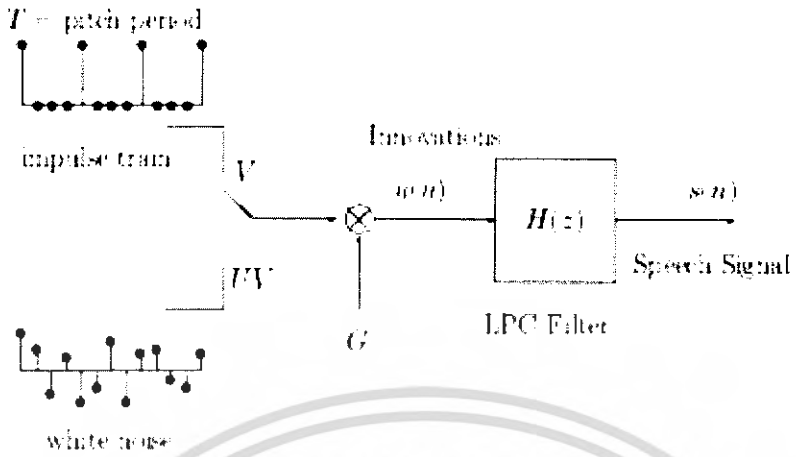


รูปที่ 2-1 แสดงกลไกการเกิดเสียงของมนุษย์

จากรูปที่ 2.1 แสดงวิธีการกำเนิดเสียงของมนุษย์ ขณะที่พูดนั้นลมจะถูกผลักออกมาจากปอดผ่านหลอดลมและออกมาทางปากกลายมาเป็นเสียง โดยลักษณะของอวัยวะออกเสียงมีผลต่อเสียงที่กำเนิด ดังนี้

1. เสียงจะมีการสั่นเป็นจังหวะ (Voiced sound) โดยคาบของการสั่นเรียกว่าพิตท์ซึ่งผู้หญิงและเด็กจะมีการสั่นของเส้นเสียงที่เร็วกว่าของผู้ชายทำให้มีเสียงที่สูงกว่าเสียงของผู้ชาย
2. เส้นเสียงจะไม่สั่นแต่จะเปิดให้ลมผ่านออกไปอย่างเดียว (Unvoiced sound)
3. รูปร่างที่เปลี่ยนไปของหลอดลมทำให้เสียงเปลี่ยนไป
4. รูปร่างของหลอดลมจะเปลี่ยนไปอย่างช้า ๆ โดยทั่วไปอยู่ในระดับ 10 – 100 mS
5. ปริมาณที่ถูกผลักออกมาจากปอดนั้นเป็นสิ่งที่กำหนดความดังของเสียง

2.3 การสังเคราะห์เสียงเชิงคณิตศาสตร์ (Mathematical Model)



รูปที่ 2-2 แสดงการสังเคราะห์สัญญาณเสียง

จากรูปที่ 2.2 แสดงแบบจำลองทางคณิตศาสตร์สำหรับการสังเคราะห์เสียง โดยอธิบายได้ดังนี้

1. สัญญาณเสียงที่สังเคราะห์ขึ้นได้จากการนำสัญญาณเสียงที่เป็นคาบ (Voiced) และ สัญญาณเสียงที่ไม่เป็นคาบ (Unvoiced) มาทำการมอดูเลท (modulate) และผ่านวงจรกรอง
2. เปรียบเทียบความสัมพันธ์ระหว่างรูปแบบทางกายภาพกับรูปแบบทางคณิตศาสตร์ได้ดังต่อไปนี้

หลอดลม (Vocal Tract)	⟺	***H(z)*** (LPC Filter)
ลม (Air)	⟺	***u(n)*** (Innovations)
การสั่นของเส้นเสียง (Vocal Cord Vibration)	⟺	***V*** (Voiced)
คาบการสั่นของเส้นเสียง (Vocal Cord Vibration Period)	⟺	***T*** (Pitch period)
เสียงที่ไม่มีการสั่นของเส้นเสียง (Fricatives and Plosives)	⟺	***UV*** (Unvoiced)
ปริมาณของลมที่ออกมาจากปอด (Air Volume)	⟺	***G*** (Gain)

ตารางที่ 2-1 ความสัมพันธ์ระหว่างรูปแบบทางกายภาพกับรูปแบบทางคณิตศาสตร์

2.3.1 การหาสัมประสิทธิ์ของวงจรกรอง LP (Linear Prediction)

แนวคิดของการหาสัมประสิทธิ์ของวงจรกรอง LP คือการประมาณให้ตัวอย่างของเสียงที่เวลาใด ๆ s(n) สามารถประมาณค่าได้จากการทำความเป็นไปได้เชิงเส้น (linear combination) ของตัวอย่างสัญญาณเสียงอินพุทก่อนหน้าจำนวน p ตัวอย่าง เช่น

$$s(n) \approx a_1 s(n-1) + a_2 s(n-2) + \ldots + a_p s(n-p) \tag{2.1}$$

ซึ่งสัมประสิทธิ์ของ $a_1,a_2,\ldots,a_p$ นั้นถือว่ามีค่าคงที่ในช่วงเฟรม ที่ทำการวิเคราะห์ เราสามารถแปลงสมการที่ (2.1) จากการประมาณให้เป็นเท่ากับได้โดยเพิ่มสัญญาณกระตุ้นเข้าไป ดังต่อไปนี้

$$s(n) = \sum_{i=1}^{p} a_i s(n-i) + Gu(n) \tag{2.2}$$

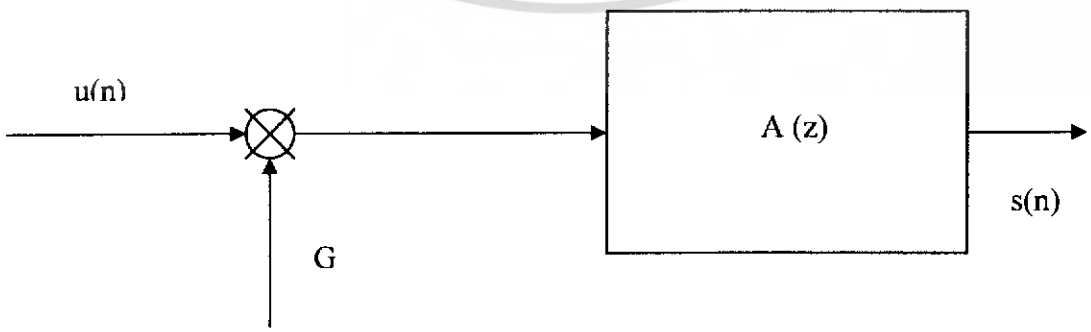
โดยที่ u(n) เป็นสัญญาณกระตุ้นที่ถูกทำให้เป็นหนึ่งหน่วย (normalize) แล้ว เมื่อ G เป็นอัตราขยายของสัญญาณกระตุ้น ซึ่งหากพิจารณาจากสมการดังกล่าวบน โดเมนของ z จะได้ความสัมพันธ์ดังนี้

$$S(z) = \sum_{i=1}^{p} a_i z^{-i} S(z) + GU(z) \tag{2.3}$$

ทำให้เราได้ฟังก์ชันถ่ายโอน (Transfer function) เป็น

$$H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1 - \sum_{i=1}^{p} a_i z^{-i}} = \frac{1}{A(z)} \tag{2.4}$$

ความหมายของสมการนี้สามารถแสดงได้ด้วยรูปที่ 2.1 ซึ่งแหล่งกำเนิดของสัญญาณกระตุ้นปล่อยสัญญาณกระตุ้น u (n) ออกมา สัญญาณจะถูกขยายด้วยค่า G และกลายเป็นสัญญาณอินพุทที่ป้อนให้กับวงจรกรองแบบ ออลโพล (all pole) เพื่อสร้างสัญญาณเสียง s(n) ออกมา จากความรู้ในปัจจุบันเราพบว่าสัญญาณกระตุ้นของเสียงพูดนั้น เป็นแบบพัลซ์เทรน (pulsed train) หรือสัญญาณเสียงที่เป็นคาบ (voiced sound) หรือเป็นสัญญาณแบบสุ่ม (unvoiced sound) ส่วนวงจรกรองนั้นเป็นการทำงานที่ผสมผสานกันของอวัยวะต่างที่ใช้ออกเสียง (ช่องคอ ทางเดินอากาศ ช่องปาก และลิ้น)



รูปที่ 2-3 แสดง linear prediction mode of speech

การประมาณแบบความเป็นไปได้เชิงเส้นของสัญญาณเสียงในอดีต โดยนิยามค่าของสัญญาณที่ประมาณ ดังนี้

$$\tilde{s}(n) = \sum_{i=1}^{p} a_i s(n-i) \qquad (2.5)$$

ค่าความผิดพลาดในการประมาณ e (n) แสดงได้โดย

$$e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k) \qquad (2.6)$$

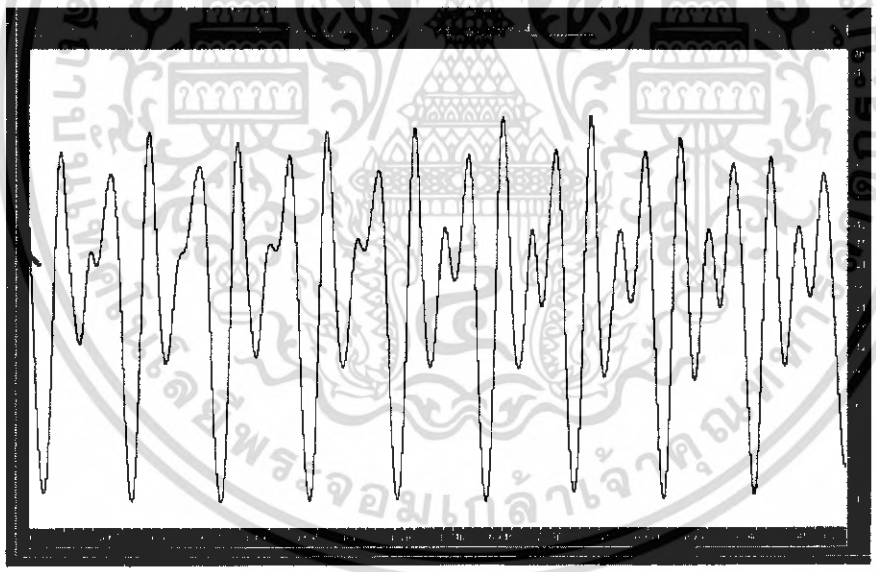
โดยที่มีค่าความผิดพลาดของฟังก์ชันถ่ายโอน (Error transfer function)

$$A(z) = \frac{E(z)}{S(z)} = 1 - \sum_{k=1}^{p} a_k z^{-k} \qquad (2.7)$$

ดังนั้นเมื่อให้ s(n) เป็นสัญญาณจริงที่ถูกสร้างขึ้นมาจากระบบเชิงเส้นตามสมการที่ (2.2) จะพบว่า e(n) จะมีค่าเท่ากับ สัญญาณกระตุ้นที่ถูกขยายแล้ว Gu(n)

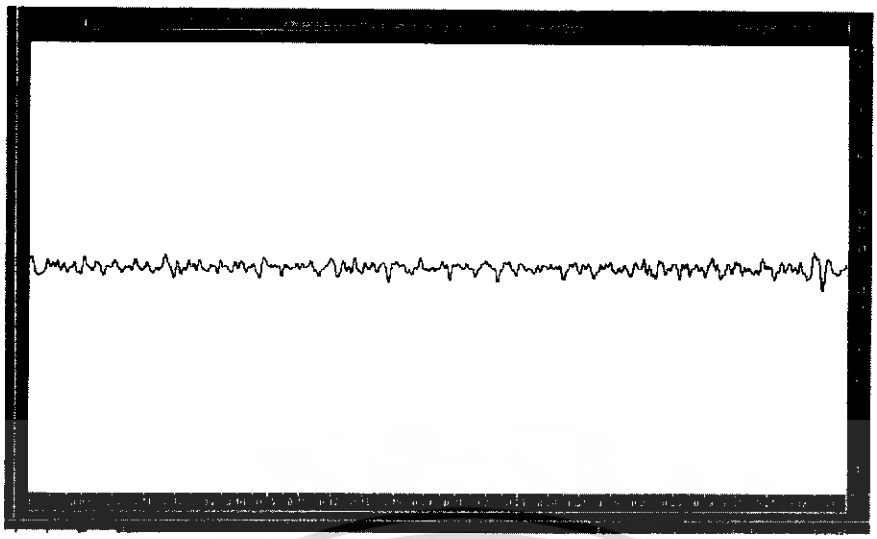
2.3.2 องค์ประกอบของสัญญาณเสียง

1. Voiced sound คือ สัญญาณที่มีคุณสมบัติซ้ำกันเป็นคาบ (Periodic signal) หากเปรียบเทียบกับการออกเสียงของเราก็คือเสียงที่ออกมาแล้วรู้สึกถึงการสั่นของเส้นเสียงที่ลำคอ เช่นการออกเสียงตัวอักษร b, d, g



รูปที่ 2-4 แสดงตัวอย่างลักษณะของสัญญาณเสียงที่เป็นคาบ

2. สัญญาณเสียงแบบสุ่ม (Unvoiced sound) ก็คือสัญญาณเสียงที่มีลักษณะของสัญญาณเป็นสัญญาณรบกวน หากเปรียบเทียบกับการออกเสียงของเราก็คือเสียงที่สร้างโดยการบังคับลมของช่องปาก เช่น การออกเสียง th, f, sh



รูปที่ 2-5 แสดงตัวอย่างลักษณะของสัญญาณ เสียงแบบสุ่ม

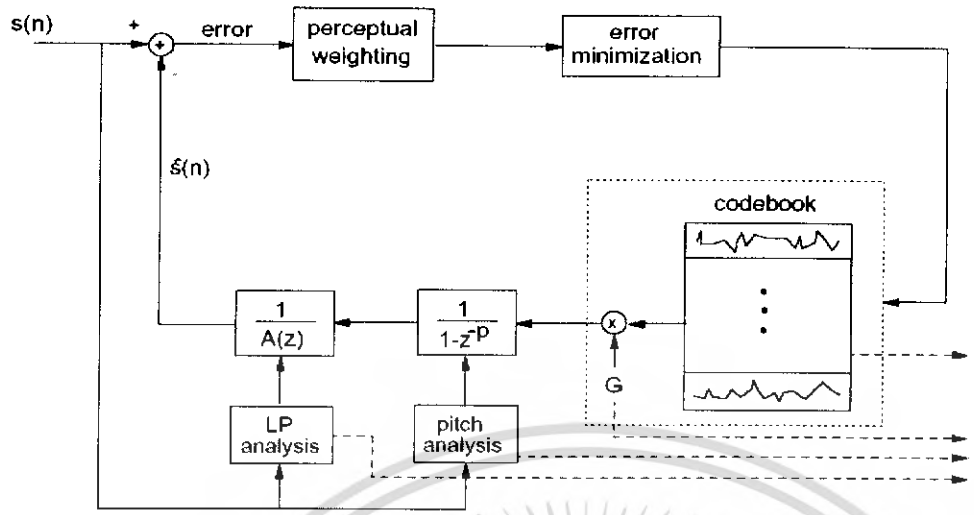
สำหรับในโครงงานนี้ซึ่งเป็นการการเข้ารหัส / ถอดรหัส ตามมาตรฐานของ ITU-T G.729 ซึ่งเป็นการเข้ารหัสสัญญาณเสียงแบบ 8 kbit/s โดยใช้ CS-ACELP โดยแบ่งเสียงออกเป็นเฟรม ซึ่งแต่ละเฟรม ได้มาจากการ sampling สัญญาณเสียงด้วย Sampling rate 8000 samples/s โดยทำการควอนไทซ์ ข้อมูลแบบ 16 Bits PCM โดยแต่ละเฟรม จะมีความยาว 10 mS โดย 1 เฟรม จะแบ่งออกเป็น 2 เฟรมย่อย ข้อมูลที่ทำการเข้ารหัสแล้วจะประกอบด้วย พารามิเตอร์ดังต่อไปนี้

Parameter	Codeword	Subframe 1	Subframe 2	Total per frame
Line spectrum pairs	*L0, L1, L2, L3*			18
Adaptive-codebook delay	*P1, P2*	8	5	13
Pitch-delay parity	*P0*	1		1
Fixed-codebook index	*C1, C2*	13	13	26
Fixed-codebook sign	*S1, S2*	4	4	8
Codebook gains (stage 1)	*GA1, GA2*	3	3	6
Codebook gains (stage 2)	*GB1, GB2*	4	4	8
Total				80

ตารางที่ 2-2 Bit allocation of the 8 kbit/s CS-ACELP Algorithms (10 mS frame)

โดยพารามิเตอร์นี้จะถูกส่งออกไป และตัวถอดรหัสจะทำการถอดรหัสพารามิเตอร์เหล่านี้เพื่อสร้างสัญญาณเสียง โดยสัญญาณเสียงที่สร้างขึ้นจะถูกกรองผ่านวงจร Short-term synthesis (LP filter), Long-term synthesis (Pitch filter) และ Post filter (เพื่อเติม ฮาร์โมนิก ของสัญญาณที่สังเคราะห์ขึ้น)

2.4 Code Excited Linear Prediction Model (CELP)



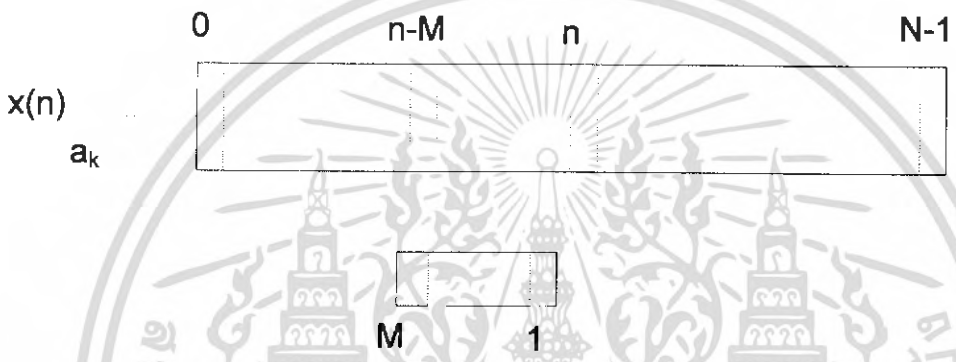
รูปที่ 2-6 แสดงแบบจำลองการสังเคราะห์เสียงแบบ CELP

G.729 จัดอยู่ในการเข้ารหัสเสียง (Speech coders) ใน category CELP (Code Excited Linear Prediction) ซึ่ง โมเดล(Model) ชนิดนี้มีพื้นฐานมาจากระบบกำเนิดเสียงของมนุษย์ โดยอวัยวะกำเนิดเสียงของคนเราจะถูกจำลองเป็นวงจรกรอง (Linear filter) ชุดหนึ่ง ซึ่งเสียงพูดนั้นจะถูกสร้างโดยอาศัยการสร้างสัญญาณที่มีคาบเท่ากับคาบการสั่นของอากาศมากระตุ้นวงจรกรองนี้ ถ้าเรามองในเชิงของ โดเมนเชิงความถี่แล้วเสียงพูดจะถูกพิจารณาจากองค์ประกอบ 2 ส่วนคือ ผลตอบสนองที่เรียบ[Smooth response (envelope)] หรือสัญญาณที่มีลักษณะเป็นคาบ (Periodic signal) มาทำการมอดูเลทกับ ชุดขององค์ประกอบของความถี่ที่ไม่เป็นคาบ (discrete frequency signal) ในการแทน รูปแบบการเข้ารหัส(Coder Model) ของ CELP นั้นจะแปรผันไปตามสัญญาณกระตุ้นที่ระบุ และค่าสัมประสิทธ์ของวงจรกรอง ซึ่งทั้ง 2 ส่วนนั้นจะหามาจากช่วงของสัญญาณเสียงที่ ตัดออกมาเป็น ยูนิท(unit) หรือเรียกอีกอย่างหนึ่งว่าเฟรม ซึ่งช่วงในการตัดสัญญาณเสียงที่นำมาพิจารณาสามารถ แปรผันได้ตั้งแต่ [1mS – 100mS] ชุดของพารามิเตอร์ที่สร้างขึ้นจะถูกส่งให้กับ โมดูลถอดรหัสในแต่ละเฟรม ของเสียง โดยที่ฝั่งเข้ารหัสต้องรอสัญญาณเสียงอย่างน้อย 1เฟรม ก่อน ถึงจะสามารถเข้ารหัสเฟรมข้อมูลได้ ซึ่งก็หมายความว่ามีการดีเลย์ของระบบเกิดขึ้น ใน G.729 นั้น 1 เฟรม ที่พิจารณาใช้เวลา 10 mS หรือ 80 ตัวอย่าง (Sampling rate 8Khz) โดยจะแบ่งย่อยเป็น 2 เฟรมย่อย (5 mS / เฟรมย่อย) ในส่วนของ พารามิเตอร์ของวงจรกรองจะคำนวณ 1 ครั้งต่อเฟรม ในขณะที่สัญญาณกระตุ้นจะหาแต่ละเฟรมย่อย ซึ่งในส่วนของสัญญาณกระตุ้นเป็นส่วนของสัญญาณอินพุท สำหรับสังเคราะห์เสียงประกอบด้วยสัญญาณเสียงแบบเป็นคาบ และ สัญญาณเสียงแบบสุ่ม

CELP Model ทำงานได้ดีกับเสียงที่เป็นคาบเนื่องจากเสียงที่เป็นคาบเป็นส่วนสำคัญที่ใช้พิจารณาในการหาสัมประสิทธิ์ของวงจรกรองแต่ในส่วนของสัญญาณกระตุ้น นั้นได้ทำการหาทั้ง 2 องค์ประกอบดังนั้นแบบจำลองชนิดนี้จึงสามารถใช้ได้กับทั้ง 2 กรณี

สำหรับวิธีการในการหาพารามิเตอร์ของวงจรกรองและสัญญาณกระตุ้น นั้นเรียกว่าการวิเคราะห์จากการสังเคราะห์ (Analysis by synthesis) ฝั่งเข้ารหัสจะทำการ ค้นหาหาพารามิเตอร์ที่เหมาะสมโดยทำการถอดรหัสกลับในแต่ละรอบของการค้นหาโดยนำสัญญาณที่สังเคราะห์ได้มาเปรียบเทียบกับสัญญาณต้นฉบับ พารามิเตอร์ชุดที่ใกล้เคียงกับสัญญาณต้นฉบับมากที่สุดจะถูกเลือกและส่งให้กับฝั่งถอดรหัส

เนื่องจากเราพิจารณาสัญญาณเสียงในช่วงเวลาสั้น ๆ เราจึงสามารถสร้างแบบจำลองในการกำเนิดเสียงในลักษณะของระบบที่ค่าสถิติทางเวลาคงที่ (Stationary system) จากคุณสมบัติของระบบดังกล่าวเราใช้วิธีประมาณความสัมพันธ์ระหว่างค่าของสัญญาณก่อนหน้ากับค่าของสัญญาณปัจจุบันโดยใช้วิธีการประมาณพันธะเชิงเส้น (Linear Prediction) เป็นค่าสัมประสิทธิ์ของวงจรกรองตามรูปที่ 2.7



Prediction of x(n) : $\hat{x}(n) = -\sum_{k=1}^{M} a_k x(n-k)$ (2.8)

Prediction error : $e(n) = x(n) - \hat{x}(n) = x(n) + \sum_{k=1}^{M} a_k x(n-k)$ (2.9)

เมื่อ a_k คือสัมประสิทธิ์ที่ได้จากการประมาณพันธะเชิงเส้น (LP coefficient)

รูปที่ 2-7 แสดงสมการที่ใช้แสดงความสัมพันธ์ในการประมาณค่าของสัญญาณโดยใช้วิธีการของการประมาณพันธะเชิงเส้น

บทที่ 3
มาตรฐานการเข้ารหัสสัญญาณ G.729

มาตรฐานการเข้ารหัสสัญญาณ G.729

ตัวเข้ารหัสสัญญาณนั้น ทำงานด้วยความเร็วต่อ 1 เฟรมของสัญญาณเสียง คือ 10 mS/frame อัตราของการ sample สัญญาณเสียงคือ 8000 sample/s ตามมาตรฐานของ G.712 ซึ่ง เป็น อนาล็อก (Analog Input) โดยใช้ PCM (Pulse Code modulate) 16 บิต และทุก ๆ 10 mS โดย 1 เฟรม ประกอบด้วย 2 เฟรมย่อย เฟรมย่อยละ 5 mS (40 samples) สัญญาณจะถูกวิเคราะห์ออกมาและวิเคราะห์พารามิเตอร์ต่าง ๆ คือ linear-prediction filter coefficient , adaptive and fixed codebook indices and gains ใน CELP model ซึ่งพารามิเตอร์ พวกนี้จะถูกเข้ารหัส และส่งออกไป และที่ การถอดรหัส (Decoder) สัญญาณเสียงจะถูกสร้างขึ้นใหม่โดยใช้การกรองสัญญาณกระตุ้น (excitation signal) ผ่าน short term synthesis filter ซึ่งมีพื้นฐานจาก 10 th Linear Prediction filter และ Long term synthesis filter โดยใช้ วิธีการของ adaptive-codebook approach และหลังจากนั้น จะผ่าน post filter เพื่อเพิ่มฮาร์โมนิก ของสัญญาณ

การเข้ารหัส (Encoder)

สัญญาณอินพุตจะถูกกรองผ่านความถี่สูงผ่าน (High pass filter) และกำหนดขนาดของสัญญาณก่อนที่จะประมวลผลสัญญาณ และใช้สัญญาณนี้ในการวิเคราะห์ LP analysis จะใช้ในการวิเคราะห์ทุก ๆ 10 mS (1 frame) เพื่อที่จะหา LP filter coefficient และค่าสัมประสิทธิ์ เหล่านี้นั้นจะถูกเปลี่ยนไปเป็น LSP และถูกควอนไทต์โดยการแทนด้วยเวกเตอร์ 18 บิต เพื่อเป็นตัวแทนของสัมประสิทธิ์ของตัวกรอง (Quantized LP coefficient) และสัญญาณเสียงจะถูกวิเคราะห์หาสัญญาณกระตุ้น โดยการเลือกสัญญาณที่นำมาแทน สัญญาณกระตุ้น นั้นจะถูกเลือกโดยใช้วิธีการ analysis by synthesis หาความผิดพลาดของสัญญาณต้นฉบับกับสัญญาณที่ถูกสร้างขึ้นมาใหม่จะต้องผิดพลาดน้อยที่สุด (error signal e (n)) โดย สัญญาณผิดพลาดนี้จะถูกกรองผ่าน perception weighting filter ซึ่ง ค่าสัมประสิทธิ์ของ ตัวกรองนั้นได้มาจาก Levinson – Durbin method (Unquantized LP coefficient) ซึ่งเป็นการปรับปรุงสัญญาณให้ตอบสนองกับความถี่ได้ดีขึ้น

สัญญาณกระตุ้น (Fixed and Adaptive-codebook parameters) จะถูกหาทุก ๆ 5 mS (ทุก ๆ 40 samples) ควอนไทต์ และไม่ควอนไทต์ LP filter coefficient จะถูกใช้สำหรับเฟรมย่อยที่สอง การหาพิทช์ นั้นจะถูกคำนวณหาทุก 10 mS โดยคร่าว ๆ (Open loop)และจะทำการคำนวณซ้ำในแต่ละเฟรมย่อย (Close loop) โดยใช้พื้นฐานของ ความคล้ายคลึงกัน (correlation) เป็นคาบของสัญญาณเสียงที่ได้รับ หลังจากนั้น Impulse response (h) ของ weighted synthesis filter จะถูกคำนวณหา และสัญญาณเป้าหมายที่เราต้องการหาคือ x (n) ซึ่งคือ อินพุตระบบ ถูกคำนวณโดยโดยใช้ LP residual r(n) ลบออกด้วยความผิดพลาดระหว่าง LP residual และสัญญาณกระตุ้น ซึ่งเปรียบได้กับการการลบออกด้วย the zero-input response แล้วนำไปผ่าน weighted synthesis filter ต่อไปเป็นขั้นตอนของ Close-loop pitch analysis ซึ่งพยายาม maximize correlation ระหว่าง x(n) กับ r(n)* h(n) เพื่อหาค่าที่ละเอียดของ พิทช์จะถูกเข้ารหัส เป็น 8 บิตสำหรับเฟรมย่อยแรก และ 5 บิต

สำหรับเฟรมย่อยที่ 2 ซึ่ง พิทช์นั้นถูกเข้ารหัส สัมพัทธ์กัน รวมถึง อัตราขยาย ของ Adaptive code-book ด้วย และ x(n) ที่ถูกกำหนดให้ลบออกด้วย adaptive code-book gain * filter adaptive code-book vector (filtered adaptive-codebook contribution) และจะได้สัญญาณเป้าหมายใหม่ คือ $x'(n)$ ซึ่งจะถูกใช้ในกระบวนหาพารามิเตอร์ ของ fixed codebook ซึ่งจะได้เวกเตอร์ 17 บิต สำหรับ fixed- codebook Gain ของ adaptive และ fixed codebook เป็นเวกเตอร์ ที่ถูกควอนไทต์ ด้วย 7 บิต (โดยใช้ MA prediction กับ fixed codebook gain) ท้ายที่สุดตัวกรอง memory จะถูกกำหนดโดยใช้ สัญญาณกระตุ้น ที่หาได้

การถอดรหัส (Decoder)

หลักการของการถอดรหัสสัญญาณ นั้น ขั้นตอนที่ 1 Parameter indices จะถูกแยกออกมาจาก bit stream ที่ได้รับ indices จะถูกถอดรหัส เพื่อที่จะได้รับ พารามิเตอร์ที่ใช้สำหรับการเข้ารหัส พารามิเตอร์เหล่านี้ประกอบด้วย LSP coefficient, 2 Fixed fractional pitch delay, 2 Fixed-codebook vectors และ Adaptive/Fixed code-book gain โดยค่าสัมประสิทธิ์ LSP นั้นจะถูกประมาณและแปลงกลับไปเป็นค่าสัมประสิทธิ์ตัวกรอง LP สำหรับแต่ละเฟรมย่อย โดยทำตามขั้นตอนต่อไปนี้

- สัญญาณกระตุ้น จะถูกสร้างขึ้นโดยบวกขนาด adaptive และ fixed codebook vector โดยขึ้นอยู่กับอัตราขยาย
- สัญญาณเสียงจะถูกสร้างขึ้นใหม่โดยใช้การกรองสัญญาณกระตุ้น ผ่านการสังเคราะห์ตัวกรอง LP
- เสียงที่ถูกสร้างขึ้นมาใหม่นั้นจะส่งผ่านไปยัง Post-processing stage ซึ่งประกอบด้วย post filter ซึ่งมีพื้นฐานมาจาก short-term และ long-term filter และ ตามด้วยการทำตัวกรองความถี่สูงผ่าน(high pass filter) และสเกล ขนาดของสัญญาณ

ดีเลย์ (Delay)

ตัวเข้ารหัสสำหรับการเข้ารหัสสัญญาณเสียงใช้ 10 mS โดยจะใช้อีก 5 mS สำหรับเฟรม ในอนาคต ทางทฤษฎีจะมีการดีเลย์ ทั้งหมด 15 mS สำหรับดีเลย์ ที่มากกว่านี้จะเกิดขึ้นจากสาเหตุดังต่อไปนี้

- เวลาที่ใช้ในการประมวลผลสำหรับการเข้ารหัส และการถอดรหัส
- เวลาในการส่งข้อมูลในสายสัญญาณ
- การทำ Multiplex delay เข้ากับข้อมูลอื่น ๆ

3.1 สัมประสิทธิ์ของวงจรกรอง (Filter coefficient)

ใน G.729 นั้นวงจรกรองที่ใช้เป็นแบบ 10^{th} order all-pole และเนื่องจากวงจรกรองชุดนี้ใช้ในการสังเคราะห์เสียงตัววงจรกรองจึงถูกเรียกว่าวงจรสังเคราะห์ (synthesis filter) ในส่วนของวงจรกรองวิเคราะห์นั้นจะใช้ส่วนกลับของวงจรกรองชนิด FIR (Finite Impulse Response) ซึ่งเมื่อผ่านเสียงเข้าไปยังวงจรกรองวิเคราะห์ ก็จะได้สัญญาณกระตุ้นสำหรับเสียงนั้นออกมา ที่จริงแล้ววงจรกรองวิเคราะห์ก็คือสัญญาณผิดพลาด

(error signal) ในการทำนายเสียงจากเสียงก่อนหน้า 10 ตัวอย่าง ซึ่งสามารถมองเป็นตัวพยากรณ์เชิงเส้น (linear predictor) ได้ในการแก้สมการตรง ๆ นั้นทำได้ลำบากจึงอาศัยวิธีการหาความสัมพันธ์จากคอร์รีเลชัน (correlation)เมทริกซ์ ของสัญญาณเสียง

ในการหาความสัมพันธ์ของสัญญาณในแต่ละเฟรม นั้นจะทำการพิจารณาช่วงของสัญญาณใน 30 mS ประกอบด้วย 15 mS จากอินพุทของเฟรม ก่อนหน้า 10 mS ของเฟรม ปัจจุบันและเฟรมล่วงหน้า อีก 5 mS ซึ่งทำให้เกิด อัลกอริทึมดีเลย์ที่ 15 mS เราจะวินโดว์ (window) สัญญาณก่อนด้วยฟังก์ชันแฮมมิ่งวินโดว์ (Hamming window) และควอเตอร์โคไซน์ (quarter cosine) แทนที่จะคำนวณการหาคอร์รีเลชันโดยตรงจากสัญญาณอินพุทที่เข้ามา เพื่อช่วยปรับรูปร่างของสัญญาณให้การหาคอรีรีเลชันเน้นส่วนที่พิจารณาให้อยู่ในส่วนของเฟรมอินพุทปัจจุบันแทนที่จะไปให้น้ำหนักกับเฟรม ก่อนหน้าหรือในอนาคตมากกว่า อีกหนึ่งขั้นตอนก่อนการหาคอร์รีเลชันเนื่องจากเสียงที่มีพิตท์ที่มีความถี่สูงเป็นองค์ประกอบ (เช่นเสียงผู้หญิง) มีสเปกตรอลเอนวิโลป (speetral envelope) สูงกว่าของพิทช์ความถี่ต่ำที่ถูกมอดูเลทเข้ามาด้วย ทำให้การประมาณหาเอนวิโลปที่ความถี่ระหว่างคาบของพิตท์ (pitch periods) นั้นต่ำไป การแก้ไขในจุดนี้ทำโดยการทำคอนโวลูชันค่าความถี่ (convolution frequeney)ของสเปกตรอลเอนวิโลปโดยเกาส์เซียนฟังก์ชันเพื่อขยายช่วงค่าสูงสุดของสเปกตรอลเอนวิโลปเพื่อเติมช่องว่างในส่วนนั้น

สัมประสิทธิ์ของวงจรกรองสังเคราะห์ LP อันดับที่ 10^{th} สามารถคำนวณหาได้โดยอาศัยคอร์รีเลชัน r(k) จากสมการของ Yule-Walker โดยใช้วิธีแก้ปัญหาแบบทำซ้ำโดยอัลกอริทึมของ Lervinson-Durbin

$$\sum_{i=1}^{10} a_i r(|i-k|) = -r_k \qquad (3.1)$$

ในขั้นตอนถัดไปก็จะทำการควอนไทต์วงจรกรอง แต่เนื่องจากเราไม่สามารถควอนไทต์มันโดยตรงได้เพราะ

1. เป็นไปได้ว่าควอนไทต์ noise อาจจะให้รากคำตอบไม่สามารถหาค่ากลับได้ทำให้มีปัญหาเรื่องความไม่เสถียรของวงจรกรอง
2. มันยากในการควอนไทต์สัญญาณรบกวนของสัมประสิทธิ์ที่ได้รับมาเนื่องจากการรับรู้สัญญาณรบกวนของคนเรามีพื้นฐานจากโดเมนเชิงความถี่

เพื่อแก้ปัญหา 2 ข้อนี้ค่าสัมประสิทธิ์จะถูกเปลี่ยนไปอยู่ใน Line Spectral Frequency หรือ LSF's โดยอาศัยสมการ

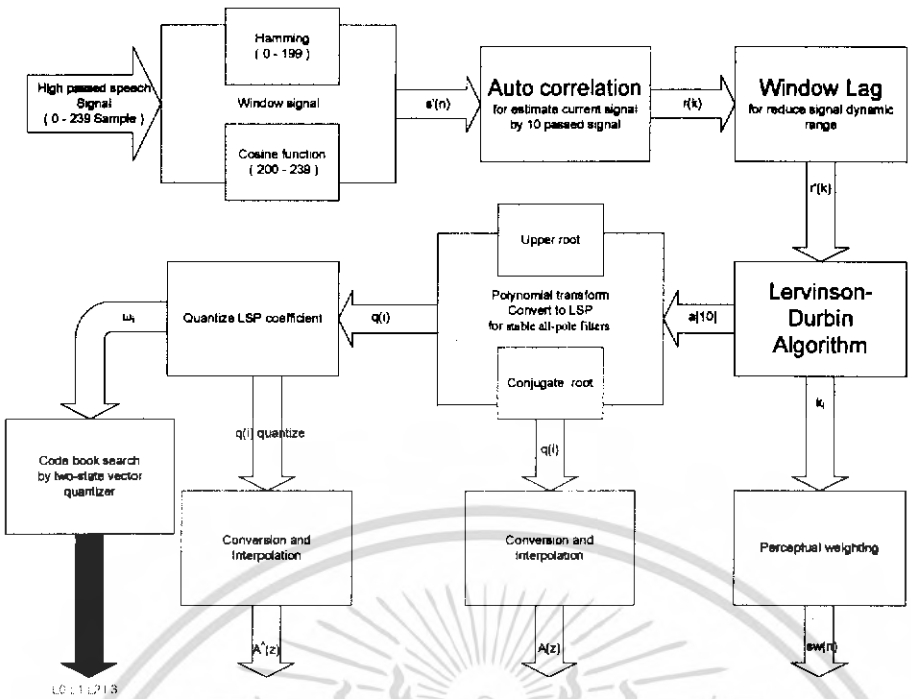
$$\begin{aligned} F_1(z) &= A(z) + z^{-11}A(z^{-1}) \\ F_2(z) &= A(z) - z^{-11}A(z^{-1}) \end{aligned} \qquad (3.2)$$

LSF's เป็นรากคำตอบของโพลิโนเมียลเหล่านี้ซึ่งมีคุณสมบัติที่สำคัญดังนี้

1. รากคำตอบอยู่บนวงกลมหนึ่งหน่วย
2. มีรากคำตอบเป็นคอนจูเกต (conjugate) ซึ่งกันและกัน (แสดงว่าสามารถหาส่วนกลับของวงจรกรองได้)
3. สำหรับคู่ของ โพลิโนเมียล ที่เป็นไปตาม 2 ข้อข้างต้นทำให้วงจรกรอง A(z) เป็นหนึ่งหน่วย (จากข้อ 1) และ วงจรสังเคราะห์กลับ (Inverse synthesis filter) เสถียร
4. การเปลี่ยนแปลง LSF ใด ๆ ทำให้เกิดการเปลี่ยนแปลงวงจรกรองวิเคราะห์ (analysis filter)ในช่วงความถี่เล็ก ๆ รอบ ๆ ความถี่ LSF นั้น

จากคุณสมบัติข้อ 3 ทำให้ตัวถอดรหัสง่ายต่อการระบุว่าวงจรกรองนั้นเสถียรหรือไม่และจากคุณสมบัติข้อ 4 ช่วยให้การควอนไทด์เซชัน LSF เกี่ยวข้องกับ frequency response ของ วงจรสังเคราะห์เพื่อลด Bandwidth ในการส่งข้อมูลฝั่งเข้ารหัสและถอดรหัสจะใช้การพยากรณ์ค่าของ LSF's โดยใช้ 4^{th} order moving average โดยจะมีชุดของ predictor ที่ใช้เลือกอยู่ 2 ชุด แล้วอาศัยบิตในการเลือกว่าจะใช้เวกเตอร์ ควอนไทด์เซชันชุดไหน การทำเวกเตอร์ ควอนไทด์เซชันแบ่งออกเป็น 2 สถานะโดยสถานะที่หนึ่ง 10-dimension codebook ประกอบด้วย 128 เอนทรีสำหรับค้นหา โดยเอนทรีที่ให้ผลลัพธ์ของการมินิไมซ์ค่า รากที่สอง ค่าเฉลี่ยของความผิดพลาด (mean square error) ระหว่างค่า LSF's จริงและสถานะที่สอง 10-dimension เวกเตอร์ จะถูกลบออกจากค่าจริง (Original LSF's) ซึ่งค่าผลต่างนี้จะถูกแบ่งออกเป็นสอง 5-dimension เวกเตอร์ โดยเอนทรีที่ให้ค่าที่ตรงกับเวกเตอร์ ชุดแรก (แทน LSF 1-5) จะถูกเลือกจาก codebook ชุดที่ 2 (โดยอาศัยการมินิไมซ์ค่ารากที่สองค่าเฉลี่ยของความผิดพลาดเช่นเดียวกับสถานะแรก) และเอนทรีที่ math กับเวกเตอร์ ชุดที่สอง (แทน LSF 6-10) ก็จะถูกเลือกออกมาจาก codebook ชุดที่ 3 ใน codebook ชุดที่ 2 และ 3 นี้ เป็นแบบ 5 – dimension ซึ่งมี 32 เอนทรีเช่นกัน

วิธีการหาแบบ two state structure นี้เรียกว่า Conjugate structure และ แทน "CS" ในชื่อของการเข้ารหัส สังเกตว่าเราจะส่งข้อมูลในการควอนไทด์เซชันไป 7 บิตเพื่อเลือก codebook ตัวแรก อีก 10 บิต สำหรับ เลือก codebook 32 เอนทรี 2 ชุด และอีก 1 บิตในการเลือกฟังก์ชัน moving average ที่ใช้ ในฝั่งถอดรหัสค่า LSF's ในเฟรมย่อย แรกจะทำการประมาณค่ากับเฟรม ที่แล้วแต่ในเฟรมย่อย ที่สอง จะใช้ค่าจริงที่ได้รับไปถอดรหัส



รูปที่ 3-1 แสดงโฟลโปรแกรมของการหาสัมประสิทธิ์ของวงจรกรอง

3.1.1 ขั้นตอนการหาสัมประสิทธิ์วงจรกรอง โดยละเอียด

3.1.1.1 เตรียมสัญญาณก่อนการประมวลผล (Pre-processing)

ประกอบด้วย 2 ฟังก์ชันที่สำคัญ คือ

1 Signal scaling ประกอบด้วยการหารด้วย 2 เพื่อป้องกันการเกิดโอเวอร์โฟล (overflow) ในกระบวนการทำ fixed point

2 วงจรกรองความถี่สูงเพื่อป้องกันสัญญาณที่มีความถี่ต่ำที่ไม่ต้องการ ตามสมการ

$$H_{hl}(z) = \frac{0.46363718 - 0.9272470z^{-1} + 0.46363718z^{-2}}{1 - 1.9059465z^{-1} + 0.9114024z^{-2}} \qquad (3.3)$$

ซึ่งเป็นสมการใน z โดเมน เมื่อแปลง z กลับจะได้สมการใน โดเมนของเวลาดังนี้

$$y(n) = 0.46363718x(n) - 0.92724705x(n-1) + 0.46363718x(n-2) - 1.9059465y(n-1) + 0.9114024y(n-2) \qquad (3.4)$$

s (n) แทน y (n) ของสัญญาณที่ผ่านขั้นตอนนี้แล้ว

3.1.1.2 วิเคราะห์หาสัมประสิทธิ์ LP และควอนไทด์เซชัน (Linear prediction analysis and quantization)

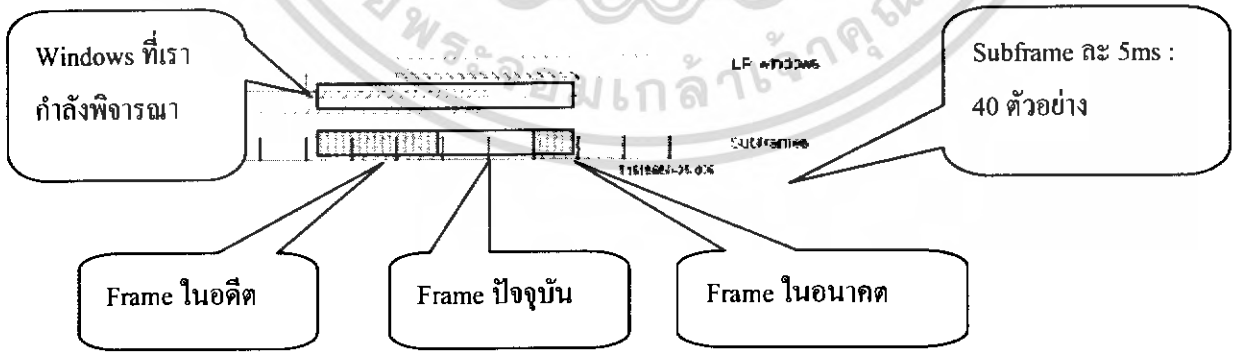
Short-term analysis และวงจรสังเคราะห์ มีพื้นฐานมาจากวงจรกรองลิเนียร์พรีดิคชันลำดับที่10 (10 th Order Linear Prediction (LP) filter) โดยจะถูกทำ 1 ครั้งทุก ๆ 1 เฟรม โดยใช้วิธีการออโตคอร์รีเลชัน (autocorrelation method) ซึ่งเป็นวินโดว์ที่ไม่สมมาตร (asymmetric window) 30 mS ทุก ๆ ค่าสัมประสิทธิ์ 80 ตัวอย่าง ของออโตคอร์รีเลชันของวินโดว์ของสัญญาณเสียงถูกคำนวณและเปลี่ยนกลับไปเป็นค่าสัมประสิทธิ์ LP โดยใช้อัลกอริทึมของ Levinson หลังจากนั้นค่าสัมประสิทธิ์ LP จะถูกเปลี่ยนไปเป็น LSPโดเมนสำหรับการประมาณและการทำควอนไทด์เซชันและ การประมาณที่ได้จากวงจรกรองชนิดควอนไทด์และ ไม่ควอนไทด์ จะถูกแปลงกลับไปเป็นค่าสัมประสิทธิ์วงจรกรอง LP อีกครั้ง เพื่อทำการสร้างและถ่วงน้ำหนัก สำหรับแต่ละ เฟรมย่อย

3.1.1.3 วินโดว์และการหาออโต้คอร์รีเลชัน (Windowing and autocorrelation computation)

LP analysis window ประกอบด้วย 2-ส่วน-โดยส่วนแรกเป็นฟังก์ชันแฮมมิ่งวินโดว์ และส่วนที่ 2 เป็น ¼ ของรอบของฟังก์ชันโคไซน์ (Cosine function)

$$w_{lp}(n) = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi n}{399}\right) & n = 0,...,199 \\ \cos\left(\dfrac{2\pi(n-200)}{159}\right) & n = 200,...,239 \end{cases} \quad (3.5)$$

โดย 5 mS ที่เพิ่มขึ้นมาใน LP analysis หมายความว่า 40 ตัวอย่าง จะต้องทำมาจาก เฟรม ในอนาคต ซึ่งการแปลงนี้จะทำให้เกิดดีเลย์ขึ้น 5 mS ในขณะทำการเข้ารหัส ดังนั้น LP analysis window ต้องการ 120 ตัวอย่าง จากเฟรมในอดีต 80 ตัวอย่าง จากเฟรมในปัจจุบันที่กำลังพิจารณา และ 40 ตัวอย่างจากเฟรมในอนาคต รวมเป็น 240 ตัวอย่างและคำนวณหา $s'(n)$



รูปที่ 3-2 แสดงเฟรม ของสัญญาณอินพุทที่ใช้ในการคำนวณ

สำหรับความสำคัญที่เราใช้ในการประมวลผลสัญญาณนั้นเราให้ไม่เท่ากันในแต่ละ ตัวอย่าง จึงต้องมีการคูณด้วยค่าถ่วงน้ำหนัก ซึ่งในที่นี้คือ ค่าของ วินโดว์ในตำแหน่งที่ตรงกับตัวอย่างนั้น ๆ

$$s'(n) = w_{lp}(n)s(n) \qquad n = 0,...,239 \tag{3.6}$$

ใช้ s′(n) ในการหาสัมประสิทธิ์ออโตคอร์รีเลชัน r (k)

$$r(k) = \sum_{n=k}^{239} s'(n)s'(n-k) \qquad k = 0,...,10 \tag{3.7}$$

เพื่อป้องกันการเกิดปัญหาค่าทางคณิตศาสตร์สำหรับ r(0) จะมีค่าต่ำกว่าขอบเขตที่กำหนด r (0) = 1.0 A ย่านของสัญญาณการขยาย 60 Hz จะถูกคูณเข้ากับ r (k)

$$w_{lag}(k) = \exp\left[-\tfrac{1}{2}\left(\frac{2\pi f_0 k}{f_s}\right)^2\right] \qquad k = 1,..,10 \tag{3.8}$$

ซึ่งแทนค่า f_0 = 60 Hz และ fs ด้วย 8000 Hz ซึ่งเป็นความถี่ในการทำสุ่มตัวอย่างและ r (0) จะต้องทำการคูณด้วยอัตราส่วนของ white noise correction คือ 1.0001 ซึ่งเทียบเท่ากับการบวกด้วยสัญญาณรบกวนที่ – 40 dB ดังนั้น r′(k) จะกลายเป็นออโตคอร์รีเลชันที่ถูกแก้ไขแล้ว

$$\begin{aligned} r'(0) &= 1.0001r(0) \\ r'(k) &= w_{lag}(k)r(k) \qquad k = 1,...,10 \end{aligned} \tag{3.9}$$

3.1.1.4 อัลกอริทึมของเลอวินสัน และเดอบิน (Levinson-Durbin Algorithms)

สมการที่แสดงความสัมพันธ์ระหว่างสัมประสิทธิ์ของออโตคอร์รีเลชันที่ถูกปรับปรุง r′(k) กับสัมประสิทธิ์ LP a_i i = 1,. …, 10

$$\sum_{i=1}^{10} a_i r'(|i-k|) = -r'(k) \qquad k = 1,...,10 \tag{3.10}$$

สำหรับวิธีการแก้สมการหานั้นใช้ความสัมพันธ์แบบรีเคอร์ซีฟดังต่อไปนี้

$$
\begin{aligned}
&E^{[0]} = r'(0) \\
&for \quad i = 1 \quad to \quad 10 \\
&\qquad a_0^{[i-1]} = 1 \\
&\qquad k_i = -\left[\sum_{j=0}^{i-1} a_j^{[i-1]} r'(i-j)\right] / E^{[i-1]} \\
&\qquad a_i^{[i]} = k_i \\
&\qquad for \quad j = 1 \quad to \quad i-1 \\
&\qquad\qquad a_j^{[i]} = a_j^{[i-1]} + k_i a_{i-j}^{[i-1]} \\
&\qquad end \\
&\qquad E^{[i]} = \left(1 - k_i^2\right) E^{[i-1]} \\
&end
\end{aligned}
\tag{3.11}
$$

สำหรับคำตอบที่ได้จะเท่ากับ $a_j = a_j^{[10]}$, $j = 0,\ldots,10$ โดย $a_0 = 1.0$

3.1.1.5 การแปลงจากโดเมนของ LP ไปยัง LSP (LP to LSP conversion)

สัมประสิทธิ์ของ LP a_i $i = 1,\ldots, 10$ จะถูกเปลี่ยนเป็นสัมประสิทธิ์ LSP สำหรับเพื่อทำควอนไทต์เซชันและ การประมาณค่า สำหรับวงจรกรอง LP ลำดับที่ 10 (10^{th} Order LP filter) นั้นสัมประสิทธิ์ LSP จะถูกนิยามอยู่ในรูปของผลรวมของรากของสมการและผลต่างของ โพลิโนเมียล

$$
\begin{aligned}
&F_1'(z) = A(z) + z^{-11} A(z^{-1}) \\
&and: \\
&F_2'(z) = A(z) - z^{-11} A(z^{-1})
\end{aligned}
\tag{3.12}
$$

โดยโพลิโนเมียล $F'_1(z)$ เป็น symmetric และโพลิโนเมียล $F'_2(z)$ เป็น anti symmetric จะสามารถพิสูจน์ได้ว่ารากทุกรากของสมการจะอยู่บนวงกลมหนึ่งหน่วยและเป็นทางเลือก (alternate) ซึ่งกันและกัน $F'_1(z)$ มีรากที่ $z = -1$ และ $F'_2(z)$ มีราก $z = 1$ ซึ่งจะทำให้กำจัดเทอมได้และ นิยามโพลิโนเมียลใหม่อีก 2 ตัว

$$
\begin{aligned}
&F_1(z) = F_1'(z)/(1 + z^{-1}) \\
&and: \\
&F_2(z) = F_2'(z)/(1 - z^{-1})
\end{aligned}
\tag{3.13}
$$

ซึ่งแต่ละสมการจะมี 5 รากที่เป็นคอนจูเกตซึ่งกันและกันบน วงกลมหนึ่งหน่วย ซึ่งสามารถเขียนอยู่ในรูปของผลคูณได้

$$F_1(z) = \prod_{i=1,3,\ldots,9} (1 - 2q_i z^{-1} + z^{-2})$$

$$and: \tag{3.14}$$

$$F_2(z) = \prod_{i=2,4,\ldots,10} (1 - 2q_i z^{-1} + z^{-2})$$

โดย q_i ตามสมการนั้นแทนด้วย cos (ω_i) ซึ่ง ω_i แต่ละพจน์ เรียกว่า Line Spectral Frequencies (LSF) และเรียงลำดับ $0 < \omega_1 < \omega_2 < \ldots < \omega_{10} < \pi$ โดย q_i แต่ละพจน์นั้นหมายถึงสัมประสิทธิ์ LSP ในโดเมนของโคไซน์เพราะว่าโพลิโนเมียล $F_1(z)$ และ $F_2(z)$ นั้นสมมาตร (symmetric) เฉพาะสัมประสิทธิ์ ห้าพจน์แรกซึ่งต้องการที่จะทำการคำนวณ โดย สัมประสิทธิ์ ดังกล่าวนั้นจะสามารถหาได้จากความสัมพันธ์แบบ รีเคอร์ซีฟ

$$\begin{aligned} f_1(i+1) &= a_{i+1} + a_{10-i} - f_1(i) \qquad i = 0,..,4 \\ f_2(i+1) &= a_{i+1} - a_{10-i} + f_2(i) \qquad i = 0,..,4 \end{aligned} \tag{3.15}$$

โดยกำหนดให้ $f_1(0) = f_2(0) = 1.0$ สัมประสิทธิ์ LSP นั้นจะหาเจอเมื่อมีการหาค่าของ $F_1(z)$ และ $F_2(z)$ แบ่งช่วงระหว่าง $0 - \pi$ ออกเป็น 60 จุดและทำการตรวจสอบเครื่องหมาย เมื่อมีการเปลี่ยนแปลงเครื่องหมายแสดงว่า เกิดรากขึ้นที่ระหว่างจุด ๆ นั้น และทำการแบ่งพื้นที่ นั้นลงไปอีก 4 ส่วนจะทำให้ได้ค่าประมาณของรากที่ดีขึ้นโพลิโนเมียลของ Chebyshev ถูกใช้ในการหาค่า $F_1(z)$ และ $F_2(z)$ ซึ่งวิธีนี้จะหารากได้เป็นโดเมนของโคไซน์

$$F(\omega) = 2e^{-j5\omega} C(x)$$

$$with: \tag{3.16}$$

$$C(x) = T_5(x) + f(1)T_4(x) + f(2)T_3(x) + f(3)T_2(x) + f(4)T_1(x) + f(5)/2$$

โดย $T_m(x) = \cos(m\omega)$ และ m เป็นอันดับของโพลิโนเมียลของ Chebyshev และ f(i) เป็นสัมประสิทธิ์ของ $F_1(z)$ และ $F_2(z)$ โพลิโนเมียล C(x) จะถูกหาค่าได้ค่าที่ $x = \cos(\omega)$ โดยใช้ความสัมพันธ์แบบรีเคอร์ซีฟ

$$\begin{aligned} &for \quad k = 4 \quad down \quad to \quad 1 \\ &\qquad b_k = 2xb_{k+1} - b_{k+2} + f(5-k) \\ &end \\ &C(x) = xb_1 - b_2 + f(5)/2 \end{aligned} \tag{3.17}$$

$$init: \qquad b_5 = 1 \quad and \; b_6 = 0$$

3.1.1.6 การควอนไทต์สัมประสิทธิ์ LSP (Quantization of the LSP coefficient)

สัมประสิทธิ์ LSP q_i นั้นถูกควอนไทต์โดยใช้ LSF ω_i และ Switched 4 th order MA นั้นถูกใช้เพื่อการพยากรณ์ค่าของสัมประสิทธิ์ LSF ของเฟรม ปัจจุบัน โดยผลต่างระหว่างค่าที่คำนวณได้กับการพยากรณ์สัมประสิทธิ์จะถูกควอนไทต์โดยใช้ two stage vector quantizer โดยสถานะแรก จะเป็น VQ 7 บิตโดยใช้ L1 สำหรับ 128 เอนทรี ส่วนสถานะที่สองจะเหลือ 10 บิตแบ่งออกเป็น VQ 5 บิต 2 ส่วน แทน L2 และ L3 ตามลำดับ ซึ่ง 5 บิตนั้นแทน 32 เอนทรี

ในกระบวนการถอดรหัสสัมประสิทธิ์ที่ได้รับนั้นจะอยู่ในรูปของผลรวมของทั้ง 2-Codebook

$$\hat{l}_i = \begin{cases} L1_i(L1) + L2_i(L2) & i = 1,2,..,5 \\ L1_i(L1) + L3_{i-5}(L3) & i = 6,7,..,10 \end{cases} \tag{3.18}$$

โดย L1,L2 และ L3 เป็นอินเด็กของ codebook เพื่อลดการเกิด sharp resonance ในการทำควอนไทต์วงจรสังเคราะห์ LP โดยสัมประสิทธิ์ของควอนไทต์เซอร์เอาท์พุท (Quantizer output) จะต้องถูกจัดเรียงจนกระทั่งสัมประสิทธิ์ที่อยู่ติดกันจะมีระยะทางน้อยกว่า J โดยวิธีการจัดเรียงตามอัลกอริทึม

$$\begin{aligned} &for \quad i = 2 \quad to \quad 10 \\ &\quad if(\hat{l}_{i-1} > \hat{l}_i - J) \\ &\quad\quad \hat{l}_{i-1} = (\hat{l}_i + \hat{l}_{i-1} - J)/2 \\ &\quad\quad \hat{l}_i = (\hat{l}_i + \hat{l}_{i-1} + J)/2 \\ &\quad end \\ &end \end{aligned} \tag{3.19}$$

การจัดเรียงจะต้องทำสองครั้ง โดยครั้งแรก J = 0.0012 และ J = 0.0006 หลังจากกระบวนการเรียงแล้ว สัมประสิทธิ์ของ LSF ที่ถูกควอนไทต์ (Quantized LSF coefficient) สำหรับเฟรม ปัจจุบัน จะสามารถหาได้จากควอนไทต์เซอร์เวกเตอร์ ของ 4 เฟรมก่อนหน้า ดังนี้

$$\hat{\omega}_i^{(m)} = \left(1 - \sum_{k=1}^{4} \hat{p}_{i,k}\right) \hat{l}_i^{(m)} + \sum_{k=1}^{4} \hat{p}_{i,k}\, \hat{l}_i^{(m-k)} \qquad i = 1,..,10 \tag{3.20}$$

ซึ่ง $\hat{p}_{i,k}$ คือ Switched MA predictor coefficient ซึ่งมี 2 สมการ จะใช้สมการไหนขึ้นอยู่กับบิต L0 โดยค่าเริ่มต้นของ $l_i^{(k)}$ เมื่อ $k < 0$ มีค่าเท่ากับ $i\pi$ /11

หลังจากการทำการหาสัมประสิทธิ์ของ LSF ที่ถูกควอนไทต์แล้ว ต้องตรวจสอบความเสถียรของรากคำตอบ โดยทำขั้นตอนดังนี้

- เรียงลำดับสัมประสิทธิ์ของ LSF ที่ถูกควอนไทต์จากน้อยไปมาก
- ถ้าสัมประสิทธิ์ของ LSF ที่ถูกควอนไทต์มีค่าน้อยกว่า 0.005 กำหนดให้ เท่ากับ 0.005
- ถ้า $\omega_{i+1} - \omega_i - 0.0391$ แล้ว กำหนดให้ $\omega_{i+1} = \omega_i + 0.0391$
- ถ้า $\omega_{10} > 3.135$ แล้ว $\omega_{10} = 3.135$

สำหรับกระบวนการเข้ารหัส LSF นั้น จะมีสมการที่ใช้ในการพยากรณ์ 2 สมการ จะให้ค่าประมาณที่ดีที่สุดของสัมประสิทธิ์ LSF ซึ่งการประมาณที่ดีที่สุดนั้นจะต้องทำให้ weighted mean-squared error (MSE)

$$E_{lsf} = \sum_{i=1}^{10} w_i(\omega_i - \hat{\omega}_i)^2 \tag{3.21}$$

สำหรับ Weight (w) นั้นหาได้จากฟังก์ชัน ของสัมประสิทธิ์ LSF ที่ไม่ควอนไทต์ ดังนี้

$$\begin{aligned} w_i &= \begin{cases} 1.0 & if \quad \omega_2 - 0.04\pi - 1 > 0 \\ 10(\omega_2 - 0.04\pi - 1)^2 + 1 & otherwise \end{cases} \\ w_i \quad 2 \le i \le 9 &= \begin{cases} 1.0 & if \quad \omega_{i+1} - \omega_{i-1} - 1 > 0 \\ 10(\omega_{i+1} - \omega_{i-1} - 1)^2 + 1 & otherwise \end{cases} \\ w_{10} &= \begin{cases} 1.0 & if \quad -\omega_9 + 0.92\pi - 1 > 0 \\ 10(-\omega_9 + 0.92\pi - 1)^2 + 1 & otherwise \end{cases} \end{aligned} \tag{3.22}$$

สำหรับ w_5 และ w_6 นั้นคูณด้วย 1.2

ควอนไทต์เซอร์เวกเตอร์ ของเฟรมปัจจุบัน ได้มาจาก สมการดังต่อไปนี้

$$l_i = \left[\omega_i^{(m)} - \sum_{k=1}^{4} \hat{p}_{i,k} \hat{l}_i^{(m-k)} \right] \Big/ \left(1 - \sum_{k=1}^{4} \hat{p}_{i,k} \right) \qquad i = 1,....,10 \tag{3.23}$$

โดย L1 จะถูกหาและ เอนทรี ของ L1 ที่ทำให้ค่าของรากที่สองค่าเฉลี่ยของความผิดพลาดน้อยที่สุดจะถูกเลือก และ L2 จะถูกหาและเลือกเช่นเดียวกับ L1 แต่จะเป็นส่วนของ 5 บิตล่างของสถานะที่สอง L2 ที่หามาได้จะต้อง และจัดเรียงเพื่อประกันระยะทางสั้นสุดคือ 0.0012 และใช้ L1 และ L2 ที่ถูกเลือกเพื่อหา L3 อีกครั้ง ต้องมีการเรียงเพื่อประกันระยะทางสั้นสุดคือ 0.0012 และเวกเตอร์ L3 จะต้องให้ผลลัพธ์ความผิดพลาดที่น้อยที่สุดควอนไทต์เซอร์เวกเตอร์ จะต้องถูกนำมาเรียงใหม่โดยรับประกันระยะทางสั้นสุด 0.0006 ซึ่ง process นี้เสร็จสิ้นจะต้องทำให้เสร็จสำหรับ MA predictor ทั้งสองสมการ จะทำการหาค่า L0 และ MA predictor จะเลือก I.0 ที่ให้ค่าของรากที่สองค่าเฉลี่ยของความผิดพลาดต่ำสุดจะถูกเลือก

3.1.1.7 การประมาณค่าของสัมประสิทธิ์ LSP (interpolation of LSP coefficient)

ควอนไทต์(และสัมประสิทธิ์ LP ที่ไม่ควอนไทต์) จะถูกใช้สำหรับเฟรมย่อยที่ 2 สำหรับ เฟรมย่อยแรกควอนไทต์ (และสัมประสิทธิ์ LP ที่ไม่ควอนไทต์) จะได้มาจากทำการประมาณค่าเชิงเส้น (Linear interpolation) ซึ่งจะเกี่ยวข้องกับพารามิเตอร์ของเฟรมย่อยที่อยู่ติดกัน กำหนดให้ $q^{(current)}$ เป็นสัมประสิทธิ์ LSP ซึ่งคำนวณได้จากเฟรม 10 mS ปัจจุบัน และ $q^{(previous)}$ เป็น LSP coefficient ซึ่งคำนวณได้จากเฟรม 10 mS ก่อนหน้า สำหรับสัมประสิทธิ์ของ LSP ที่ไม่ควอนไทต์ ของแต่ละ 2 เฟรมย่อย ดังนี้

$$\begin{aligned} &Subframe\,1: \quad \hat{q}_i = 0.5 * \hat{q}_i^{(previous)} + 0.5 * \hat{q}_i^{(current)} \qquad i = 1,2,...,10 \\ &Subframe\,2: \quad \hat{q}_i = \hat{q}_i^{(current)} \qquad\qquad\qquad\qquad\qquad\quad i = 1,2,...,10 \end{aligned} \tag{3.24}$$

3.1.1.8 การแปลงจากโดเมน ของ LSP ไปยัง LP (LSP to LP conversion)

แต่ละครั้งที่สัมประสิทธิ์ LSP ถูกควอนไทต์ และหาค่า จะต้องถูกเปลี่ยนกลับมาเป็นสัมประสิทธิ์ LP ซึ่งขั้นตอนแสดงความสัมพันธ์แบบรีเคอซีฟตาม อัลกอริทึม

$$\begin{aligned} &for \;\; i = 1 \;\; to \;\; 5 \\ &\quad f_1(i) = -2q_{2i-1}f_1(i-1) + 2f_1(i-2) \\ &\quad for \;\; j = i-1 \;\; down \;\; to \;\; 1 \\ &\qquad f_1^{[i]}(j) = f_1^{[i-1]}(j) - 2q_{2i-1}f_1^{[i-1]}(j-1) + f_1^{[i-1]}(i-2) \\ &\quad end \\ &end \end{aligned} \tag{3.25}$$

กำหนดให้ $f_1(0) = 1$ และ $f_1(-1) = 0$ และ $f_2(i)$ ก็สามารถคำนวณได้เช่นเดียวกันแต่ต้องแทนค่า q_{2i+1} ด้วย q_{2i}

แต่ละครั้งที่เจอค่าสัมประสิทธิ์ $f_1(i)$ และ $f_2(i)$ $F_1(z)$ และ $F_2(z)$ จะต้องถูกคูณด้วย $(1 + z^{-1})$ และ $(1 - z^{-1})$ เพื่อที่จะได้ $F'_1(z)$ และ $F'_2(z)$ จะได้

$$\begin{aligned} f'_1(i) &= f_1(i) + f_1(i-1) \qquad i = 1,..,5 \\ f'_2(i) &= f_2(i) + f_2(i-1) \qquad i = 1,..,5 \end{aligned} \tag{3.26}$$

ท้ายที่สุดสัมประสิทธิ์ LP จะถูกคำนวณจาก $f'_1(i)$ และ $f'_2(i)$

$$a_i = \begin{cases} 0.5f'_1(i) + 0.5f'_2(i) & i = 1,...,5 \\ 0.5f'_1(11-i) + 0.5f'_2(11-i) & i = 6,...,10 \end{cases} \tag{3.27}$$

ซึ่งเป็นวิธีโดยตรง ซึ่งหามาได้จากความสัมพันธ์ A (z) = ($F'_1(z) + F'_2(z)$) / 2 เพราะว่า $F'_1(z)$ และ $F'_2(z)$ นั้นเป็น symmetric และ anti symmetric

3.2 สัญญาณกระตุ้น (Excitation)

ในขั้นตอน ต่อไปจะกล่าวถึงวิธีการที่ใช้หาสัญญาณกระตุ้นซึ่งจะหาสำหรับแต่ละเฟรมย่อย สัญญาณกระตุ้นถูกแทนด้วยผลรวมขององค์ประกอบสองส่วน อย่างที่แรกก็คือ Version ที่ถูกดีเลย์ของ สัญญาณกระตุ้นที่ใช้ ในส่วนที่สองคืออิมพัลซ์ (impulse) ในตำแหน่งต่าง ๆ ของสัญญาณ 4 ตำแหน่ง ส่วนประกอบส่วนแรกนั้นเรียกว่า adaptive codebook เป็นแบบจำลองของสัญญาณเสียงที่มีคุณสมบัติเป็นคาบ ซึ่งจริง ๆ แล้วก็คือพิตช์ดีเลย์ ในสัญญาณเสียงนั้น

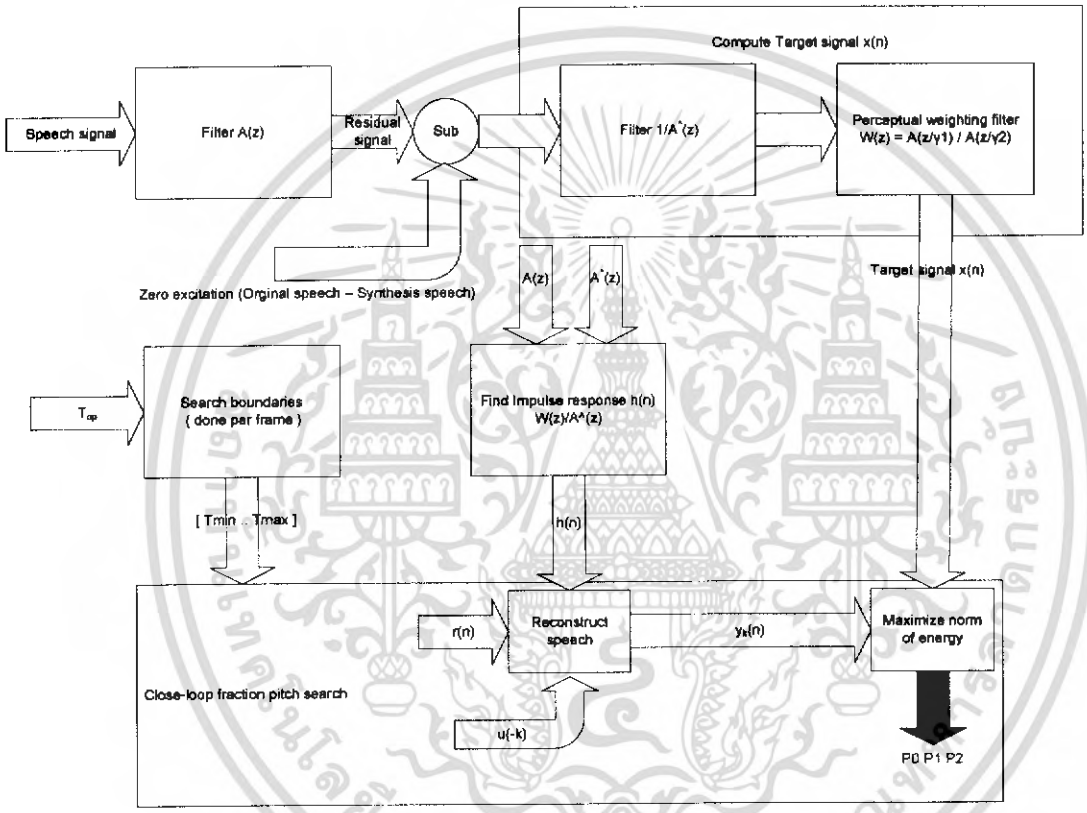
ในขั้นตอนแรกของการหาพิตช์ดีเลย์นั้น จะหาพิตช์ดีเลย์แบบหยาบ ๆ ก่อนโดยจะนำสัญญาณเสียงมาทำออโตคอร์รีเลชันซึ่งแบ่งช่วงของการทำเป็น 3 ช่วงแล้วเลือกอินเด็ก(delay) ของการทำออโตคอร์รีเลชันที่ให้ค่าสูงสุดในแต่ละช่วงมาเพื่อเลือกอีกทีโดยจะให้น้ำหนักกับ Harmonic ที่อยู่ในช่วงต่ำกว่ามากกว่า Harmonic ช่วงสูง เรียกการค้นหาแบบหยาบ ๆ ในขั้นตอนนี้ว่า Open-loop pitch analysis แล้วนำค่าพิตช์แบบหยาบ ๆ ที่ได้มาหาแบบละเอียดรอบ ๆบริเวณนั้น ในขั้นตอนต่อไปโดย การนำสัญญาณกระตุ้นของเฟรม ก่อนหน้ามาผ่านวงจรสังเคราะห์ LP แล้วนำผลลัพธ์ที่ได้มาทำการคอร์รีเลชันกับสัญญาณเสียงจริง หารด้วยขนาดของเอาท์พุทของวงจรสังเคราะห์ (เพื่อไม่ต้องพิจารณาอัตราขยายสัญญาณในขั้นตอนการ search)ดีเลย์ที่ทำให้พลังงานสูงที่สุด ค่าอัตราขยายสัญญาณจะหาได้จากผลลัพธ์ที่ได้เลือกมา เราจะใช้ส่วนที่เหลือจากการนำดีเลย์ที่เหมาะสมถูกเลือกขึ้นมาผ่าน วงจรสังเคราะห์ LP ลบออกจาก สัญญาณอินพุทแล้วนำส่วนที่เหลือนี้ไปใช้หาสัญญาณกระตุ้นส่วนที่เหลือ (unvoiced component) ต่อไป

ในขั้นตอนนี้เรียกว่า fixed codebook contribution โดยสัญญาณกระตุ้นที่เหลือนี้จะถูกแทนด้วยอิมพัลซ์ที่มีแอมพลิจูด ±1 อยู่ 4 ตัว codebook เวกเตอร์ ที่ถูกเลือกโดยอิมพัลซ์ เหล่านั้นจะถูกเพิ่มองค์ประกอบของ Harmonic เพื่อเพิ่มคุณภาพของสัญญาณเสียงที่จะสังเคราะห์ขึ้นมาโดยวงจรกรองแบบฮาร์โมนิก (Harmonic filter) ในการค้นหานั้นจะได้ตำแหน่งของพัลซ์ และเครื่องหมาย โดยจะเลือกผลลัพธ์ของเมทริกซ์ที่ค้นหาที่ให้ค่ามากที่สุดของการกระตุ้นวงจรกรองด้วยสัญญาณกระตุ้นไปยังวงจรสังเคราะห์ คูณด้วย target signal (speech input ส่วนที่เหลือจากการหา Open-loop) หารด้วยขนาดพลังงาน ของเอาท์พุท จากวงจรสังเคราะห์ (เพื่อไม่ต้องพิจารณาอัตราขยายสัญญาณในขั้นตอนการ search) หลังจากนั้นจึงหาอัตราขยายสัญญาณจากสัญญาณกระตุ้นที่ได้เลือกไว้

ในขั้นนี้แต่ละเฟรมย่อย ได้หา พารามิเตอร์พิตช์ดีเลย์, อัตราขยายสัญญาณ adaptive codebook, สัญญาณกระตุ้นของ fixed codebook และอัตราขยายสัญญาณ ซึ่งพารามิเตอร์เหล่านี้จะถูกควอนไทต์และส่งไปให้กับฝั่งถอดรหัสในเฟรมย่อยแรกพิตช์ดีเลย์จะถูกส่งออกไปปรกติ (8 บิต) แต่ในเฟรมย่อยที่ 2 นั้นจะใช้ค่า rclative จากเฟรมย่อยแรกทำให้ใช้เพียง (5 บิตในการส่ง) ส่วนของ fixed codebook นั้นก็ทำการส่งออกไปตรง ๆ โดยใช้ 4 บิตสำหรับเครื่องหมายและ 13 บิตสำหรับการระบุตำแหน่งของพัลซ์ แต่ในส่วนของอัตราขยายสัญญาณนั้น (อัตราขยายสัญญาณ adaptive codebook และ fixed codebook) ใช้การทำนายจากเฟรม ก่อนหน้าโดยอาศัยองค์ประกอบอัตราขยายสัญญาณ (gain factor) ซึ่งเป็นความสัมพันธ์ของอัตราขยายสัญญาณทั้งสอง

ในรูปแบบของผลคูณเพื่อทำนายความผิดพลาดโดยองค์ประกอบอัตราขยายสัญญาณ และอัตราขยายสัญญาณ fixed codebook ใช้การรวมความสัมพันธ์แล้ว ควอนไทต์แบบ two stage vector ควอนไทต์เซชัน สถานะแรกประกอบด้วย 3 บิต two dimension codebook หนึ่งชุด สถานะถัดไปประกอบด้วย 4 บิต two dimension หนึ่งชุด ซึ่งผลรวมของ codebook ทั้งคู่จะแทนองค์ประกอบอัตราขยายสัญญาณ และอัตราขยายสัญญาณ fixed codebook ที่ใช้ในการส่งออกไป

3.2.1 การหาองค์ประกอบสัญญาณเสียงที่เป็นคาบ (Voice sound)



รูปที่ 3-3 แสดงวิธีการหาสัญญาณกระตุ้นที่มีลักษณะของสัญญาณเป็นคาบ (Voice sound)

3.2.1.1 การปรับลักษณะของสัญญาณ (Perceptual weighting)

Perceptual weighting นั้นมีพื้นฐานมาจากสัมประสิทธิ์ของ LP ที่ไม่ควอนไทต์

$$W(z) = \frac{A(z/\gamma 1)}{A(z/\gamma 2)} = \frac{1 + \sum_{i=1}^{10} \gamma_1^i a_i z^{-i}}{1 + \sum_{i=1}^{10} \gamma_2^i a_i z^{-i}} \qquad (3.28)$$

โดยค่าของ γ_1 และ γ_2 นำไปสามารถนำไปหาผลของการตอบสนองเชิงความถี่ของวงจรกรอง W(z) สำหรับการปรับที่ค่าของ γ_1 และ γ_2 เหมาะสมจะทำให้การทำถ่วงน้ำหนักเกิดประสิทธิภาพมาก ซึ่งจะต้องทำให้ γ_1 และ γ_2 นั้นเป็นฟังก์ชัน spectral แบบเดียวกับสัญญาณอินพุทซึ่งการทำนี้จะทำเพียงครั้งเดียวใน 10 mS และมีการประมาณในเฟรมย่อยแรกของแต่ละเฟรมจะทำให้การปรับสัญญาณให้เรียบขึ้น สำหรับฟังก์ชัน Spectral นั้นจะสามารถหาได้จากวงจรกรองแบบการทำนายเชิงเส้นอันดับที่สอง (2^{nd} order linear prediction) ซึ่งได้มาจากผลพลอยได้ของ Lervinson – Durbin recursion โดยสัมประสิทธิ์การสะท้อน (reflection coefficient) k ถูกเปลี่ยนไปเป็นสัมประสิทธิ์ของอัตราส่วน Log Area (LAR) o

$$o_i = \log\frac{(1.0+k_i)}{(1.0-k_i)} \qquad i=1,2 \tag{3.29}$$

ค่าของสัมประสิทธิ์ LAR ในเฟรมปัจจุบัน ใช้ในเฟรมย่อยที่ 2 สัมประสิทธิ์ LAR สำหรับเฟรมย่อยแรกนั้น ค่าที่ใช้แทนนั้นหาได้จากการประมาณค่าจากเฟรมก่อนหน้ากับเฟรมปัจจุบัน

$$\begin{aligned} &Subframe1: \; o_i^{(1)} = 0.5o_i^{(previous)} + 0.5o_i^{(current)} && i=1,2 \\ &Subframe2: \; o_i^{(2)} = o_i^{(current)} && i=1,2 \end{aligned} \tag{3.30}$$

โดยคุณสมบัติของ Spectral Envelope ของสัญญาณ แบ่งออกเป็น flat (flat = 1) หรือ tilt (flat = 0) สำหรับแต่ละเฟรมย่อย คุณสมบัติเหล่านี้ขึ้นอยู่กับสัมประสิทธิ์ LAR ดังสมการ

$$flat^{(m)} = \begin{cases} 0 & if \;\; o_1^{(m)} < -1.74 \;\; and \;\; o_2^{(m)} > 0.65 \;\; and \;\; flat^{(m-1)} = 1 \\ 1 & if \;\; (o_1^{(m)} > -1.52 \;\; or \;\; o_2^{(m)} < 0.43) \;\; and \;\; flat^{(m-1)} = 1 \\ flat^{(m-1)} & otherwise \end{cases} \tag{3.31}$$

ดังที่กล่าวมาข้างต้นว่า γ_1 และ γ_2 มีค่าขึ้นอยู่กับฟังก์ชัน สเปคตรัม ดังนั้น ค่าของ γ_1 และ γ_2 กำหนด ดังนี้

- flat (flat = 1) weight factor γ_1 จะเท่ากับ 0.94 และ γ_2 จะเท่ากับ 0.6
- tilted (flat = 0) weight factor γ_1 จะเท่ากับ 0.98 และ γ_2 จะปรับเปลี่ยนโดยขึ้นอยู่กับความแข็งของ เรโซแนนซ์ใน วงจรสังเคราะห์ LP แต่อยู่ในช่วง 0.4 – 0.7 ถ้า แข็งแรงมากมาก γ_2 ก็จะเข้าใกล้ 0.7 การปรับนี้จะมากหรือน้อยโดยขึ้นอยู่กับ ระยะทางที่น้อยที่สุดระหว่างสัมประสิทธิ์ LSP ของเฟรมนั้น

$$d_{\min} = \min[\omega_{i+1} - \omega_i] \qquad i=1,...,9 \tag{3.32}$$

โดยค่าของ γ_2 จะมีความสัมพันธ์กับ d ดังนี้

$$\gamma_2 = -6.0d_{\min} + 1.0 \qquad bounded\ \ by\ \ 0.4 \le \gamma_2 \le 0.7 \tag{3.33}$$

Weighted speech signal (sw (n)) หาได้ตามสมการ ดังต่อไปนี้

$$sw(n) = s(n) + \sum_{i=1}^{10} a_i \gamma_1^i s(n-i) - \sum_{i=1}^{10} a_i \gamma_2^i s(n-i) \qquad n = 0,...,39 \tag{3.34}$$

3.2.1.2 การหาค่าพิตช์อย่างหยาบ (Open loop pitch analysis)

เพื่อที่จะลดความซับซ้อนในการหาสำหรับดีเลย์ของ Adaptive codebook (Pitch) ดังนั้นจุดในการหาจะถูกกำหนดให้อยู่รอบ ๆ พิตช์คร่าว ๆ จากการทำ Open loop pitch analysis ซึ่งการทำขั้นตอนนี้จะทำเพียงครั้งเดียวต่อเฟรม ซึ่งจะต้องใช้หลักการของความเหมือนกัน (correlation) ของสัญญาณเป็นคาบ ๆ

$$R(k) = \sum_{i=0}^{79} sw(n)sw(n-k) \tag{3.35}$$

แบ่งช่วงออกเป็น 3 ช่วง ดังนี้

$i = 1:$ 80,...,143
$i = 2:$ 40,...,79
$i = 3:$ 20,...,39

โดย R′ (k) คือ R(k) ที่ถูกทำให้เป็นหนึ่งหน่วยตามสมการ

$$R'(t_i) = \frac{R(t_i)}{\sqrt{\sum_n sw^2(n-t_i)}} \qquad i = 1,...,3 \tag{3.36}$$

โดยเลือกค่าที่ทำให้ R′ (k) มีค่ามากที่สุด ช่วงละ 1 ตัว ดังนั้น จะมีตัวที่ถูกเลือกขึ้นมา 3 ตัว สำหรับการเลือกนั้นอัลกอริทึม จะมีแนวโน้มจะเลือกค่าที่มีการดีเลย์น้อย โดยวิธีการเลือกมีดังต่อไปนี้

$$\begin{aligned}
&T_{op} = t_1 \\
&R'(T_{op}) = R'(t_1) \\
&if \quad R'(t_2) \ge 0.85R'(T_{op}) \\
&\qquad R'(T_{op}) = R'(t_2) \\
&\qquad T_{op} = t_2 \\
&end \\
&if \quad R'(t_3) \ge 0.85R'(T_{op}) \\
&\qquad R'(T_{op}) = R'(t_3) \\
&\qquad T_{op} = t_3 \\
&end
\end{aligned} \tag{3.37}$$

3.2.1.3 การคำนวณหาผลการตอบสนองเชิงอิมพัลซ์ (Computation of the impulse response)

การตอบสนองอิมพัลส์ h (n) ของการถ่วงน้ำหนักวงจรสังเคราะห์ถูกใช้ในการหา Adaptive codebook และ Fix codebook h (n) ถูกคำนวณไว้สำหรับแต่ละเฟรมย่อย โดยการกรองสัญญาณซึ่งประกอบไปด้วยสัมประสิทธิ์ของวงจรกรอง A (z/γ_1) และ ทำการวงจรกรอง ต่อโดยใช้ 1/ $\hat{A}(z)$ (z) และ 1/A (z/γ_2)

3.2.1.4 การคำนวณหาสัญญาณกระตุ้น (Computation of the target signal)

Target signal x (n) ใช้สำหรับหา Adaptive codebook สามารถคำนวณได้จากนำ LP residual signal (r (n)) ลบออกด้วยความผิดพลาดของสัญญาณเสียงจริงกับสัญญาณเสียงที่สังเคราะห์ขึ้น และกรอง LP residual signal r(n) ผ่านการความเป็นไปได้ ระหว่างวงจรสังเคราะห์ 1/ $\hat{A}(z)$ (z) และวงจรกรองถ่วงน้ำหนัก A(z/γ_1) / A(z/γ_2) จะได้สัญญาณกระตุ้นที่ผ่านมา (y(n)) x(n) และ y(n) ถูกเลื่อนออกไปด้วยดีเลย์ k แล้วหาความเหมือนกันเป็นคาบ เพื่อหา พิตช์ที่ละเอียดขึ้น

3.2.1.5 การค้นหาแค็บทีพโคดบุ๊ก (Adaptive codebook search)

พารามิเตอร์ของ Adaptive codebook ประกอบด้วย พิตช์และอัตราขยายสัญญาณโดย adaptive codebook นั้นถูกสร้างขึ้นโดยใช้สัญญาณกระตุ้นของวงจรกรองของพิตช์ (pitch filter) โดยการหา Adaptive codebook จะกระทำทุก 5 mS หรือทุก ๆ เฟรมย่อย ในเฟรมย่อยแรกนั้นพิตช์ T1 ถ้าประมาณแล้วมีค่าต่ำกว่า 85 จะต้องถูกหาให้ละเอียดขึ้น โดยการประมาณค่าเศษส่วนซึ่งมีความละเอียด 1/3 ([19 1/3, 84 2/3]) และกรณีที่มากกว่า 85 จะประมาณเป็นจำนวนเต็มเท่านั้น ([85,143]) สำหรับเฟรมย่อย ที่ 2 ดีเลย์ T2 จะถูกใช้ร่วมกับการประมาณเศษส่วนด้วยความละเอียด 1/3 และอยู่ในช่วงของ ([int(T1) – 5 2/3, int(T1) + 4 2/3])

สำหรับพิตช์ของแต่ละเฟรมย่อยแรกใช้ Closed loop analysis เพื่อลดรากที่สองค่าเฉลี่ยของความผิดพลาดที่ถ่วงน้ำหนักพบว่าจะมีค่าอยู่รอบ Open loop pitch ซึ่งจะไม่เกิน 6 ตัวอย่าง ดังนั้นจะช่วงลดการหาพิตช์ลงไปได้โดยกำหนดช่วงของการหาได้โดยอยู่ระหว่าง tmin และ tmax

$$
\begin{aligned}
& t_{\min} = T_{op} - 3 \\
& \textit{if} \quad t_{\min} < 20 \quad \textit{then} \quad t_{\min} = 20 \\
& t_{\max} = t_{\min} + 6 \\
& \textit{if} \quad t_{\max} > 143 \quad \textit{then} \\
& \qquad t_{\max} = 143 \\
& \qquad t_{\min} = t_{\max} - 6 \\
& \textit{end}
\end{aligned}
\tag{3.38}
$$

สำหรับเฟรมย่อยที่ 2 ค่าของพิตช์จะอยู่รอบ ๆ ของพิตช์ที่ถูกเลือกโดย T1 ขอบเขตของการค้นหาจะอยู่ในช่วงของ tmin – 2/3 และ tmax + 2/3 โดย tmin และ tmax

$$
\begin{aligned}
&t_{\min} = \mathrm{int}(T_1) - 5 \\
&if \quad t_{\min} < 20 \quad then \quad t_{\min} = 20 \\
&t_{\max} = t_{\min} + 9 \\
&if \quad t_{\max} > 143 \quad then \\
&\qquad t_{\max} = 143 \\
&\qquad t_{\min} = t_{\max} - 9 \\
&end
\end{aligned}
\tag{3.39}
$$

สำหรับการหา Close loop pitch จะต้องลดรากที่สองค่าเฉลี่ยของความผิดพลาดที่ถ่วงน้ำหนักระหว่าง x (n) กับ y (n) ซึ่ง ซึ่งต้องพยายามทำให้สมการที่ 3.40 มีค่ามากที่สุด

$$
R(k) = \sum_{i=0}^{39} x(n) y_k (n-k) / \sqrt{\sum_{i=0}^{39} y_k (n) y_k (n)} \tag{3.40}
$$

โดย y (n) ถูกเลื่อนไปเรื่อย ๆ โดย k อยู่ในช่วงของ tmin + 1 ถึง tmax ซึ่งค่าที่ได้จะเป็นพิตช์ที่ละเอียดขึ้น

โดย u (n) คือบัฟเฟอร์ของสัญญาณกระตุ้น และ y_{k-1} (-1) = 0 ซึ่งในขั้นตอนในการหานั้น แต่ละตัวอย่างของ u (n), n = 0...39 นั้นยังไม่รู้ และยังต้องการให้พิตช์ดีเลย์น้อยกว่า 40 และเพื่อทำให้การค้นหาง่ายขึ้น LP residual จะถูกกำหนดค่าให้กับ u (n) จึงทำให้เกิดความสัมพันธ์ ดังต่อไปนี้

$$
y_k (n) = y_{k-1} (n-1) + u(-k) h(n) \qquad n = 39,...,0 \tag{3.41}
$$

สำหรับการหาค่า T2 และ T1 ถ้าพิตช์ที่หาได้มีค่าน้อยกว่า 85 จะต้องทำการหาต่อด้วยความละเอียด 1/3 รอบ ๆ พิตช์ที่หามาได้จะถูกทดสอบ หาค่ามากที่สุด โดยการประมาณค่าโดยใช้วงจรกรองชนิด FIR b12 มีพื้นฐานจากฟังก์ชัน ฟังก์ชันแฮมมิ่งวินโดว์ซิงก์ (Hamming windowed sinc) ด้วย ปิด sinc +11 หรือ –11 และ pad ด้วย zeros ที่ + 12 หรือ –12 (b_{12} (12) = 0) โดยวงจรกรอง มีความถี่ cut off ที่ –3 dB ที่ 3600 Hz และประมาณค่า R (k) สำหรับ fraction –2/3, -1/3, 0, 1/3, 2/3

$$
R(k)_t = \sum_{i=0}^{3} R(k-i) b_{12} (t+3i) + \sum_{i=0}^{3} R(k+1+i) b_{12} (3-t+3i) \qquad t = 0,1,2 \tag{3.42}
$$

โดยที่ T = 0, 1, 2 เกี่ยวข้องกับ fraction 0, 1/3, 2/3 ตามลำดับ

3.2.1.6 การสร้างอแด็บทีพโคดบุ๊คเวกเตอร์ (Generation of the adaptive-codebook vector)

เมื่อหาพิทช์ดีเลย์ได้แล้ว เวกเตอร์ v (n) จะถูกคำนวณ โดยแท้จริงแล้ว v (n) คือ u (n) ที่มีค่าพิตช์เป็นดีเลย์ k fraction t เท่านั้น

$$v(n)=\sum_{i=0}^{9}u(n-k+i)b_{30}(t+3i)+\sum_{i=0}^{9}u(n-k+1+i)b_{30}(3-t+3i) \qquad n=0,...,39 \quad t=0,1,2 \tag{3.43}$$

โดยการประมาณค่าใช้วงจรกรอง b 30 มีพื้นฐานมาจากฟังก์ชัน ฟังก์ชันแฮมมิ่งวินโดว์ซิงก์ (Hamming windowed sinc) ปิดเศษที่ +29 หรือ –29 และเติมด้วยศูนย์ที่ + 30 หรือ –30 (b_{30} (30) = 0) โดยมีความถี่ cut off = -3 dB ที่ 3600 Hz

3.2.1.7 การคำนวณหาดีเลย์ของอแด็บทีพโคดบุ๊ก (Codeword computation for adaptive-codebook delays)

พิทช์ T1 นั้นเข้ารหัส 8 บิทในเฟรมย่อยแรก และพิตช์ของเฟรมย่อยที่ 2 ซึ่งสัมพัทธ์กับเฟรมย่อยแรกจะเข้ารหัสด้วย 5 บิทดีเลย์ที่เป็นเศษส่วน T ถูกแสดงโดยตัวที่เป็นจำนวนเต็มของ T int (T) และส่วนที่เป็นเศษส่วน frac/3, frac = -1,0,1 พิตช์อินเด็ก P1 จะถูกเข้ารหัส ดังนี้

$$P1=\begin{cases}3(\text{int}(T_1)-19)+frac-1 & if \quad T_1=[19,...,85],\ frac=[-1,0,1]\\ (\text{int}(T_1)-85)+197 & if \quad T_1=[86,...,143],\ frac=0\end{cases} \tag{3.44}$$

สำหรับ T2 ก็เช่นเดียวกับ T1 แต่สัมพัทธ์กับ T1

$$P2=3(\text{int}(T_2)-t_{\min})+frac+2 \tag{3.45}$$

เพื่อความทนของตัวเข้ารหัส ในการได้รับการสุ่มความผิดพลาดต่าง ๆ จึงมีการเพิ่ม พาริตี้บิท P0 เข้าไปโดยคำนวณจาก P1 พาริตี้บิตจะถูกสร้างโดย XOR โอเปอร์เรชันโดยใช้ 6 บิต สูงของ P1 ที่ตัวถอดรหัสพาริตี้บิทจะถูกคำนวณกลับ และถ้าไม่ตรงกับที่ส่งมา ตัวถอดรหัสจะทำการกลบเกลื่อนความผิดพลาด (error concealment)

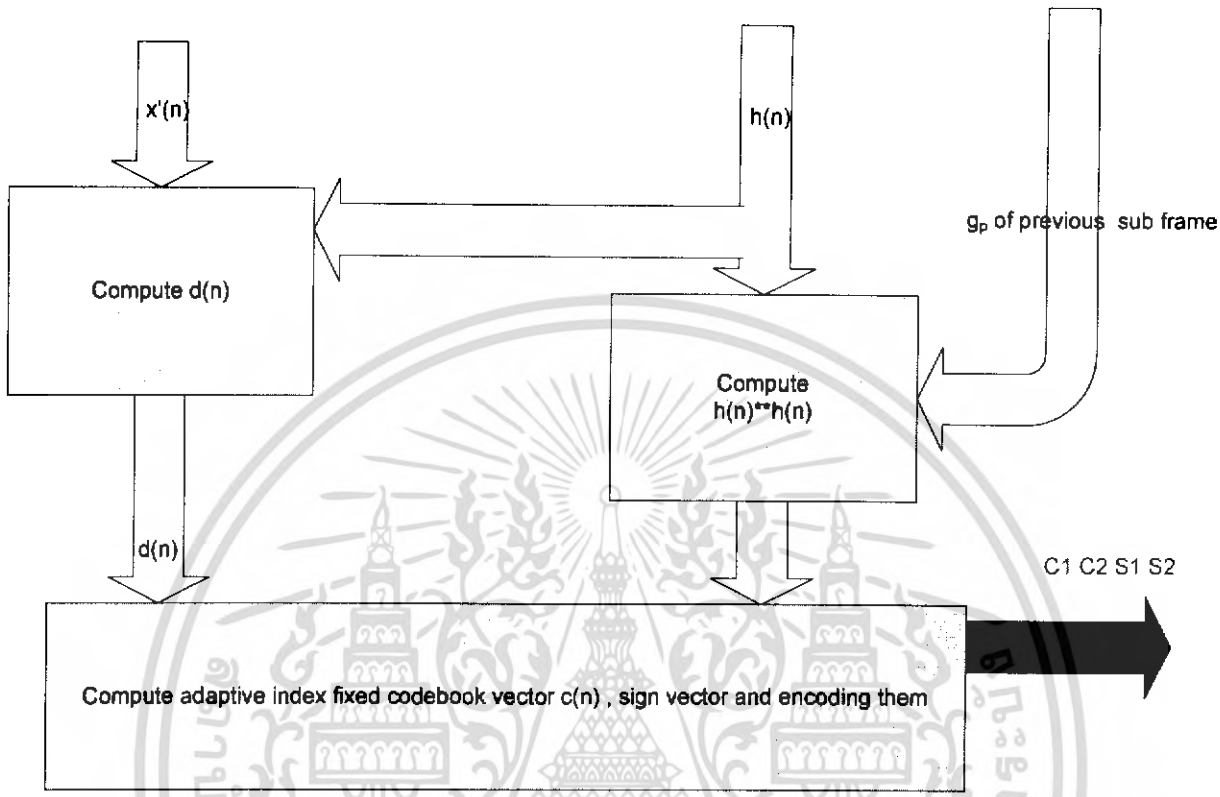
3.2.1.8 การคำนวณหาอัตราขยายของอแด็บทีพโคดบุ๊ก (Computation of the adaptive-codebook gain)

เมื่อดีเลย์ของ Adaptive codebook ถูกคำนวณแล้ว จะต้องมีการหาอัตราขยายสัญญาณ adaptive-codebook

$$\begin{aligned}g_p &= \sum_{n=0}^{39}x(n)y(n)/\sum_{n=0}^{39}y(n)y(n)\\ y(n) &= \sum_{i=0}^{n}v(i)h(n-i) \qquad\qquad n=0,...,39\end{aligned} \tag{3.46}$$

ขณะที่ x (n) เป็น target signal y (n) เป็น adaptive codebook เวกเตอร์ ที่ถูกกรอง ซึ่งจะได้มาจากเป็นการคอนโวลูชันระหว่าง v (n) กับ h (n)

3.2.2 การหาองค์ประกอบสัญญาณเสียงสุ่ม (Unvoiced sound)



รูปที่ 3-4 แสดงการหาองค์ประกอบสัญญาณเสียงสุ่ม (Unvoiced sound)

3.2.2.1 ฟิกซ์โคดบุ๊ค (Fix codebook)

Fix codebook มีพื้นฐานมาจาก algebraic codebook โดยใช้ ISPP ซึ่ง codebook แต่ละ codebook เวกเตอร์ จะประกอบด้วยพัลส์ ที่ไม่ใช่ 0 4 พัลส์ ซึ่งแต่ละพัลส์ จะมีทั้งแอมพลิจูดที่เป็น +1 หรือ –1

Pulse	Sign	Positions
i0	***s0:* ± 1**	***m*0: 0, 5, 10, 15, 20, 25, 30, 35**
i1	***s1:* ± 1**	***m*1: 1, 6, 11, 16, 21, 26, 31, 36**
i2	***s2:* ± 1**	***m*2: 2, 7, 12, 17, 22, 27, 32, 37**
i3	***s3:* ± 1**	***m*3: 3, 8, 13, 18, 23, 28, 33, 38 4, 9, 14, 19, 24, 29, 34, 39**

ตารางที่ 3-1 แสดงโครงสร้างของ fixed codebook

Codebook เวกเตอร์ c (n) ถูกสร้างขึ้นโดยเป็นเวกเตอร์ 0 ทั้งหมด 40 และมี 4 พัลส์หนึ่งหน่วย (unit pulse) ที่ตำแหน่งที่เจอ และคูณด้วยเครื่องหมายที่เกี่ยวข้องในตาราง

$$c(n) = s_0\delta(n-p_0) + s_1\delta(n-p_1) + s_2\delta(n-p_2) + s_3\delta(n-p_3) \qquad n = 0,...,39 \tag{3.47}$$

เมื่อ $\delta(n)$ แทนฟังก์ชันอิมพัลส์หนึ่งหน่วย (unit impulse function)

s แทนเครื่องหมายแสดงค่า +/- ของแอมพลิจูด

Codebook ที่ถูกเลือกมาจะต้องผ่าน adaptive pre-filter P (z) ซึ่งจะเพิ่ม harmonic คอมโพเนนท์และพัฒนาคุณภาพของสัญญาณเสียง

$$P(z) = 1/(1-\beta z^{-T}) \tag{3.48}$$

เมื่อ T คือเวลาที่เป็นจำนวนเต็มของพิตช์ในเฟรมย่อยนั้น ๆ และ β คือ อัตราขยายสัญญาณพิตช์ค่าของ β หาได้จาก นำอัตราขยายสัญญาณของ adaptive codebook ที่ถูกควอนไทต์ของเฟรมย่อยก่อนหน้า

$$\beta = \hat{g}_p^{(m-1)} \qquad bounded \quad by \quad 0.2 \le \beta \le 0.8 \tag{3.49}$$

สำหรับพิตช์ที่น้อยกว่า 40 Codebook c (n) จะถูกแก้ไข ดังต่อไปนี้

$$c(n) = \begin{cases} c(n) & n = 0,...,T-1 \\ c(n) + \beta c(n-T) & n = T,...,39 \end{cases} \tag{3.50}$$

เช่นเดียวกันกับ h (n) ที่ต้องถูกแก้ไข เช่นเดียวกัน

$$h(n) = \begin{cases} h(n) & n = 0,...,T-1 \\ h(n) + \beta h(n-T) & n = T,...,39 \end{cases} \quad (3.51)$$

3.2.2.2 การค้นหาฟิกซ์โคดบุ๊ก (Fixed codebook search procedure)

Fixed codebook ถูกค้นหาโดย ลด MSE ระหว่าง ความเข้มของสัญญาณเสียงที่เข้ามาตาม x′ (n) กับ สัญญาณที่เราจะสร้างขึ้นมาคือ c (n)

$$x'(n) = x(n) - g_p y(n) \qquad n = 0,...,39 \quad (3.52)$$

เมื่อ y (n) เป็น adaptive codebook เวกเตอร์ ที่ถูก และ g_p เป็นอัตราขยายสัญญาณ adaptive codebook Matrix H ถูกนิยามเป็น lower triangular Toepliz convolution เมทริกซ์ กับ diagonal h (0) และ lowers diagonal h (1)... h (39) เมทริกซ์ $\phi = H^t H$ ประกอบด้วยคอร์รีเลชันของ h (n) และอีลีเมนต์ของเมทริกซ์ที่สมมาตร

$$\phi(i,j) = \sum_{n=j}^{39} h(n-i)h(n-j) \qquad i = 0,...,39 \qquad j = i,..39 \quad (3.53)$$

สัญญาณคอร์รีเลชัน (Correlation signal d (n)) ได้มาจาก x′(n) และการตอบสนองอิมพัลส์ (Impulse response)h (n)

$$d(n) = \sum_{i=n}^{39} x'(n).h(i-n) \qquad , n = 0,...,39 \quad (3.54)$$

ถ้า c_k เป็นพจน์ที่ k ของ Fix – codebook เวกเตอร์ แล้ว codebook จะถูกหาโดยต้องทำให้correlation ระหว่าง d (n) และ c (n) สมการที่ (3.55) มีค่ามากที่สุด เมื่อ t คือสัญลักษณ์ของการทรานส์โพส

$$\frac{C_k^2}{E_k} = \frac{\left(\sum_{n=0}^{39} d(n)c_k(n)\right)^2}{c_k' \phi c_k} \quad (3.55)$$

Algebraic ของ codebook ทำให้การค้นหารวดเร็วเพราะว่า codebook เวกเตอร์ c_k ประกอบด้วยเฉพาะพัลส์ ที่ไม่ใช่ 0 จำนวน 4 พัลส์ จึงทำให้คอร์รีเลชันในที่เป็นตัวเศษของสมการลดรูปลงเหลือเพียง

$$C = \sum_{i=0}^{3} s_i d(m_i) \quad (3.56)$$

โดย m แทนตำแหน่งของพัลส์และ s แทนแอมพลีจูดของพัลส์ และ พลังงาน ที่เป็นตัวหารนั้น จะกลายเป็น

$$E = \sum_{i=0}^{3} \phi(m_i, m_i) + 2\sum_{i=0}^{2} \sum_{j=i+1}^{3} s_i s_j \phi(m_i, m_j) \tag{3.57}$$

เพื่อทำให้การค้นหาง่ายลงแอมพลีจูดของพัลส์ จะถูกคาดเดาก่อนโดยการทำควอนไทซ์ สัญญาณ d (n) ซึ่งทำได้โดยกำหนดแอมพลีจูดของพัลส์ ในตำแหน่งที่แน่นอนที่เท่ากับเครื่องหมายของ d(n) ที่ตำแหน่งนั้น ๆ ก่อนทำการหา codebook ขั้นตอนแรกจะต้องสร้างกลับ (decompose d(n)) ออกเป็น 2 ส่วน โดยแบ่งเป็น ขนาดของ |d(n)| และเครื่องหมาย ขั้นตอนที่ 2 ทำการแก้เมทริกซ์ ϕ โดยการรวมเครื่องหมายด้วย

$$\phi'(i,j) = sign[d(i)]sign[d(j)]\phi(i,j) \qquad i = 0,..39 \qquad j = i+1,...,39 \tag{3.58}$$

และทแยงมุมหลัก (Main diagonal) ใน ϕ จะถูกสเกลโดยเอาอัตราส่วน ของ 2 ออก

$$\phi'(i,j) = 0.5\phi'(i,i) \qquad i = 0,...,39 \tag{3.59}$$

3.2.2.3 การเข้ารหัสฟิกซ์โคดบุ๊ก (Codeword computation of the fixed codebook)

ตำแหน่งของพัลส์ i_0 i_1 i_2 จะเข้ารหัส โดยใช้ 3บิต ในขณะที่ i_3 นั้นจะเข้ารหัส โดยใช้ 4บิต แต่ละพัลส์ แอมพลีจูดนั้นจะใช้ 1บิต ในการเข้ารหัส ทำให้ทั้งหมดมี 17บิต สำหรับ 4 พัลส์ โดย s = 1 ถ้าเครื่องหมายเป็น + และ s = 0 ถ้าเครื่องหมายเป็น – sign codeword

$$S = s_0 + 2s_1 + 4s_2 + 8s_3 \tag{3.60}$$

และ Fixed codebook ดังนี้

$$C = (m_0/5) + 8(m_1/5) + 64(m_2/5) + 512(2(m_3/5) + jx) \tag{3.61}$$

โดยที่ jx = 0 ถ้า m3 = 3,8,…,38 และ jx = 1 ถ้า m3 = 4,9,…,39

3.2.2.4 การทำควอนไทต์เซรันของอัตราขยาย (Quantization of the gain)

อัตราขยายสัญญาณ Adaptive codebook (pitch gain) และ fix codebook เป็น เวกเตอร์ที่ถูกควอนไทต์ โดยใช้ 7-bit ในการหา codebook ของ อัตราขยายสัญญาณจะต้องทำให้ MSE ระหว่างสัญญาณเสียงเดิมกับสัญญาณเสียงที่สร้างขึ้นมาใหม่ให้น้อยที่สุด สมการ

$$E = x'x + g_p^2 y'y + g_c^2 z'z - 2g_p x'y - 2g_c x'z + 2g_p g_c y'z \tag{3.62}$$

โดยที่ x เป็นเวกเตอร์ target y เป็นวงจรกรอง adaptive codebook เวกเตอร์ และ z เป็น fixed codebook เวกเตอร์ ที่ทำการคอนโวลฟ์ (convolved) กับ h (n)

$$z(n) = \sum_{i=0}^{n} c(i)h(n-i) \qquad n = 0,...,39 \tag{3.63}$$

3.2.2.5 การพยากรณ์อัตราขยาย (Gain prediction)

อัตราขยายสัญญาณ Fixed codebook g_c สามารถหาได้จาก

$$g_c = \gamma . g_c' \tag{3.64}$$

โดย g'_c คืออัตราขยายสัญญาณที่ได้จากการพยากรณ์โดยใช้พลังงานของ fixed codebook ของเฟรม ก่อนหน้า และ γ คืออัตราส่วนของความถูกต้อง

พลังงานเฉลี่ยของ Fixed codebook contribution นั้น กำหนดโดยสมการ

$$E = 10.\log\left(\frac{1}{40}\sum_{n=0}^{39} c(n)^2\right) \tag{3.65}$$

หลังจากทำ Scale ของ c (n) ด้วยอัตราขยายสัญญาณ fixed codebook g_c พลังงานของสเกล fixed codebook คือ 20 log g_c + E สมมติให้ E_m เป็น Mean removed energy (dB) ของ Scaled fixed codebook contribution ที่เฟรมย่อยที่ m จะได้ดังสมการ ดังนี้

$$E^{(m)} = 20\log g_c + E - \overline{E} \tag{3.66}$$

โดยที่ E_{av} = 30 dB ซึ่งเป็นค่าเฉลี่ยพลังงานของสัญญาณกระตุ้น Fixed codebook โดย g_c สามารถเขียนอยู่ในรูปฟังก์ชัน ของพลังงานได้ ดังนี้

$$g_c = 10^{(E_m + \bar{E} - E)/20} \tag{3.67}$$

อัตราขยายสัญญาณที่ใช้สำหรับการพยากรณ์ g_c (predict gain) ถูกคาดการณ์ได้จาก log ของพลังงานของ Fixed codebook contribution ก่อนหน้า โดยใช้ 4th Order MA Prediction

$$\tilde{E}^{(m)} = \sum_{i=1}^{4} b_i \hat{U}^{(m-i)} \tag{3.68}$$

โดย [b1 b2 b3 b4] = [0.68 0.58 0.34 0.19] คือสัมประสิทธิ์ MA และ $\tilde{E}^{(m)}$ คือควอนไทต์ของความผิดพลาดจากการพยากรณ์ (prediction error) ที่เฟรมย่อยที่ m

$$U^{(m)} = E^{(m)} - \hat{E}^{(m)} \tag{3.69}$$

3.2.2.6 การค้นหาอัตราขยายที่ควอนไทต์ (Codebook search for gain quantization)

อัตราขยายสัญญาณ Adaptive codebook g_p และอัตราส่วนของ γ เป็นเวกเตอร์ ที่ถูกควอนไทต์โดยใช้ two stage conjugate structured codebook โดยสถานะแรกประกอบด้วย 3บิต 2 เวกเตอร์ codebook GA และสถานะที่สองประกอบด้วย 4 บิต 2 เวกเตอร์ codebook GB โดยเอลิเมนต์แรกของแต่ละ codebook นั้นแทน quantized อัตราขยายสัญญาณ adaptive codebook *gp* แล้วเอลิเมนต์ที่สองแทน quantized fixed codebook gain correction factor $\hat{\gamma}$ กำหนดให้งอินเด็กของ codebook GA และ GB สำหรับ **GA** และ **GB** อัตราขยายสัญญาณ adaptive codebook ที่ถูกควอนไทต์

$$\begin{aligned} &\hat{g}_p = GA_1(GA) + GB_1(GB) \\ &and \quad the \quad quantized \quad fixed - codebook \quad gain \quad by: \\ &\hat{g}_c = g_c' \hat{\gamma} = (GA_2(GA) + GB_2(GB)) \end{aligned} \tag{3.70}$$

3.2.2.7 การเข้ารหัสอัตราขยายที่ถูกควอนไทต์แล้ว (Codeword computation for gain quantizer)

Codeword GA และ GB สำหรับควอนไทต์เซอร์ของอัตราขยายสัญญาณ จะได้รับโดยอินเด็ก ที่เกี่ยวข้อง เพื่อที่จะช่วยลดผลกระทบความผิดพลาดแบบบิตเดียว (single bit error) ในอินเด็กของ codebook ที่ทำการ map ไป

3.2.2.8 การปรับปรุงข้อมูลในหน่วยความจำ (Memory Update)

การอัพเดตสถานะเพื่อทำการสังเคราะห์และถ่วงน้ำหนักที่ต้องการคำนวณเพื่อหา target signal ในเฟรมย่อยถัดไป หลังจากที่อัตราขยายสัญญาณทั้ง 2 ถูกควอนไทต์แล้ว u(n) ในเฟรมย่อยปัจจุบันจะหาได้จาก

$$u(n) = \hat{g}_p v(n) + \hat{g}_c c(n) \qquad n = 0,...,39 \tag{3.71}$$

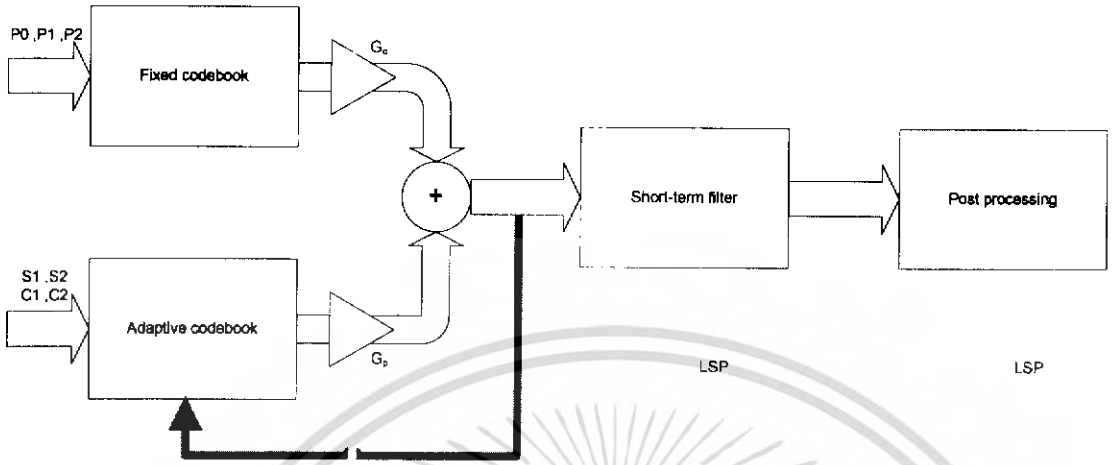
สถานะของวงจรกรองจะต้องถูก update โดยการกรอง r(n) – u(n) หรือความแตกต่างระหว่าง residual กับสัญญาณกระตุ้นผ่านวงจรกรอง $1/\hat{A}(z)$ และ A (z/γ_1)*A (z/γ_2) สำหรับ 40 ตัวอย่าง เฟรมย่อย และทำการเก็บสถานะของวงจรกรองไว้ โดย สร้างสัญญาณขึ้นมาใหม่ s(hat)(n) ซึ่งจะถูกคำนวณโดยการกรองสัญญาณกระตุ้นผ่าน $1/\hat{A}(z)$ เอาท์พุท ของวงจรกรองขึ้นอยู่กับ r(n) – u(n) ซึ่งจะเท่ากับ e(n) = s(n) – $\hat{s}$(n) ดังนั้นสถานะของวงจรสังเคราะห์ $1/\hat{A}(z)$ จะถูกให้โดย e(n) โดยที่ n = 30,...,39 อัพเดตสถานะ ของวงจรกรอง A (z/γ_1)*A (z/γ_2) โดยการกรองสัญญาณความผิดพลาดผ่านวงจรกรองเพื่อนำไปหา ew(n)

$$ew(n) = x(n) - \hat{g}_p y(n) - \hat{g}_c z(n) \tag{3.72}$$

Parameter	codeword	Subframe 1 (bits)	Subframe 2 (bits)	Total bits per frame
Line spectrum pairs	*L0, L1, L2, L3*			18
Adaptive-codebook delay	*P1, P2*	8	5	13
Pitch-delay parity	*P0*	1		1
Fixed-codebook index	*C1, C2*	13	13	26
Fixed-codebook sign	*S1, S2*	4	4	8
Codebook gains (stage 1)	*GA1, GA2*	3	3	6
Codebook gains (stage 2)	*GB1, GB2*	4	4	8
Total				80

ตารางที่ 3-2 แสดงบิตฟิลด์ที่ส่งให้ฝั่งถอดรหัส

3.3 การถอดรหัส



รูปที่ 3-5 แสดงโฟลโปรแกรมของการถอดรหัส

การสังเคราะห์สัญญาณเสียงกลับนั้นวิธีการได้อยู่ในฝั่งเข้ารหัส อยู่แล้วเมื่อฝั่งถอดรหัส ได้รับพารามิเตอร์แล้วสังเคราะห์สัญญาณเสียงออกมา สัญญาณเสียงที่ได้จะถูกปรับปรุงคุณภาพโดยอาศัย Post filter

3.3.1 การปรับปรุงคุณภาพของสัญญาณ (Post Processing)

เมื่อตัวถอดรหัส ได้รับพารามิเตอร์แล้วสร้างสัญญาณเสียงขึ้นมา สัญญาณเสียงที่สร้างขึ้นจะถูกนำมาปรับปรุงคุณภาพโดยใช้ Post Processing ซึ่งมีองค์ประกอบด้วยกัน 4 ส่วนคือ

1. Long term post filter แทนโดยสัญลักษณ์ Hp(z)
2. Short term post filter แทนโดยสัญลักษณ์ Hf(z)
3. Tilt compensation filter แทนโดยสัญลักษณ์ Ht(z)
4. Gain compensation factor แทนโดยสัญลักษณ์ g

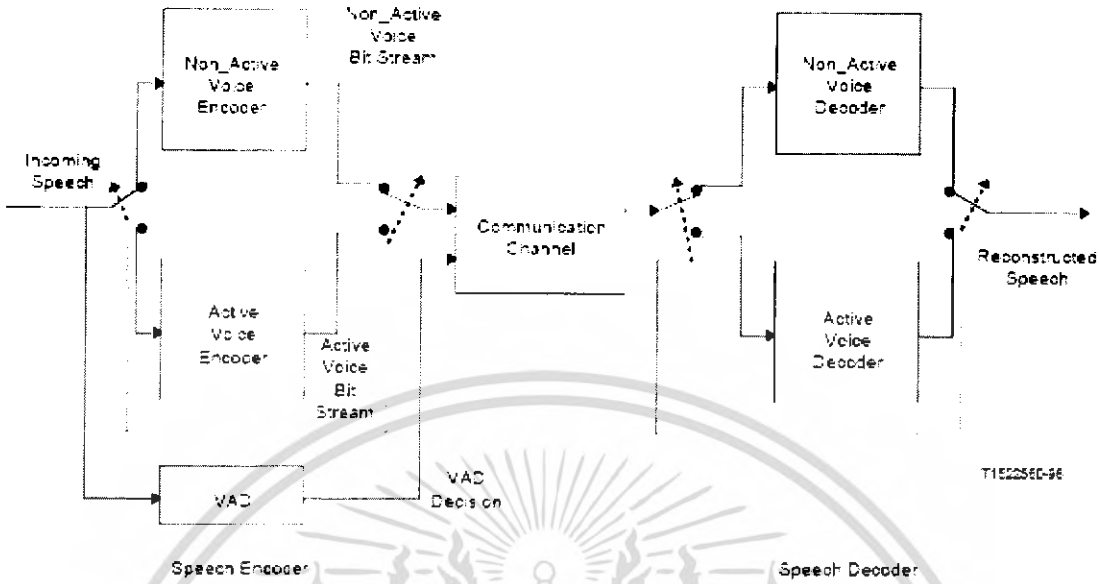
Long term post filter สร้างจากถอดรหัสหาอัตราขยายของสัญญาณและพารามิเตอร์ของพิทช์ดีเลย์เป็นวงจรกรองฟังก์ชัน 1st order all-zero ที่ช่วยเน้นช่วงความถี่รอบ ๆ คาบของพิตช์

Short term post filter ออกแบบมาให้ช่วยเน้น formants ซึ่งเป็นช่วงความถี่ของพลังงานในวงจรสังเคราะห์

Tilt คือความชัน (slope) ของพลังงานใน โดเมนเชิงความถี่ ในสัญญาณเสียงโดยวงจรกรองแบบ Tilt compensation ออกแบบมาเพื่อช่วยลดความผิดเพี้ยนที่เกิดจากการทำ Short term postfilter

Gain compensation factor เป็นผลหารของพลังงานของการถอดรหัสแบบเดิมหารด้วยพลังงานหลังผ่านส่วนของ post filter แล้ว เป็นตัวที่ใช้เก็บความเพี้ยนของสัญญาณเสียง

3.4 การตรวจสอบว่ามีการสนทนาหรือไม่ (Voiced Activity Detection (VAD))



รูปที่ 3-6 แสดง Speech communication system with VAD

G.729 Annex B เป็นส่วนเพิ่มเติมจาก G.729 สำหรับการเข้ารหัสสัญญาณเสียงเฟรม ที่ไม่ใช่เสียงพูด โดยการแทนรหัสสัญญาณในเฟรม นั้นเป็นสัญญาณสัญญาณรบกวน โดยปกติสัญญาณเสียงจะประกอบด้วยช่วงที่มีเสียงพูด (talk spurts) และตามด้วยช่วงที่ไม่มีเสียงพูด (silence period) ส่วนที่เพิ่มเติมเข้าไปสำหรับ Annex B นี้คือ การแยกได้ว่าช่วงใดที่มีเสียงพูดและช่วงใดที่ไม่มีเสียงพูด ด้วยการเข้ารหัส G.729 เดิมนั้นเฟรมที่ไม่มีเสียงพูด G.729 นั้นจะมีการส่งสัญญาณที่เข้ารหัสแล้ว 2 แบบ คือไม่ส่งค่าพารามิเตอร์อะไรออกไปเลย หรือ ส่งค่าพลังงานของสัญญาณเสียงนั้น เพื่อที่ตัวถอดรหัสจะสามารถแทนสัญญาณที่เข้ารหัสมาด้วย white noise แต่อย่างไรก็ตาม ในสภาพแวดล้อมที่มีเสียงดังและประกอบด้วยสัญญาณรบกวนที่มีค่าสถิติไม่คงที่ (non stationary background noise) นั้น การส่งสัญญาณทั้ง 2 ทั้ง 2 แบบนั้น ไม่มีประสิทธิภาพเพียงพอ จึงต้องมี Annex B เพิ่มเติมขึ้นมาสำหรับการตัดสินใจว่าสัญญาณเฟรม ที่สนใจนั้นเป็นช่วงที่มีเสียงพูดหรือไม่นั้น ใช้วิธีการของ Voice Activity Detection (VAD) ในแต่ละเฟรม การตัดสินใจว่าเฟรม ใด เป็นเสียงพูดหรือไม่นั้น ใช้ค่าเฉลี่ยของพารามิเตอร์4 ตัวประกอบการตัดสินใจ โดยพารามิเตอร์ทั้ง 4 นั้นมีดังต่อไปนี้

1. LSF ของช่วงที่ไม่ใช่เสียงพูด
2. ค่าพลังงาน Full band ของสัญญาณเสียง ซึ่งหาได้จาก Log ของ สัมประสิทธิ์ตัวแรกของการทำออโตคอร์รีเลชันของช่วงที่ไม่ใช่เสียงพูด
3. ค่าพลังงาน Low band ของสัญญาณเสียง ซึ่งหาได้จากการนำค่าสัมประสิทธิ์ของการทำออโตคอร์รีเลชันกรองผ่านวงจรกรองชองช่วงที่ไม่ใช่เสียงพูด
4. ค่า zero crossing rate ของสัญญาณเสียง ของช่วงที่ไม่ใช่เสียงพูด

ในแต่ละเฟรม พารามิเตอร์ที่กล่าวนั้นจะถูกคำนวณและเปรียบเทียบกับค่าเฉลี่ย ผลต่างของค่าจริงและค่าเฉลี่ยนั้นจะใช้ในการตัดสินใจว่าเฟรม ใดเป็นเสียงพูดหรือไม่ใช่เสียงพูด มากกว่านั้นค่าที่คำนวณได้จะนำไป Update ค่าเฉลี่ย จะได้ค่าเฉลี่ยใหม่สำหรับใช้ในการคำนวณในเฟรมถัดไป

หลังจากนั้นสัญญาณเสียง (เสียงพูด และ ไม่ใช่เสียงพูด) จะถูกกรอง โดยใช้พารามิเตอร์ที่ได้จาก 2 เฟรม ก่อนหน้า ถ้าตัดสินใจได้ว่าเฟรม ปัจจุบันนั้นเป็นเฟรม ที่ไม่ใช่เสียงพูด ขั้นตอนถัดไปคือตัดสินใจว่า จะส่งเฟรมข้อมูลแบบ Silence Insertion Description (SID) หรือไม่ส่งข้อมูลใดเลย (a null frame) สำหรับเฟรม SID นั้นประกอบด้วยพารามิเตอร์ที่ให้ตัวถอดรหัสสามารถสร้าง comfort noise ขึ้นมาได้ โดยประกอบด้วย พลังงานของสัญญาณกระตุ้น (5 บิต) และความผิดพลาดของสัมประสิทธิ์ LSF (10 บิต) เฟรม SID จะส่งออกไปก็ต่อเมื่อพบว่าพารามิเตอร์ของ background noise มีการเปลี่ยนแปลงไปหลังจากการส่งเฟรม SID ครั้งที่แล้ว โดยพิจารณาโดยตัวเข้ารหัส จากสัมประสิทธิ์ของวงจรกรองและพลังงานที่เปลี่ยนแปลงเกินค่า thresholds

การแยกความแตกต่างระหว่างเฟรม ทั้ง 3 ประเภท speech (G.729 เดิม), SID, และ null โดยดูจาก header ของเฟรม นั้น ๆ คือขนาดของเฟรม 80, 15, 0 ตามลำดับ

บทที่ 4

การออกแบบและการสร้าง

4.1 การปรับปรุงความเร็วในการเข้ารหัส (OPTIMIZATION)

4.1.1 แนวทางในการพัฒนา

ในการปรับปรุงความเร็วของ อัลกอริทึมนั้นเราสามารถแบ่งแนวทางที่เป็นไปได้อยู่ 2 รูปแบบคือ การแก้ที่ตัวมาตรฐานของชุดเข้ารหัส แต่วิธีการนี้อาจจะมีข้อจำกัดในเรื่องของระดับของคุณภาพเสียงที่ได้ด้วยจึงต้องพิจารณาควบคู่กันไป ส่วนแนวทางการพัฒนาอีกทางเป็นการปรับปรุงโดยอาศัยความสามารถของเครื่องมือ (Tool) ที่ใช้พัฒนาคือพยายามดึงความสามารถของเครื่องมือของ ดี เอส พี บอร์ด ที่ใช้ออกมาให้มากที่สุด

4.1.2 การปรับปรุงความเร็วโดยการแก้ไขที่มาตรฐานการเข้ารหัส

ในการพัฒนาในส่วนนี้จำเป็นต้องทราบก่อนว่าชุดหรือกลุ่มของการคำนวณส่วนใดมีการใช้งานมากเป็นพิเศษ หรือใช้เวลาในการคำนวณนานที่สุดซึ่งจากการทดลองทำให้ทราบว่าเวลาส่วนใหญ่ของการเข้ารหัสอยู่ที่โมดูลของการคำนวณหา Fixed codebook

Module	เวลาเฉลี่ยต่อเฟรม (us)
d4i40_17()	1155.91
cor_h_cp()	63
ACELP_codebook()	16.51
corr_xy2()	8.94
cor_h_x()	35.99
update_bwd()	7.03
tst_bwd_dominant()	5.43
ener_DB()	23.27
vad()	72.81
MakeDec()	4.82

ตารางที่ 4-1 แสดงเวลาที่ใช้ในแต่ละโมดูล

มีผู้ที่เสนอทฤษฎีต่าง ๆ ในการที่จะลดความซับซ้อนในการคำนวณการค้นหาตำแหน่งของอินเด็ก ใน Codebook ไว้หลายวิธี หนึ่งในนั้นก็คือ Candidate Scheme for fast ACELP search ซึ่งเป็นวิธีที่อาศัย pilot ฟังก์ชัน ในการลดจำนวนบริเวณที่ใช้ในการค้นหา (search region) ลงก่อนทำการค้นหาซึ่งก็ต้องแลกกับ

คุณภาพของเสียงที่จะลดลงด้วยดังนั้นจะต้องทำการทดลองหาจำนวนพื้นที่ที่จะลดลงแลกกับคุณภาพของเสียงที่ยอมรับได้

4.1.2.1 Candidate Scheme for fast ACELP search

การหา Algebraic code excited linear prediction (ACELP) ถูกใช้ในหลาย ๆ มาตราฐานการเข้ารหัสสัญญาณเสียง เนื่องจากมีความซับซ้อนในการคำนวณต่ำกว่าวิธีอื่น ๆ และได้สัญญาณที่มีคุณภาพสูงด้วยวิธีการออพติไมซ์แบบ analysis-by-synthesis สำหรับการลดความซับซ้อนในการคำนวณของ ACELP นั้น อาศัยหลักการออกแบบฟังก์ชัน pilot เพื่อพยากรณ์ตำแหน่งของพัลซ์ที่เป็นไปได้ก่อน ทำให้ลดการคำนวณหาฟังก์ชันของคอร์รีเลชันของพัลซ์ที่ไม่จำเป็น ทำให้ตัวเข้ารหัสแบบ ACELP ซึ่งใช้สำหรับมาตราฐานของ ITU G.723.1 และ ITU G.729 ลดความซับซ้อนในการคำนวณลง 50-80 % โดยที่สัญญาณเสียงมีคุณภาพไม่ต่างจากเดิมมากนัก

1 . วิธีการ Candidate position

เพื่อที่จะสร้างสัญญาณ เสียงแบบสุ่ม ตามมาตรฐานนั้น เราจำเป็นที่จะต้องคำนวณค่า ϕ และค่า d สำหรับแต่ละ d (m) นั้น เราใช้การคูณทั้งหมด (L-m) ครั้ง และการบวกทั้งหมด (L-m-1) ครั้ง ดังนั้น ถ้าเราจะหา d ทั้งหมดนั้น เราจะต้องใช้การคูณทั้งหมด (L+1)*L/2 ครั้ง และการบวกทั้งหมด (L-1)*L/2 ครั้ง

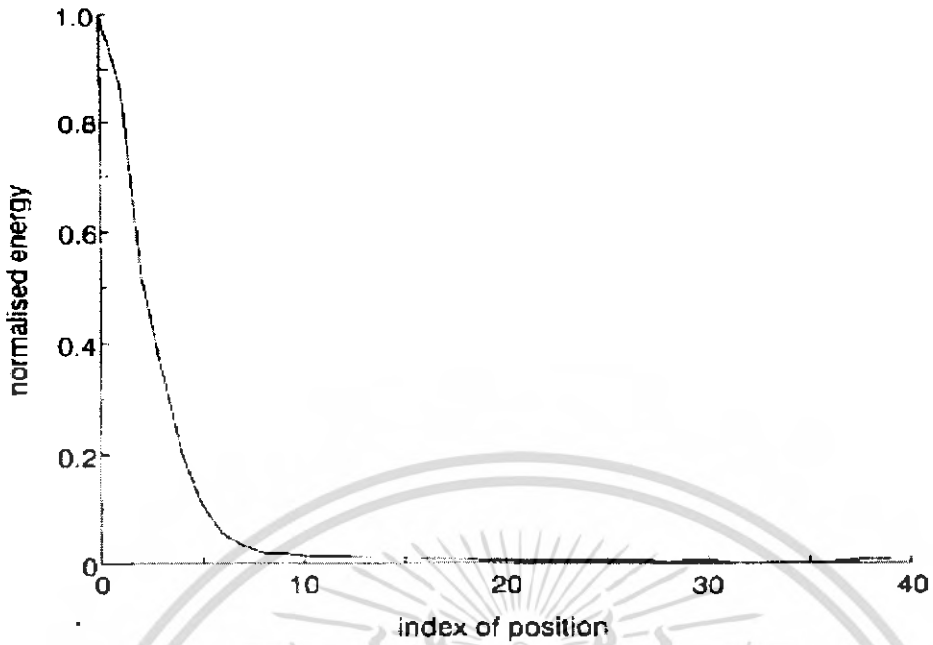
$$d(m) = \sum_{i=n}^{L-1} r[i]h[i-m], \qquad m = 0,1,...,L-1 \tag{4.1}$$

ในการหา ϕ(I,j) เราใช้การคูณทั้งหมด (L-max(I,j)) และการบวกทั้งหมด (L-max(I,j)-1) ครั้ง และสำหรับการคำนวณค่า ϕ เริ่มต้นจาก ϕ(L-1,L-1) = h(0)h(0) เราสามารถเขียนสมการแบบรีเคอร์ซีฟดังนี้

$$\phi(i,j) = \sum_{n=j}^{L-1} h[n-i]h[n-j], \qquad i = 0,1,...,L-1, \quad and \quad j = i, i+1,...,L-1 \tag{4.2}$$

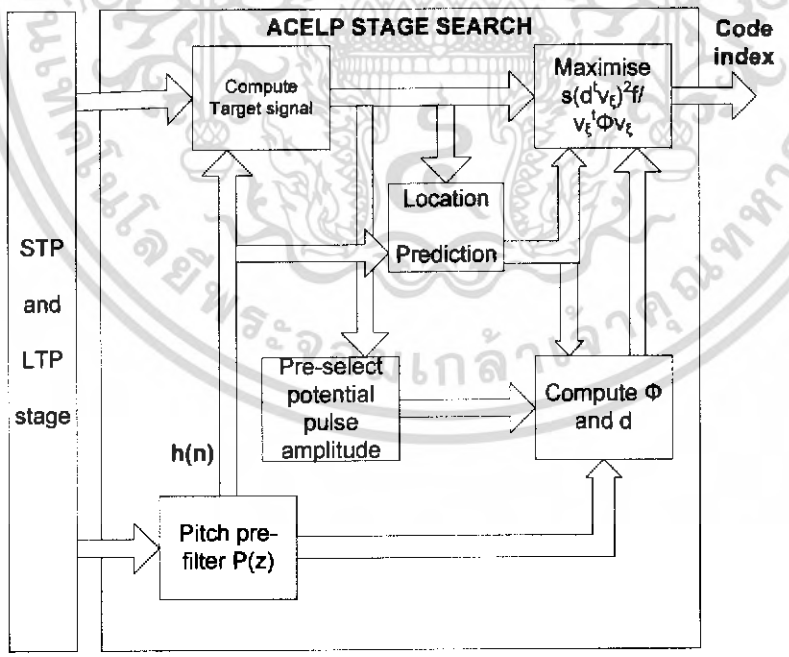
ดังนั้น เราจะต้องใช้การคูณทั้งหมด (L+1)*L/2 ครั้ง และการบวกทั้งหมด (L-1)*L/2 ครั้ง ถ้าเราสามารถประมาณว่าตำแหน่งของพัลซ์ มีโอกาสเป็นสมาชิกของเสียงแบบสุ่มเท่า ๆ กันทั้งหมด แล้วเราสามารถที่จะลดขั้นตอนการค้นหาตำแหน่งของพัลซ์ ได้ และท้ายที่สุดการคำนวณหาค่า ϕ และค่า d ก็สามารถลดลงได้ด้วยเช่นกัน

เราเสนอวิธีการของ Candidate position method ว่ามีประสิทธิภาพในการพยากรณ์ตำแหน่งที่มีโอกาสเป็นพัลซ์ ในส่วนของสัญญาณเสียงที่ไม่เป็นคาบ (unvoiced sound) ได้ เพราะว่าการวิเคราะห์โดยใช้ LPC นั้นเป็นการแทนแบบจำลองของทางเดินเสียง (vocal tract models) เพื่อที่จะเป็นมินิมัมเฟสฟิลเตอร์ (minimum phase filter) และวงจรสังเคราะห์ h[n] จะต้องมีความเสถียรภาพและมีมินิมัมเฟสแลก (minimum phase-lag) และ มินิมัมพลังงานดีเลย์พรอบเพอร์ตี (delay properties) ตามรูปที่ 4-1 แสดงเส้นโค้งของ $E[h[n]]^2$



รูปที่ 4-1 แสดงเส้นโค้งของ $E[h[n]]^2$

หลังจากใช้ค่าเฉลี่ยของ 10000 vocal tract models พบว่า มากกว่า 95 % ของพลังงานทั้งหมดตกอยู่ใน 5 พัลซ์ แรก ๆ ดังนั้นลักษณะของสัญญาณที่จะสังเคราะห์ได้นั้นจะต้องใกล้เคียงกับตำแหน่งของสัญญาณกระตุ้น และด้วยการสังเกตนี้เราจึงได้ออกแบบวิธีการเพื่อออพติไมซ์



รูปที่ 4-2 แสดง ACELP STAGE SEARCH

จากรูปที่ 4-2 เป็นการออกแบบโครงสร้างของโปรแกรมเพื่อปรับปรุงมาตรฐานการเข้ารหัสที่ใช้การหาสัญญาณกระตุ้นด้วย ACELP แบบเดิม ประยุกต์เข้ากับวิธีการใหม่ คือ Candidate Position Method

ถ้าเราสามารถคาดเดาได้ว่าพื้นที่ใดที่มีโอกาสเกิดพัลซ์ ที่เราต้องการแทนในสัญญาณกระตุ้น เราจะสามารถลดการคำนวณหา d และ ϕ ลงได้อย่างมาก เพื่อที่จะคาดการถึงตำแหน่งที่เป็นไปได้ของพัลซ์ นั้น เราจะแบ่งเฟรมย่อยของเสียงที่จะทำการวิเคราะห์จำนวน L ตัวอย่างออกเป็น K ส่วน ดังนั้นสัญญาณ target r(n) จะเป็นสมการดังนี้

$$\hat{r}(k) = \sum_{n \in R_k} \Lambda(r(n)), \qquad k = 0,1,...,K-1 \tag{4.3}$$

โดยที่ Rk นั้นแทนพื้นที่ลำดับที่ k และ Λ แทน ฟังก์ชัน condensing โดยค่า K จะน้อยกว่า L และเพื่อที่จะดึงเอาส่วนที่มีความถี่ต่ำออกมาฟังก์ชัน condensing จะต้องทำการกรองข้อมูลโดยการลดความถี่ของการสุ่มตัวอย่าง (down sampling) เพื่อที่จะคำนวณหาพื้นที่ที่โดดเด่นของ r(n) ซึ่งถ้าเราใส่ฟังก์ชัน condensing ลงไปใน vocal tract model เราจะได้สมการ ดังนี้

$$\hat{h}(k) = \sum_{n \in R_k} \Lambda(h(n)), \qquad k = 0,1,...,K-1 \tag{4.4}$$

ถ้าความละเอียดถูกลดลงโดยค่าที่มากกว่า 5 เราจะพบว่า $h(k) \approx \delta(k)$ เพราะว่าพลังงานที่มากที่สุดจะปรากฏอยู่ใน 5 พัลส์ แรก สำหรับวิธีนี้ พบว่าจำนวนพื้นที่ ที่เหมาะสม K จะประมาณได้เท่ากับ L/5 ในทางเดียวกัน ถ้าเราใส่ฟังก์ชัน condensing ลงไปกับสัญญาณกระตุ้น จะได้รับพื้นที่ที่มีความเด่นชัดมากที่สุด ซึ่งในกรณีนี้สัญญาณกระตุ้นซึ่งอยู่ในรูปของตำแหน่งของพัลซ์ นั้น จะถูกแบ่งออกเป็น K พื้นที่

	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7
m_0	0	5	10	15	20	25	30	35
m_1	1	6	11	16	21	26	31	36
m_2	2	7	12	17	22	27	32	37
m_3	3	8	13	18	23	28	33	38
	4	9	14	19	24	29	34	39

ตารางที่ 4-2 แสดงตำแหน่งพัลซ์ และพื้นที่ ของการค้นหาอินเด็ก

จากตารางแสดงให้เห็นถึงพื้นที่ที่ถูกแบ่ง (R, i = 0... 7) ในมาตรฐานของ G.729 เนื่องจาก ตัวเข้ารหัส G.729 นั้น L=40 และเลือก K=8

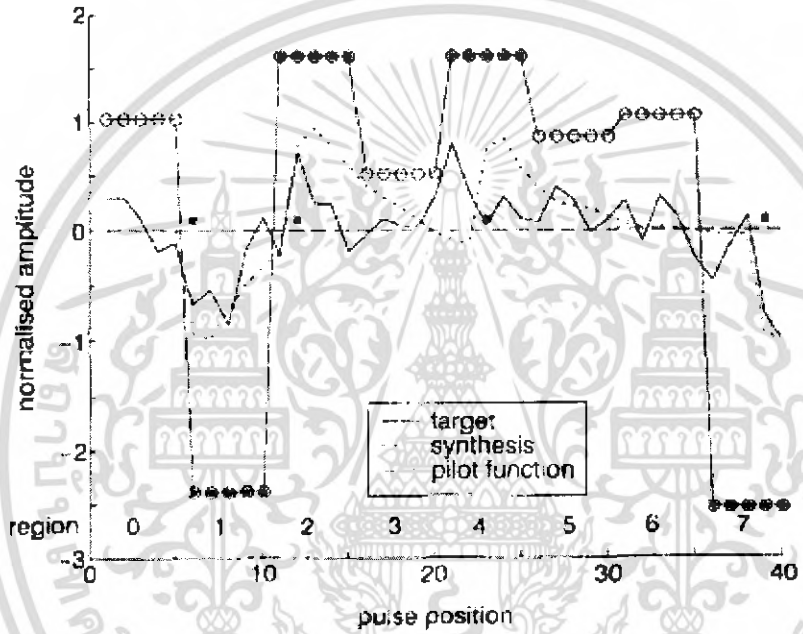
ในภาพรวมแล้ว ถ้าเรามีสัญญาณ target ที่มีความละเอียดต่ำ (r(n)) และการตอบสนองของวงจรกรอง ($h(n) \approx \delta(k)$) เราสามารถสังเกต r(k) เพื่อที่จะพยากรณ์ตำแหน่งของพัลซ์ที่จะเกิดขึ้น และขั้นตอนถัดไปเราจะหาฟังก์ชัน condensing ที่เหมาะสม

สำหรับวิธีของ Candidate position scheme นั้น เราต้องพัฒนาฟังก์ชัน condensing ที่เหมาะสม เพื่อที่จะพยากรณ์ตำแหน่งของพัลซ์ได้อย่างมีประสิทธิภาพมากที่สุด ถ้าพื้นที่ของสัญญาณที่ถูกเลือกแล้ว เราจะ

เริ่มต้นทำการนิยามของฟังก์ชัน pilot E(k) เพื่อที่จะแสดงลักษณะของพลังงานที่อยู่ในพื้นที่แต่ละพื้นที่ ดังสมการต่อไปนี้

$$E(k) = p_k \sum_{n \in R_k} |r(n)|, \qquad k = 0,1,2,\ldots, K-1 \tag{4.5}$$

โดยที่ $p_k = sign(\sum_{n \in R_k} r(n))$ เพื่อที่จะแสดงขั้วของ E(k) ในพื้นที่ที่ k จากรูปแสดงกราฟของสัญญาณ target , สัญญาณที่สังเคราะห์ได้ , ฟังก์ชัน pilot , และตำแหน่งของพัลซ์ที่ได้จากการค้นโดยใช้วิธีการเดิม แสดงโดยรูปที่ 4-3



รูปที่ 4-3 แสดงฟังก์ชัน Pilot, target และสัญญาณ synthesis ใน ตัวเข้ารหัสแบบ ACELP

ในเฟรมย่อยโดยทั่วไป แล้วพื้นที่ใดที่ให้ค่าพลังงาน E(k) มีแอมพลิจูดมากที่สุด มักจะเกิดพัลซ์มากที่สุด ดังนั้นถ้าเราเลือกที่จะค้นหาพื้นที่ M พื้นที่เราจะเลือกพื้นที่ (region) ที่ให้ค่า E(k) สูงสุด M อันดับแรกมาพิจารณาหาพัลซ์เฉพาะภายในพื้นที่ที่เลือกขึ้นมาเท่านั้น โดยตำแหน่งที่จะถูกค้นหาลดลงเหลือ ML/K เท่านั้นแทนที่จะต้องหาทุก ๆ ความเป็นไปได้ที่เกิดขึ้น คือ 2^{13} จะลดลงเหลือเพียง $2M^4$ เนื่องจากตามมาตราฐานของ G.729 จะมีการเปลี่ยนแปลงสัญญาณกระตุ้น v(n) และผลตอบสนองของวงจรกรอง h(n) ในกรณีที่ ค่าพิตท์ของสัญญาณเสียงน้อยกว่าความยาวของเฟรมย่อยดังนั้นฟังก์ชันจะถูกเปลี่ยน ในกรณีที่ค่าพิตท์น้อยกว่าความยาวของเฟรมย่อยดังสมการดังนี้

$$\hat{r}(k)=\widetilde{E}(k)=\begin{cases}E(k)+\beta E(k+\widetilde{T}) & k=0,\ldots,(K-1-\widetilde{T})\\ E(k) & k=(K-\widetilde{T}),\ldots,(\widetilde{T}-1)\\ (1-\beta p_k p_{k-\widetilde{T}})E(k) & k=\widetilde{T},\ldots,(K-1)\end{cases} \quad (4.6)$$

โดยที่ T = T/5

ท้ายที่สุด เมื่อเราเลือกพื้นที่ได้แล้ว การค้นหาตำแหน่งของพัลซ์นั้นเราจะค้นหาเฉพาะพื้นที่ที่เราคาดว่าจะเกิดพัลซ์ ขึ้นเท่านั้น

สำหรับการคำนวณนั้น สามารถลดการคำนวณ หาค่า d, ϕ และการค้นหาได้อย่างมาก ยกตัวอย่างเช่น ถ้า M = 3 เราจะคำนวณ เฉพาะ ตำแหน่งที่เกี่ยวข้องกับ 3 พื้นที่ที่เราเลือกเท่านั้น ในตาราง เราแสดงความซับซ้อนในการคำนวณของวิธีเดิมเมื่อเทียบกับวิธี Candidate position method พบว่า การคำนวณลดลงประมาณ 27 – 48 % ยกตัวอย่างเช่น เมื่อเลือก M=5 พบว่า การค้นหาจะลดลงจาก 1440 เหลือเพียง 220 ครั้ง สำหรับ G.729 และ 320 เหลือเพียง 140 สำหรับ G.729A ที่ใช้การค้นหาแบบ dept-first-search

No. of selected regions	Exhaustive search		Focused search(maximum)		Depth-first tree search	
	Original	Proposed	Original	Proposed	Original	Proposed
M	8192	$2M^4$	<1440	$<2M\lfloor 90/(8/M)^3 \rfloor$	320	$4(2M+M^2)$
4	8192	512	<1440	<96	320	96
5	8192	1250	<1440	<220	320	140

ตารางที่ 4-3 แสดงจำนวนของความเป็นไปได้ ในวิธีการค้นหาแบบ ACELP สำหรับ G.729 coders

สำหรับค่า M ที่เลือกขึ้นมา จะมีผลต่อคุณภาพของเสียงที่สังเคราะห์ขึ้นเช่นเดียวกัน ถ้าเลือก M มากจะทำให้คุณภาพเสียงไม่เพี้ยนมากนัก แต่ถ้าเลือก M น้อย ๆ พบว่าคุณภาพเสียงก็จะแย่ลง ดังนี้

Fast ACELP	M=8	M=7	M=6	M=5	M=4	M=3	M=2
MOS degradation	0	~0*	~0	~0	~0	-0.02	-0.1

* ~0: หมายถึงผู้ฟังไม่สามารถบอกระดับความแตกต่างของเสียงถอดรหัสต้นฉบับและเสียงที่ถูกแก้ไข

ตารางที่ 4-4 MOS degradation for the proposed candidate

จากการทดลองพบว่าระดับที่เหมาะสมในด้านของความซับซ้อนในการคำนวณและคุณภาพของเสียงควรจะเลือกค่า M ที่เท่ากับ 5 แต่ถึงแม้จะลดลงไปมากกว่านี้ คุณภาพของเสียงก็ไม่แตกต่างจากเดิมมากนัก ดังนั้นสำหรับโครงงานนี้ เราจะเลือกการออพติไมซ์โดยที่ M=1

4.1.3 การปรับปรุงความเร็วโดยการแก้ไขที่เครื่องมือ (การจัดการหน่วยความจำ & DSP BIOS Tool)

งานทางด้านการประมวลผลสัญญาณดิจิตอล (Digital signal processing) นั้นเป็นงานที่ต้องใช้การคำนวณทางคณิตศาสตร์สูงดังนั้นจึงมีผู้ผลิตอุปกรณ์ที่ใช้งานทางด้านนี้โดยเฉพาะอยู่หลายเจ้าด้วยกัน เท็กซัสอินสตรูเมนต์ ก็ถือว่าเป็นบริษัทที่มีชื่อเสียงและถือส่วนแบ่งในการตลาดในการผลิตอุปกรณ์ทางด้านนี้มากกว่า 50 % ซึ่งทางผู้จัดทำได้ใช้บอร์ดรุ่น TMS320C6713 เป็นบอร์ดที่สามารถคำนวณเลขทศนิยม (floating point unit) ได้และมี เอ แอล ยู (ALU) ถึง 4 ชุด สรุปคุณสมบัติที่สำคัญ ๆ ดังนี้

	TMS320C6713-225
Cycle time (ns)	4.4
Data/Program memory (bits)	4KByte L1D Data Cache; 4KByte L1P Program Cache; 64KByte L2 Cache; 192 Kbytes L2 SRAM
SDRAM	8 MByte
DMA	16 (EDMA)
External memory interface	GDP: 32 bits
Host port /Exp. Bus / PCI	(1)16-bits HPI
McBSP	2
McASP	2
I2C	2
Timers	(2)32-bits
GPIO	(1)16-bits
Core Supply (Volts)	1.26
IO Supply (Volts)	3.3

ตารางที่ 4-5 แสดงคุณสมบัติของบอร์ดที่ใช้

จากความสามารถของบอร์ดที่ใช้ในการทดลองดังกล่าวข้างต้นจึงมีความจำเป็นที่จะต้องเข้าใจและดึงเอาความสามารถของบอร์ดที่ใช้อยู่ออกมาให้ได้มากที่สุดซึ่งแบ่งออกได้เป็น 2 กรณีดังนี้

1. การออพติไมซ์โดยอาศัยออฟชันของเครื่องมือที่ใช้พัฒนา (build option)
2. การออพติไมซ์โดยใช้ความสามารถของ Real-time os ในเรื่องของการจัดการหน่วยความจำ

4.1.3.1 การออพติไมซ์โดยอาศัยความสามารถของเครื่องมือที่ใช้พัฒนา

เนื่องจากเครื่องมือที่ใช้ในการพัฒนาโปรแกรม (Code composer studio V2.0) มีความสามารถในการเลือกระดับในการ build code ของโปรแกรมได้หลายระดับรูปแบบด้วยกันซึ่งสามารถดูผลได้จากบทที่เกี่ยวกับการทดลองทำการออพติไมซ์โมดูล G.729 โดยอาศัยเครื่องมือ Code Composer Studio V2.0

4.1.3.2 การออพติไมซ์โดยใช้ความสามารถของ Real-time OS ในเรื่องของการจัดการหน่วยความจำ

เมื่อเทียบเวลาในการเข้าถึงข้อมูลระหว่างหน่วยความจำภายในกับหน่วยความจำภายนอก หน่วยประมวลผลแล้วจะพบว่าการเข้าถึงข้อมูลของหน่วยความจำภายใน(Internal memory) เร็วกว่าหน่วยความจำภายนอกถึง 25 wait state (ดี เอส พี บอร์ด รุ่น TMS320C6713) ดังนั้นจึงควรที่จะนำ อัลกอริทึมไปเก็บในหน่วยความจำภายใน แต่ เนื่องจากทรัพยากรของหน่วยความจำภายในหน่วยประมวลผลนั้นมีจำกัดดังนั้นหากขนาดของโปรแกรมที่ทำการพัฒนามีขนาดใหญ่แล้วก็เป็นไปไม่ได้ที่จะนำโค้ดเซคชันและเซคชันข้อมูลทั้งหมดเข้าไปใช้ในหน่วยความจำภายในได้จึงจำเป็นที่จะต้องให้ความสำคัญในการจัดการกับทรัพยากรทางด้านนี้เป็นอย่างมากซึ่งวิธีที่ได้นำมาใช้ในการจัดการนั้นมีดังนี้

4.1.3.2.1 Overlay

การทำ Overlay เป็นการแบ่งเซคชันของโปรแกรมออกเป็นส่วนย่อย ๆ โดยจะโหลดเซคชันส่วนที่ใช้งานเข้าไปไว้ในหน่วยความจำหลักโดยแบ่งออกเป็นสองกรณีคือ ส่วนของโค้ดเซคชันและเซคชันข้อมูลในส่วนของโค้ดเซคชันนั้นมีปัจจัยในการพิจารณาว่าจำเป็นที่ต้องทำ overlay หรือไม่คือ

1. ดี เอส พี บอร์ด ที่ใช้นั้นมีแคช (Cache) หรือไม่ หากมีแคชส่วนนั้นมีขนาดเท่าไหร่และมีชนิดอย่างไร (Program-Cache , Data-Cache ,P/D-Cache)
2. กรณีที่มีโปรแกรมแคช หรือ P/D-Cache แล้วต้องพิจารณาต่อว่าขนาดของแคช นั้นเพียงพอที่จะเก็บส่วนของโมดูลของวัตถุที่ใหญ่ที่สุดของเราได้หรือไม่
3. หากมีขนาดของแคช เพียงพอก็ไม่จำเป็นที่จะต้องทำ Overlay โค้ดเซคชัน

เนื่องจากแคช ของ ดี เอส พี บอร์ด ที่ใช้เป็นแบบ P/D แคชซึ่งมี L1Pแคชโปรแกรม ขนาด 4KByte ซึ่งเมื่อพิจารณาดูแล้วอัลกอริทึมที่ได้พัฒนาขึ้นมานั้นฟังก์ชัน ที่มีขนาดใหญ่ที่สุดมีขนาดไม่เกิน 4Kbyte จึงไม่จำเป็นต้องทำ Overlay ในส่วนของโค้ดเซคชัน

การทำ Overlay ในส่วนของเซคชันข้อมูลนั้นเราใช้วิธีของมาตราฐานการเขียนอัลกอริทึมของผู้ผลิตไลบราลี โดยการรวมกลุ่มของข้อมูลที่ใช้ในช่วงเดียวกันให้อยู่ด้วยกัน โดยอนุญาติให้ผู้ใช้งานอัลกอริทึมสามารถที่จะคัดลอกเซคชันข้อมูล (data-section) ส่วนที่ต้องการใช้เข้ามาไว้ในหน่วยความจำภายใน ได้ ส่วนโมดูล ที่ไม่ถูกใช้งานก็สามารถคัดลอกออกไปเก็บไว้ที่พื้นที่ภายนอกได้ซึ่งการอิมพลีเมนต์ นั้นจะอยู่ในส่วนของ ALG_MOVE ในภาคผนวกเรื่อง มาตราฐานการเขียนอัลกอริทึมของผู้ผลิตไลบราลี

4.1.3.1.2 Pragma directive

Pragma จริงแล้วก็คือการบอกอะไรบางอย่างกับ compiler เพื่อเป็นประโยชน์ในการสร้าง build code และ การลิงก์โมดูล ต่าง ๆ เช่น ใช้ในการกำหนดเซคชันของฟังก์ชัน ต่าง ๆ ว่าจะจัดกลุ่มให้อยู่ในเซคชันส่วนใดทำให้เราสามารถ แบ่งเซคชันที่ใช้ในการ inint กับ Exit ออกมาเก็บ ใน Write-once บัฟเฟอร์ได้ซึ่งเมื่อโมดูลส่วนนี้ถูกใช้งานไปครั้งหนึ่งแล้วก็จะสามารถนำหน่วยความจำกลับมาใช้ใหม่ ข้อดีอีกอย่างหนึ่งก็คือกรณีที่ในไฟล์หนึ่ง ๆ นั้นมีหลายฟังก์ชัน หากเราไม่ได้กำหนด pragma ให้แต่ละฟังก์ชัน แล้ว การโหลด code ขึ้นมานั้นจะถูกโหลดขึ้นมาทั้ง page หรือทั้งโมดูลของวัตถุ ทำให้เสียเวลาและพื้นที่ไปกับฟังก์ชัน ที่ไม่ได้ถูกเรียกใช้งาน แต่หากเรากำหนด pragma กำหนดเซคชันให้กับฟังก์ชัน symbol ก็จะช่วยให้การโหลดนั้นLoad เฉพาะฟังก์ชันที่ใช้งานจริง ๆ ไม่ได้โหลดมาทั้งไฟล์

4.2 การพัฒนาให้เป็นมัลติแชนเนล

เนื่องจากในการพัฒนาโปรแกรมหรืองานหนึ่ง ๆ นั้นผู้พัฒนาไม่จำเป็นต้องเริ่มพัฒนาตั้งแต่ต้นใหม่ตลอดการนำฟังก์ชัน ที่เคยเขียนมา reuse ใหม่ช่วยให้การพัฒนาเป็นไป ได้อย่างรวดเร็ว concept การทำงานในลักษณะนี้ทำให้มีผู้พัฒนาไลบรารี ออกมาขายให้กับผู้ที่ไม่ต้องการพัฒนาในรายละเอียดบางอย่าง งานด้าน Digital Signal Processing ก็เช่นเดียวกันได้มีผู้พัฒนา อัลกอริทึมต่าง ๆ ออกมาขายอย่างมากมายแต่ เมื่อมองในมุมมองของความเป็นไปได้ในการประกอบรวมอัลกอริทึมจากผู้ผลิตต่างเจ้าก็เป็นไปได้ยากหรือแทบเป็นไปไม่ได้เลยที่จะไม่ต้องแก้ไขโค้ดบางส่วนเพื่อประกอบรวมอัลกอริทึมเหล่านั้น จึงมีความสำคัญที่จะต้องมีมาตรฐานในการเรียกฟังก์ชัน สำหรับ อัลกอริทึมของ DSP นั้นจำเป็นที่จะต้องมีคุณสมบัติสำคัญอีกประการหนึ่งคือมัลติแชนเนิลซึ่งการทำงานจะเป็นหลาย ๆ Thead พร้อม ๆ กัน โดยจะต้องมีคุณสมบัติเป็น reentrant ซึ่งจำเป็นต้องมีการเก็บสถานะ และค่าของตัวแปร รวมทั้งเป็น relocatable อีกด้วย

4.2.1 อัลกอริทึมแบบมัลติแชนเนิล

อัลกอริทึมแบบมัลติแชนเนิลมักจะถูกใช้เพื่อตอบสนองความต้องการดัง ต่อไปนี้

- การใช้ อัลกอริทึมของ ดี เอส พี (DSP algorithm) ตัวเดียวกันกับหลายช่องสัญญาณ
- ชนิดของ อัลกอริทึมและ ช่องสัญญาณ ที่ใช้สามารถ เปลี่ยนแปลงได้ระหว่าง run-time

โดยที่ อัลกอริทึมลักษณะนี้จะต้องมีคุณสมบัติ Reentrant และ Relocatable

4.2.1.1 โปรแกรมที่มีคุณสมบัติ Reentrant

คือคุณสมบัติที่โปรแกรมนั้นอาจจะถูกเรียกเข้าไปทำงานซ้ำ ๆ หรืออาจจะถูกเรียกเข้าไปทำงานก่อนที่การเรียกก่อนหน้านั้นจะเสร็จสมบูรณ์ โดยที่การทำงานนั้นไม่ขึ้นต่อกัน หรืออีกนัยหนึ่งก็คือ ในส่วนของโค้ดเซคชันและเซคชันข้อมูลนั้นสามารถอ่านได้อย่างเดียว

โดยทั่วไปแล้วชนิดของตัวแปรมีอยู่ 2 แบบคือ

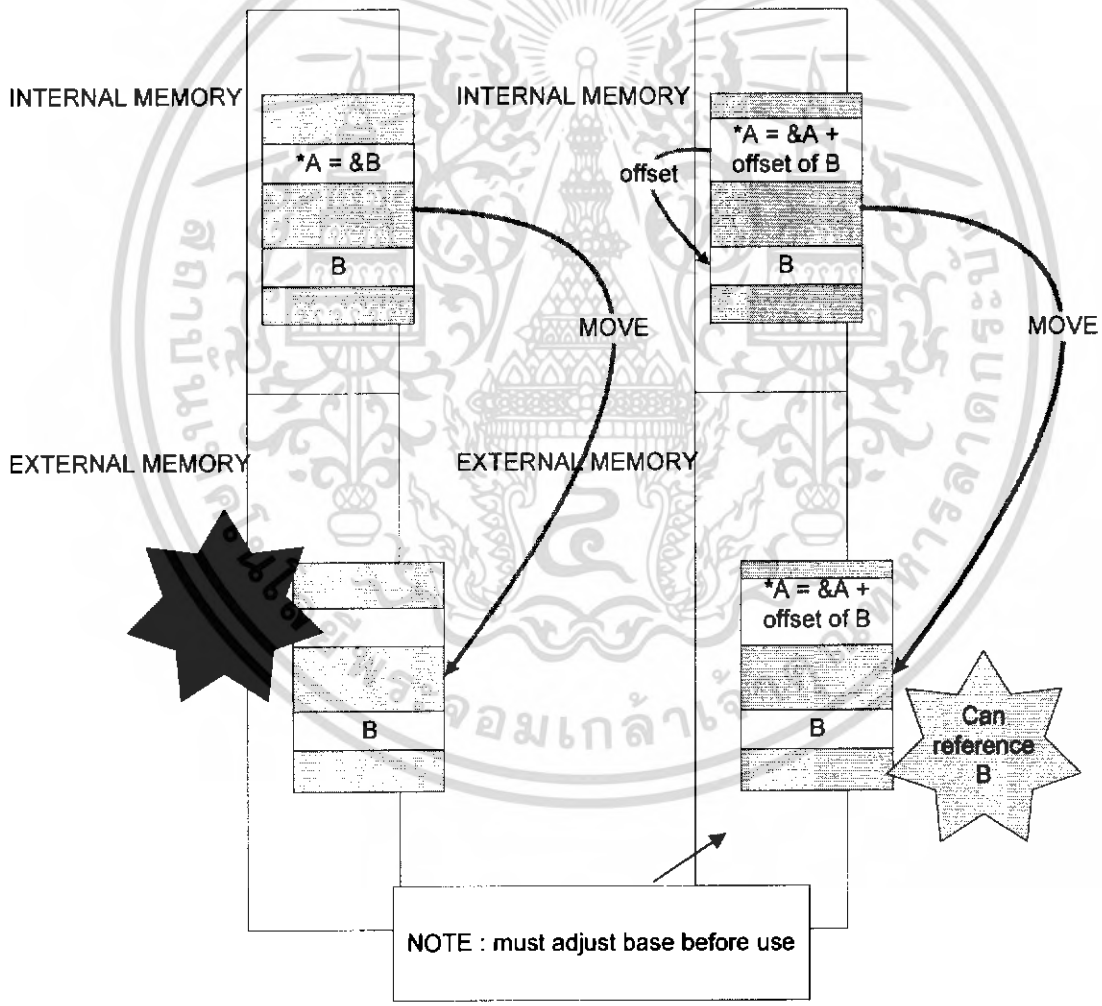
- ช่วงชีวิตของตัวแปรโลคอล (Local lifetime) ตัวแปรชนิดนี้จะถูกจองใหม่ลงใน block ที่ได้ ประกาศไว้ในแต่ละครั้งที่ execution control ผ่านและถูกคืนเมื่อรันคำสั่ง return()

- ช่วงชีวิตของตัวแปรโกลบอล (Global lifetime) ตัวแปรชนิดนี้จะถูกจองเมื่อเริ่มต้นโปรแกรม และทำการคืนเมื่อจบการทำงานของโปรแกรม

ตัวแปรชนิดโกลบอลจะต้องเป็น constant เพื่อที่จะทำให้โปรแกรม เป็น reentrant ในแต่ละ Channel จำเป็นที่จะต้องมีตัวแปรที่ใช้ในการเก็บ States โดยที่การส่งค่าตัวแปรนั้นอาศัยการส่งโดยใช้ตำแหน่งของตัวแปร (pass by reference หรือ โดย pointer)

4.2.1.2 Relocatable

เนื่องจากปัญหาทางด้านข้อจำกัดของหน่วยความจำภายในมีขนาดไม่เพียงพอที่จะนำอัลกอริทึมทุกตัวเข้าไปทำงานพร้อมกันได้จึงจำเป็นที่ข้อมูลทุกส่วนของอัลกอริทึมต้องเป็น Relocatable ซึ่งจะช่วยให้เราสามารถเคลื่อนย้าย อัลกอริทึมไปมาระหว่างหน่วยความจำภายนอกและภายใน (off-chip และ on-chip) ระหว่างการทำ context switching ได้



รูปที่ 4-4 แสดงการทำ Data Relocatable

จากรูปที่ 4-4 รูปทางขวามือได้ทำการแก้ไขโปรแกรมให้มีคุณสมบัติของ Data Relocatable โดยแก้ไขตัวแปร pointer ในโครงสร้างของเราให้เก็บ offset ที่ชี้ไปยังหน่วยความจำส่วนที่อ้างถึงแทนที่จะเก็บ Absolute

address เหมือนลักษณะการเขียนของรูปทางซ้ายมือซึ่งทำให้การอ้างตำแหน่งของหน่วยความจำของ pointer จะผิดหากมีการย้ายตำแหน่ง block ของหน่วยความจำ แต่รูปทางขวามือจำเป็นต้องปรับการชี้ตำแหน่งก่อนเรียกใช้

ซึ่งในการที่เราจะทำให้เกิดคุณสมบัติที่กล่าว เราจำเป็นที่จะออกแบบโครงสร้างที่ใช้เก็บข้อมูลที่เหมาะสม โดยการออกแบบโครงสร้างนั้น เราแบ่งตามการทำงานในแต่ละ Process ที่ตัวแปรตัวนั้นเกี่ยวข้องอยู่ ซึ่งแสดงได้ดังต่อไปนี้

Memtab	Size(Byte)
coderMemBlk	10,376
preProcMemBlk	32
exc_err_mblk	32
vad_mblk	520
cod_cng_mblk	768
bwfwfunc_mblk	8
pwf_mblk	24
acelp_cp_d4i40_17_mblk	4
q_gaincp_mblk	32
phdisp_mblk	72
Total	11,868

ตารางที่ 4-6 แสดงขนาดของ Memtab แต่ละตารางหน่วยความจำของตัวเข้ารหัส

Memtab	Size(Byte)
decld8cp_mblk	4,000
post_pro_mblk	32
exc_err_mblk	32
phdisp_mblk	72
pstcp_mblk	2,288
bwfwfunc_mblk	8
degaincp_mblk	32
dec_sid_mblk	96
decodecp_mblk	2,696
Total	9,256

ตารางที่ 4-7 แสดงขนาดของ Memtabแต่ละตารางหน่วยความจำของตัวถอดรหัส

Table Name	Size(Byte)
G729_DTX_TABLE	808
G729_LD8K_TABLE	3,962
G729_LD8CP_TABLE	2,260

ตารางที่ 4-8 แสดงขนาดของตารางค่าคงที่

4.2.1.3 การแก้ไขเชิงปฏิบัติในการทำมัลติแชนเนิล (Practical Approaches)

4.2.1.3.1 การแก้ไขที่ ฟังก์ชันที่มีตัวแปรสแตติกหรือโกลบอลเป็นสมาชิก (Functions with Multiple Static/Global Variables)

ตัวแปรที่เป็นโกลบอลและสแตติกต้องถูกจัดกลุ่มเข้าด้วยกันเป็นโครงสร้างดังตัวอย่าง

```
// File Pre_Proc.c
static FLOAT x0, x1;        /* high-pass fir memory        */
static FLOAT y1, y2;        /* high-pass iir memory        */
void init_pre_process( void  )
{
x0 = x1 = (F)0.0;
y2 = y1 = (F)0.0;
return;
}
```

แก้ไขใหม่ดังนี้

```
typedef struct{
FLOAT x0;
FLOAT x1;
FLOAT y1;
FLOAT y2;
}preProcMemBlk,*preProeMemBlk_handle ;

void init_pre_process( preProcMemBlk_handle hmem)
{
hmem->x0 = hmem->x1 = (F)0.0;
```

ในกรณีที่โมดูล ใหญ่มีการเรียกใช้โมดูลย่อยซึ่งแต่ละโมดูล ก็มีสแตติกหรือตัวแปรชนิดโกลบอล ในกรณีนี้มีวิธีการจัดการอยู่ 2วิธีคือ

1. ประกาศตัวแปรสแตติกและโกลบอลของทุกโมดูล เป็นโครงสร้างใหญ่โครงสร้างเดียวเลยแล้วอาศัยการส่ง pointer ให้แก่ทุก ๆ โมดูลย่อย (sub module)
2. ทำการจัดกลุ่มตัวแปรสแตติกของแต่ละ โมดูลย่อยให้แยกเป็นโครงสร้างของใครของมัน โดยให้ ตัวแปรโกลบอลและวัตถุของโมดูลย่อย อยู่ในโครงสร้างของโมดูลของโนดพาร์เรน แล้วส่งให้แก่โมดูลย่อยที่ต้องการ

```
// File codercp.c
typedef struct{
coderMemBlk_handle coder_handle;
preProcMemBlk_handle preProc_handle;
exc_err_mblk_handle exc_err_handle;
vad_mblk_handle vad_handle;
cod_cng_mblk_handle cod_eng_handle;
bwfwfunc_mblk_handle bwfwfunc_handle;
pwf_mblk_handle pwf_handle;
acelp_ep_d4i40_17_mblk_handle acelp_ep_d4i40_17_handle;
q_gaincp_mblk_handle q_gaincp_handle;
phdisp_mblk_handle phdisp_handle;

}ENCODER_G729_MEM_BLK,*ENCODER_G729_MEM_BLK_HANDLE;

void ENCODER_G729_MEM_BLK_init(ENCODER_G729_MEM_BLK_HANDLE *hmem );
void ENCODER_G729_MEM_BLK_free(ENCODER_G729_MEM_BLK_HANDLE hmem);
```

```
void ENCODER_G729_MEM_BLK_free(ENCODER_G729_MEM_BLK_HANDLE hmem){
        coderMemBlk_free(hmem->coder_handle);
        preProcMemBlk_free(hmem->preProc_handle);
        exc_err_mblk_free(hmem->exc_err_handle);
        vad_mblk_free(hmem->vad_handle);
        cod_cng_mblk_free(hmem->cod_cng_handle);
        bwfwfunc_mblk_free(hmem->bwfwfunc_handle);
        acelp_cp_d4i40_17_mblk_free(hmem->acelp_cp_d4i40_17_handle);
        pwf_mblk_free(hmem->pwf_handle);
        q_gaincp_mblk_free(hmem->q_gaincp_handle);
        phdisp_mblk_free(hmem->phdisp_handle);

        free(hmem);
}
```

4.2.1.3.2 ตารางค่าคงที่ (Constant Tables)

แม้ว่าตารางค่าคงที่ จะไม่ถูกแก้ไขในการประมวลผล แต่ว่ามันก็ต้อง support relocatable ด้วย เพื่อที่จะได้ย้ายเข้าไปอยู่ หน่วยความจำภายในก่อนการประมวลผลช่วยให้ประมวลผลได้รวดเร็วขึ้นและสามารถย้ายออกมาได้เมื่อพื้นที่ไม่เพียงพอ โดยทั่วไปแล้วตารางจะถูกประกาศไว้ที่หน่วยความจำข้อมูลภายนอก (external data memory) แล้วค่อยย้ายเข้ามาในหน่วยความจำภายในก่อนการประมวลผล ซึ่งต้องใช้เวลามากสำหรับ อัลกอริทึมที่มีตารางค่าคงที่ หลายตาราง ในการระบุว่า subroutine ไหนใช้ตารางไหน เราใช้วิธีการเช่นเดียวกับการจัดกลุ่มตารางที่เป็นค่าคงที่ (context data constant table) ก็สามารถทำให้ relocatable ได้ในช่วงเวลาสั้น ๆ

วิธีการ

1. ทำการจัดกลุ่มค่าคงที่เข้ามาใน โครงสร้างเพียงโครงสร้างเดียว
2. ประกาศข้อมูล pointer กับชนิดของโครงสร้างซึ่ง Pointer ตัวนี้จะเป็นตัวแปรชนิดโกลบอล
3. เพิ่มการนิยามข้อมูลลงไปในไฟล์ที่ก่อนแก้ไขที่มีค่าคงที่อยู่

```
// Table.h
typedef struct{
/* VAD constants */
FLOAT lbf_corr[NP+1];
/* SID gain quantization */
FLOAT fact[NB_GAIN+1];
FLOAT tab_Sidgain[32];
/* SID LSF quantization */
int PtrTab_1[32];
int PtrTab_2[2][16];
FLOAT noise_fg[MODE][MA_NP][M];
FLOAT noise_fg_sum[MODE][M];
FLOAT noise_fg_sum_inv[MODE][M];

FLOAT Mp[MODE];
}G729_DTX_TABLE,*G729_DTX_TABLEPTR;

G729_DTX_TABLEPTR G729_DTX_TablePtr ;
void initTab_dtx();
/* VAD constants */
#define lbf_corr G729_DTX_TablePtr->lbf_corr
/* SID gain quantization */
#define fact G729_DTX_TablePtr->fact
#define tab_Sidgain G729_DTX_TablePtr->tab_Sidgain
```

```
/* SID LSF quantization */
#define PtrTab_1 G729_DTX_TablePtr->PtrTab_1
#define PtrTab_2 G729_DTX_TablePtr->PtrTab_2
#define noise_fg G729_DTX_TablePtr->noise_fg
#define noise_fg_sum G729_DTX_TablePtr->noise_fg_sum
#define noise_fg_sum_inv G729_DTX_TablePtr->noise_fg_sum_inv
#define Mp G729_DTX_TablePtr->Mp
```

4.2.1.3.3 Data Relocation

กรณีที่มีตัวแปรที่เป็น static/global/constant pointers ต้องทำการปรับวิธีการอ้างถึงค่าของตัวแปรโดยการเก็บ off set แทนโดยอาศัย base จากตำแหน่งของโครงสร้างที่ถูกจัดกลุ่ม

ตัวอย่าง

```
/*-----------------------------------------------*
 *       Staticหน่วยความจำallocation.            *
 *-----------------------------------------------*/

/* Speech vector */
static FLOAT old_speech[L_TOTAL];
static FLOAT *speech, *p_window;
FLOAT  *new_speech;                /* Global variable */
/* Weighted speech vector */
static FLOAT old_wsp[L_FRAME+PIT_MAX];
static FLOAT *wsp;
```

```
/* Filter's memory */
static FLOAT mem_err[M_BWD+L_SUBFR], *error;
/* Weighted speeeh vector */
static FLOAT old_wsp[L_FRAME+PIT_MAX];
static FLOAT *wsp;


/* Excitation vector */
static FLOAT old_exc[L_FRAME+PIT_MAX+L_INTERPOL];
static FLOAT *exc;
/* Zero vector */
static FLOAT ai_zero[L_SUBFR+M_BWDP1];
static FLOAT *zero;

void init_coder_ld8c(
   int dtx_enable   /* input : DTX enable flag */
){
  new_speech = old_speech + L_TOTAL - L_FRAME;       /* New speech     */
   speech     = new_speech - L_NEXT;                 /* Present frame  */
   p_window   = old_speech + L_TOTAL - L_WINDOW;     /* For LPC window */

   /* Initialize static pointers */
   wsp    = old_wsp + PIT_MAX;
   exc    = old_exc + PIT_MAX + L_INTERPOL;
   zero   = ai_zero + M_BWDP1;
   error  = mem_err + M_BWD;
return ;
}
```

แก้ไขใหม่ได้ดังนี้

```
void coderMemBlk_init(coderMemBlk_handle *hmem){


FLOAT lsp_oldTmp[M]=
{ (F)0.9595, (F)0.8413, (F)0.6549, (F)0.4154, (F)0.1423,
(F)-0.1423, (F)-0.4154, (F)-0.6549, (F)-0.8413, (F)-0.9595};
int lag_bufTmp[5]={20,20, 20, 20,20};
 FLOAT pgain_bufTmp[5]={(F)0.7,(F)0.7, (F)0.7, (F)0.7,(F)0.7};


   (*hmem) = (coderMemBlk *)malloc(sizeof(coderMemBlk));
          copy(lsp_oldTmp,(*hmem)->lsp_old,M);
          copyArrI(lag_bufTmp,(*hmem)->lag_buf , 5);
          copy(pgain_bufTmp,(*hmem)->pgain_buf , 5);


          /* New speech    */
   (*hmem)->new_speech = (FLOAT *)(((char *)&((*hmem)->old_speech)) + ((L_TOTAL -
L_FRAME)*sizeof(FLOAT)) -(int)(*hmem));
     /* Present frame  */
   (*hmem)->speech     = (FLOAT *)(((char *)&((*hmem)->new_speech ))-
(L_NEXT*sizeof(FLOAT)) -(int)(*hmem));
     /* For LPC window */
   (*hmem)->p_window   = (FLOAT *)(((char *)&((*hmem)->old_speech ))+ ((L_TOTAL -
L_WINDOW)*sizeof(FLOAT))-(int)(*hmem));
```

```
   /* Initialize static pointers */
   (*hmem)->wsp   = (FLOAT *)(((char *)&((*hmem)->old_wsp ))+
(PIT_MAX*sizeof(FLOAT))-(int)(*hmem));
   (*hmem)->exc   = (FLOAT *)(((char *)&((*hmem)->old_exc ))+ ((PIT_MAX +
L_INTERPOL)*sizeof(FLOAT))-(int)(*hmem));
   (*hmem)->zero  = (FLOAT *)(((char *)&((*hmem)->ai_zero ))+
(M_BWDP1*sizeof(FLOAT))-(int)(*hmem));
   (*hmem)->error = (FLOAT *)(((char *)&((*hmem)->mem_err ))+
(M_BWD*sizeof(FLOAT))-(int)(*hmem));
          return ;
}
```

บทที่5

การทดลองและผลการทดลอง

การทดลองที่ 5.1 การทดลองตรวจสอบความถูกต้องของ ชุดเข้า/ถอดรหัส G.729 ที่โปรแกรมลงบน ดี เอส พี บอร์ด

วัตถุประสงค์

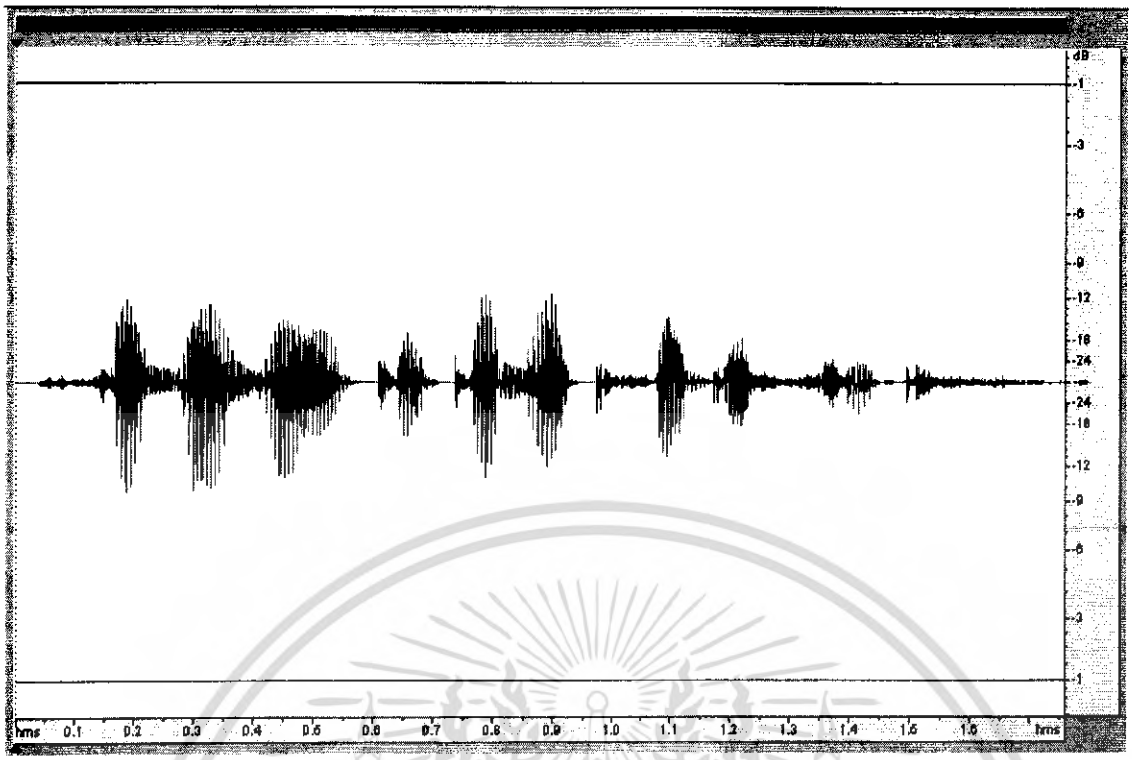
1. เพื่อตรวจสอบชุดเข้ารหัส/ ถอดรหัส เมื่อทำการโปรแกรมลงบน ดี เอส พี บอร์ด

วิธีการทดลอง

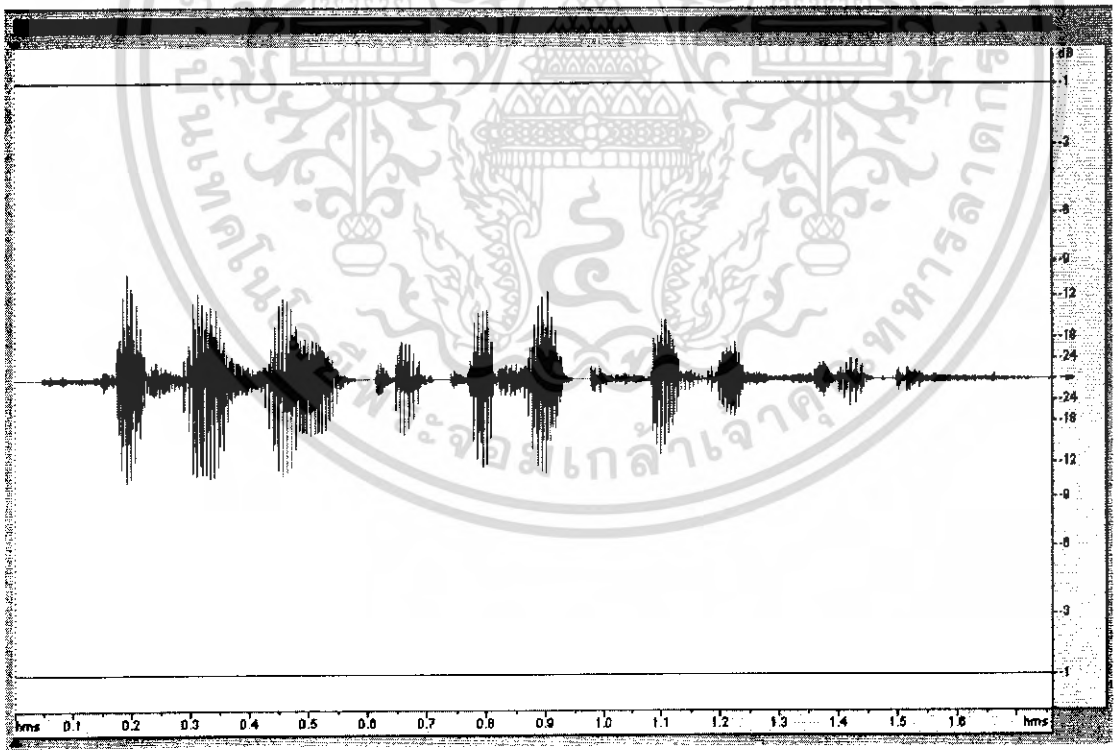
1. นำไฟล์เสียงอินพุตที่ 1 มาทำการเข้ารหัสบน ดี เอส พี บอร์ด ผ่านทางโปรแกรม Code Composer Studio version 2.0
2. จากนั้นนำเอาท์พุตที่ได้มาทำการถอดรหัสโดยใช้ตัวถอดรหัสบน ดี เอส พี บอร์ดจะได้ไฟล์เสียงเอาท์พุตออกมา
3. นำไฟล์เสียงอินพุท เทียบกับไฟล์เสียงที่ผ่านการเข้ารหัสและถอดรหัส G.729 มาเปรียบเทียบกันโดยใช้โปรแกรม Adobe Audition 1.5 โดยการฟังเสียงแล้วเปรียบเทียบรูปกราฟที่ได้
4. นำสัญญาณเสียงอินพุตที่ 2 มาทำการทดลองตามข้อ 1 ถึง 3 อีกครั้ง

ผลการทดลอง

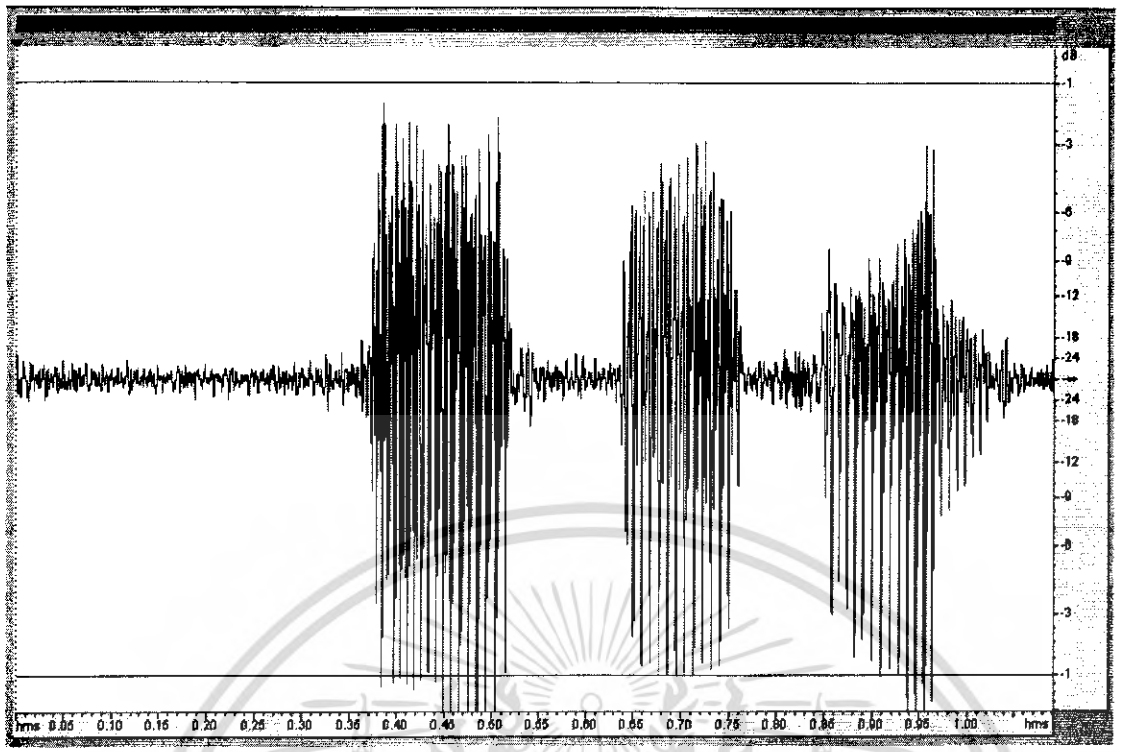
1. จากการฟังพบว่าเสียงสัญญาณที่ได้จากการสังเคราะห์เทียบกับเสียงสัญญาณอินพุต ได้ยินเสียงไม่แตกต่างกัน (ไม่สามารถแยกแยะความแตกต่างได้)
2. เมื่อนำไฟล์เสียงอินพุท เทียบกับไฟล์เสียงที่ผ่านการเข้ารหัสและถอดรหัส G.729 พบว่ามีลักษณะของสัญญาณเป็นดังรูปต่อไปนี้



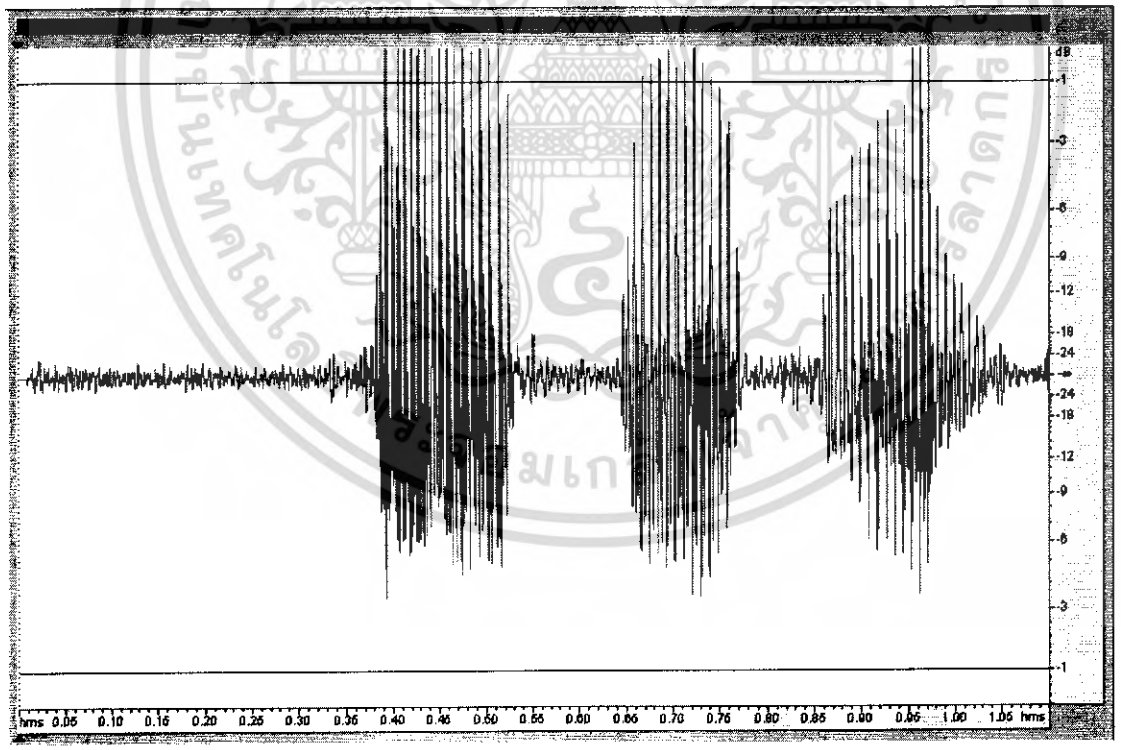
รูปที่ 5.1แสดงเสียงสัญญาณต้นฉบับที่ 1



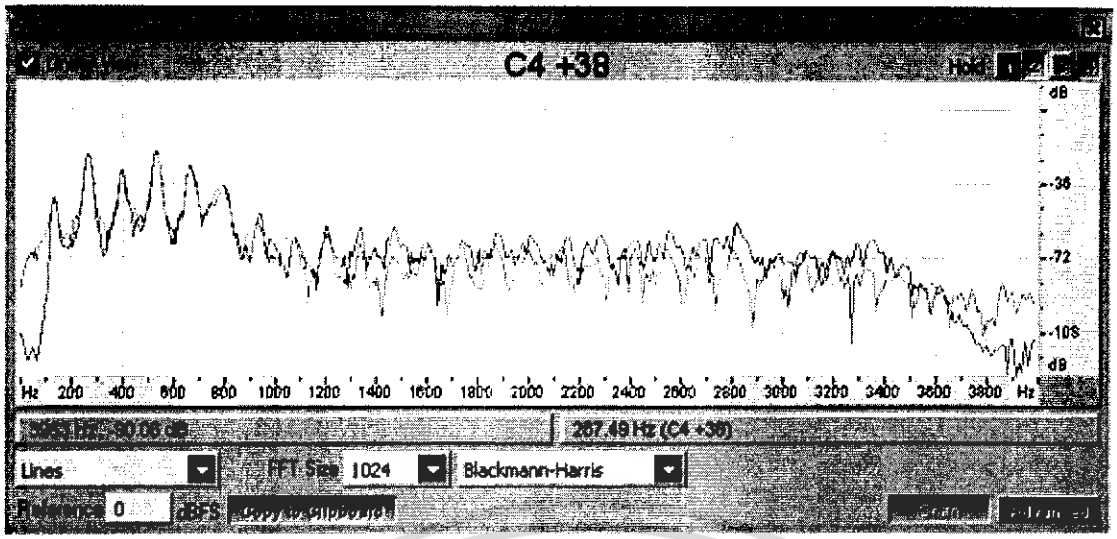
รูปที่ 5.2 แสดงสัญญาณเสียงสังเคราะห์ที่ 1โดยโมดูล G.729



รูปที่ 5.3 แสดงเสียงสัญญาณต้นฉบับที่ 2



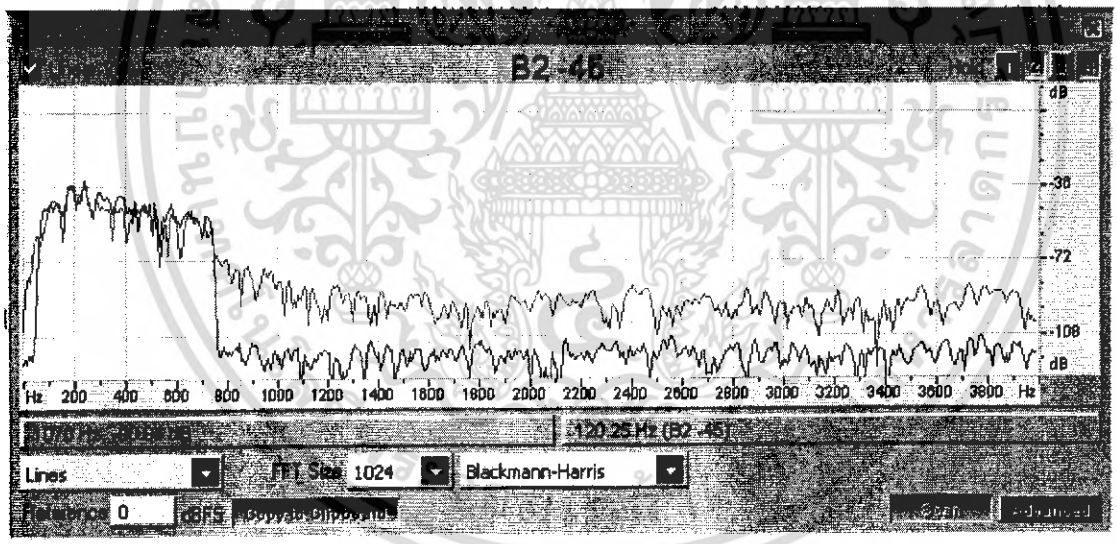
รูปที่5.4 แสดงสัญญาณเสียงสังเคราะห์ที่ 2โดยโมดูล G.729



รูปที่ 5.5 เปรียบเทียบสัญญาณต้นฉบับกับสัญญาณสังเคราะห์ในโดเมนความถี่ (สัญญาณที่ 1)

โดยที่ อินพุต = สัญญาณ Input 1

อินพุต = สัญญาณ synthesis 1



รูปที่5.6 เปรียบเทียบสัญญาณต้นฉบับกับสัญญาณสังเคราะห์ในโดเมนความถี่ (สัญญาณที่ 2)

โดยที่ อินพุต = สัญญาณ Input 2

อินพุต = สัญญาณ synthesis 2

การทดลองที่5.2 การทดลองปรับปรุงความเร็วของชุดเข้ารหัสและถอดรหัส G.729 โดยอาศัย Build ออพชัน ของ Code Composer Studio V2.0

วัตถุประสงค์

1. เพื่อศึกษาระดับการออพติไมซ์ของเครื่องมือ code composer ว่าแต่ละระดับจะมีผลต่อประสิทธิภาพการทำงานอย่างไร
2. ตรวจสอบว่าเมื่อเลือกระดับการออพติไมซ์แบบต่าง ๆ แล้วผลลัพธ์การทำงานยังคงถูกต้องอยู่หรือไม่

วิธีการทดลอง

1. เลือกระดับของการ ออพติไมซ์สำหรับการ compile และ build สำหรับ code composer
2. ทำการทดลองที่ 1 ซ้ำโดยการเปรียบเทียบไฟล์ในลักษณะเดิมเพื่อดูว่าไฟล์ที่ได้หลังจากการออพติไมซ์แล้วนั้นยังคงเหมือนเดิมอยู่หรือไม่
3. บันทึกขนาดของไฟล์ที่ใช้ในการรัน ในการออพติไมซ์แบบต่าง ๆ
4. บันทึกเวลาการทำงานของโมดูล G.729 ในการออพติไมซ์แบบต่าง ๆ

ผลการทดลอง

แสดงผลการทดลองการระดับของการออพติไมซ์สำหรับโมดูล การเข้าและถอดรหัส

Optimize Level	File size (KB)	Max (mS)	Avg (mS)
size most critical (None)	672	16.65	14.25
size most critical (Register)	619	10.37	8.82
size most critical (Local)	588	9.35	7.64
size most critical (Function)	592	8.08	7.13
size most critical (File)	590	8.04	6.83
speed critical (None)	675	16.43	13.68
speed critical (Register)	622	10.10	8.6
speed critical (Local)	591	9.10	7.79
speed critical (Function)	613	5.38	4.76
speed critical (File)	613	5.25	4.52
speed most critical (None)	675	16.47	13.29
speed most critical (Register)	622	10.17	8.21
speed most critical (Local)	592	9.09	7.46
speed most critical (Function)	617	5.42	4.65
speed most critical (File)	617	5.36	4.59

ตารางที่ 5-1 แสดงผลการทดลองเมื่อเลือกระดับของการออพติไมซ์แบบต่าง ๆสำหรับโมดูล การถอดรหัส G.729

<u>สรุปผลการทดลอง</u>

1. เมื่อเปรียบเทียบด้านขนาดของไฟล์พบว่า

- การเข้ารหัสระดับของการออพติไมซ์แบบ size most critical (Local) ให้ผลลัพธ์ที่มีขนาดไฟล์ที่เล็กที่สุด
- การถอดรหัสระดับของการออพติไมซ์แบบ size most critical (Function / File) ให้ผลลัพธ์ที่มีขนาดไฟล์ที่เล็กที่สุด

2. เมื่อเปรียบเทียบด้านความเร็ว พบว่า

- การเข้ารหัสระดับของการออพติไมซ์แบบ speed critical (File) ให้ผลลัพธ์ที่ใช้เวลาน้อยที่สุด
- การถอดรหัสระดับของการออพติไมซ์แบบ speed critical (File) ให้ผลลัพธ์ที่ใช้เวลาน้อยที่สุด

3. เมื่อเปรียบเทียบได้คุณสมบัติ ขนาดของไฟล์และ ความเร็วในการทำงาน พบว่า

- การเข้ารหัสระดับของการออพติไมซ์แบบ speed critical (File) ให้ผลลัพธ์ที่ดีที่สุด
- การถอดรหัสระดับของการออพติไมซ์แบบ speed critical (File) ให้ผลลัพธ์ที่ดีที่สุด

การทดลองที่ 5.3 การทดลองเพื่อทดสอบความถูกต้องของโปรแกรม หลังแก้ไขให้โปรแกรมมีคุณสมบัติ Reentrant และ Relocatable

วัตถุประสงค์

1. เพื่อตรวจสอบว่าชุดเข้ารหัสและถอดรหัสที่ได้แก้ไขให้เป็นโปรแกรมที่เป็น Reentrant และ Relocatable แล้วสามารถใช้งานได้อย่างถูกต้อง

2 เป็นพื้นฐานในการพัฒนาโปรแกรมแบบ Multi-channel

วิธีการทดลอง

1. ทำการแก้ไขโปรแกรมการเข้ารหัสและถอดรหัสให้มีคุณสมบัติของการ Reentrant โดยจัดกลุ่มกลุ่มของตัวแปรที่เป็นโกลบอลและสแตติกให้อยู่ในโครงสร้างตามที่ได้ออกแบบไว้

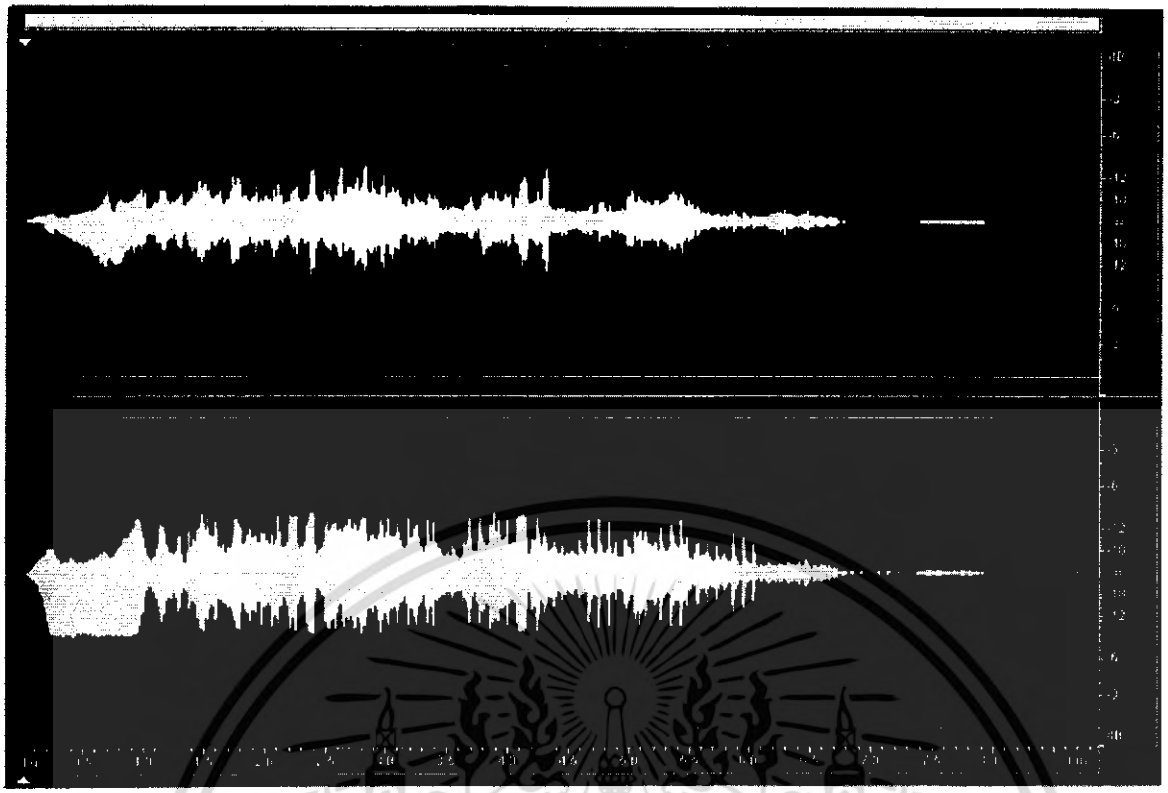
2. ทำการแก้ไขโปรแกรมการเข้ารหัสและถอดรหัสให้มีคุณสมบัติของการ Relocatable โดยแก้ไขตัวแปร pointer ในโครงสร้างของเราให้เก็บ offset ที่ชี้ไปยังหน่วยความจำส่วนที่อ้างถึงแทนที่จะเก็บ Absolute address โดยจะปรับการชี้ตำแหน่งเมื่อเรียกใช้

3. นำไฟล์เสียงที่เป็นแบบ 2 ช่องสัญญาณ มาทำการเข้ารหัสและถอดรหัส

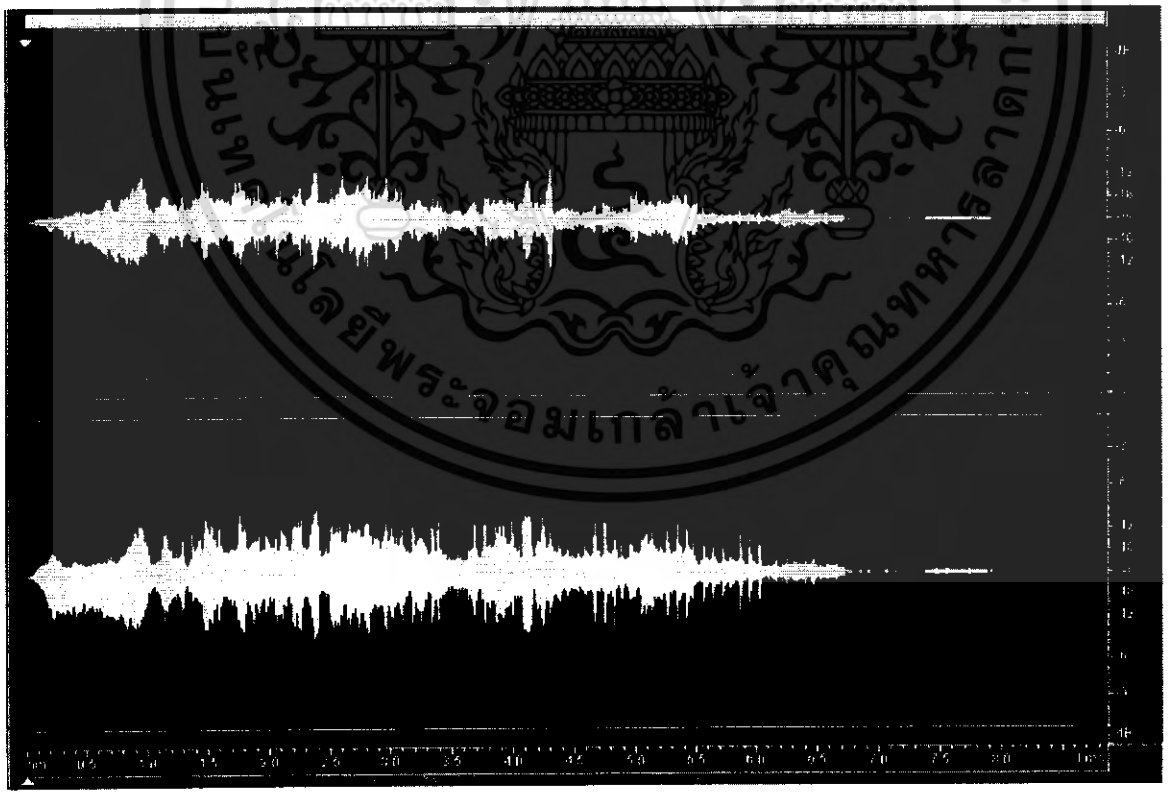
4. ตรวจสอบสัญญาณเสียงเอาท์พุทที่ได้จากการถอดรหัสว่าสัญญาณเอาท์พุตที่ได้เป็นแบบ 2 ช่องสัญญาณเช่นเดียวกับอินพุท

ผลการทดลอง

สัญญาณเสียงเอาท์พุทที่ได้จากการถอดรหัสว่าสัญญาณเอาท์พุตที่ได้เป็นแบบ 2 ช่องสัญญาณเช่นเดียวกับอินพุท



รูปที่ 5-7 รูปสัญญาณ Input ที่ใช้ในการ ปรับปรุงให้มีคุณสมบัติ Reentrant และ Relocatable



รูปที่ 5-8 สัญญาณ Output ที่ได้จากโปรแกรมหลังจากแก้ไขให้มีคุณสมบัติ Reentrant และ Relocatable แล้ว

<u>สรุปผลการทดลอง</u>

ผลที่ได้จากการทดลองทำ Reentrant และ Relocatable ทำให้สัญญาณเอาท์พุตที่ได้เป็นแบบ 2 ช่องสัญญาณเช่นเดียวกับอินพุท สรุปได้ว่าโปรแกรมของเรานั้นเป็น Reentrant แล้วเพราะหากโปรแกรมยังมีการเก็บสถานะโดยอาศัยตัวแปรชนิดโกลบอลอยู่ผลลัพธ์ที่ได้จากการเข้ารหัสและถอดรหัสของไฟล์ นั้นต้องมีผลลัพธ์ที่แตกต่างไปจากชุดต้นฉบับ และ Reentrant เพราะหากโปรแกรมยังมีการเก็บสถานะโดยอาศัยตัวแปรชนิดโกลบอลอยู่รวมทั้งมีการย้ายตำแหน่งของบล็อคของหน่วยความจำ (memory block) ผลลัพธ์ที่ได้จากการเข้ารหัสและถอดรหัสของไฟล์ 2 ไฟล์นั้นต้องมีผลลัพธ์ที่แตกต่างไปจากชุดต้นฉบับ

จากคุณสมบัติเรื่องของการทำ Relocatable และ Reentrant ทำให้โปรแกรมของเราสามารถสร้างเป็นอัลกอริทึม แบบมัลติแชนเนิลได้

การทดลองที่5.4 การทดลองทำการวัดเวลาสำหรับโมดูล ต่าง ๆ สำหรับการเข้ารหัส

<u>วัตถุประสงค์</u>

เพื่อวัดว่าโมดูล ใดมีการทำงานมากที่สุดสำหรับการเข้ารหัสเพื่อเป็นแนวทางในการเลือกโมดูล สำหรับการออพติไมซ์ต่อไป

<u>วิธีการทดลอง</u>

1. ทำการ enable clock ของระบบก่อน จาก Menu profiler->enable clock
2. เลือกเครื่องมือที่ใช้ในการวัด performance จาก Menu profiler->start new session

<u>ผลการทดลอง</u>

Module	เวลาเฉลี่ยต่อเฟรม (us)
d4i40_17()	1155.91
cor_h_cp()	63
ACELP_codebook()	16.51
corr_xy2()	8.94
cor_h_x()	35.99
update_bwd()	7.03
tst_bwd_dominant()	5.43
ener_DB()	23.27
vad()	72.81
MakeDec()	4.82

ตารางที่ 5-2 แสดงจำนวนครั้งในการเรียกใช้งานฟังก์ชัน เรียงลำดับการเรียกใช้จากมากไปน้อย

<u>สรุปผลการทดลอง</u>

1. พบว่าการคำนวณส่วนใหญ่จะอยู่ในฟังก์ชั่น d4i40_17() ซึ่งใช้ในการหาตำแหน่งของ พัลซ์ ที่เป็นไปได้จำนวน 4 พัลซ์ในสัญญาณสุ่ม (unvoiced sound)
2 เราควรที่จะออพติไมซ์โมดูล ในส่วนของการค้นหา fix codebook

การทดลองที่ 5.5 การทดลองทำการวัดเวลาหลังจากทำการออพติไมซ์ตามหัวข้อวิจัยของ F.-K CHEN, J.-F YANG และ Y. –L YAN เรื่อง Candidate scheme for fast ACELP search

<u>วัตถุประสงค์</u>

เพื่อวัดเวลาหลังจากการออพติไมซ์เปรียบเทียบกับก่อนการออพติไมซ์เวลาที่วัดได้จากการเข้าและถอดรหัสรวมกัน

<u>วิธีการทดลอง</u>

1. ทำการสร้างวัตถุของ STS ของ DSP Bios เพื่อใช้ในการช่วยวัดเวลา
2. กำหนด environment ให้ทั้งโค้ดและเซคชันข้อมูลทำงานอยู่ที่หน่วยความจำภายนอก
3. แทรกโค้ดสำหรับวัดเวลาไว้ที่หัวและท้ายของ โมดูล ที่ต้องการวัด
4. เรียก Statistical view ออกมา
5. รันโปรแกรมตามปกติ

<u>ผลการทดลอง</u>

จำนวนพื้นที่ในการออพติไมซ์สำหรับการ search	ALL MAX(mS)	ALL AVG(ms)	ACELP MAX(mS)	ACELP AVG(mS)
8(แบบเดิม)	5.96	4.69	1.16	0.85
7	5.30	4.38	0.96	0.67
6	4.90	4.11	0.82	0.52
5	4.52	4.07	0.70	0.40
4	4.25	3.75	0.46	0.32
3	4.09	3.64	0.35	0.27
2	4.04	3.71	0.30	0.24
1	4.01	3.64	0.28	0.23

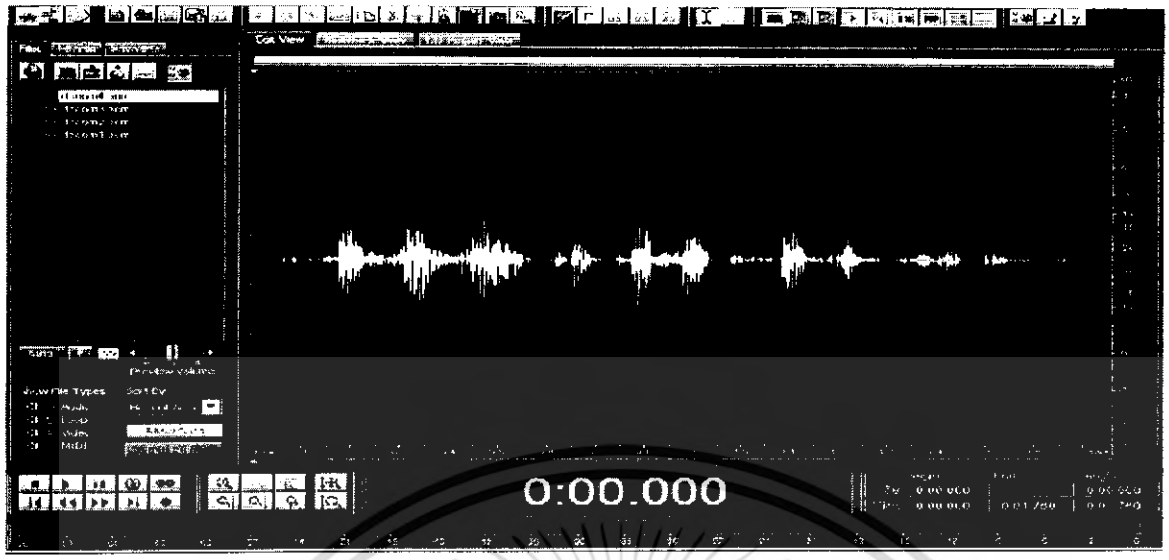
ตารางที่ 5-3 แสดงผลการทดลองการวัดเวลาหลังจากทำการออพติไมซ์โดยใช้วิธีของ Candidate scheme for fast ACELP search

รูปที่ 5-9 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสตามมาตรฐาน

รูปที่ 5-10 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสที่ใช้จำนวนพื้นที่สำหรับการค้นหา = 1

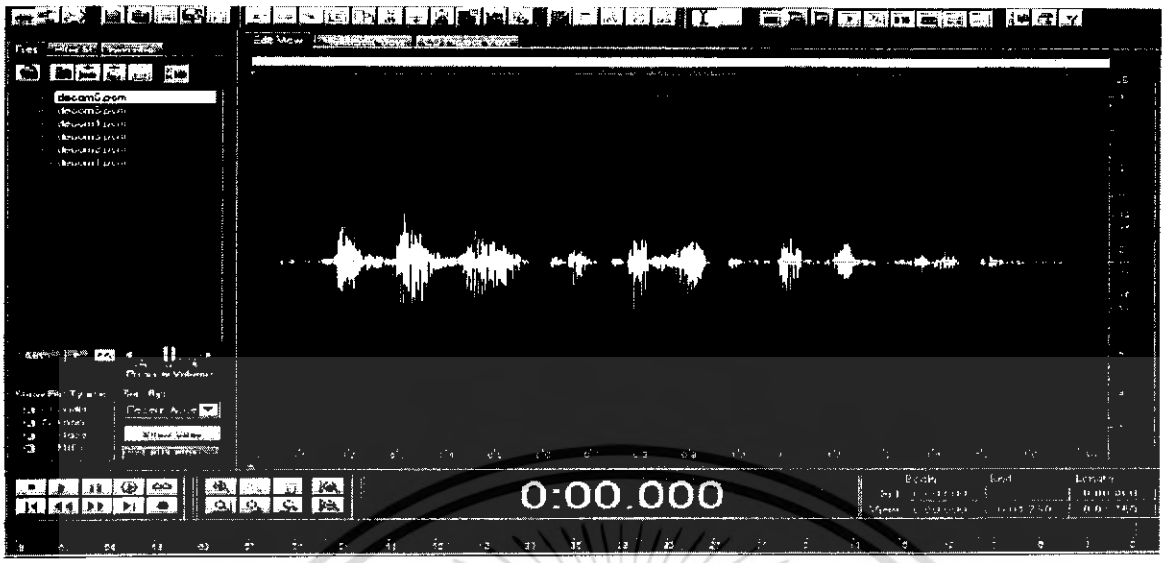
รูปที่ 5-11 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสที่ใช้จำนวนพื้นที่สำหรับการค้นหา = 2

รูปที่ 5-12 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสที่ใช้จำนวนพื้นที่สำหรับการค้นหา = 3



รูปที่ 5-13 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสที่ใช้จำนวนพื้นที่สำหรับการค้นหา = 4

รูปที่ 5-14 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสที่ใช้จำนวนพื้นที่สำหรับการค้นหา = 5



รูปที่ 5-15 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสที่ใช้จำนวนพื้นที่สำหรับการค้นหา = 6

รูปที่ 5-16 แสดงสัญญาณเสียงที่ถูกถอดรหัสโดยชุดถอดรหัสที่ใช้จำนวนพื้นที่สำหรับการค้นหา = 7

<u>สรุปผลการทดลอง</u>

1. พบว่าจำนวนพื้นที่สำหรับการค้นหา ลดลงทำให้ความเร็วในการเข้าและถอดรหัสสัญญาณเสียงเร็วขึ้น

2. ถึงแม้ว่าจำนวนพื้นที่สำหรับการค้นหา จะลดลงแต่ลักษณะของสัญญาณเสียงแทบจะไม่แตกต่างจากของเดิม และเมื่อเปรียบเทียบจากการฟังพบว่าหูคนไม่สามารถแยกความแตกต่างนี้ได้

3. เราตัดสินใจเลือกการออพติไมซ์ด้วยวิธีนี้โดยเลือกพื้นที่สำหรับการค้นหา = 1 เพราะการเข้าและถอดรหัสสามารถทำได้เร็วที่สุดโดยที่คุณภาพของเสียงที่ได้ยังยอมรับได้

บทที่ 6
วิเคราะห์และวิจารณ์

จากการศึกษาเราได้เรียนรู้ว่าการเข้ารหัสสัญญาณเสียงมีหลายประเภทอันได้แก่เวฟฟอร์มควอนไทต์เซชัน (Wave form quantization) และพาราเมทริกซ์ควอนไทต์เซชัน (Parametric quantization) ซึ่งแต่ละวิธีนั้นก็เหมาะกับการประยุกต์ใช้งานที่แตกต่างกันเช่น งานที่ต้องการสัญญาณเสียงที่มีคุณภาพสูงและไม่สนใจเรื่องของการส่งผ่านเครือข่ายก็ควรใช้เวฟฟอร์มควอนไทต์เซชันในขณะที่การส่งสัญญาณเสียงผ่านเครือข่ายนั้นจะเหมาะกับการเข้ารหัสเสียงแบบพาราเมทริกซ์ควอนไทต์เซชันเนื่องจากข้อมูลที่ใช้ในการส่งมีขนาดเล็กกว่า เวฟฟอร์มควอนไทต์เซชันแต่ก็มีข้อเสียที่ต้องอาศัยทฤษฎีการคำนวณมากทำให้เราต้องใช้เวลาในการศึกษาการทำงานของการเข้ารหัสและถอดรหัส ในขณะเดียวกันนั้น แอพพลิเคชันที่ทำงานเกี่ยวกับสัญญาณเสียงนั้นส่วนใหญ่จะเป็นแบบเรียลไทม์แอพพลิเคชันซึ่งเราต้องคำนึงถึงเดทไลน์ (Deadline) และความซับซ้อนในการประมวลผลสัญญาณซึ่งถ้าการประมวลผลนั้นเกินระยะเวลาเดทไลน์ออกไปจะมีผลต่อคุณภาพเสียงที่ได้รับ ดังนั้นจึงมีความจำเป็นที่จะต้องลดความซับซ้อนในการประมวลผลลง โดยในโครงงานนี้ได้ศึกษาวิธีการจากวารสาร IEEE และนำทฤษฎีมาประยุกต์ใช้บน DSP ไมโครโพรเซสเซอร์พบว่าสามารถลดความซับซ้อนลงได้อย่างมากแต่ก็ต้องแลกกับคุณภาพของสัญญาณเสียงที่ลดลงทั้งนี้ต้องขึ้นอยู่กับผู้นำเอาอัลกอริทึมไปใช้งานว่าจะยอมรับได้ถึงระดับไหน

สำหรับในโครงงานนี้ซึ่งเป็นการการเข้ารหัส / ถอดรหัส ตามมาตรฐานของ ITU-T G.729 ซึ่งเป็นการเข้ารหัสสัญญาณเสียงแบบ 8 kbit/s โดยใช้ CS-ACELP โดยแบ่งเสียงออกเป็นเฟรมซึ่งแต่ละเฟรมได้มาจากการแซมปลิง สัญญาณเสียงด้วยอัตราการแซมปลิง 8000 samples/s โดยทำการควอนไทต์ข้อมูลแบบ 16 บิต PCM สำหรับผลที่ได้ชุดเข้ารหัสและถอดรหัส G.729 ที่โปรแกรมลงบน ดี เอส พี บอร์ด ทำงานได้อย่างถูกต้อง และสามารถที่จะประมวลผลสัญญาณ 2 ช่องสัญญาณได้ในเวลาเดียวกัน และทำให้สามารถประมวลผลได้เร็วขึ้นโดยการทำออพติไมต์โปรแกรม ซึ่งทำให้การทำงานแบบเรียลไทม์ได้ผลที่ดีขึ้น และ โดยเมื่อพิจารณาอย่างละเอียดพบว่า สัญญาณเสียงที่ผ่านชุดเข้ารหัสและถอดรหัส G.729 จะมีลักษณะของสัญญาณที่แตกต่างไปจากสัญญาณเสียงต้นฉบับเล็กน้อย เพราะโมดูล G.729 เป็นการเข้ารหัสสัญญาณแบบ lossy ซึ่งเป็นการเข้ารหัสสัญญาณที่ยอมให้ผลลัพธ์หลังจากการถอดรหัสกลับมาแล้วไม่จำเป็นต้องเหมือนกับสัญญาณต้นฉบับทุกประการ

แนวในการพัฒนา

หลังจากที่เราได้พัฒนาโปรแกรมเรียบร้อยแล้วพบว่าการนำโปรแกรมเข้าไปใช้ร่วมกับ แอพพลิเคชันอื่นนั้นทำได้ยากเนื่องจากโปรแกรมที่ได้พัฒนามายังไม่มีฟังก์ชัน มาตรฐานในการติดต่อระหว่าง แอพพลิเคชันดังนั้นเราจึงได้ศึกษาการโปรแกรมตามมาตรฐานของผู้ผลิต (Texas instrument Standard) และได้ปรับปรุงโปรแกรมให้เป็นตามมาตรฐานของผู้ผลิต โดยการนำไลบรารี ที่เกี่ยวข้องเช่น DTMF, Echo Canceller นำมา integrate เข้าด้วยกันเป็นระบบ VOIP ได้ในที่สุด

บรรณานุกรม

[1] ITU-T Recommendation G.729 Annex A , B and C+ , 1996

[2] Venkatraman Atti and Andreas Spanias,A Simulation tool for introducing algebraic CELP (ACELP) coding concepts in a DSP course ,

[3] Antti Kiviluoto, Speech Coding Standards

[4] Climent Nadeu, Speech analysis

[5] Jonathan D. Rosenberg, Lucent Technologies, Bell Laboratories & Columbia University, G.729 Error Recovery for Internet Telephony

[6] F.-K Chen, J.-F Yang and Y.-L Yan, Candidate scheme for fast ACELP search, IEEE 2002

[7] TMS320 DSP Algorithm Standard API Reference, Literature Number: SPRU360C

[8] TMS320 DSP Algorithm Standard Rule & Guidelines, Literature Number: SPRU352E

[9] TMS320 DSP Algorithm Standard Algorithm Standard Developer's Guide , Literature Number: SPRU424C

[10] A Consumer's Guide to using eXpressDSP-Compliant Algorithms for DSP Software Products, Literature Number: SPRA810

[11] A Technical Overview of eXpressDSP-Complianct Algorithms for DSP Software Producers, Literature Number: SPRA579C

[12] Using the TMS320 DSP Algorithm Standard in Static System, Literature Number: SPRA557B

[13] Using the TMS320 DSP Algorithm Standard in Dynamic System, Literature Number: SPRA580B

[14] Techniques for implementing Shared Relocatable Buffers Using the TMS320 DSP Algorithm Standard, Literature Number: SPRA790

[15] TMS320C6000 Assembly Language Tools User's Guide, Literature Number: SPRU186I , April 2001

ภาคผนวก

ภาคผนวก ก คู่มือการใช้งาน Code composer studio V2.0

ก. คู่มือการใช้งานโปรแกรม Code composer studio V2.0

ในการใช้งาน Code composer studio ทาง TI มีตัวอย่างโปรแกรมสำหรับผู้เริ่มต้นได้ฝึกทดลองใช้เครื่องมือต่าง ๆ ที่ code composer มีให้ซึ่งโปรเจคสำหรับสอน (Project tutorial) อยู่ในโฟลเดอร์ของ path ที่ลง code composer\ti\tutorial

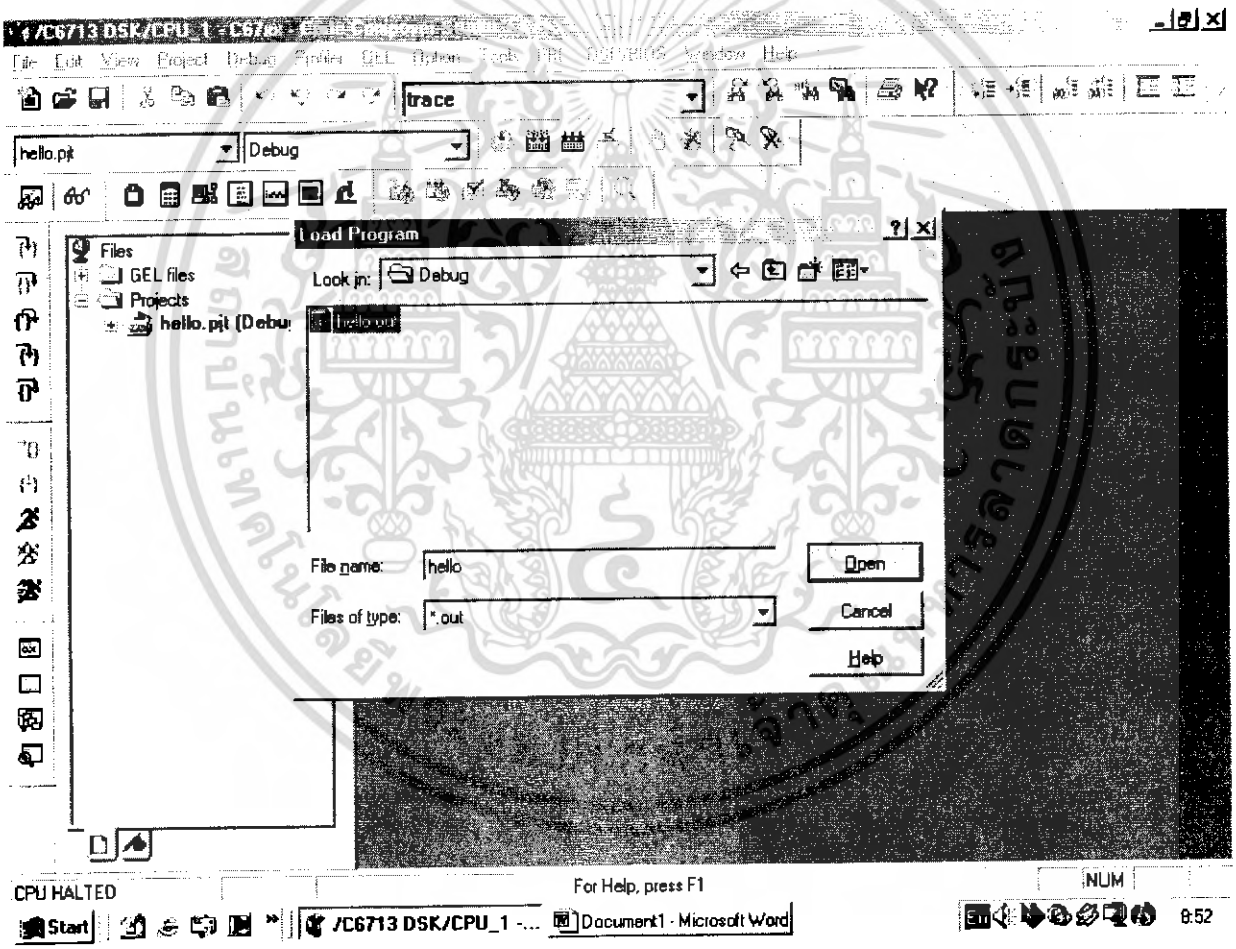
ก.1 การทดลอง Hello World (...\ti\tutorial\dsk6713\hello1)

เป็นการเรียกใช้ไลบรารี พื้นฐานของภาษา C ธรรมดา ได้แก่ printf, scanf โดยผ่าน lib stdio.h

วิธีการ Compile

เลือก Project -> Rebuild all

ทำการโหลดโปรแกรมโดยผ่าน FILE -> Load Program -> Debug -> *.out open



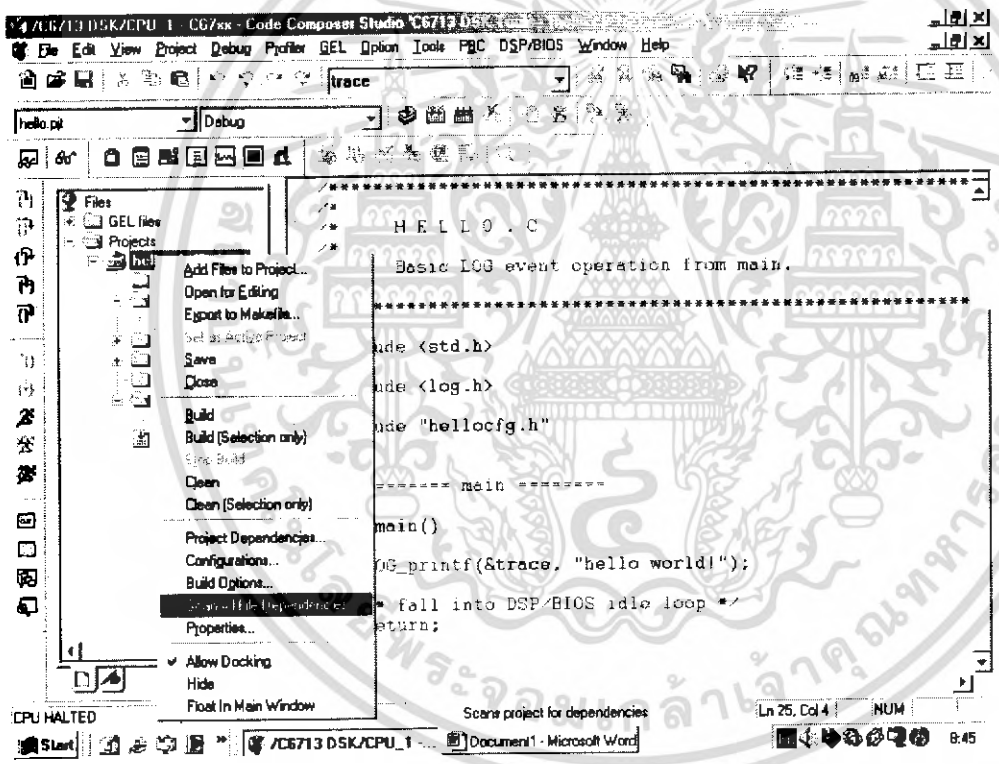
รูปที่ ก-1 แสดงโหลดไฟล์ลงบน ดี เอส พี บอร์ด

ก.2 การทดลอง Hello World 2 (...\ti\tutorial\dsk6713\hello2)

เนื่องจากการเรียกใช้ไลบรารี พื้นฐานของภาษา C มี Overhead ในการทำงานสูงดังนั้นดังนั้นจึงเปลี่ยนมาใช้ฟังก์ชัน ของ DSP BIOS สำหรับการใช้แทนฟังก์ชัน printf โดยใช้ LOG_Obj

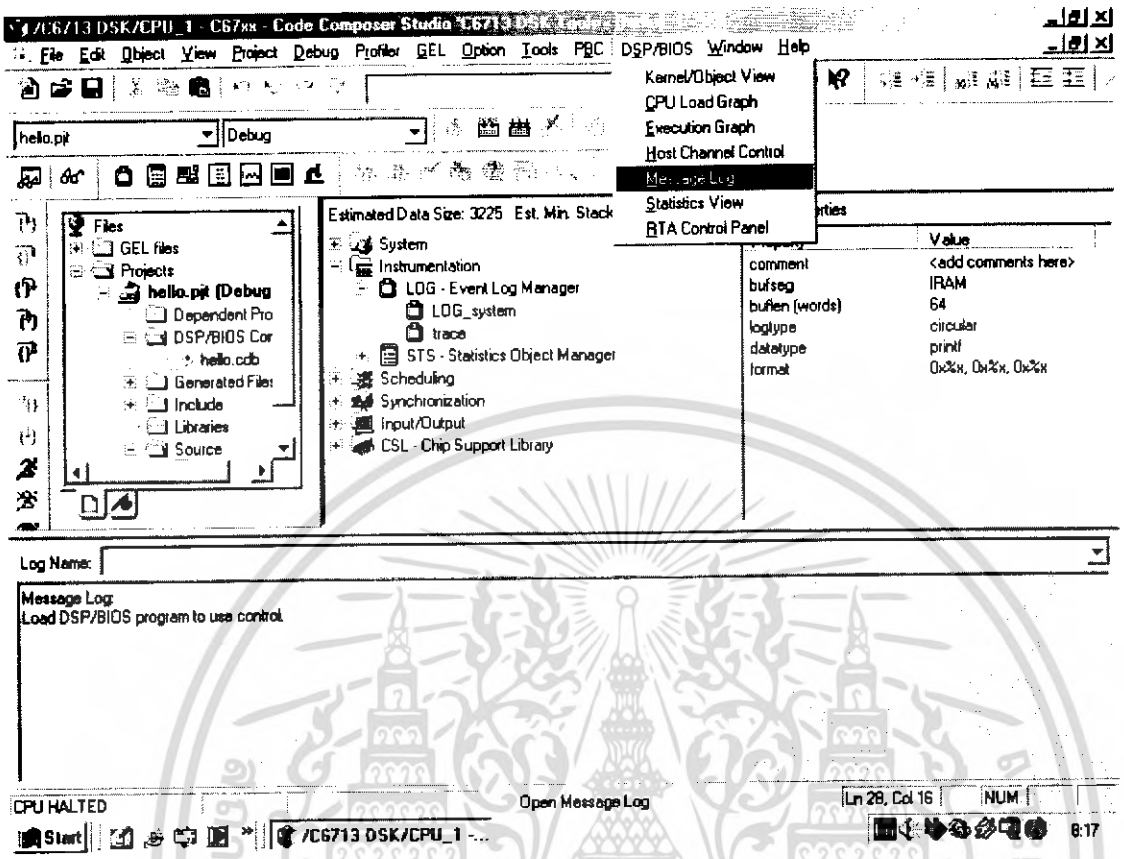
LOG_printf (&trace, "hello world!"); // trace เป็นชื่อวัตถุของ Log

สำหรับวิธีการสร้างนั้นเริ่มต้นเข้าไปที่ FILE -> NEW-> DSP BIOS Configuration เลือกตระกูลของบอร์ดทดลองที่ใช้ ในที่นี้ได้ใช้ dsk6713.cdb ซึ่งเป็นไฟล์ สำหรับเก็บคอนฟิกกูเรชันจากนั้นเลือก Instrumentation-> -> LOG – Event Log Manager click ขวาเลือก Insert Log ทำการตั้งชื่อของวัตถุและกำหนดพารามิเตอร์โดย Click ขวาแล้วเลือก property หลังจากกำหนดพารามิเตอร์เรียบร้อยแล้วจะต้องทำการ scan dependency เพื่อนำไฟล์อื่น ๆ ที่เกี่ยวข้อง Include เข้ามาในโปรเจคโดย Click ขวาที่ ชื่อโปรเจคแล้วเลือก Scan All File Dependency



รูปที่ ก-2 แสดงการทำ Scan dependencies

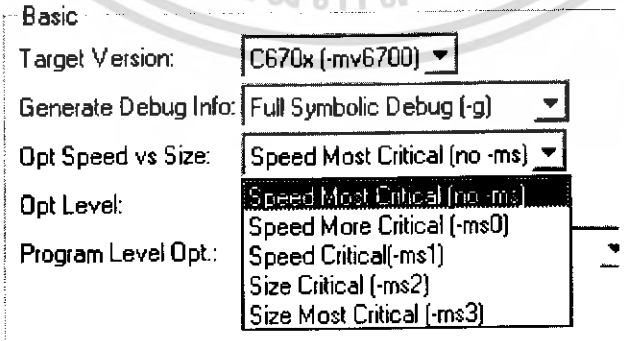
เครื่องมือที่ใช้ในการ view message อยู่ที่ DSP/BIOS -> Message Log ดังรูป



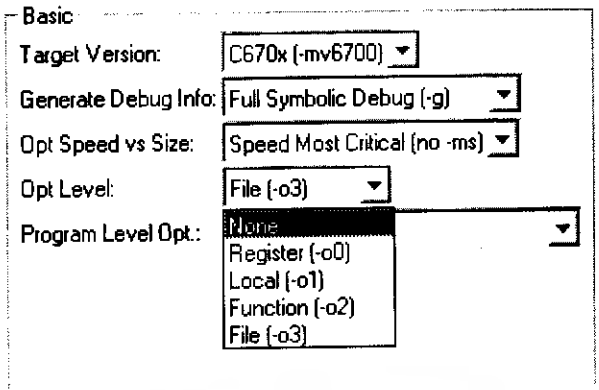
รูปที่ ก-3 แสดงการใช้งาน Message log

ก.3 การทดลอง Compile Optimizer

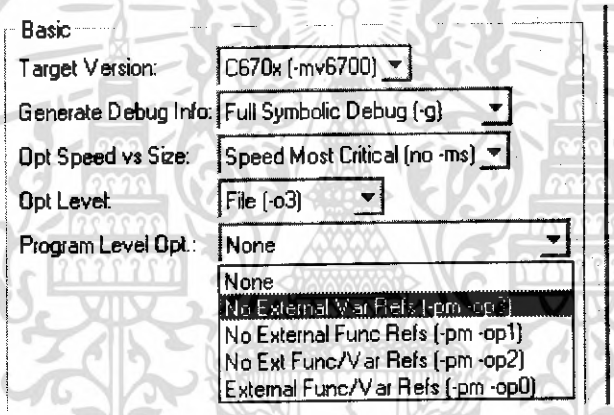
Lab นี้ได้เรียนรู้เกี่ยวกับการ กำหนดพารามิเตอร์ที่ใช้สำหรับการ compile โดยมี trace off อยู่ 2 อย่าง คือหน่วยความจำ และ Speed โดยเลือกออพติไมซ์ตามความเหมาะสมของโปรแกรมเรา โดยอาศัยเครื่องมือของ TI ที่ช่วยในการวัดขนาด รวมทั้งความเร็ว ของส่วนต่าง ๆ ในโปรแกรมรวมทั้งจำนวน Instruction ด้วย



รูปที่ ก-4 แสดงการเลือกระดับของการออพติไมซ์เมื่อเทียบขนาดของไฟล์กับความเร็ว



รูปที่ ก-5 แสดงการเลือกระดับของการออพติไมซ์

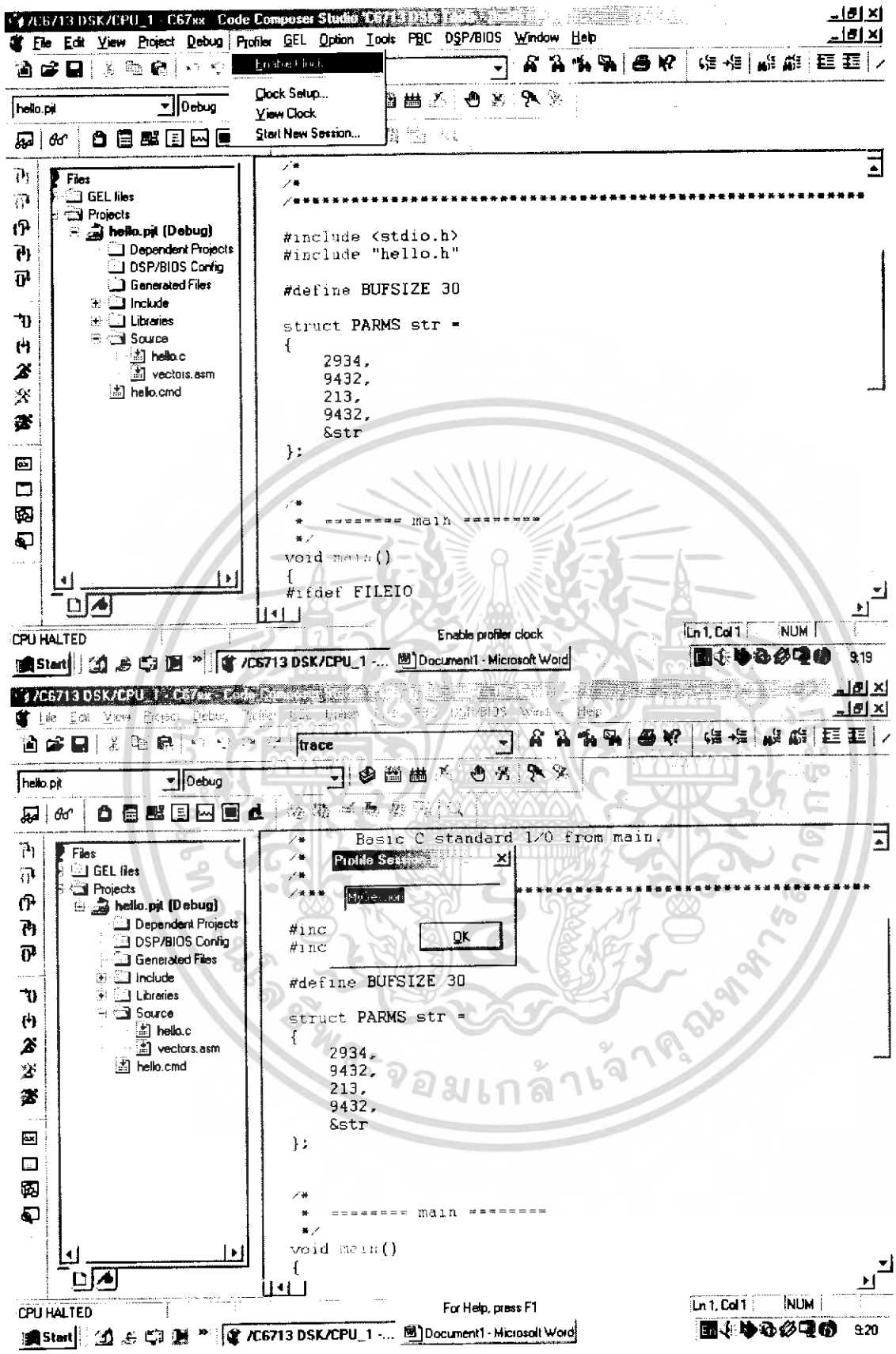


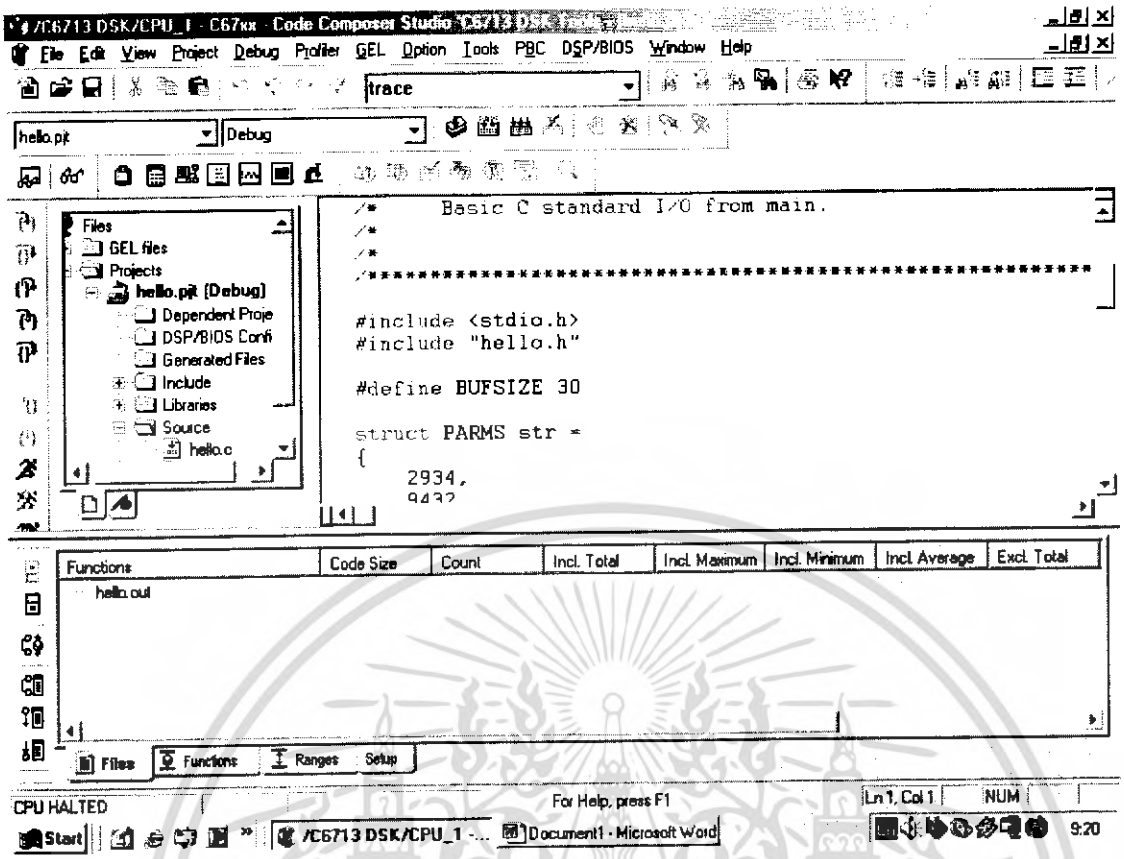
รูปที่ ก-6 แสดงการเลือกระดับของการออพติไมซ์แบบระดับโครงสร้างของโปรแกรม

ในการวัดประสิทธิภาพของโปรแกรม Code composer มีเครื่องมือที่ช่วยในการวัดซึ่งทำได้ตามขั้นตอนต่อไปนี้

1. ทำการ Enable Clock
2. ตั้งชื่อ Session

ซึ่งการวัดประสิทธิภาพนั้น เราสามารถวัดได้ในหลายระดับ เช่นไฟล์,ฟังก์ชัน , Ranges





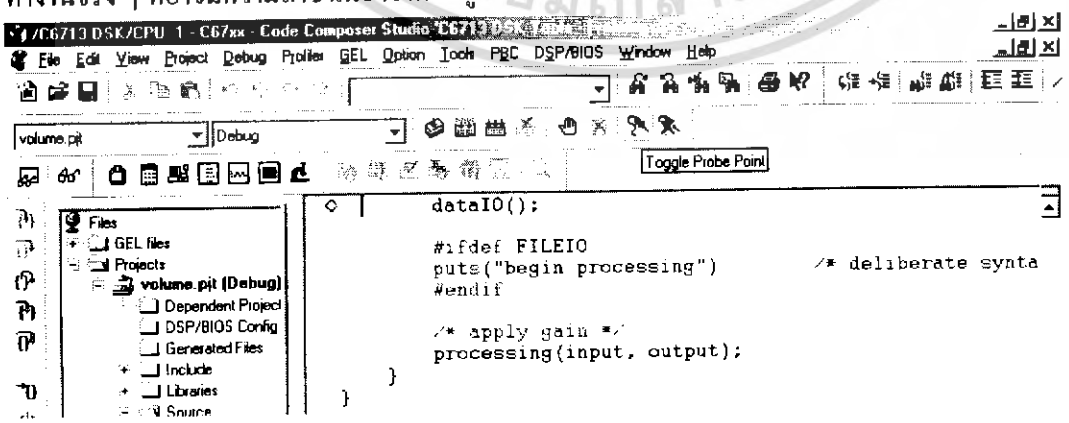
รูปที่ ก-7 แสดงการใช้เครื่องมือประสิทธิภาพของโปรแกรม ในการวัด

ก.4 การทดลอง Volumn1 (...\ti\tutorial\dsk6713\volume1)

<u>วัตถุประสงค์</u>

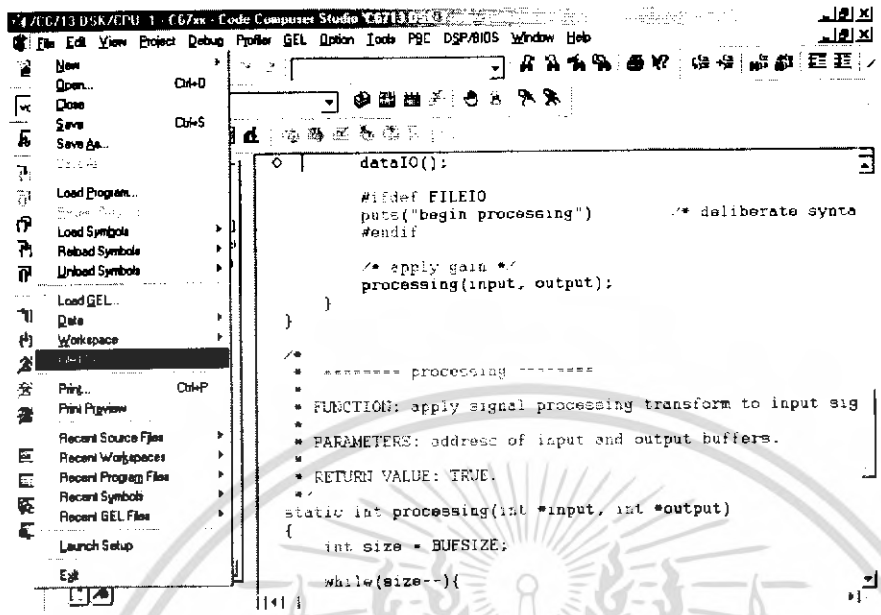
เพื่อศึกษาการใช้ Probe point และโหลดฟังก์ชัน

ในการจำลองการทำงานเพื่อศึกษา Process การทำงานของ อัลกอริทึมต่าง ๆ นั้นเราไม่จำเป็นต้องนำอินพุท จาก source จริงมาใช้ในการ process เราอาจใช้อินพุท จากไฟล์ ที่เตรียมไว้เพื่อใช้ทำการศึกษาได้ จากคุณสมบัติของ probe point และเราสามารถกำหนดโหลดให้กับโปรแกรมของเราได้เพื่อจำลองสภาวะการทำงานจริง ๆ ที่อาจมีความล่าช้าเนื่องจากโมดูล อื่น



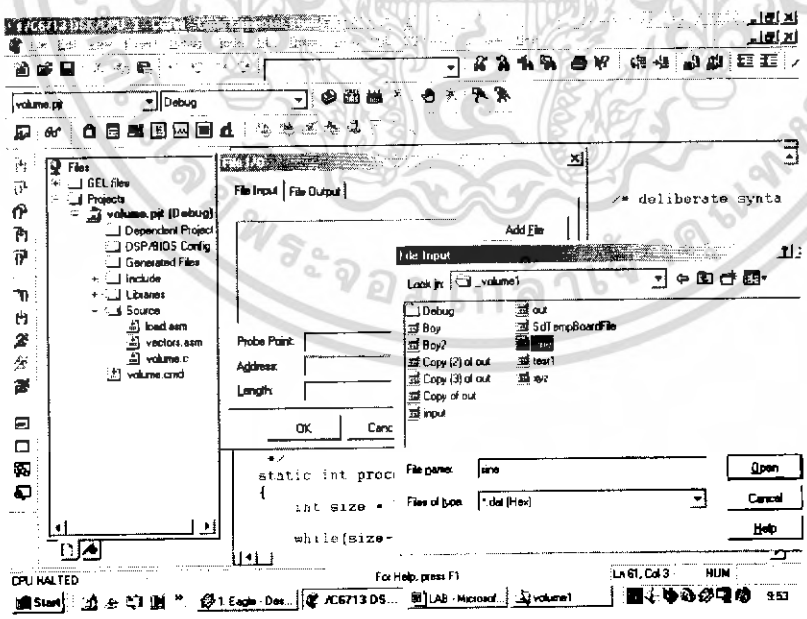
รูปที่ ก-8 แสดงการ add probe point

insert probe point นำ pointer ไปวางไว้ในตำแหน่งที่ต้องการนำสัญญาณที่จะนำไป process click



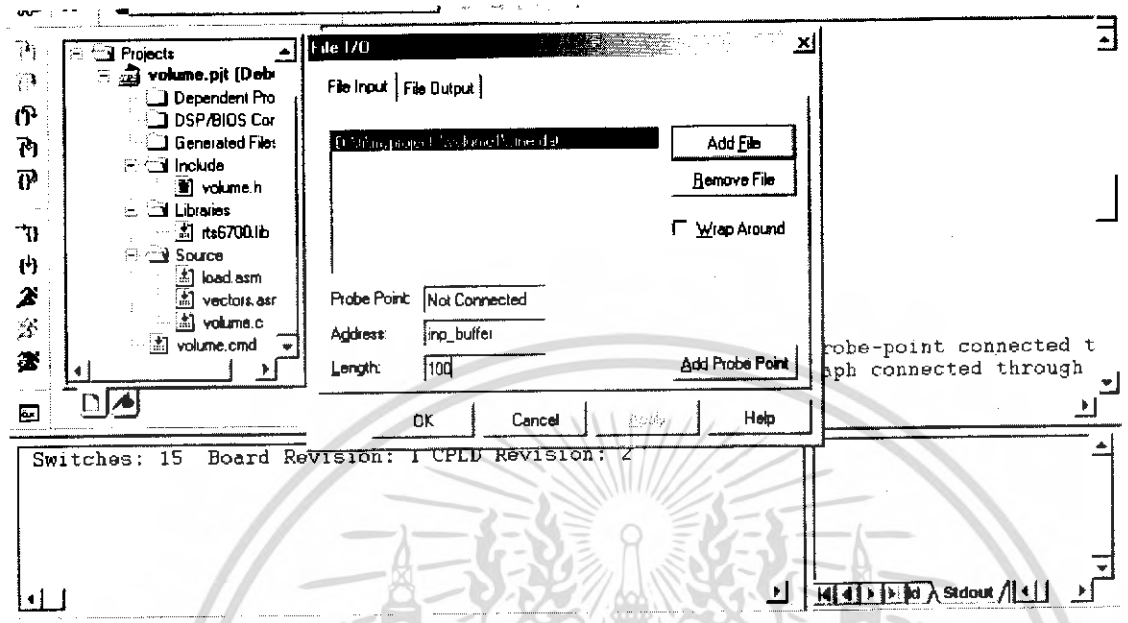
รูปที่ ก-9 แสดงการเลือก Menu file I/O

ทำการเพิ่มไฟล์อินพุท เข้ากับตำแหน่ง probe point โดยเลือกที่ File I/O



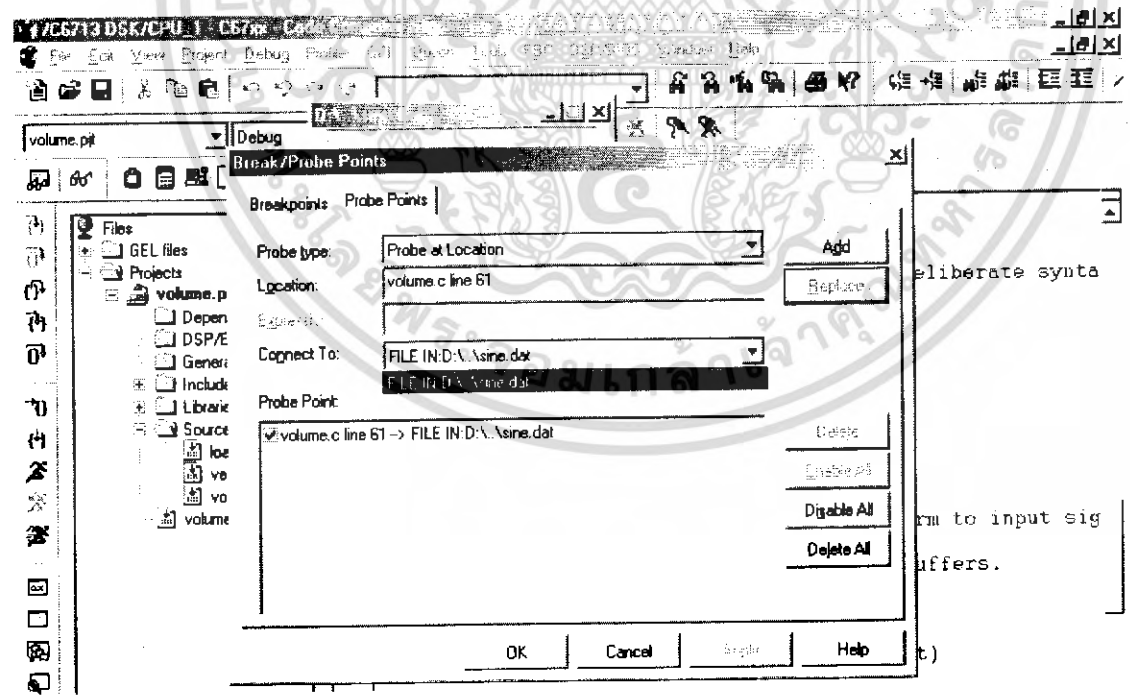
รูปที่ ก-10 แสดงวิธีการ add file I/O

เลือก raw ไฟล์ที่ต้องการใช้เป็นอินพุทพร้อมทั้งกำหนดขนาดของบัฟเฟอร์ที่ต้องการอ่านค่าจาก ไฟล์เข้ามาต่อครั้ง และกำหนดค่าที่อ่านเข้ามาให้กับพารามิเตอร์



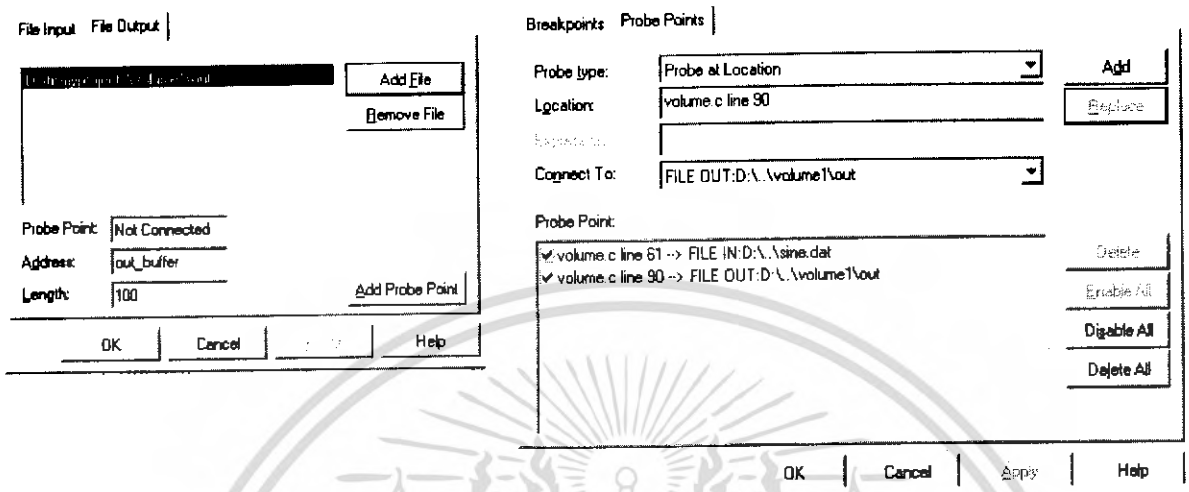
รูปที่ ก-11 แสดงวิธีการ Map ไฟล์อินพุทเข้ากับพารามิเตอร์

ทำการลิงค์ไฟล์ของอินพุทเข้ากับ Probe point



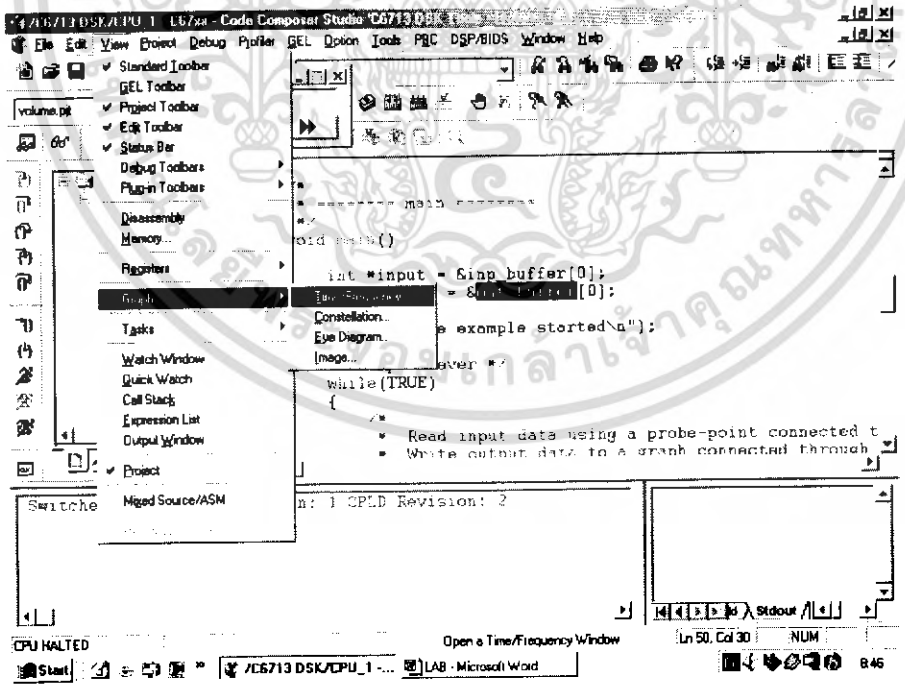
รูปที่ ก-12 แสดงการ Map file เข้ากับ Probe point

ทำลักษณะเช่นเดียวกันกับไฟล์เอาท์พุทที่ต้องการ

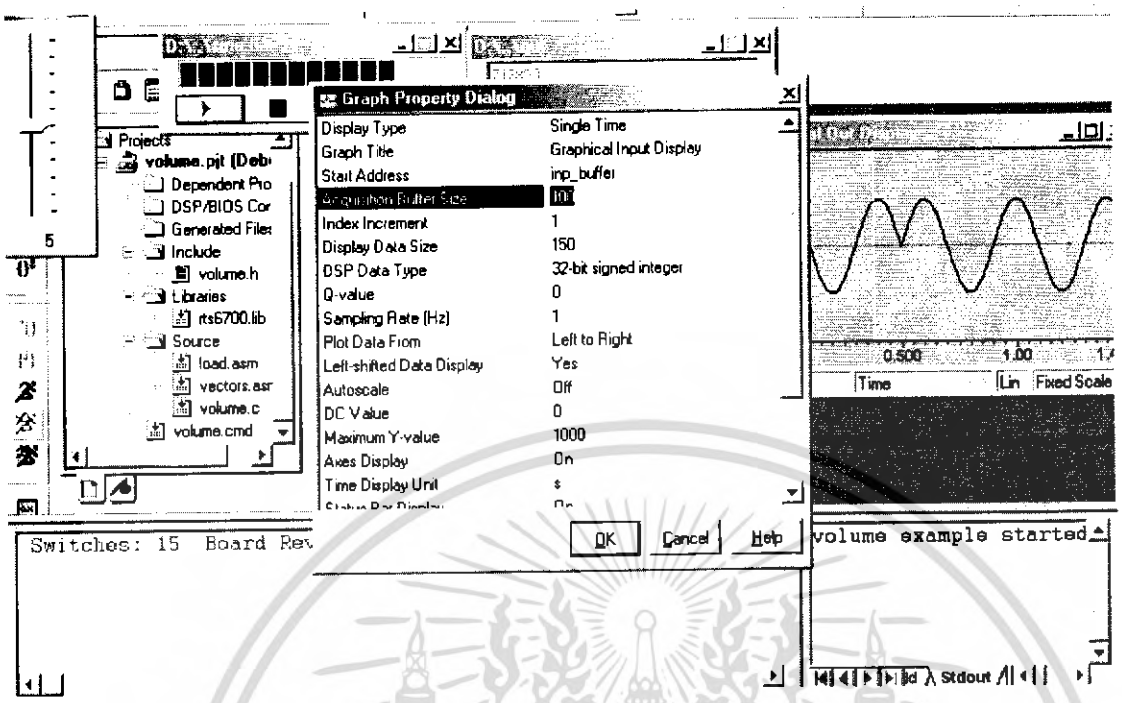


รูปที่ ก-13 แสดงวิธีการคอนฟิก file I/O ของเอาท์พุท

ทำการ Load graph เพื่อแสดงลักษณะของ อินพุทและเอาท์พุท โดยกำหนดพารามิเตอร์ดังนี้

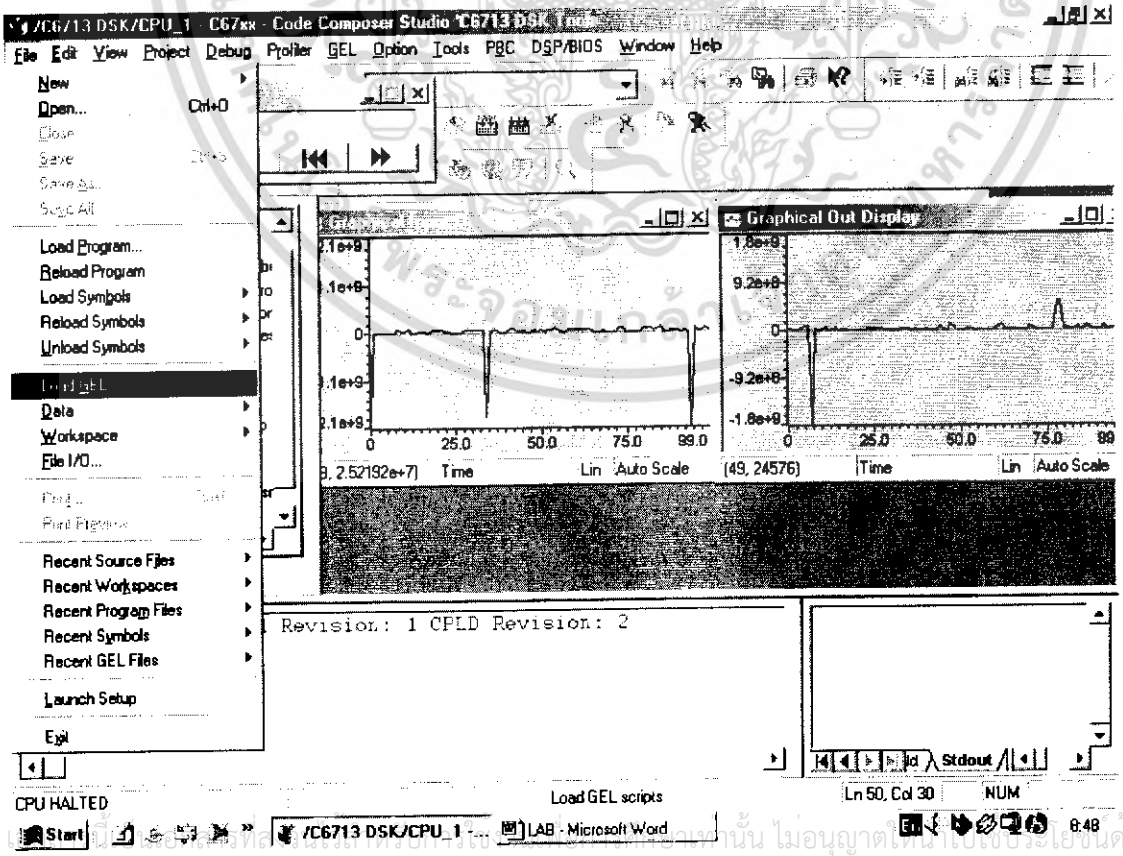


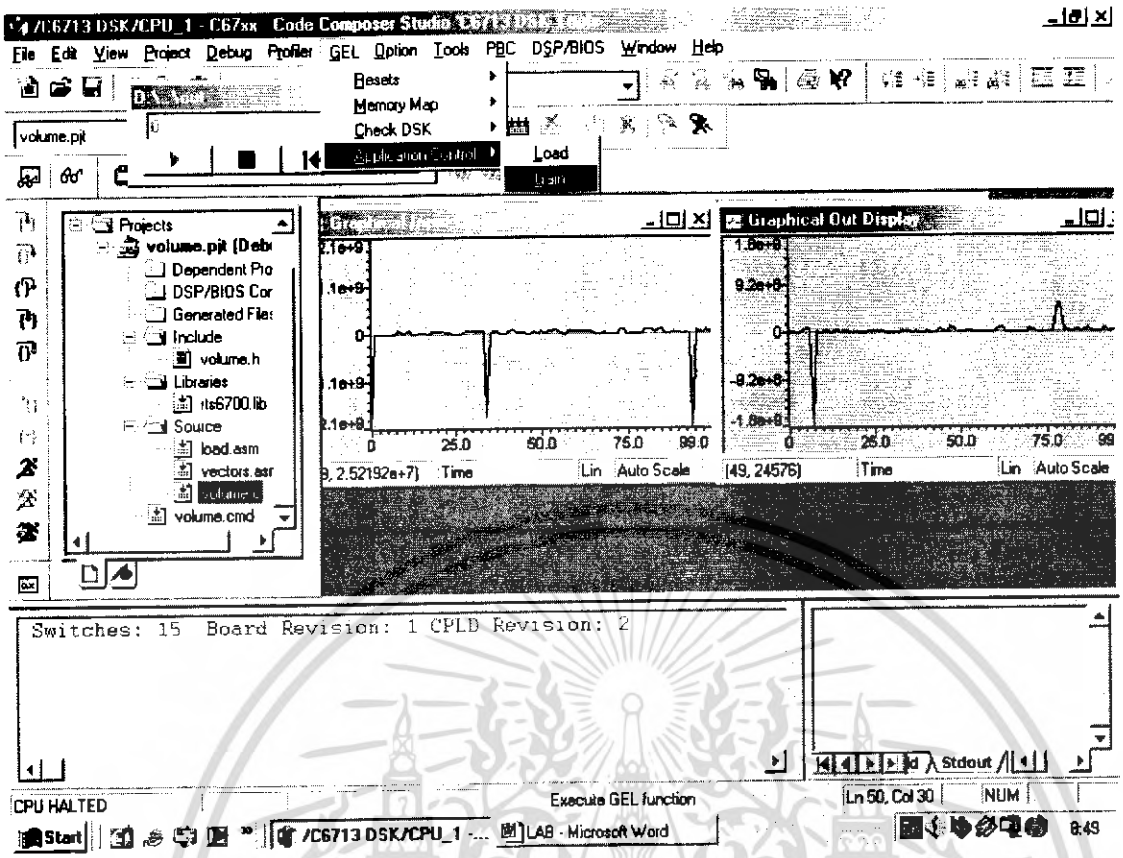
รูปที่ ก-14 แสดงวิธีการเรียกดู Graph ของ Input



รูปที่ ก-15 แสดง properties page ของ graph ที่เลือก

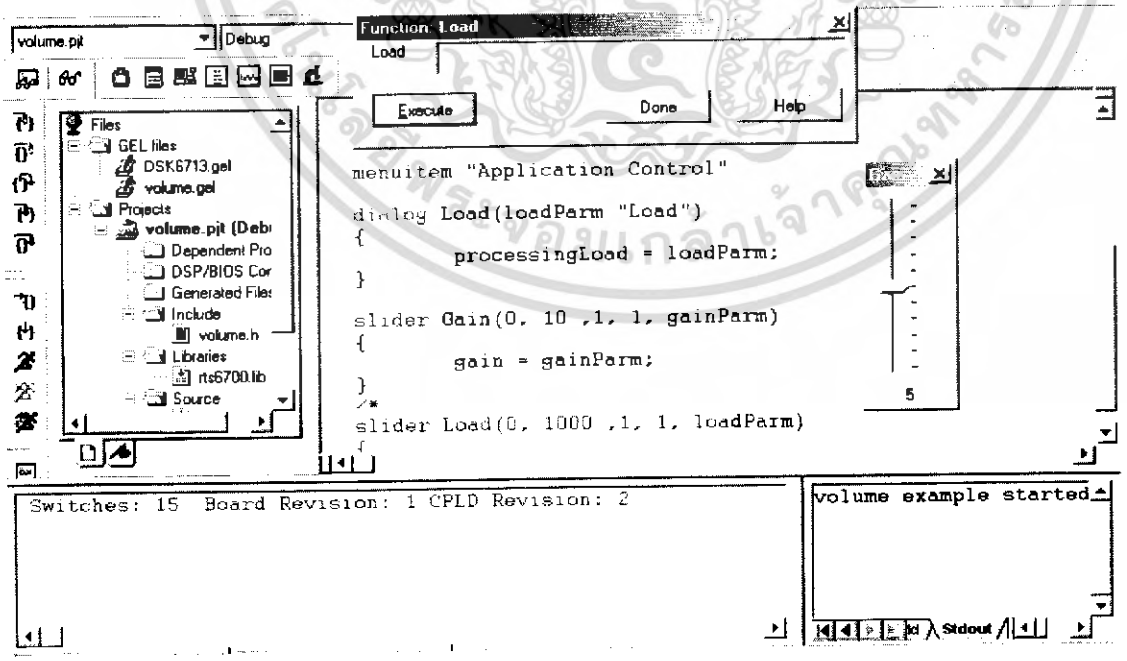
ทำการโหลด GEL เข้ามาเพื่อใช้ในการปรับค่าพารามิเตอร์ขณะรันโปรแกรม



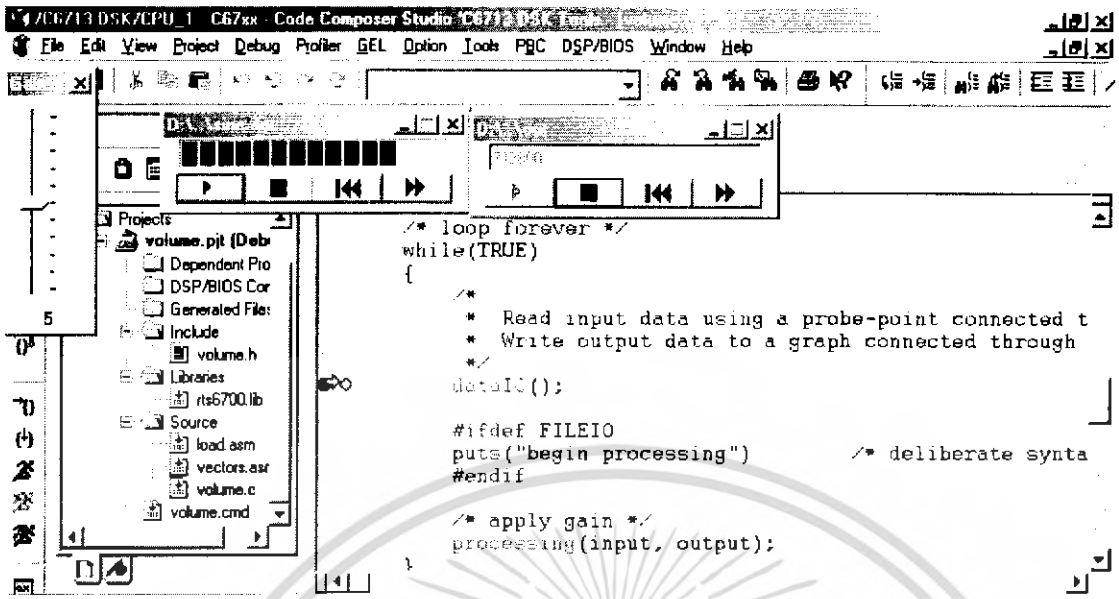


รูปที่ ก-16 แสดงวิธีการเรียกใช้ GEL

ลองเข้าไปดูที่ไฟล์ volume.gel เราอาจเข้าไปทำการแก้ไขหรือเพิ่มได้หลังจากนั้น ทำการบันทึกแล้วโหลดเข้าไปใหม่



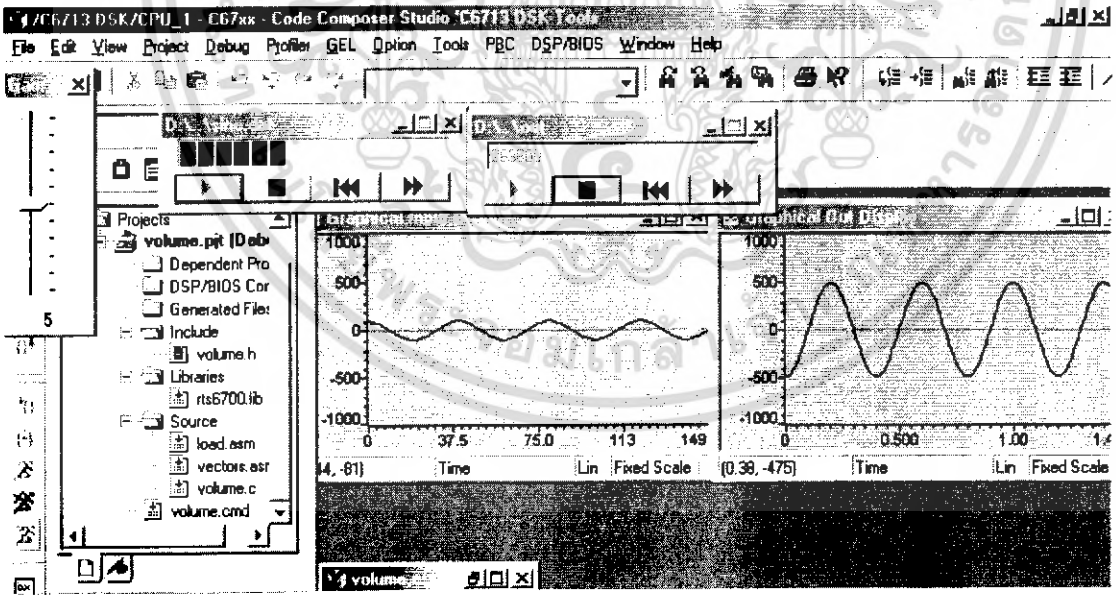
รูปที่ ก-17 แสดงวิธีการใช้ GEL ฟังก์ชัน



รูปที่ ก-18 แสดงการรัน แบบ animate โดยการกำหนด break point

ทำการกำหนด break point เพื่อช่วยในการรัน animate

Rebuild all แล้วทำการโหลดโปรแกรมเพื่อรัน ต่อไป ทดลองปรับอัตราขยายสัญญาณและโหลดสังเกตผลที่ได้



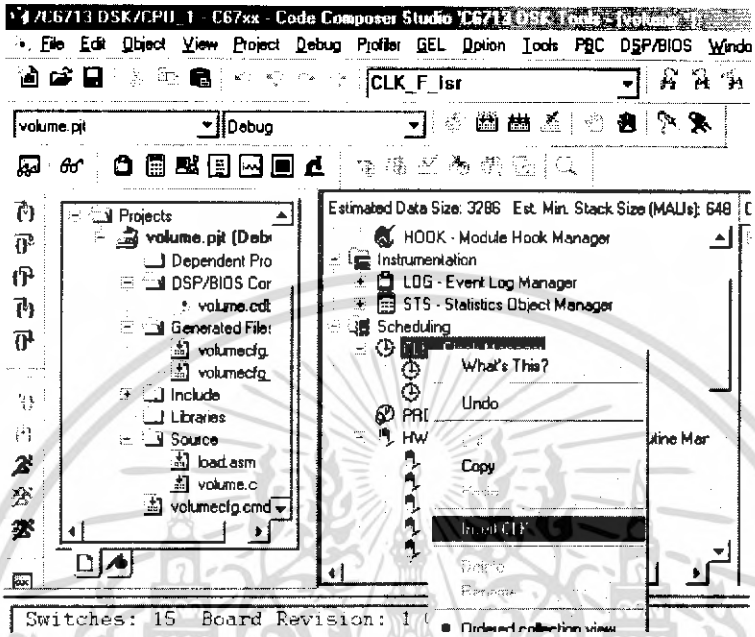
รูปที่ ก-19 แสดง Graph ทั้งอินพุทและเอาท์พุท ที่ได้จากการรัน

ก.5 การทดลอง Volumn 2 (...\tl\tutorial\dsk6713\volume2)

<u>วัตถุประสงค์</u>

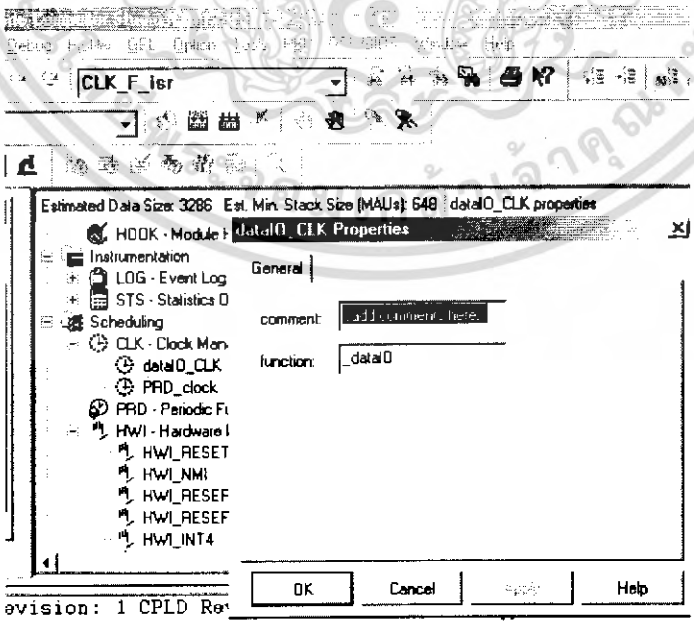
เพื่อศึกษาการใช้งาน software interrupt ผ่านทาง DSP bios

เริ่มต้นทำการสร้าง CLK เพื่อใช้ในการรองรับ HWI Interrupt จาก timer เพื่อ map เข้ากับ SWI



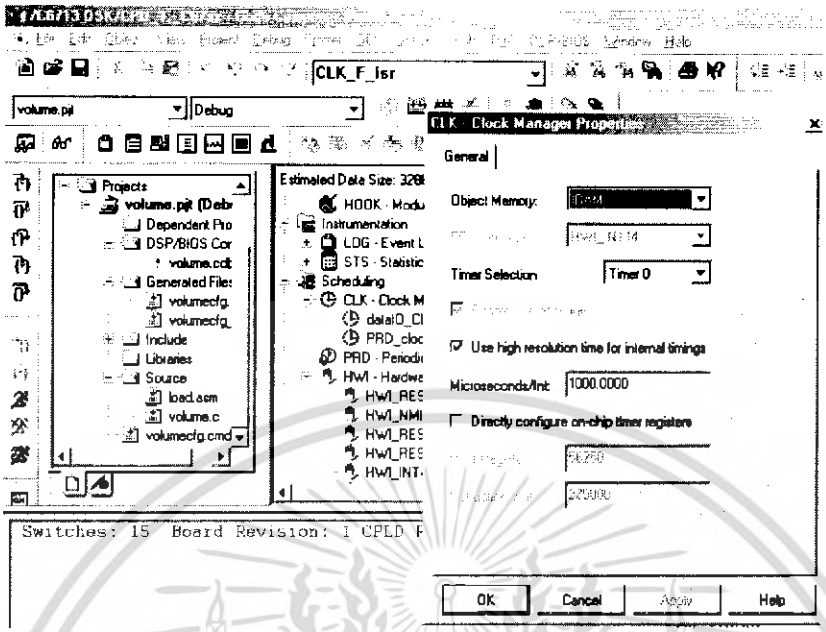
รูปที่ ก-20 แสดงวิธีการสร้างวัตถุของ clock เพื่อใช้ในการ trick ฟังก์ชัน

ทำการกำหนด map ฟังก์ชัน ที่ต้องการให้ทำเมื่อเกิด HWI ที่ property ของวัตถุของ timer ที่สร้างขึ้นมาใหม่ สังเกตว่าฟังก์ชัน ที่เรียกต้องขึ้นต้นด้วย "_" เนื่องจากการเรียกฟังก์ชัน เป็นการเรียกผ่าน assembly code



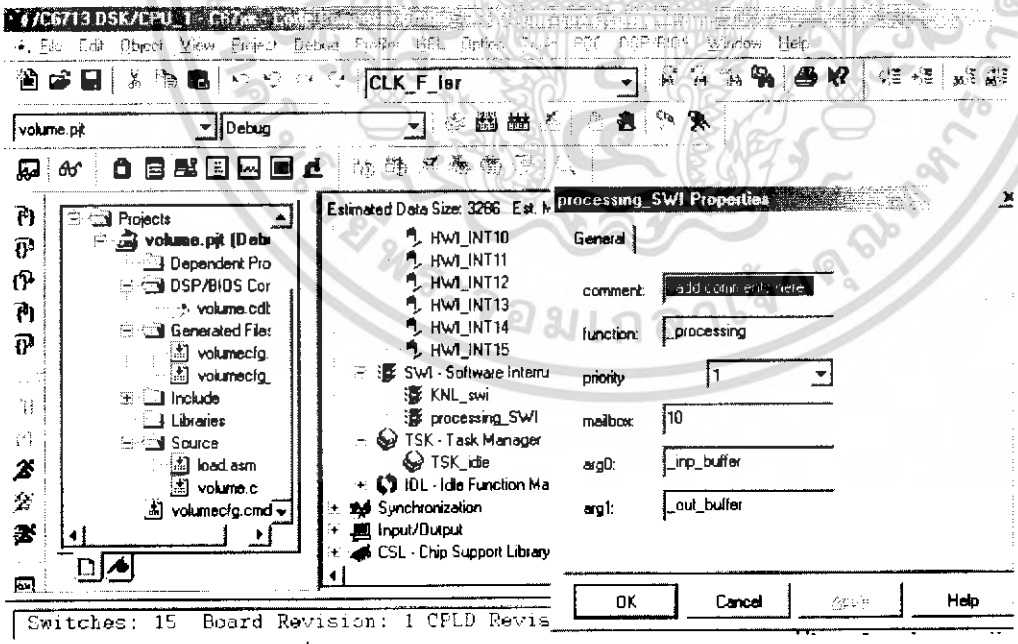
รูปที่ ก-21 แสดงการกำหนดฟังก์ชันที่จะเรียกทำงาน

เราสามารถกำหนด การ Interrupt ของ Hardware ได้ที่ property ของ CLK Manager



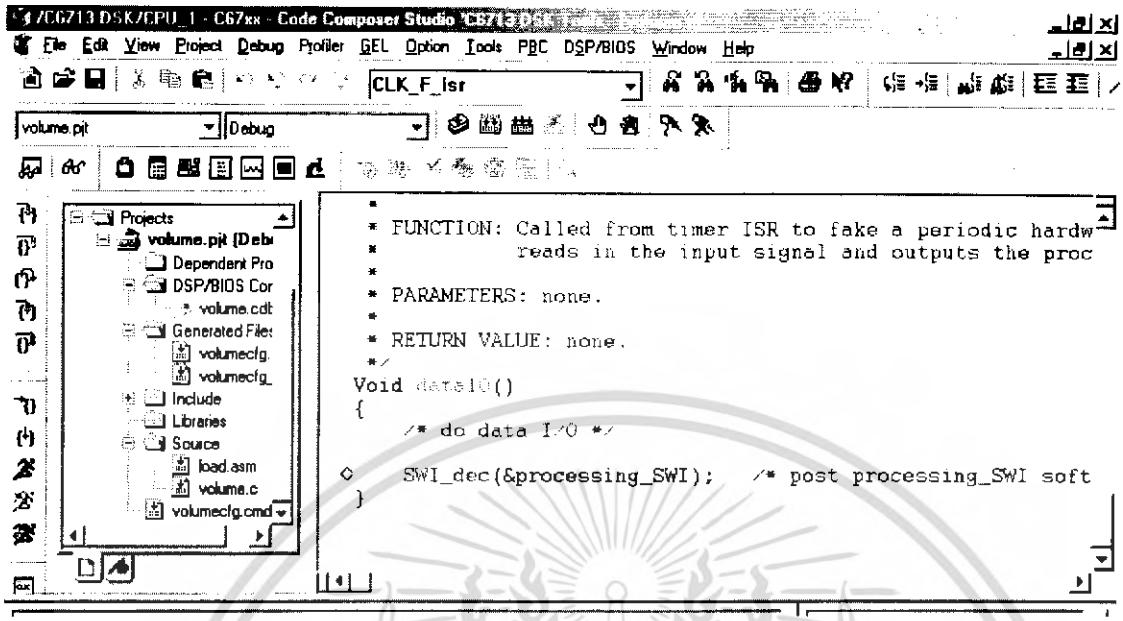
รูปที่ ก-22 แสดงการกำหนดช่วงเวลาที่จะทำการ trick ฟังก์ชัน

ทำการเพิ่มส่วนที่เป็น SWI พร้อมทั้งกำหนดขนาดของ mail box และฟังก์ชัน ที่ต้องการเรียก รวมถึงกำหนดอาร์กิวเมนต์ที่จะส่งให้โดยฟังก์ชัน จะถูกเรียกขึ้นมาทำงานเมื่อค่าใน mail box ลดลงจนเป็น 0



รูปที่ ก-23 แสดงการกำหนด mail box ของ software interrupt

สังเกตจากโค้ดว่าเมื่อมี clk ของ timer0 ที่เข้ามาทุก ๆ 1 mS ฟังก์ชัน dataIO() จะถูกเรียกขึ้นมาทำงาน เพื่อทำการลดค่าใน mailbox โดย ฟังก์ชัน SWI_dec()

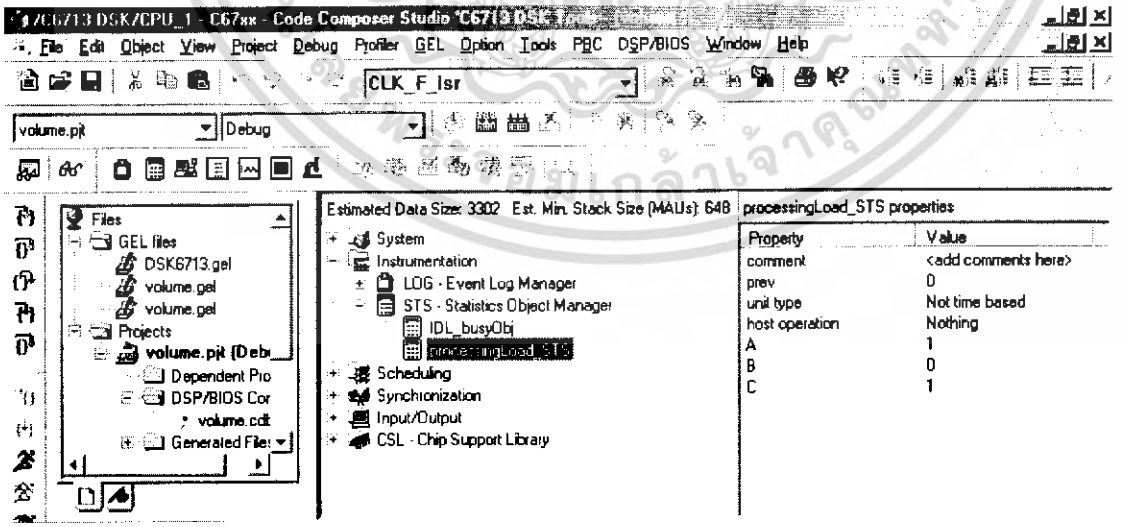


รูปที่ ก-24 แสดงโค้ดส่วนที่ทำงานเมื่อ dataIO ถูกเรียกใช้งาน

ก.6 การทดลอง Volumn3 (...\ti\tutorial\dsk6713\volume3)

<u>วัตถุประสงค์</u>

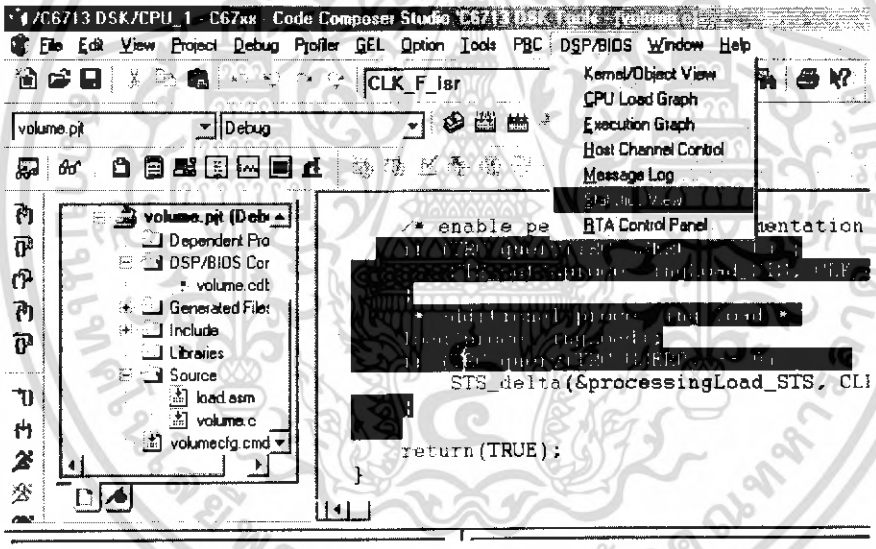
เพื่อศึกษาการใช้งานฟังก์ชัน ของ DSP bios เพื่อวัด performance ของฟังก์ชัน ใด ๆ ที่เราต้องการศึกษา โดยการเพิ่มวัตถุของ STS



รูปที่ ก-25 แสดง properties ของ Statistic object

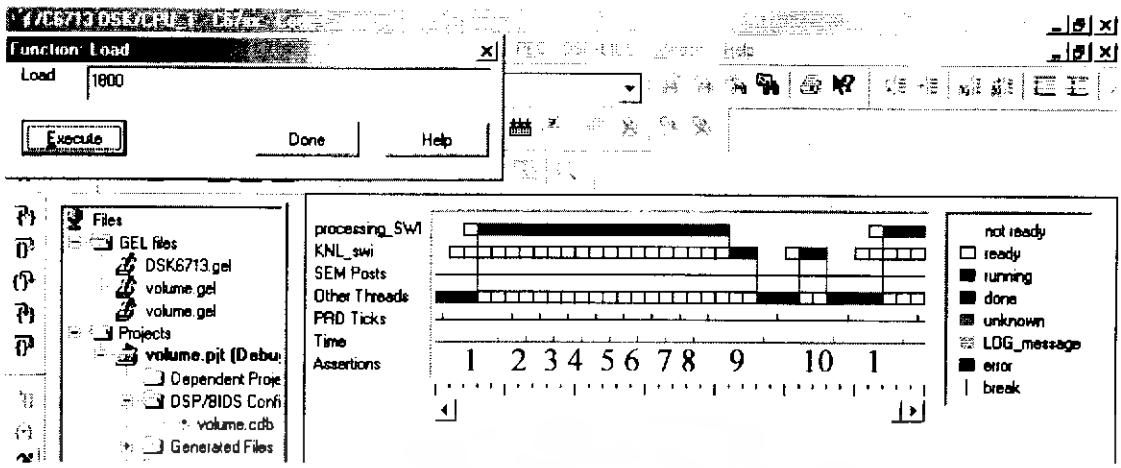
```
if (TRC_query(TRC_USER0) == 0) {
    STS_set(&processingLoad_STS, CLK_gethtime());
}
/* additional processing load */
load(processingLoad);
if (TRC_query(TRC_USER0) == 0) {
    STS_delta(&processingLoad_STS, CLK_gethtime());
}
```

เมื่อเราต้องการวัด Performance ที่ส่วนใด ๆ เช่นในตัวอย่าง เราต้องการหาว่า method load นั้นใช้กี่ instruction เวลาที่ใช้ในการประมวลผลเท่าไหร่ เราจะทำการ set เวลาเริ่มต้นก่อนเข้า method และเมื่อสิ้นสุดการทำงานของ method นั้น เราก็ set เวลาสิ้นสุดมาหา เวลาที่แตกต่างกันระหว่าง เวลาเริ่มต้นกับเวลาสุดท้าย เราจะได้ระยะเวลาการทำงานทั้งหมด รวมถึงจำนวน instruction ที่ใช้อีกด้วย เราสามารถดูข้อมูลของ processingLoad ได้โดย เลือก DSP/BIOS เลือก statistic view



รูปที่ ก-26 แสดงการเรียกใช้เครื่องมือในการดูค่าทางสถิติ

สังเกตจาก PRD Ticks ที่ใช้ Timer0 เช่นเดียวกับ PRD_clk เมื่อครบ 10 tick ก็ทำการเรียก processing_SWI ขึ้นมารัน 1 ครั้ง



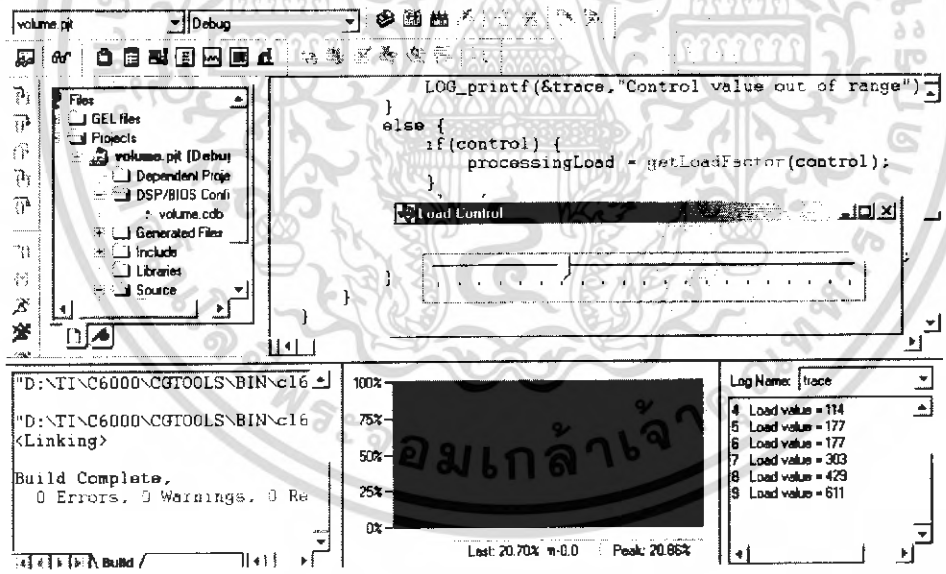
รูปที่ ก-27แสดง execution graph

ก.7 การทดลอง Volumn4 (...\ti\tutorial\dsk6713\volume4)

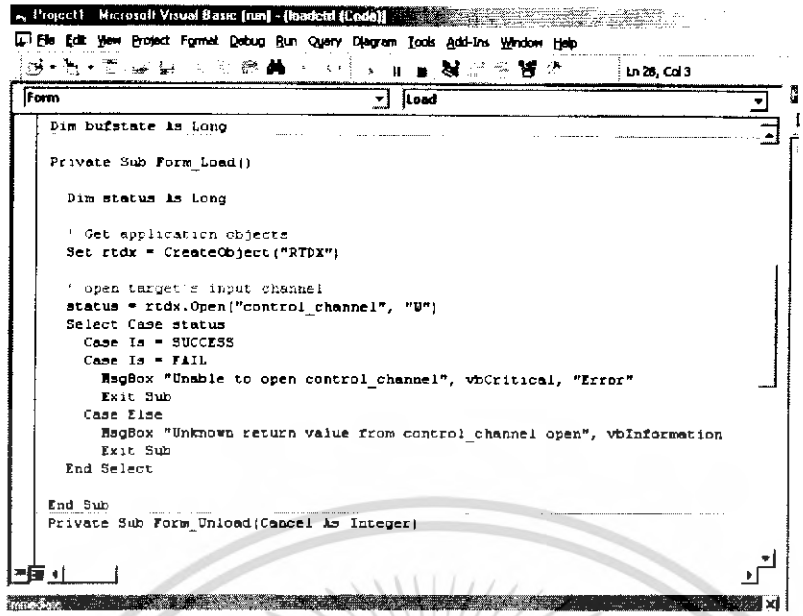
<u>วัตถุประสงค์</u>

เพื่อศึกษาการใช้ RTDX (real time data exchange) โดยผ่าน Object Linking and Embedding (OLE) อินเตอร์เฟสกับ Texas Instruments' debugger (Code Composer)

ทดลองรันโปรแกรม แล้วเรียก VB ขึ้นมารัน ทดลองเปลี่ยนโหลดสังเกตค่า Log พารามิเตอร์ ดู



รูปที่ ก-28 แสดง CPU Load graph



รูปที่ ก-29 แสดงโค้ด VB ที่ใช้ในการติดต่อกับ GEL ฟังก์ชัน

ภาคผนวก ข คู่มือโปรแกรมเมอร์

ข. คู่มือโปรแกรมเมอร์

ข.1 โครงสร้างของตัวแปรที่ใช้ในไลบรารี ของการเข้ารหัสและถอดรหัสตามมาตรฐาน G.729

ข.1.1 ENCODE_G729_MEM_BLK

Context Name	Array	Size(Bytes)
Prm		(18+1)*4 = 76
*new_speech		4
*preProcMemBlk (Pre_Proc.c)		
x0		4
x1		4
y1		4
y2		4
*coderMemBlk (codld8cp.c)		
old_speech		240*4 = 960
*speech		4
*p_window		4
*new_speech		4
old_wsp		(80+143)* 4 = 892
*wsp		4
old_exc		(80+143+11)* 4=936
*cxc		4
ai_zero		(40+31)*4= 284
*zero		4
lsp_old		10*4 = 40
lsp_old_q		10*4 = 40
mem_syn		10*4 = 40
mem_w0		10*4 = 40
mem_w		10*4 = 40
mem_err		70*4 = 280
*error		4
pit_sharp		4
G.729B		
pastVad		4

ppastVad	4
seed	2
/* for G.729E */	
/* for the backward analysis */	
prev_filter	31*4 = 124
rexp	31*4 = 124
synth	(65+80)*4 = 580
*synth_ptr	4
prev_lp_mode	4
gamma1	2*4=8
gamma2	2*4=8
A_t_bwd_mem	31*4 = 124
bwd_dominant	4
C_int	4
glob_stat	2
stat_bwd	2
val_stat_bwd	2
/* Last backward A(z) for case of unstable filter */	
old_A_bwd	31*4 = 124
old_rc_bwd	2*4=8
/* Last forkward A(z) for case of unstable filter */	
old_A_fwd	11*4 =44
old_rc_fwd	2*4 = 8
freq_prev	4*10*4 =160
lag_buf	5*4 = 20
pgain_buf	5*4 = 20
*exc_err_mblk (taming.c)	
exc_err	4*4=16
*vad_mblk (vad.c)	
MeanLSF	10*4 = 40
Min_buffer	16*4 = 64
Prev_Min	4
Next_Min	4

Min	4
MeanE	4
MeanSE	4
MeanSLE	4
MeanSZC	4
prev_energy	4
count_sil	4
count_update	4
count_ext	4
flag	4
v_flag	4
less_count	4
*cod_cng_mblk (dtx.c)	
/* Static Variables */	
lspSid_q	10*4 =40
pastCoeff	11*4 = 44
RCoeff	11*4 = 44
Acf	22*4 = 88
sumAcf	33*4 = 132
ener	2*4 = 8
fr_cur	4
cur_gain	4
nb_ener	4
sid_gain	4
flag_chang	4
prev_energy	4
count_fr0	4
//*Music	
*bwfwfunc_mblk (bwfwfunc.c)	
count_bwd2	4
count_fwd2	4
*pwf_mblk (pwf.c)	
smooth	4
lar_old	2*4 = 8

*acelp_cp_mblk (acelp_cp.c)	
cxtra	4
* q_gaincp_mblk (q_gaincp.c)	
past_qua_en	4*4 = 16
*phdisp_mblk (phdisp.c)	
prevDispState	4
gainMem	6*4 = 24
prevCbGain	4
onset	4

9.1.2 DECODE_G729_MEM_BLK

*decld8cpMblk (decld8cp.c)	
old_exc	[80+143+11]*4=936
*exc	4
lsp_old	10*4 = 40
mem_syn	30*4 = 120
sharp	4
gain_code	4
gain_pitch	4
prev_t0	4
prev_t0_frac	4
/* for G.729B */	
seed_fer	2
past_ftyp	4
seed	2
sid_sav	4
/* for the backward analysis */	
rexp	31*4 = 124
A_bwd_mem	31*4 = 124
A_t_bwd_mem	31*4 = 124
prev_voicing	4

prev_bfi	4
prev_lp_mode	4
c_fe	4
c_int	4
prev_filter	31*4 = 124
prev_pitch	4
stat_pitch	4
pitch_sta	4
frac_sta	4
/* Last backward A(z) for case of unstable filter */	
old_A_bwd	31*4 = 124
old_rc_bwd	2*4 = 8
gain_pit_mem	4
gain_cod_mem	4
c_muting	4
count_bfi	4
stat_bwd	4
freq_prev	4*10*4 = 160
prev_ma	4
prev_lsp	10*4 = 40
*pstcp_mblk(pstcp.c)	
apond2	32*4 = 128
mem_stp	30*4 = 120
mem_zero	30*4 = 120
res2	[154+40]*4 = 776
*res2_ptr	4
*ptr_mem_stp	4
*post_pro_mblk(post_pro.c)	
x0	4
x1	4
y1	4
y2	4
*phdisp_mblk(phdisp.c)	
prevDispState	4

gainMem	6*4 = 24
prevCbGain	4
onset	4
*exc_err_mblk (taming.c)	
exc_err	4*4=16
*degaincp_mblk (degaincp.c)	
past_qua_en	4*4 = 16
*dec_sid_mblk (dec_sid.c)	
cur_gain	4
lspSid	10*4 = 40
sid_gain	4

ข.1.3 G729_DTX_TABLE Table

/* VAD constants */	
FLOAT lbf_corr[NP+1];	4*(12+1) = 52
/* SID gain quantization */	
FLOAT fact[NB_GAIN+1];	4*(2+1) = 12
FLOAT tab_Sidgain[32];	4*32 = 128
/* SID LSF quantization */	
int PtrTab_1[32];	
int PtrTab_2[2][16];	4*2*16 = 128
FLOAT noise_fg[MODE][MA_NP][M];	4*2*4*10 = 320
FLOAT noise_fg_sum[MODE][M];	4*2*10 = 80
FLOAT noise_fg_sum_inv[MODE][M];	4*2*10 = 80
FLOAT Mp[MODE];	4*2 = 8

ข.1.4 G729_LD8K_TABLE

FLOAT hamwindow[L_WINDOW];	4*240 =960

FLOAT grid[GRID_POINTS+1];	4*(60+1)=244
FLOAT lspcb1[NC0][M]; /*First Stage Codebook */	4*14*10=560
FLOAT lspcb2[NC1][M]; /*Second Stage Codebook*/	4*10*10=400
FLOAT fg[MODE][MA_NP][M]; /*MA prediction coef. */	4*2*4*10 = 320
FLOAT fg_sum[MODE][M]; /*present MA prediction coef.*/	4*2*10=80
FLOAT fg_sum_inv[MODE][M]; /*inverse coef. */	4*2*10=80
FLOAT inter_3[FIR_SIZE_ANA];	4*13=52
FLOAT inter_3l[FIR_SIZE_SYN];	4*31=124
FLOAT pred[4];	4*4=16
FLOAT coef[2][2];	4*2*2=16
FLOAT thr1[NCODE1-NCAN1];	4*(4)=16
FLOAT thr2[NCODE2-NCAN2];	4*(8)=32
FLOAT gbk1[NCODE1][2];	4*8*2=54
FLOAT gbk2[NCODE2][2];	4*16*2=128
FLOAT tab_hup_s[SIZ_TAB_HUP_S];	4*7*4=112
FLOAT tab_hup_l[SIZ_TAB_HUP_L];	4*7*16 =448
int map1[NCODE1];	4*8=32
int map2[NCODE2];	4*16 =64
int imap1[NCODE1];	4*8 =32
int imap2[NCODE2];	4*16 =64
int bitsno[PRM_SIZE];	4*11 =44
FLOAT b140[3];	4*3 = 12
FLOAT a140[3];	4*3 = 12
FLOAT b100[3];	4*3 = 12
FLOAT a100[3];	4*3 = 12

ช.1.5 G729_LD8CP_TABLE

int bitsno_B[PRM_SIZE_SID];	4*4 =16
int bitsno_D[PRM_SIZE_D];	4*10 = 40
int bitsno_E_fwd[PRM_SIZE_E_fwd-1];	4*(18-1) = 68
int bitsno_E_bwd[PRM_SIZE_E_bwd-1];	4*(16-1) = 60

/* 11.8k */	
FLOAT lag_bwd[M_BWD];	4*30 =120
FLOAT hw[NRP+L_FRAME+M_BWD];	4*(35+80+30) = 580
int ipos[16];	4*16 = 64
/* 6.4k (for NTT CS-VQ)*/	
int trackTable0[16];	4*16 = 64
int trackTable1[32];	4*32 =128
int posSearched[2];	4*2=8
int grayEncode[32];	4*32=128
int grayDecode[32];	4*32 = 128
FLOAT ph_imp_low[L_SUBFR];	4*40 = 160
FLOAT ph_imp_mid[L_SUBFR];	4*40 = 160
FLOAT ph_imp_high[L_SUBFR];	4*40 = 160
FLOAT gbk1_6k[NCODE1_6K][2];	4*8*2 = 64
FLOAT gbk2_6k[NCODE2_6K][2];	4*8*2 = 64
FLOAT coef_6k[2][2];	4*2*2 = 16
FLOAT thr1_6k[NCODE1_6K-NCAN1_6K];	4*2 =8
FLOAT thr2_6k[NCODE2_6K-NCAN2_6K];	4*2 = 8
int map1_6k[NCODE1_6K];	4*8 =32
int imap1_6k[NCODE1_6K];	4*8 = 32
int map2_6k[NCODE2_6K];	4*8 = 32
int imap2_6k[NCODE2_6K];	4*8 = 32
FLOAT freq_prev_reset[M];	4*10 = 40
FLOAT lwindow[M+2];	4*12 = 48

ภาคผนวก ก คู่มือ Reference framework 3

ค. คู่มือ Reference framework 3

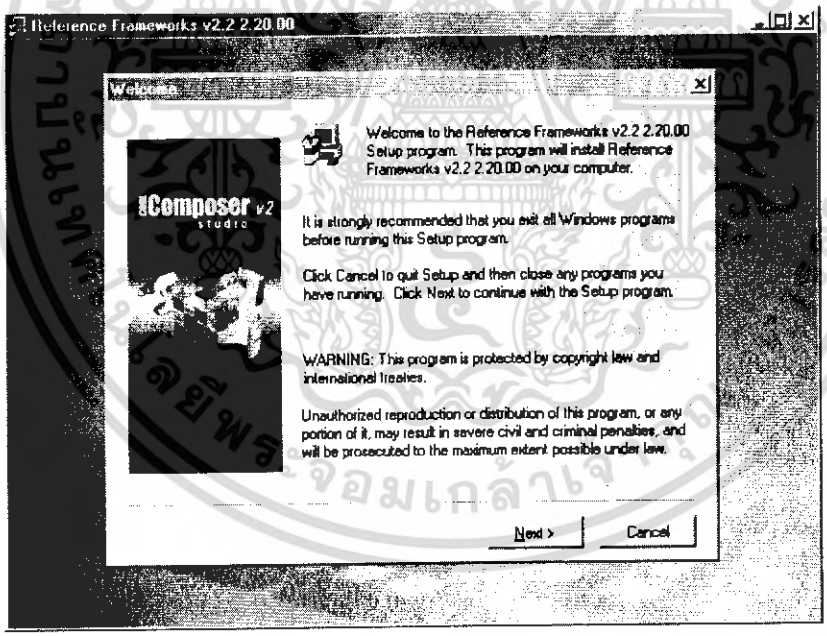
ค.1 โครงสร้างการทำงานของโปรแกรม

การพัฒนาโปรแกรมบน ดี เอส พี บอร์ด ของ เท็กซัส อินสตรูเมนต์ นั้น เริ่มต้นจากการพัฒนาบนโครงของโปรแกรมที่ทางบริษัทนั้นพัฒนาออกมา โดยการเพิ่มเติม และเปลี่ยนแปลงโปรแกรมบางส่วนทำให้เราไม่ต้องเริ่มต้นจากการพัฒนาใหม่ทั้งหมด ซึ่งการพัฒนาในโครงการนี้ เราจะพัฒนาบน platform ที่ทาง เท็กซัส อินสตรูเมนต์ ที่ให้มา คือ reference framework

ค.1.1 Reference framework

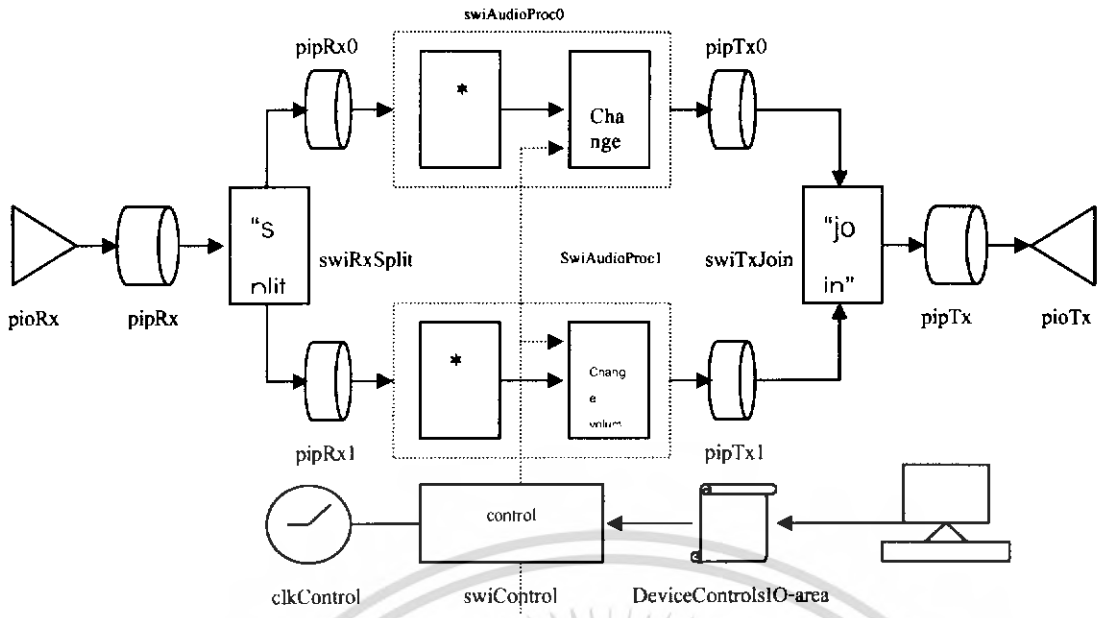
ในการทำงานกับ DSP/BIOS นั้น การที่เราจะเขียนโปรแกรมควบคุมตั้งแต่เริ่มแรกนั้น เราจะต้องทำการศึกษา และต้องเข้าใจการทำงานของ DSP ในทุกขั้นตอน ทำให้ต้องเสียเวลามากในการจัดการโปรแกรมหนึ่ง ๆ เพราะฉะนั้น จึงมีผู้ที่ทำการเขียนโปรแกรมพื้นฐานคร่าว ๆไว้ หลากให้เลือกใช้งาน ตามความต้องการจะนำไปประยุกต์ ซึ่งเรียกว่า Reference Frameworks ซึ่งการใช้งานอธิบายได้ดังนี้

1. ติดตั้งเฟรมเวิร์ค ที่ดาวน์โหลด มา และทำตามคำแนะนำที่ระบุไว้ ซึ่งภายในจะเป็นโค้ดให้ใช้



รูปที่ ค-1 แสดงหน้าจอการ ติดตั้ง โปรแกรม

2. ซึ่งในที่นี้จะใช้ Reference Framework (RF3) ของบริษัท เท็กซัส อินสตรูเมนต์ โดยโครงสร้างของ RF3 เป็นดังนี้



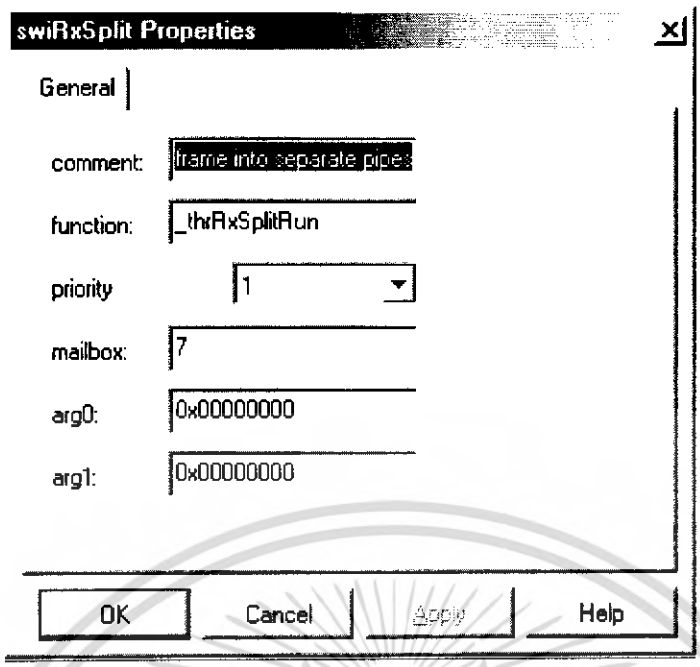
รูปที่ ค-2 แสดงโครงสร้างของ Reference Framework 3

หมายเหตุ * แทนกระบวนการเข้ารหัสและถอดรหัส

ค.1.1.1 การทำงานของเฟรมเวิร์ค

จะทำเกี่ยวกับการรับสัญญาณอินพุทที่เป็นเสียงเข้ามาทาง port LineIn บน ดี เอส พี บอร์ด (pioRX) และจะนำมาไว้ใน บัฟเฟอร์(pipRx) เมื่อมีข้อมูลแล้ว thread "split" left & right จะทำการ แยกออกเป็น 2 channel ไปไว้ในบัฟเฟอร์ pipRx0 และ pipRx1 สลับกันไปเรื่อย ๆตามขนาดเฟรม ตามลำดับ แล้วจะนำไป process การจัดการกับเสียง โดยทั้ง 2 channel นี้จะมีการทำงานเหมือนกันคือ จะมี การ change volume หลังจากทำการเข้ารหัสและถอดรหัสแล้วแล้ว แล้วจะนำมาเก็บไว้ที่บัฟเฟอร์ pipTx0 และ pipTx1 ตามลำดับ หลังจากนั้น จะนำข้อมูลทั้งสองมารวมเข้าด้วยกันโดย thread "joint" left & right หลังจากนั้นจะส่งไปที่ บัฟเฟอร์ pipTx แล้วจะส่งไปยัง port headphone บน ดี เอส พี บอร์ด ผ่าน pioTx โดยการทำงานทั้งหมด จะควบคุมโดย thread control ดังนั้นเสียงที่ออกมาจะเป็น 2 ช่องสัญญาณ

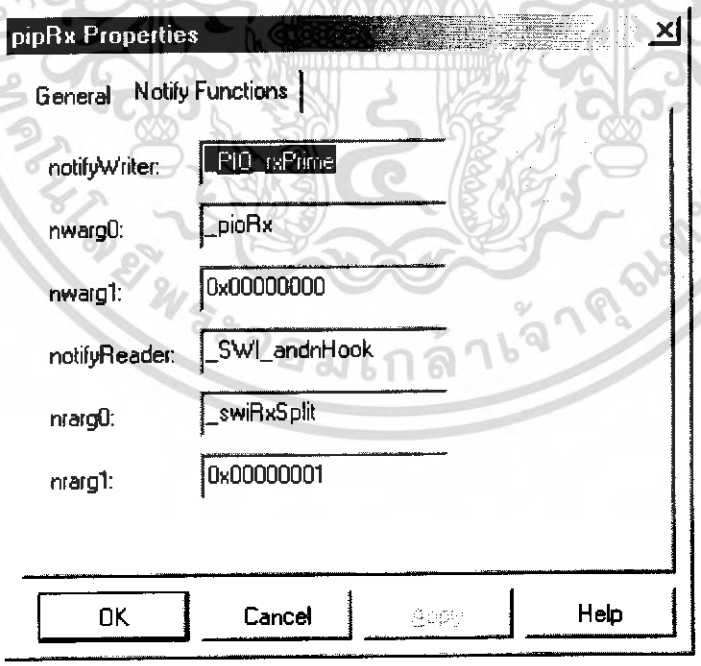
ในขั้นตอนของ thread "split" left & right จะต้องตรวจสอบก่อนว่า ใน pipRx มีข้อมูลเต็มหรือไม่ และใน pipRx0 และ pipRx1 จะต้องเป็นบัฟเฟอร์ว่าง คือ เป็น บัฟเฟอร์ ที่ไม่มีข้อมูล หมายถึงจะต้องมี 3 เงื่อนไขที่จะต้องตรวจสอบ



รูปที่ ค-3 แสดง properties ของ sw

iRxSplit

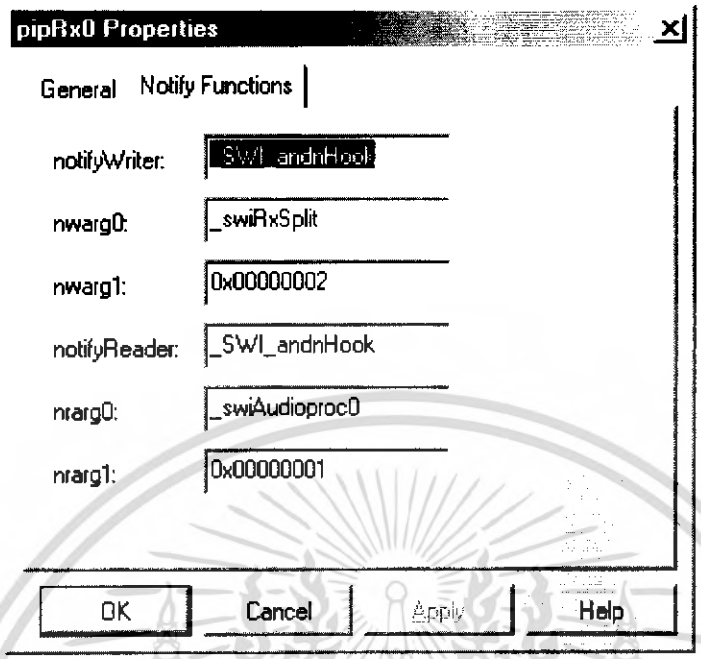
โดยการตรวจสอบจะเป็นการดูจากค่า mailbox ซึ่งมีค่าเท่ากับ 7 (0111_2) เป็น ตรวจสอบบิทและจะทำงานเมื่อเท่ากับ 0 ถ้ามีเงื่อนไขหนึ่ง เช่นบัฟเฟอร์ pipRx มีข้อมูลก็จะทำตามที่กำหนดใน property page ของ pipRx



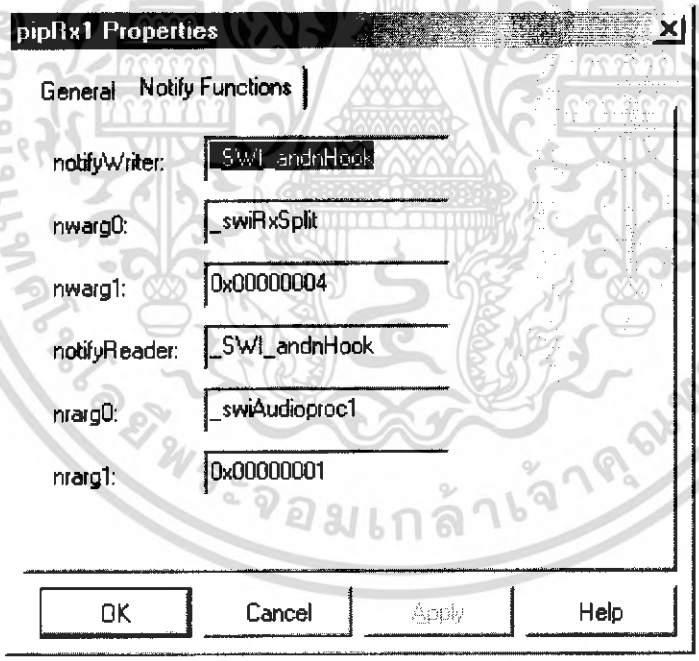
รูปที่ ค-4 แสดงหน้า property ของ pipRx

ใน property page นี้ สามารถบอกได้ว่า เมื่อมี ข้อมูลมาอยู่ใน pipRx ก็จะเรียกฟังก์ชัน *SWI*_andnHook โดยมีพารามิเตอร์คือ _swiRxSplit และ 0x0000001 โดยการทำงานคือ จะ NAND ค่า mailbox

ของ _swiRxSplit ด้วยค่า 0x00000001 โดยจะทำหน้าที่เหมือน maskbit จะทำให้ค่าของ mailbox กลายเป็น 0110 และ pip อีก 2 pip (pipRx0, pipRx1) ก็ทำหน้าที่เหมือนกัน



รูปที่ ค-5 แสดง properties ของ pipRx0



รูปที่ ค-6 แสดง properties ของ pipRx1

แต่จะแตกต่างกันโดย ส่วนนี้จะเป็นการเรียก *SWI*_andnHook ทาง notifyWriter เพราะว่า_swiRxSplit จะทำการเขียน pipRx1 และจะทำการ NAND ด้วย 0004 (0100_2) และ 0002 (0010_2) จะทำให้ mailbox ของ _swiRxSplit กลายเป็น 0000 ทำให้ thread thrRxSplit ทำงาน