การสังเคราะห์สัญญาณเสียงโดยใช้บอร์ด TMS320C6211

VOICE SYNTHESIS WITH TMS320C6211

โดย

นาย สาธิต ชวลิตรุจิวงษ์

นาย สานนท์ เดชอภินันท์ชัย

นาย สุขรักษ์ แซ่เจี่ย

อาจารย์ที่ปรึกษา

รศ.ดร. ปราโมทย์ วาดเขียน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร ปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การสังเคราะห์สัญญาณเสียงโดยใช้บอร์ด TMS320C6211

VOICE SYNTHESIS WITH TMS320C6211

โดย

นาย สาธิต ชวลิตรุจิวงษ์

นาย สานนท์ เดชอภินันท์ชัย

นาย สุขรักษ์ แซ่เจี่ย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร ปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ภาควิชา
วิศวกรรมโทรคมนาคม

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การสังเคราะห์สัญญาณเสียงโดยใช้บอร์ด TMS320C6211**

VOICE SYNTHESIS WITH TMS320C6211

ผู้จัดทำ

1.นาย สาธิต ชวลิตรุจิวงษ์ 44010524

2. นาย สานนท์ เดชอภินันท์ชัย 44010525

3. นาย สุขรักษ์ แซ่เจี่ย 44010537

..... อาจารย์ที่ปรึกษา

(รศ. ดร.ปราโมทย์ วาดเขียน)

การสังเคราะห์สัญญาณเสียงโดยใช้บอร์ด TMS320C6211

Voice synthesis with TMS320C6211

โดย นาย สาธิต ชวลิตรุจิวงษ์ 44010524

นาย สานนท์ เดชอภินันท์ชัย 44010525

นาย สุขรักษ์ แซ่เจี่ย 44010537

อาจารย์ที่ปรึกษา รศ. ดร.ปราโมทย์ วาดเขียน

บทคัดย่อ

โครงงานนี้นำเสนอเกี่ยวกับการสังเคราะห์สัญญาณเสียงโดยอาศัยหลักการ LPC (linear prediction coding) ด้วย การนำเสียงต้นแบบมาหาค่าสัมประสิทธิ์ LPC และนำค่าสัมประสิทธิ์ที่ได้มาสังเคราะห์ให้เป็นสัญญาณเสียงกลับคืนมาสำหรับการประมวลผลจะใช้บอร์ด TMS320C6211 ซึ่งโครงงานนี้สามารถนำมาพัฒนาเป็นเครื่องบีบอัดเสียง เพื่อช่วยลด bandwidth ในการติดต่อสื่อสารได้

ABSTRACT

The speech synthesize based on linear predictive coding is present. Which speech prototype is used to be input to generate LPC coefficients and they are used to reconstruct the speech back. TMS320C6211 board is used to be processor in this project. For the application, it can apply to voice compressor that help to decrease bandwidth for communication.

สารบัญ

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

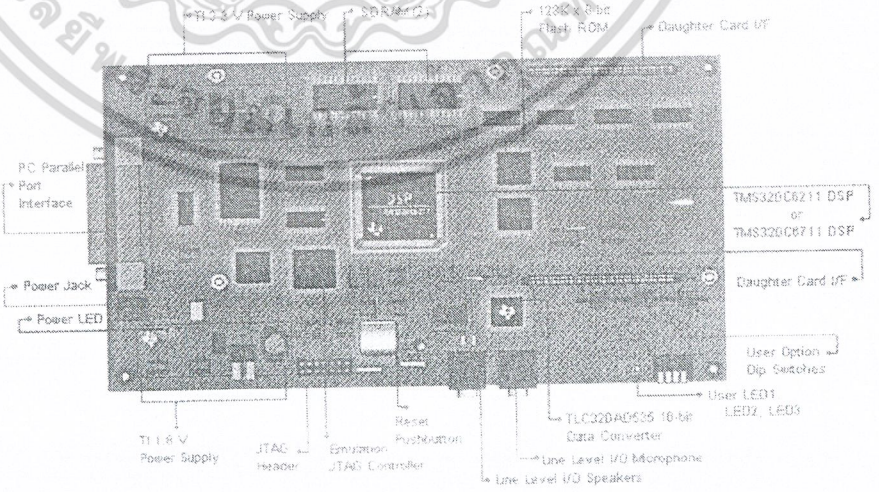
สารบัญรูปภาพ(ต่อ)

สารบัญตาราง

บทที่ 1

เทคโนโลยีทางด้านวิศวกรรมไฟฟ้า และคอมพิวเตอร์ในปัจจุบันได้ก้าวหน้าไปอย่างรวดเร็ว และบทบาทสำคัญของความก้าวหน้าเหล่านั้น ก็มาจากการเปลี่ยนแปลงของเทคโนโลยีหลายอย่างที่เคยทำด้วยวงจรแอนะลอกมาเป็นวงจรดิจิตอล เรามักจะได้ยินคนพูดเสมอถึงเทคโนโลยีในโลกปัจจุบันว่าเป็น "ยุคแห่งดิจิตอล" หรือ "โลกดิจิตอล" ซึ่งผู้พูดส่วนใหญ่ก็มักมองจากสินค้าทันสมัยต่างๆ ที่ออกมาในท้องตลาด เช่น การมีคอมพิวเตอร์ที่มีความเร็วสูงขึ้น การมีระบบมัลติมีเดียดิจิตอลที่ดีขึ้น การเชื่อมต่อของระบบคอมพิวเตอร์ผ่านเครือข่ายอินเตอร์เน็ต การมีอุปกรณ์สื่อสาร เช่น โทรศัพท์มือถือและเพจเจอร์ที่ดีขึ้น และอุปกรณ์อื่นๆอีกมากมาย ซึ่งแน่นอนว่าอุปกรณ์ต่างๆ เหล่านี้มีพื้นฐานมาจากทฤษฎีของอิเล็กทรอนิกส์และคอมพิวเตอร์ แต่ยังมีทฤษฎีอีกเรื่องหนึ่งที่เป็นพื้นฐานและเป็นส่วนประกอบที่สำคัญของเทคโนโลยีเหล่านี้นั่นก็คือ การประมวลผลสัญญาณดิจิตอล

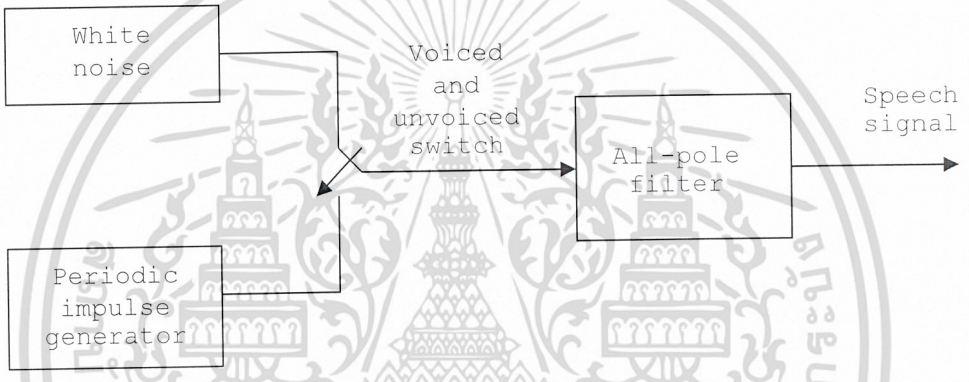
วิศวกรไทยมักมองว่าการประมวลผลสัญญาณดิจิตอลเป็นทฤษฎีขั้นสูง และเป็นเรื่องไกลตัวซึ่งก็มาจากการที่เรายังไม่มีการสอนเรื่องการประมวลผลสัญญาณดิจิตอล ในระดับชั้นปริญญาตรีกันเท่าไรนัก เนื่องจากมองว่าเป็นวิชาที่ยาก ซึ่งเต็มไปด้วยคณิตศาสตร์กับทฤษฎีที่มองไม่เห็นภาพ แต่จริงๆแล้วการประมวลผลสัญญาณดิจิตอลเป็นวิชาที่มีการประยุกต์ใช้งานอย่างกว้างขวางมาก ในปัจจุบันการประมวลผลสัญญาณดิจิตอลเป็นส่วนสำคัญที่เสริมกับเทคโนโลยีในสาขาวิชาต่างๆ ทั้งด้านวิศวกรรมอิเล็กทรอนิกส์ วิศวกรรมโทรคมนาคม วิศวกรรมควบคุม วิศวกรรมไฟฟ้าแรงสูง และวิศวกรรมคอมพิวเตอร์ ดังนั้นวิศวกรในระดับชั้นระดับปริญญาตรี จึงควรมีความเข้าใจในระดับพื้นฐานเกี่ยวกับการประมวลผลสัญญาณดิจิตอล ในรายงานฉบับนี้จะมุ่งเน้นไปทางด้านโทรคมนาคม โดยจะกล่าวถึงการทำ linear predictive coding เป็นหลักโดยเป็นการทดลองที่ใช้ MATLAB และการทดลองโดยใช้โปรแกรม Visual C++ เพื่อควบคุมการทำงานระหว่างคอมพิวเตอร์กับชิพ DSP (TMS320C6211)



รูปที่ 1.1 ชิพ DSP (TMS320C6211)

บทที่ 2

ในบทที่ 2 นี้ เราจะกล่าวถึงในเรื่องทฤษฎีและหลักการของกระบวนการสังเคราะห์สัญญาณเสียงโดยใช้หลักการ LPC (Linear Predictive Coding) ซึ่งหลักการนี้เราจะต้องคำนวณค่า parameters คือ ค่าสัมประสิทธ์ ค่า gain และค่า pitch period โดยเราจะใช้ค่าสัมประสิทธ์และค่า gain ในการสร้างตัวกรองสัญญาณในที่นี้เราจะมองตัวกรองสัญญาณชนิด IIR (Infinite Impulse Response) เป็นระบบที่เราสนใจ โดยเราจะใช้ค่า pitch period เป็นตัวตัดสินว่าจะใช้อินพุทตัวใดระหว่าง Impulse trains หรือ white noise และเอาท์พุทที่ได้จะเป็นสัญญาณ voiced หรือ unvoiced ตามลำดับ



รูปที่ 2.1 บล็อกไดอะแกรมของการสังเคราะห์สัญญาณเสียง

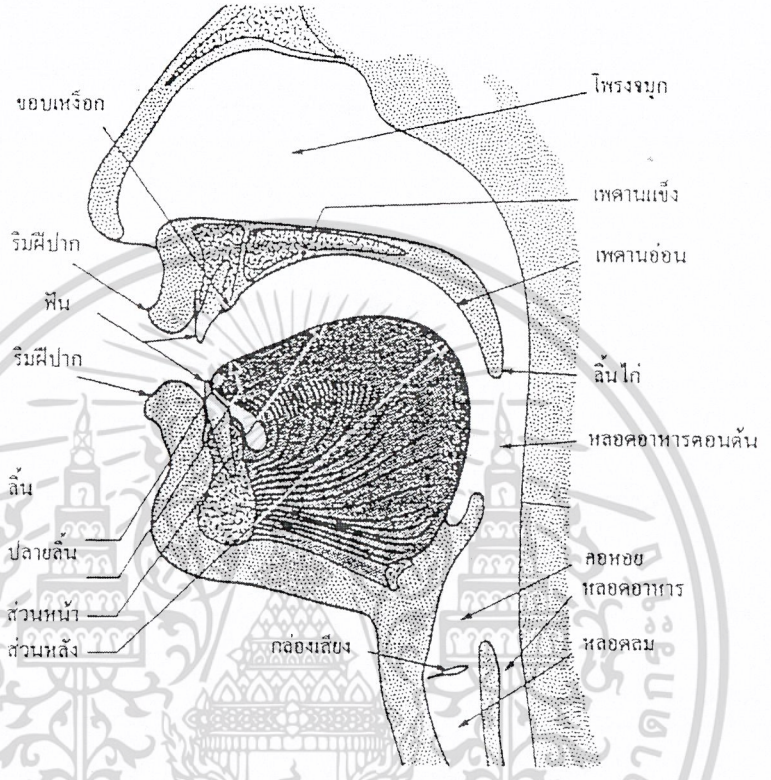
ดังนั้นในบทนี้เราจะกล่าวถึงระบบการพูดและเสียงพูดของมนุษย์ และจะกล่าวถึงภาพรวมของการประมวลผลสัญญาณดิจิตอลว่าด้วยเรื่อง สัญญาณต่อเนื่อง, สัญญาณไม่ต่อเนื่อง, การประมวลผลแบบเวลาจริง กับการเลือกใช้ตัวประมวลผลสัญญาณ, การสุ่มสัญญาณ และการสร้างสัญญาณคืน, ตัวกรองความถี่แบบ IIR และรายละเอียดของชิพ DSP (TMS320C6211)

ระบบการพูดและเสียงพูดของมนุษย์

จากการศึกษาด้านกายวิภาคศาสตร์ของมนุษย์ (Human anatomy) คือวิชาที่ว่าด้วย เสียงของภาษา (Phonetics) และศาสตร์ทางด้านเสียง (Acoutics) ช่วยให้เราเข้าใจ ขั้นตอนการทำงานร่วมกันของอวัยวะต่างๆในการเปล่งเสียงพูด ตลอดจนลักษณะทางกายภาพของเสียงพูด เพื่อนำมาทำการวิเคราะห์และสร้างแบบจำลองเลียนแบบเสียงพูดของมนุษย์

2.1 กายวิภาคของระบบการเปล่งเสียงของมนุษย์

การทำให้เกิดเสียงเป็นหน้าที่หนึ่งของระบบหายใจ การออกเสียงหรือการพูดของมนุษย์แต่ละครั้งจะต้องมีการทำงานร่วมกันของอวัยวะต่างๆของร่างกายดังรูปที่ 2.2 อันประกอบด้วย

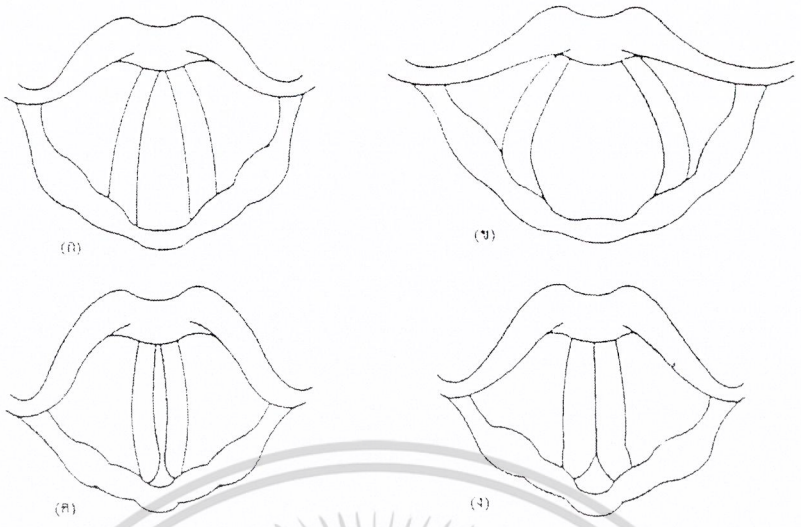


รูปที่ 2.2 ภาพตัดขวางแสดงอวัยวะในระบบการพูดของมนุษย์

2.1.1 ปอดและกระบังลม ทำหน้าที่สำคัญในการหายใจ และเป็นต้นกำเนิดการไหลของอากาศในกระบวนการผลิตเสียง

2.1.2 หลอดลม ทำหน้าที่นำอากาศจากปิดผ่านกล่องเสียง และเป็นอวัยวะที่อยู่ด้านหน้าของหลอดอาหาร

2.1.3 กล่องเสียง เป็นอวัยวะพิเศษที่ทำหน้าที่เป็นทางเดินอากาศเวลาหายใจ และเป็นตัวผลิตพัลส์ (Pulse) ของอากาศขณะเปล่งเสียง ซึ่งประกอบด้วยเส้นเสียง (Vocal cords) และช่องสายเสียง (Glottis) รูปร่างของกล่องเสียงและเส้นเสียงในลักษณะต่างๆ แสดงในรูปที่ 2.3



รูปที่ 2.3 กล่องเสียงขณะ (ก) หายใจปกติ (ข) หายใจเข้าลึกๆ
(ค) กำลังส่งเสียง (ง) ส่งเสียงกระซิบหรือเสียงแผ่ว

2.1.4 ช่องปากและส่วนของหลอดอาหารตอนต้น อวัยวะกลุ่มนี้อยู่ต่อจากกล่องเสียง อาจเรียกว่าอวัยวะกำทอนเสียง (Vocal tract) ทำหน้าที่กำทอนเสียง โดยให้กำทอนทั้งเสียงที่เกิดจากกล่องเสียงและเสียงที่เกิดภายในช่องปาก ขนาดของอวัยวะกำทอนเสียงขึ้นอยู่กับตำแหน่งของลิ้น ริมฝีปาก ขากรรไกร และเพดานอ่อน เปลี่ยนแปลงไปตามการออกเสียง

2.1.5 โพรงจมูก เริ่มจากเพดานอ่อนจนถึงรูจมูกทั้งสองทำหน้าที่กำทอนเสียงร่วมกับช่องปากเมื่อมีการเปล่งเสียงที่ออกทางจมูก (Nosal sounds) เช่นเสียง /ม/ /น/ และ /ง/ เป็นต้น

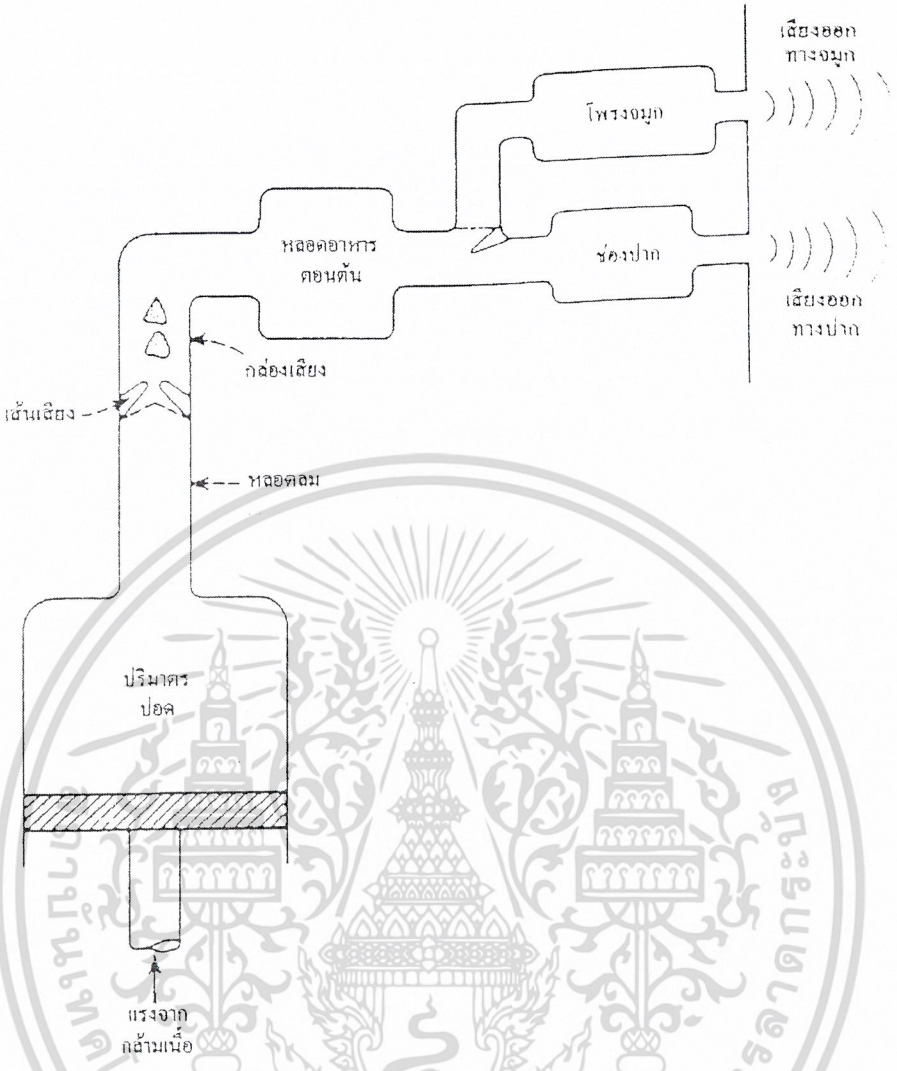
2.2 กระบวนการผลิตเสียงพูด

จากระบบเสียงพูด สามารถแสดงเป็นแผนภาพของระบบกำเนิดเสียง ดังรูปที่ 2.4 ซึ่งเราสามารถจำแนกกลไกการสร้างเสียงพูดของมนุษย์ได้ 3 แบบ ดังนี้

2.2.1 อากาศไหลจากปอดจะถูกมอดูเลท (Modulate)โดยการสั่นของเส้นเสียงทำให้เกิดคลื่นเสียงลักษณะคล้ายพัลส์ที่มีคาบเวลาแบบควอไซ (Quasi-periodic pulse-like excitation)

2.2.2 อาการที่ไหลจากปอดถูกทำให้ปั่นป่วนด้วยการบังคับให้ไหลผ่านช่องแคบอันเกิดจากการบีบตัวของอวัยวะในช่องปากทำให้เกิดเสียงลักษณะคล้ายเสียงรบกวน (Nose-like excitation)

2.2.3 อากาศที่ไหลถูกกัก และเกิดแรงดันอยู่ภายในส่วนของช่องปากที่ปิด จากนั้นจึงปล่อยให้อากาศที่มีแรงดันพุ่งออกไปอย่างรวดเร็ว ทำให้เกิดการกระตุ้นเป็นเสียงในช่วงเริ่มต้น (Transient excitation)



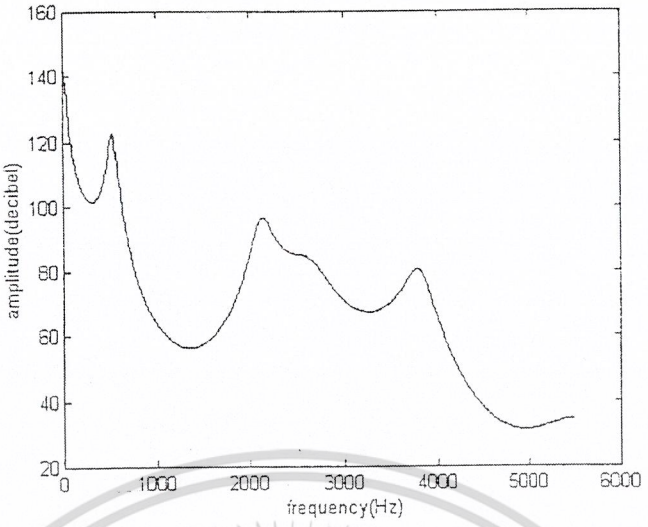
รูปที่ 2.4 แผนภาพระบบเสียงพูดของมนุษย์

2.3 เสียงพูดมนุษย์

เสียงพูดเป็นคลื่นตามยาว (Longitudinal wave) เกิดจากการสั่นของอนุภาคตัวกลางนั่นคืออากาศ และทิศทางการสั่นของอนุภาคจะอยู่ในทิศเดียวกันกับทิศทางของการเคลื่อนที่ โดยที่คลื่นเสียงเป็นคลื่นที่เปลี่ยนแปลงไปตามเวลา

2.4 ความถี่ฟอร์แมนท์ (Fotmant frequency)

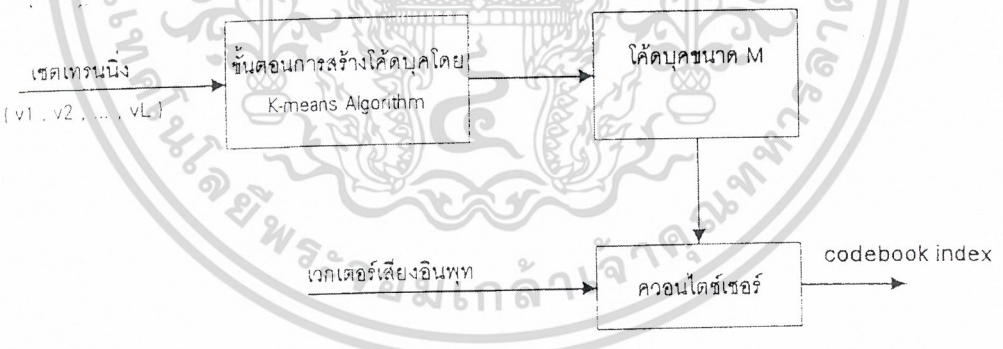
ความถี่ฟอร์แมนท์ คือ ความถี่กำทอน (Resonance frequency) ที่เกิดจากกลุ่มอวัยวะกำทอนเสียง ความถี่ฟอร์แมนท์ของเสียงจะมีค่าเท่าใดขึ้นอยู่กับการบังคับขนาดและรูปทรงของอวัยวะส่วนนี้ อันเกิดเป็นเสียงความถี่ที่แตกต่างกันนั่นคือ เสียงพูดต่างๆนั่นเอง ความถี่ฟอร์แมนท์สำหรับเสียงพูดคำหนึ่งๆอาจมีได้หลายค่าดังรูปที่ 2.5 เป็นสเปคตรัมของเสียง /a/ ซึ่งมีความถี่ฟอร์แมนท์ที่ 1, 2, 3 และ 4 อยู่ ณ ตำแหน่งความถี่ที่ 550, 2150, 2600 และ 3750 Hz ตามลำดับ



รูปที่ 2.5 สเปคตรัมของเสียง /a/ ที่ได้จากสัมประสิทธิ์ LPC

2.5 แบบจำลองระบบกำเนิดเสียงพูด

จากภาพรูปที่ 2.4 เราสามารถแสดงบล็อกไดอะแกรมจำลองระบบกำเนิดเสียงเบื้องต้นดังรูปที่ 2.6 จากรูปมีการแยกภาคแหล่งกำเนิดสัญญาณกระตุ้นออกจากส่วนกำทอนเสียง ซึ่งแทนด้วยระบบเชิงเส้นแปรผันตามเวลา



รูปที่ 2.6 บล็อกไดอะแกรมจำลองระบบกำเนิดเสียงเริ่มต้น

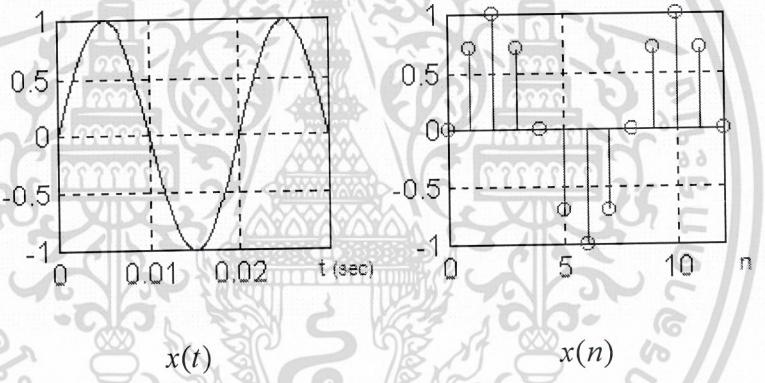
จากรูปที่ 2.6 แหล่งกำเนิดสัญญาณกระตุ้นทำหน้าที่แทนการทำงานของปอด และกล่องเสียงส่วนนี้จะผลิตขบวนพัลส์ที่มีคาบเวลาพิทช์ขณะเปล่งเสียงว้อยซ์ และให้กำเนิดเสียงซึ่งคล้ายเสียงรบกวนขณะเปล่งเสียงอันว้อยซ์

ส่วนที่สองเป็นท่อกำทอนเสียงจะแทนการทำงานของช่องปากและโพรงจมูก ทำหน้าที่เสมือนเป็นตัวกรองสัญญาณ (Filter) ที่ยอมให้ความถี่ฟอร์แมนท์ผ่านได้ ซึ่งสามารถแทนด้วยระบบเชิงเส้นแปรผันตามเวลา (Time-varying linear system)

2.6 สัญญาณต่อเนื่อง กับสัญญาณไม่ต่อเนื่อง

สิ่งแรกที่จะต้องทำความเข้าใจกันก่อนก็คือ คำว่าสัญญาณต่อเนื่อง (continuous-time signal) และสัญญาณไม่ต่อเนื่อง (discrete-time signal) หมายความว่าอย่างไร คำว่าต่อเนื่องหรือไม่ต่อเนื่องนี้หมายถึงสัญญาณนั้นๆมีค่าต่อเนื่องในทางเวลาหรือไม่ สัญญาณต่อเนื่องก็คือ สัญญาณที่เราพบเห็นใช้วิตประจำวันทั่วๆ ไป หรือที่เห็นบนหน้าจอออสซิโลสโคป เช่น สัญญาณเสียง, สัญญาณไฟบ้าน 50 Hz, และอื่นๆ ถ้าแทนสัญญาณด้วยสัญลักษณ์ x และแทนเวลาด้วยสัญลักษณ์ t เราจะกล่าวว่า x เป็นฟังก์ชั่นของ t หรือ x มีค่าที่เวลา t ใดๆ เขียนแทนสัญญาณนี้ได้ว่า $x(t)$ ซึ่งเป็นฟังก์ชั่นที่ต่อเนื่อง สัญญาณต่อเนื่องนี้เรียกอีกอย่างหนึ่งว่า สัญญาณแอนะลอก (analog signal)

สัญญาณไม่ต่อเนื่องเป็นสัญญาณที่มีค่าเพียงบางจุดของเวลา โดยทั่วไปเกิดจากการสุ่มสัญญาณต่อเนื่องด้วยคาบเวลาของการสุ่มคงที่ เราจะใช้สัญลักษณ์ n แทนเวลาแบบไม่ต่อเนื่องโดย n เป็นตัวแปรที่มีเป็นค่าจำนวนเต็มเท่านั้น คือ n =...,-2,-1,0,1,2,3,... และสัญญาณไม่ต่อเนื่องจะเป็นฟังก์ชั่นของ n ดังนั้นจะเขียนแทนสัญญาณนี้ได้ว่า $x(n)$



รูปที่ 2.7 สัญญาณต่อเนื่องและสัญญาณไม่ต่อเนื่อง (โปรดสังเกตหน่วยของแกนนอนด้วย)

รูปของสัญญาณไม่ต่อเนื่องที่แสดงเปรียบเทียบกับสัญญาณต่อเนื่องดังในรูปที่ 2.7 เป็นรูปที่เขียนเพื่อแสดงให้เห็นรูปร่างของสัญญาณแต่จริงๆแล้ว เราจะไม่สามารถเห็นสัญญาณนี้ได้โดยตรงเหมือนกับที่เห็นสัญญาณแอนะลอกในออสซิโลสโคป แต่เราจะมองสัญญาณไม่ต่อเนื่องในลักษณะของ “ลำดับของค่า” หรือ “ลำดับของข้อมูล” โดยข้อมูลแต่ละตัวก็แทนค่าแต่ละค่าของสัญญาณนั่นเอง ซึ่งลำดับของค่าเหล่านี้นอกจากมันจะเป็นตัวแทนที่ถูกต้องของสัญญาณต่อเนื่องที่ถูกสุ่มมาแล้ว มันยังสามารถถูกนำไปประมวลผลได้ด้วยคอมพิวเตอร์หรือวงจรทางดิจิตอลได้ด้วย ซึ่งในอดีตได้เคยมีผู้คิดว่าการประมวลผลสัญญาณดิจิตอลให้ผลลัพธ์เป็นเพียงการประมาณของการประมวลผลทางแอนะลอก แต่เนื่องจากทฤษฎีที่ถูกต้องที่ได้มีการคิดค้นกันมาทำให้การประมวลผลทางดิจิตอลให้ผลลัพธ์ที่ถูกต้อง และสามารถพิสูจน์ได้ว่าไม่ใช่ค่าประมาณของทางแอนะลอก ส่วนคำว่าสัญญาณดิจิตอลกับสัญญาณไม่ต่อเนื่องนั้นโดยทั่วไปหมายถึงสัญญาณในลักษณะเดียวกันแต่มีความหมายต่างกันเล็กน้อยตามความรู้สึกและประสบการณ์ของผู้

พูด เมื่อพูดถึงคำว่าสัญญาณไม่ต่อเนื่อง เราจะหมายถึงลำดับของข้อมูลดังที่ได้กล่าวมาแล้วซึ่งข้อมูลแต่ละตัวนั้นอาจมีขนาดเท่าไรก็ได้โดยไม่จำกัดความละเอียด ซึ่งค่าของสัญญาณนี้เมื่อนำไปใช้งานจริงก็จะถูกแทนด้วยค่าดิจิตอลที่มีจำนวนบิทจำกัด เช่น ในรูปที่ 2.8 ได้แสดงให้เห็นว่า สัญญาณ $x(n)$ แต่ละค่าเมื่อนำไปใช้สามารถแทนได้ด้วย 8 บิท เป็นต้น เมื่อพูดถึงคำว่าสัญญาณดิจิตอล เรามักหมายถึงสัญญาณไม่ต่อเนื่องที่แต่ละค่าของสัญญาณถูกแทนค่าด้วยเลขฐานสองที่มีจำนวนบิทจำกัด อยู่ในรูป 0 กับ 1 แล้ว

ทฤษฎีของการประมวลผลสัญญาณดิจิตอลที่เราจะได้ศึกษาต่อไป และถึงแม้ชื่อที่คนนิยมเรียกจะเรียกว่า การประมวลผล “สัญญาณดิจิตอล” (Digital Signal Processing) แต่ถ้าดูความหมายที่แท้จริงของทฤษฎีแล้ว น่าจะเรียกว่าการประมวลผลสัญญาณไม่ต่อเนื่อง (Discrete-time Signal Processing)มากกว่า ทั้งนี้เพราะว่า ทฤษฎีของการประมวลผลสัญญาณเป็นการกระทำโดยมองสัญญาณขาเข้าเป็นลักษณะของลำดับของข้อมูล (ซึ่งคือสัญญาณไม่ต่อเนื่อง) โดยนำข้อมูลเหล่านี้มาประมวลผล เช่น บวก ลบ คูณ หาร เพื่อหาสัญญาณขาออกในลักษณะเป็นลำดับข้อมูลเช่นเดียวกัน ซึ่งค่าของสัญญาณเหล่านี้เมื่อนำไปใช้งานสามารถแทนได้ด้วยข้อมูลดิจิตอล เช่น 00110110, 00111000, 01100011, …

… ≡ x(0), x(1), x(2), x(3), x(4), …
1.21, 1.33, 1.74, 0.45, -1.56, …

ค่าของสัญญาณเหล่านี้ เมื่อนำไปใช้งานสามารถแทนได้ด้วยข้อมูลดิจิตอล เช่น 00110110, 00111000, 01100011, …

รูปที่ 2.8 สัญญาณไม่ต่อเนื่อง ก็คือ ลำดับของข้อมูลนั่นเอง

ส่วนประกอบในระบบประมวลผลสัญญาณดิจิตอล

ระบบประมวลผลสัญญาณโดยส่วนใหญ่ แสดงในรูปที่ 2.9 ซึ่งประกอบด้วยส่วนต่าง ๆ ดังต่อไปนี้

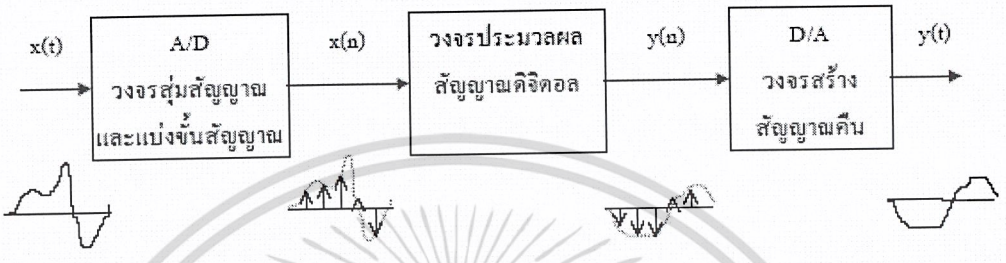
2.6.1 วงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล ซึ่งสามารถแบ่งได้เป็น 2 กระบวนการย่อย คือ

1 วงจรสุ่มสัญญาณ (Sampler) สัญญาณขาเข้าของวงจรนี้จะเป็นสัญญาณแบบแอนะลอก $x(t)$ ส่วนสัญญาณขาออกเป็นสัญญาณไม่ต่อเนื่อง $x(n)$ พารามิเตอร์วงจรสุ่มสัญญาณนี้ก็คือ ค่าอัตราการสุ่ม (sampling rate) หรือความถี่ในการสุ่ม ใช้สัญลักษณ์แทนว่า f_s ค่านี้เป็นตัวกำหนดว่าวงจรสุ่มจะสุ่มสัญญาณด้วยอัตรากี่ครั้งต่อวินาทีหรือกี่เฮิรซ์ (Hz)

2 วงจรแบ่งขั้นสัญญาณ (Quantizer) สัญญาณ $x(n)$ ที่ได้จากวงจรสุ่มสัญญาณ ถือว่ามีความละเอียดเต็มที่ในทางขนาด ซึ่งในทางปฏิบัติเมื่อนำไปใช้งานจะต้องลดความละเอียดของ $x(n)$ ลงให้สามารถแทนได้ด้วยสัญญาณดิจิตอลที่มีจำนวนบิตจำกัด กระบวนการลดความละเอียดนี้เรียกว่า การแบ่งขั้นของสัญญาณ (quantization) ความละเอียดที่ได้จากการแบ่งขั้นสัญญาณขึ้นอยู่กับจำนวนบิตที่จะใช้การ

แบ่งขั้นสัญญาณทำให้ค่าสัญญาณที่ได้คลาดเคลื่อนไป $x(n)$ จริงซึ่งจะส่งผลเหมือนมีสัญญาณรบกวนเข้ามาในระบบ

วงจรสุ่มสัญญาณรวมกับวงจรแบ่งขั้นสัญญาณ ในทางปฏิบัติก็คือตัวแปลงสัญญาณแอนะลอกเป็นดิจิตอล (A/D converter) นั่นเอง ซึ่งจะรวมสองกระบวนการนี้อยู่ในวงจรเดียวกัน และโดยทั่วไปเราจะใช้ตัวแปลงสัญญาณแอนะลอกเป็นดิจิตอลในรูปของวงจรรวมสำเร็จรูป (IC)



รูปที่ 2.9 ส่วนประกอบในระบบประมวลผลสัญญาณดิจิตอล

2.6.2 วงจรประมวลผลสัญญาณ

ส่วนนี้เป็นหัวใจหลัก ซึ่งจะทำหน้าที่ประมวลผลสัญญาณ $x(n)$ เพื่อกระทำผลบางอย่างกับสัญญาณ เช่น เป็นวงจรกรองความถี่บางย่านออกและให้ผลลัพธ์ของการประมวลผลเป็นสัญญาณขาออก $y(n)$ วงจรประมวลผลสัญญาณนี้ถ้าจะพิจารณากันอย่างง่ายๆแท้ที่จริงก็คือตัวคำนวณนั่นเอง กล่าวได้ว่ามันกระทำการคำนวณหาสัญญาณขาออกจากสัญญาณขาเข้า โดยมองเห็นสัญญาณขาเข้าในลักษณะของ "ลำดับของค่า" ตัวอย่างการประมวลผลง่ายๆ เช่น สมการสำหรับการประมวลผล คือ

$$y(n) = 0.5(x(n) + x(n-1)) \qquad (2.6.1)$$

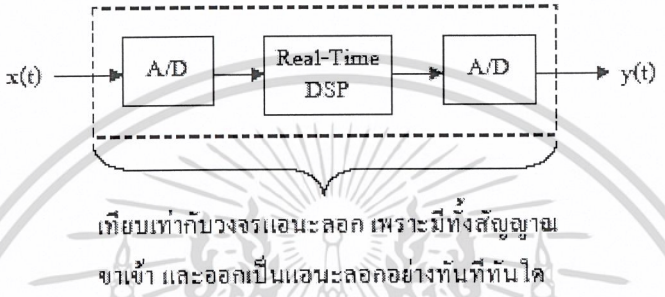
2.6.3 วงจรสร้างสัญญาณคืน (Signal Reconstruction)

ใช้ในระบบที่มีสัญญาณขาออกสุดท้ายเป็นสัญญาณต่อเนื่อง (การประมวลผลสัญญาณบางอย่างต้องการสัญญาณขาออกเป็นไม่ต่อเนื่องก็ไม่จำเป็นต้องมีส่วนที่ 2.6.3 นี้) โดยทำหน้าที่แปลงสัญญาณไม่ต่อเนื่อง $y(n)$ ให้กลับเป็นสัญญาณต่อเนื่อง (t) ซึ่งจะเป็นสัญญาณขาออกสุดท้ายของระบบ วงจรประเภทนี้ก็คือตัวแปลงสัญญาณดิจิตอลเป็นแอนะลอก (D/A converter) นั่นเอง ซึ่งก็มีในรูปแบบวงจรรวมสำเร็จรูปเช่นกัน

2.7 การประมวลผลแบบเวลาจริงกับการเลือกใช้ตัวประมวลผลสัญญาณ

การประมวลผลแบบเวลาจริง (Real-Time Signal Processing) หมายถึง การประมวลผลที่กระทำที่อัตราจริงของสัญญาณขาเข้าและให้สัญญาณขาออกทันกับสัญญาณขาเข้าที่เข้ามา เช่น ในระบบที่มีอัตราการสุ่มของสัญญาณขาเข้าและขาออกเท่ากัน เมื่อมีสัญญาณขาเข้าเข้ามา 1 ค่าระบบจะต้องประมวลผลให้

ได้สัญญาณขาออก 1 ค่าก่อนที่สัญญาณขาเข้าตัวถัดไปจะเข้ามา เป็นต้น การประมวลผลแบบเวลาจริงนี้มีการประยุกต์ใช้งานอย่างมาก และเป็นตัวแทนที่แท้จริงของระบบที่เคยใช้เป็นแบบแอนะลอกดังแสดงในรูปที่ 2.10 อย่างไรก็ตามการที่มีการประมวลผลแบบเวลาจริงไม่จำเป็นต้องมีสัญญาณขาเข้าและออก เป็นสัญญาณแอนะลอกทั้งคู่เสมอไป ยกตัวอย่างเช่น การถอดรหัสสัญญาณเสียงที่ถูกบีบอัดข้อมูลมา ในกรณีนี้สัญญาณขาเข้าเป็นดิจิตอลซึ่งคือข้อมูลเสียงที่บีบอัดมาแล้ว และส่วนสัญญาณขาออกคือสัญญาณเสียงแอนะลอกที่ต้องส่งออกที่ลำโพง ดังนั้นการประมวลผลจะต้องเกิดที่อัตราสุ่มจริงของสัญญาณเสียงขาออก อันนี้ก็ถือว่าเป็นการประมวลผลแบบเวลาจริง



รูปที่ 2.10 การประมวลผลแบบเวลาจริงทำให้ DSP ทำหน้าที่เหมือนเป็นวงจรแอนะลอกได้

ส่วนการประมวลผลแบบไม่เป็นเวลาจริงนั้นไม่มีข้อบังคับทางด้านเวลาในการประมวลผล ยกตัวอย่างเช่นการจำลองระบบประมวลผลด้วย MATLAB ในคอมพิวเตอร์ในที่นี้ถือว่าคอมพิวเตอร์เป็นตัวประมวลผล ซึ่งถ้าใช้คอมพิวเตอร์ที่เร็วเราก็ได้ผลลัพธ์เร็ว แต่ถ้าใช้คอมพิวเตอร์ที่ช้าเราก็จะได้ผลลัพธ์ช้า แต่ผลลัพธ์ที่ได้ไม่แตกต่างกันเลยไม่ว่าจะเร็วหรือช้า ทั้งนี้เพราะการประมวลผลไม่ได้เกิดขึ้นที่อัตราการสุ่มจริงของสัญญาณขาเข้าหรือขาออก ตัวอย่างอีกอันหนึ่ง เช่น การใช้โปรแกรมพวกตบแต่งรูปภาพ (ภาพนิ่ง) เช่น PhotoShop ซึ่งภาพนิ่งนี้ถือเป็นสัญญาณไม่ต่อเนื่องสองมิติและโปรแกรมพวกนี้ก็ถือเป็นโปรแกรมที่มีฟังก์ชันในการประมวลผลภาพ (Image Processing) เนื่องจากภาพนิ่งไม่มีอัตราการสุ่มของข้อมูลที่เทียบต่อเวลา ดังนั้นการประมวลผลภาพนิ่งจึงถือได้ว่าไม่มีข้อบังคับทางด้านเวลา (ถ้าไม่เอาอารมณ์ของผู้ใช้มาเป็นเกณฑ์ด้วย) จึงไม่เป็นการประมวลผลแบบเวลาจริง

การประมวลผลสัญญาณแบบเวลาจริงนั้นจะทำให้เกิดข้อกำหนดที่สำคัญขึ้นมาต่อการเลือกใช้ตัวประมวลผลสัญญาณ นั่นก็คือในการที่ต้องมีตัวประมวลผลที่เร็วพอที่จะประมวลผลสัญญาณให้ทันได้ โดยเฉพาะอย่างยิ่งถ้าสัญญาณที่ต้องการประมวลผลมีอัตราการสุ่มที่สูงหรืออัลกอริธึมที่ใช้มีความซับซ้อนในการคำนวณมาก ก็จำเป็นที่จะต้องใช้ตัวประมวลผลที่มีความเร็วสูงมากยิ่งขึ้น มีทางเลือกใหญ่ๆอยู่ 3 ทางในการทำตัวประมวลผล คือ

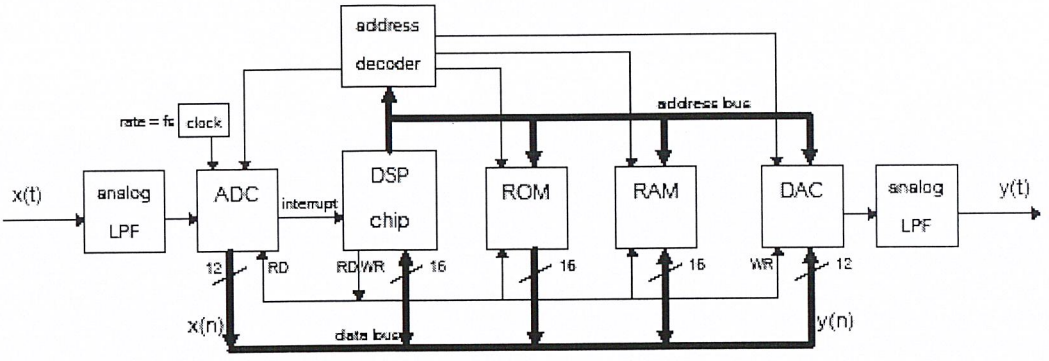
2.7.1 การเขียนซอฟท์แวร์เพื่อใช้กับคอมพิวเตอร์หรือใช้กับชิพไมโครโปรเซสเซอร์ทั่ว ๆ ไป

ถึงแม้ว่าคอมพิวเตอร์หรือไมโครโปรเซสเซอร์จะไม่ได้ออกแบบมาเฉพาะสำหรับการประมวลผลสัญญาณ แต่เราก็สามารถนำมันมาใช้ได้ในงานที่ต้องการอัตราการประมวลผลไม่มากนัก หรือในการ

ประมวลผลแบบไม่เป็นเวลาจริง อย่างไรก็ตามในปัจจุบันคอมพิวเตอร์ส่วนบุคคลมีความเร็วสูงมากจนสามารถนำมาใช้ทำการประมวลผลแบบเวลาจริงหลายๆอย่างได้ ตัวอย่างที่เห็นได้ชัดเช่น การถอดรหัสของสัญญาณเสียงหรือวีดีโอที่ถูกบีบอัดข้อมูลมาด้วยมาตรฐานMPEG ซึ่งแต่ก่อนเราต้องใช้ฮาร์ดแวร์พิเศษในการถอดรหัส แต่ปัจจุบันใช้เพียงซอฟท์แวร์ก็สามารถทำได้แล้ว โดยอาศัย CPU ที่มีความเร็วสูงขึ้น

2.7.2 การใช้ซอฟท์แวร์ร่วมกับชิพ DSP

ชิพ DSP เป็นชื่อเล่นของชิพประมวลผลสัญญาณ (Digital Signal Processor) ซึ่งก็คือไมโครโปรเซสเซอร์ที่ถูกออกแบบมาสำหรับงานประมวลผลสัญญาณแบบเวลาจริงโดยเฉพาะ และสำหรับไมโครโปรเซสเซอร์ประเภทนี้จะมีสถาปัตยกรรมที่เอื้ออำนวยต่อการคำนวณ และการโอนถ่ายข้อมูลที่มีประสิทธิภาพและมีความเร็วสูง เช่น การมีคำสั่งพิเศษในการคูณ, การบวกสะสม, หรือการอ้างข้อมูลแบบ circular buffer เป็นต้น บางชนิดยังสามารถทำการประมวลผลหลายๆส่วนได้พร้อมกันในตัวเดียว (multi-processing) อีกด้วย บริษัทที่เป็นผู้นำด้านการผลิตชิพ DSP ได้แก่ Texas Instruments, Motorola, Analog Devices,และ AT&T เป็นต้น ซึ่งชิพ DSP นี้มีทั้งประเภทที่เป็นการประมวลผลข้อมูลแบบจำนวนเต็ม (fixed-point)และประเภทที่ประมวลผลข้อมูลแบบเลขอิงดรรชนี (floating-point) การใช้งานชิพ DSP นั้นทำได้โดยเขียนเป็นโปรแกรมภาษาแอสเซมบลี หรือภาษาซีแล้วใช้คอมไพล์เลอร์แปลเป็นแอสเซมบลี ข้อดีของการเขียนเป็นภาษาแอสเซมบลีโดยตรงคือ สามารถควบคุมการทำงานของชิพได้เต็มที่ ทำให้สามารถออกแบบโปรแกรมให้ทำงานได้เร็วกว่า และมีขนาดโปรแกรมเล็กกว่าการใช้ภาษาซี แต่ข้อเสียก็คือ ภาษาแอสเซมบลีเขียนยากกว่า และไม่สามารถโอนย้ายโปรแกรมไปทำงานได้ในชิพต่างตระกูลกันหรือต่างผู้ผลิตกันได้ การต่อวงจรเพื่อใช้งานชิพ DSP ก็ทำเช่นเดียวกับการต่อวงจรไมโครโปรเซสเซอร์ทั่วๆไปเพียงแต่มีตัวแปลงสัญญาณแอนะลอกเป็นดิจิตอล (ADC) และดิจิตอลเป็นแอนะลอก (DAC) เพิ่มขึ้นมาเท่านั้น ในรูปที่ 2.11 เป็นแผนภาพทั่วไปของวงจร ซึ่งใช้ชิพ DSP แบบ fixed-point 16 บิต เช่น TMS320C50 ของ Texas Instruments โดยมันจะมีบัสข้อมูลขนาด 16 บิต มีตัวคูณและประมวลผลอื่นๆขนาด 16 บิตอยู่ภายใน ในรูปนี้เราใช้ DAC และ ADC ขนาด 12 บิต ซึ่งมีขาข้อมูล 12 ขาเพื่อส่งข้อมูลแบบขนานและต่อเข้ากับ 12 บิตล่างของบัสข้อมูล สังเกตว่ามีสัญญาณนาฬิกาซึ่งมีความถี่ f_s ป้อนให้กับ ADC เพื่อเป็นตัวกำหนดอัตราการสุ่มสัญญาณแอนะลอก ซึ่งก็คืออัตราของข้อมูลที่จะต้องถูกอ่านเข้าชิพ DSP ไปประมวลผล ชิพ DSP หลายยี่ห้อมีหน่วยความจำ ROM และ RAM บางส่วนอยู่ภายในตัวเอง ทำให้เพิ่มความเร็วในการทำงานและสะดวกในการใช้งานมาก โดยงานที่ไม่ต้องการใช้ปริมาณ ROM และRAM มากนัก ก็อาจไม่จำเป็นต้องต่อหน่วยความจำภายนอกเลย



รูปที่ 2.11 แผนภาพแสดงตัวอย่างของวงจรที่ใช้งานชิพ DSP

2.7.3 การใช้ฮาร์ดแวร์หรือไอซีที่ออกแบบเฉพาะงาน ฮาร์ดแวร์ในที่นี้ก็หมายถึง วงจรดิจิตอลซึ่งสามารถออกแบบให้ทำการประมวลผลข้อมูลได้เช่นเดียวกัน อัลกอริธึมที่เป็นที่นิยม เช่น FFT (Fast Fourier Transform) หรือตัวกรองดิจิตอลนั้น เราอาจสามารถหาซื้อได้ทั่วไปเป็นไอซีสำเร็จรูปที่ทำเฉพาะฟังก์ชั่นนั้นๆ แต่ถ้าต้องการอัลกอริธึมที่เฉพาะมากขึ้น ก็อาจต้องทำการออกแบบเป็นไอซีเฉพาะงานเอง (Application Specific Integrated Circuits หรือ ASIC) ซึ่งแน่นอนว่าต้นทุนในการออกแบบสำหรับทางเลือกนี้ค่อนข้างสูง ทางเลือกอีกทางหนึ่งคือ การใช้ไอซีดิจิตอลประเภทโปรแกรมได้ หรือ FPGA (Field Programmable Gate Array) ซึ่งปัจจุบันมีขนาดใหญ่มากพอที่จะนำมาใช้ทำการประมวลผลสัญญาณได้ การใช้ FPGA จะมีต้นทุนในการออกแบบที่ถูกกว่า ASIC การเลือกใช้ตัวประมวลผลแต่ละแบบก็ขึ้นอยู่กับลักษณะของงาน ความเร็วที่ต้องการและต้นทุน ถ้าต้องการทำอุปกรณ์ที่มีการประมวลผลแบบเวลาจริง โดยทั่วไปการใช้ชิพ DSP จะดีที่สุด (ซึ่งชิพ DSP ก็มีหลากหลายขนาดและความเร็วให้เลือกใช้อีก) แต่หากการประมวลผลไม่ซับซ้อน หรืออัตราข้อมูลไม่สูงมากจนสามารถใช้ไมโครโปรเซสเซอร์ธรรมดาได้ การใช้ไมโครโปรเซสเซอร์ก็จะทำให้ต้นทุนต่ำลงได้ ในกรณีที่ต้องการอัตราการประมวลผลสูงมากๆ เราก็อาจต้องใช้ฮาร์ดแวร์ในการประมวลผล ซึ่งโดยทั่วไปก็จะมีต้นทุนที่สูงขึ้น

2.8 งานที่มีการประยุกต์ใช้การประมวลผลสัญญาณดิจิตอล

ปัจจุบันมีงานหลายอย่างที่ได้นำเอาการประมวลผลสัญญาณดิจิตอลไปใช้งาน คงจะสามารถยกได้เพียงตัวอย่างแค่ส่วนหนึ่งของมันเท่านั้น ซึ่งได้แก่

1. การประมวลผลเสียง เช่น การบีบอัดเสียง หรือเข้ารหัสเสียง (speech coding), การรู้จำเสียง (speech recognition), การเติมเอฟเฟกต์เสียง (sound effect), การผสมเสียง, การกรองเสียงรบกวน, การสังเคราะห์เสียงดนตรี (music synthesizer) เป็นต้น

2. ในระบบสื่อสาร ได้แก่ modulation/demodulation, การชดเชยผลของช่องสัญญาณ (channel equalizer) ในอุปกรณ์โมเดมและโทรศัพท์มือถือ, การกรองเสียงสะท้อนในระบบโทรศัพท์ทางไกลและระบบการประชุมทางไกล (video conferencing), สายอากาศแบบปรับรูปแบบการรับได้เอง, ระบบเรดาร์และโซน่าร์, ระบบนำทาง (navigation system), GPS เป็นต้น

2.9 ข้อดีของการใช้การประมวลผลสัญญาณดิจิตอล

ข้อดีของการใช้การประมวลผลสัญญาณดิจิตอลที่เหนือกว่าการใช้วงจรในระบบแอนะลอกมีดังนี้

1. ความสามารถในการโปรแกรมได้ทำให้ง่ายต่อการออกแบบ, เปลี่ยนแปลงแก้ไข, และทดสอบสำหรับวงจรแอนะลอก ถ้าต้องการเปลี่ยนคุณสมบัติอะไรบางอย่างอาจหมายถึงการต้องออกแบบวงจรใหม่เลย

2. ความถูกต้องแม่นยำที่ดีกว่า ความถูกต้องของการประมวลผลสัญญาณดิจิตอลขึ้นอยู่กับจำนวนบิทที่ใช้แทนสัญญาณและพารามิเตอร์ต่างๆ ซึ่งมีความยืดหยุ่นและควบคุมได้ง่ายคือ ในงานที่เราต้องการความแม่นยำสูงเราก็จะใช้จำนวนบิทที่มากขึ้น อีกทั้งในช่วงของการออกแบบการจำลองระบบที่ออกแบบในคอมพิวเตอร์ จะให้ผลที่ตรงกับความเป็นจริงเมื่อนำไปสร้างเป็นวงจรจริง

3. สามารถทำฟังก์ชั่นที่พิสดารที่ไม่สามารถทำได้ด้วยวงจรแอนะลอกหรือทำได้ยากมาก เช่นตัวกรองแบบปรับตัวได้ (adaptive filter) ตามสภาวะของสัญญาณรบกวน, สายอากาศที่ปรับทิศทางการรับเองได้, การเติมเอฟเฟกต์เสียงให้เป็นเสียง 3 มิติ เป็นต้น

4. มีเสถียรภาพที่ไม่ขึ้นกับเวลา และอุณหภูมิ

5. DSP เกี่ยวข้องโดยตรงกับเทคโนโลยีคอมพิวเตอร์และ VLSI (ชิพ DSP ก็จัดเป็นชิพประเภท VLSI) ซึ่งเทคโนโลยีเหล่านี้กำลังเจริญก้าวหน้าอย่างรวดเร็ว ทั้งในด้านความเร็วที่สูงขึ้นความจุของชิพที่มากขึ้น การกินกำลังไฟที่ต่ำลงและราคาก็ถูกลง ข้อดีนี้ถึงแม้เป็นผลพลอยได้แต่ก็มีความสำคัญมากเพราะมันหมายถึงว่า จะทำให้ต้นทุนของการใช้การประมวลผลสัญญาณดิจิตอลจะต่ำลงๆตามความก้าวหน้าของเทคโนโลยี เราจะเห็นได้ว่างานบางอย่างที่ในอดีตการใช้วงจรแอนะลอกให้ต้นทุนที่ต่ำกว่า แต่ในปัจจุบันกลับใช้แบบดิจิตอลแล้วคุ้มค่ากว่า หรืออัลกอริธึมบางอย่างที่มีผู้คิดค้นได้ในอดีต เช่น Kalman Filter แต่ไม่สามารถนำมาใช้ได้ในเวลานั้น เนื่องจากมีความซับซ้อนของการคำนวณมากทำให้ไม่คุ้มค่ากับการนำมาใช้ ก็ปรากฏว่า อัลกอริธึมเหล่านั้นกลับนำมาใช้งานได้จริงในปัจจุบัน ทั้งนี้เป็นผลโดยตรงจากความก้าวหน้าของเทคโนโลยีคอมพิวเตอร์และ VLSI

2.10 ขีดจำกัดของการประมวลผลสัญญาณดิจิตอล

ทุกอย่างที่มีข้อดีก็มักจะมีขีดจำกัดด้วยเสมอ เทคโนโลยีของดิจิตอลที่มีข้อดีต่างๆมากมายตามที่กล่าวมาก็เช่นเดียวกัน ขีดจำกัดของการใช้การประมวลผลสัญญาณดิจิตอลพอจะแจกแจงได้ดังนี้

1. สัญญาณแอนะลอกที่มีแถบความถี่ (bandwidth) สูงมากๆ ไม่สามารถใช้กับการประมวลผลสัญญาณดิจิตอลได้ เนื่องจากสัญญาณพวกนี้ต้องการอัตราการสุ่มที่สูงมากเพื่อแปลงเป็นดิจิตอลทำให้ต้องการตัวประมวลผลที่เร็วมากทำให้ไม่คุ้มค่าต่อการใช้งาน และนอกจากนี้ขีดจำกัดที่อาจสำคัญมากกว่าตัวประมวลผลก็คือ การที่ต้องมีตัวแปลงสัญญาณระหว่างแอนะลอกกับดิจิตอลที่มีความเร็วสูงมาก ซึ่งในปัจจุบัน เทคโนโลยีของการแปลงสัญญาณแอนะลอกเป็นดิจิตอลสามารถทำได้ที่อัตราสุ่มสูงสุดในช่วงความถี่ *MHz* ซึ่งหมายความว่า เราไม่สามารถประมวลผลสัญญาณมีแถบความถี่สูงกว่าช่วงความถี่ *MHz* ได้ (ครึ่งหนึ่งของอัตราการสุ่ม)

2. งานที่ต้องการกินกำลังไฟที่ต่ำมากๆอาจจะต้องทำด้วยวงจรแอนะลอกอยู่ ในปัจจุบันถึงแม้ชิพ VLSI จะกินกำลังไฟต่ำลงมากเมื่อเทียบกับอดีต ประกอบกับเทคโนโลยีของแบตเตอรี่ที่ก้าวหน้าไปมาก ทำให้อุปกรณ์พกพาหลายๆอย่างที่มี DSP เป็นส่วนประกอบ เช่น โทรศัพท์มือถือมีขนาดเล็กกระทัดรัดลงมากแถมยังใช้งานได้นานขึ้นอีก แต่อย่างไรก็ตามวิธีการประมวลผลก็ยังจัดเป็นกระบวนการที่กินกำลังไฟไปพอสมควร อุปกรณ์ที่ต้องการให้มีขนาดเล็กมากๆ ที่ซึ่งไม่ต้องการใส่แบตเตอรี่ที่มีขนาดใหญ่ลงไป เช่น อุปกรณ์ช่วยได้ยิน ก็จะมีข้อได้เปรียบของการออกแบบเป็นชิพแอนะลอกในด้านที่จะสามารถกินกำลังไฟได้ต่ำกว่า

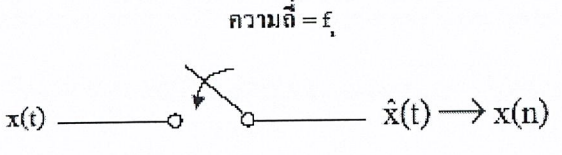
3. อุปกรณ์บางอย่างถึงแม้ทำได้ดีกว่าด้วยเทคโนโลยีดิจิตอล แต่ก็ด้วยต้นทุนที่สูงกว่าจึงมีตลาดที่จำกัดอยู่เฉพาะผู้ใช้ที่มีกำลังซื้อ อุปกรณ์เหล่านี้จึงยังคงมีใช้อยู่ทั้งแบบดิจิตอลและแอนะลอกเช่น โทรทัศน์ดิจิตอล กับโทรทัศน์แอนะลอก, เครื่องเล่น DVD กับเครื่องเล่นวีดีโอเทป, และออสซิลโลสโคปดิจิตอลกับออสซิลโลสโคปแอนะลอก, ฯลฯ เป็นต้น

4. ขีดจำกัดในข้อที่ 1 ถึง 3 จะค่อยน้อยลงๆตามความเจริญของเทคโนโลยีคอมพิวเตอร์ และVLSI ดังที่ได้กล่าวมาแล้ว แต่อย่างไรก็ตามมีวงจรบางประเภทที่ต้องสร้างด้วยเทคโนโลยีแอนะลอกเสมอ (ถึงแม้ในอนาคตก็ตาม) และจริงๆแล้วระบบประมวลผลสัญญาณดิจิตอลก็ต้องพึ่งพาวงจรเหล่านี้ด้วย นั่นคือ วงจรขยายสัญญาณต่างๆ, ดิจิตอลเป็นแอนะลอก, ตัวกรองแอนะลอกในส่วน front-end (ก่อนตัวแปลงแอนะลอกเป็นดิจิตอล) และตัวกรองแอนะลอกในส่วน back-end (หลังตัวแปลงดิจิตอลเป็นแอนะลอก) เป็นต้น

2.11 การสุ่มสัญญาณ และการสร้างสัญญาณคืน

2.11.1 การสุ่มสัญญาณ (Sampling)

แนวคิดอย่างง่ายๆของการสุ่มก็คือ การนำเอาสัญญาณขาเข้าแบบต่อเนื่องมาผ่านสวิทช์อุดมคติที่ต่อวงจรที่ตำแหน่งเวลาเท่ากับ $\ldots,-2T,-T,0,T,2T,3T,\ldots$ และเปิดวงจรที่เวลาอื่นๆ นั่นคือมีความถี่ของการตัดต่อวงจรเท่ากับ f_s ครั้งต่อวินาที หรือมีคาบของการสุ่มเท่ากับ $T = 1/f_s$ ดังในรูปที่ 2.12 เรียกสัญญาณขาออกว่า $\hat{x}(t)$



รูปที่ 2.12 การสุ่มด้วยสวิทช์อุดมคติ

แน่นอนว่าถ้ามีสวิทช์อย่างนี้จริง เราจะได้สัญญาณขาออก $\hat{x}(t)$ ซึ่งเป็นสัญญาณแอนะลอกที่มีลักษณะเป็นอุดมคติคือ มีค่าที่เฉพาะเวลา $...,-2T,-T,0,T,2T,3T,...$ เท่านั้น ส่วนที่เวลาอื่นๆมีค่าเป็นศูนย์ สัญญาณนี้หน้าตาเหมือนสัญญาณไม่ต่อเนื่องดังที่กล่าวมาแล้วข้างต้นทุกประการ เพียงแต่มองมันเป็นสัญญาณแอนะลอกในแกนเวลา t ซึ่งถึงแม้เราจะไม่ใช้สัญญาณนี้ในลักษณะเป็นสัญญาณแอนะลอก แต่การมองนี้ก็เพื่อการวิเคราะห์ในเชิงความถี่ของสัญญาณ ซึ่งจะมีประโยชน์ในการบอกถึงขีดจำกัดของกระบวนการสุ่มได้เราจะลองมาวิเคราะห์หาสเปกตรัมของสัญญาณ $\hat{x}(t)$ ดูว่าสอดคล้องกับสเปกตรัมของ $x(t)$ อย่างไรโดยจะพิสูจน์ในแง่คณิตศาสตร์ ถ้านิยามว่ามีสัญญาณอิมพัลส์ (impulse signal) หรือเขียนแทนด้วยสัญลักษณ์ว่า $\delta(t)$ เป็นสัญญาณที่มีค่าเท่ากับ 1 ที่เวลา $t=0$ และเป็น 0 ที่เวลาอื่นๆดังนี้

$$\delta(t) = \begin{cases} 1, t = 0 \\ 0, t = \text{other} \end{cases} \tag{2.11.1}$$

เราอาจจะมองว่าการสุ่มคือ การนำเอาสัญญาณขาเข้ามาคูณเข้ากับสัญญาณอิมพัลส์หลายๆลูกที่มีคาบเท่ากับ T (ระยะห่างระหว่างอิมพัลส์แต่ละลูก) ขอนิยามสัญญาณอิมพัลส์หลายๆลูกนี้เป็นสัญญาณ $s(t)$ ซึ่งรูปร่างของมันแสดงในรูปที่ 2.13 ในทางคณิตศาสตร์สามารถเขียน $s(t)$ ได้ว่าเป็นผลรวมของสัญญาณอิมพัลส์ที่ตำแหน่งเวลาต่างๆ $...,-2T,-T,0,T,2T,3T,...$ ดังนี้

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{2.11.2}$$

เมื่อนำสัญญาณ $s(t)$ นี้คูณเข้ากับสัญญาณขาเข้า $x(t)$ จะได้สัญญาณที่เป็นขาออกของตัวสุ่มสัญญาณ

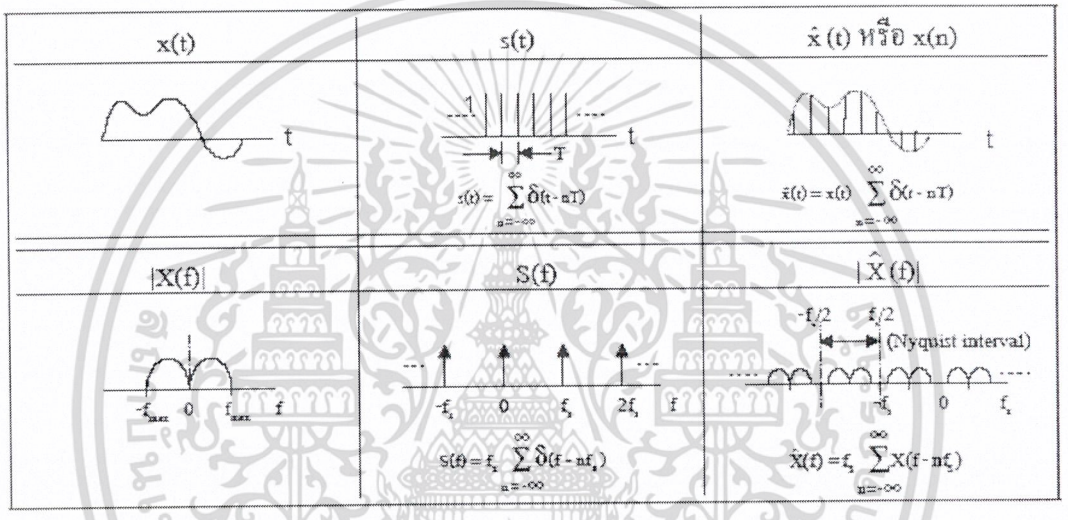
$$\hat{x}(t) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{2.11.3}$$

ตัวอย่างของสัญญาณ $\hat{x}(t)$ มีลักษณะดังแสดงในรูปที่ 2.13 สัญญาณนี้เราสามารถนิยามให้มันเป็นสัญญาณไม่ต่อเนื่องหรือเป็น "ลำดับของค่า" ดังที่ได้กล่าวมาซึ่งก็จะใช้สัญญลักษณ์เป็น $x(n)$ โดยใช้ค่า n เป็นตัวชี้แทน t จะได้ว่า $x(n)$ มีความสัมพันธ์กับ $x(t)$ ดังนี้

$$x(n) = x(t)\big|_{t=nT} \tag{2.11.4}$$

ขอย้ำให้เข้าใจอีกครั้งว่าสัญญาณ $\hat{x}(t)$ กับ $x(n)$ นั้นหมายถึงสัญญาณตัวเดียวกัน เพียงแต่สัญญาณ $\hat{x}(t)$ เป็นการมองสัญญาณนี้ในลักษณะเป็นสัญญาณแอนะลอกในแกนของเวลา t ซึ่งเราจะเห็น

ว่าสัญญาณมีค่าที่เวลา t =...,-2T,-T,0,T,2T,3T,... แต่ว่าสัญญาณ $x(n)$ เป็นการมองสัญญาณเป็นสัญญาณไม่ต่อเนื่องหรือเป็นลำดับคือ มีค่าที่ n =...,-2,-1,0,1,2,... สัญญาณ $x(n)$ นี้ทิ้งความหมายของเวลาแบบแอนะลอกไปเรียบร้อยแล้วกลายเป็นเหมือนลำดับของข้อมูลเฉยๆ ในส่วนต่อๆไปเราจะทำงานกับสัญญาณประเภทนี้ และจะเห็นต่อไปว่าตัวประมวลผลสัญญาณดิจิตอลก็ไม่จำเป็นต้องรู้ค่าเวลาที่แท้จริงของสัญญาณเลยหรือไม่ต้องใช้ค่า f_s เลยในการประมวลผล แต่อย่างไรก็ตามเราต้องจดบันทึกไว้ว่า $x(n)$ ที่เป็นสัญญาณขาเข้าของระบบนี้ เกิดจากการสุ่มสัญญาณแอนะลอกมาด้วยอัตรา f_s เท่าไรเพื่อที่จะใช้วิเคราะห์ในเรื่องผลตอบสนองเชิงความถี่ของระบบ และใช้เป็นพารามิเตอร์ในการออกแบบระบบดังจะได้กล่าวในบทต่อๆ ไป



รูปที่ 2.13 สรุปสัญญาณต่าง ๆ ในการสุ่มทั้งภาคเวลาและความถี่

ลองมาดูในภาคความถี่บ้างว่าเกิดอะไรขึ้น สมมติว่าสัญญาณ $x(t)$ เมื่อใช้การแปลงฟูริเยร์ได้สเปกตรัมเป็น $X(f)$ ซึ่งสมมติว่ามีแถบความถี่ (bandwidth) จำกัดและมีองค์ประกอบความถี่สูงสุดอยู่ที่ $f_{\max}$ ดังในรูป 2.13 ส่วนสัญญาณ $s(t)$ จะสามารถพิสูจน์โดยใช้การแปลงฟูริเยร์ได้ว่ามีสเปกตรัมเป็นสัญญาณอิมพัลส์หลายลูกที่มีคาบคงที่เช่นกัน โดยคาบในที่นี้เท่ากับ f_s และมีขนาดคงที่เท่ากับ f_s เขียนเป็นสมการได้ดังสมการที่ 2.11.5

$$S(f) = f_s \sum_{n=-\infty}^{\infty} \delta(f - nf_s) \qquad (2.11.5)$$

สำหรับสเปกตรัมของสัญญาณขาออกคือ $\hat{X}(f)$ อาจหาได้โดยกฎที่ว่า การคูณในเชิงเวลาเท่ากับการคอนโวลูชัน (convolution) ในเชิงความถี่ นั่นคือเมื่อ $\hat{x}(t)$ เป็นผลคูณระหว่างสัญญาณ $s(t)$ กับ $x(t)$ จะได้ว่าสเปกตรัมของ $\hat{x}(t)$ คือ $\hat{X}(f)$ จะเท่ากับผลคูณคอนโวลูชันระหว่าง $X(f)$ กับ $S(f)$ ดังนี้

$$\hat{X}(f) = X(f) * S(f) \tag{2.11.6}$$

เมื่อแทนค่า $S(f)$ ลงไปแล้วจัดให้เป็นรูปอย่างง่ายจะได้

$$\hat{X}(f) = X(f) * f_s \sum_{n=-\infty}^{\infty} \delta(f - nf_s)$$

$$= f_s \sum_{n=-\infty}^{\infty} X(f) * \delta(f - nf_s)$$

$$\hat{X}(f) = f_s \sum_{n=-\infty}^{\infty} X(f - nf_s) \tag{2.11.7}$$

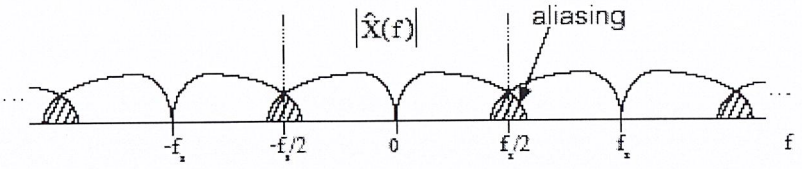
สมการนี้ประกอบด้วยผลบวกของเทอม $X(f - nf_s)$ ซึ่งเทอมนี้คือสเปกตรัมของสัญญาณขาเข้าที่เลื่อนจุดศูนย์กลางไปอยู่ที่ตำแหน่งความถี่ nf_s โดย n เป็นจำนวนเต็มตั้งแต่ $-\infty$ จนถึง $+\infty$ นั่นหมายถึงว่า สเปกตรัมของสัญญาณหลังการสุ่มคือ $\hat{X}(f)$ นั้นจะประกอบด้วยสำเนาของสเปกตรัมของสัญญาณก่อนการสุ่มคือ $X(f)$ อยู่อย่างค่อนข้างมากมาย โดยที่สำเนาเหล่านี้จะเกิดขึ้นที่ความถี่ $\ldots,-2f_s,-f_s,0,f_s,2f_s,\ldots$ เป็นศูนย์กลาง โดยสำเนาแต่ละตัวนี้มีชื่อเรียกอีกอย่างหนึ่งว่า เป็นภาพฉาย (image) ของสัญญาณ รูปร่างของ $\hat{X}(f)$ ที่ได้แสดงตัวอย่างดังในรูปที่ 2.13 ขอให้สังเกตว่าสเปกตรัม $\hat{X}(f)$ ที่หามาได้นี้ก็มีลักษณะที่เป็นอุดมคติในเชิงสัญญาณแอนะลอก ทั้งสัญญาณ $x(t)$ และ $\hat{X}(f)$ (ถ้ามองในเชิงแอนะลอก)เป็นสัญญาณที่เราสร้างขึ้นในทางคณิตศาสตร์ เพื่อประโยชน์ในการวิเคราะห์ขีดจำกัดของกระบวนการสุ่มต่อไป

ทฤษฎีการสุ่มสัญญาณ (Sampling Theorem)

ทฤษฎีการสุ่มสัญญาณระบุไว้ว่า ถ้าสัญญาณที่ต้องการสุ่มมีความถี่สูงสุดที่ $f_{\max}$ เพื่อให้ได้สัญญาณที่สุ่มแล้วเป็นตัวแทนที่ถูกต้องของสัญญาณนี้ ความถี่ในการสุ่มจะต้องมีค่ามากกว่าสองเท่าของความถี่สูงสุดในสัญญาณ หรือ

$$f_s > 2f_{\max} \tag{2.11.8}$$

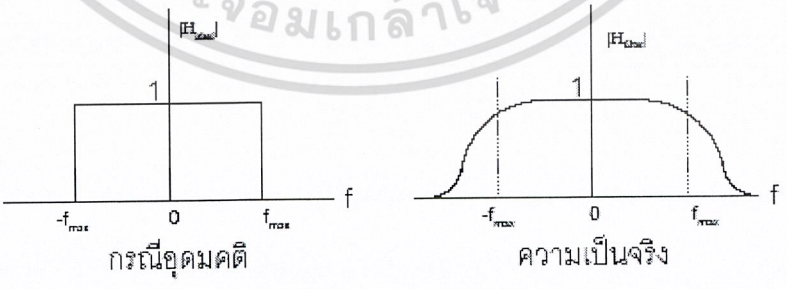
ซึ่งความถี่ที่ $2f_{\max}$ นี้มีชื่อเรียกพิเศษว่า "ความถี่ไนควิสท์" (Nyquist frequency) หรือเรียกอัตราไนควิสท์ ลองพิจารณาว่าถ้าเราใช้ f_s ต่ำกว่าค่าความถี่ไนควิสท์จะเกิดอะไรขึ้น ซึ่งผลที่เกิดขึ้นนี้จะดูได้อย่างชัดเจนในภาคความถี่ ถ้าเรายึดหลักที่ว่าได้พิสูจน์มาในหัวข้อที่แล้วว่าหลังการสุ่มจะเกิดสเปกตรัมของสัญญาณก่อนการสุ่มอยู่ที่ความถี่ $\ldots-2f_s,-f_s,0,f_s,2f_s,\ldots$ เป็นศูนย์กลาง แล้วลองวาดรูปคร่าวๆ ออกมาจะได้ดังแสดงในรูป 2.14



รูปที่ 2.14 สัญญาณในเชิงความถี่ที่เกิดขึ้น เมื่อใช้ f_s ต่ำกว่า $2f_{\max}$

เมื่อ f_s ต่ำกว่า $2f_{\max}$ จะเห็นได้ว่า สำเนาของ $X(f)$ ที่เกิดขึ้นจะมีช่วงของความถี่ส่วนปลายที่ซ้อนทับกัน เรียกองค์ประกอบความถี่ส่วนที่ซ้อนทับกันนี้ว่า aliasing (อ่านว่า เอเลียสซิ่ง) ซึ่ง aliasing นี้เป็นผลร้ายอย่างยิ่งกับระบบประมวลผลสัญญาณ เนื่องจากในระบบแบบไม่ต่อเนื่องเราจะสนใจความถี่ในช่วง $-f_s/2$ จนถึง $f_s/2$ ซึ่งเรียกว่า ช่วงไนควิสท์หรือย่านไนควิทซ์ (Nyquist interval) ซึ่งถ้าเกิด aliasing ซ้อนทับในช่วงไนควิสท์นี้ ก็จะถือว่าสัญญาณขาเข้าที่สุ่มมาได้มีความผิดเพี้ยนไปก่อนที่จะเข้าไปในส่วนของการประมวลผลเสียอีก หรือพูดอีกอย่างหนึ่งก็คือสัญญาณที่สุ่มมาได้ ไม่เป็นตัวแทนที่สมบูรณ์ของสัญญาณแอนะลอกขาเข้า จึงจำเป็นอย่างยิ่งที่เราต้องพยายามไม่ให้ aliasing เกิดขึ้น หรือให้เกิดน้อยที่สุดเท่าที่จะทำได้ ซึ่งกระทำได้สองวิธีคือ

1. การใช้ตัวกรองเพื่อป้องกัน aliasing (anti-aliasing filter) ในกรณีที่เราไม่มั่นใจว่าสัญญาณที่จะทำการสุ่มไม่มีความถี่จำกัดอยู่ที่ $f_{\max}$ เช่น อาจมีสัญญาณรบกวนความถี่สูง หรือสัญญาณอื่นๆปนอยู่ด้วย วิธีแก้ทำได้โดยการใช้ตัวกรองแอนะลอกแบบความถี่ต่ำผ่าน เพื่อจำกัดความถี่ของสัญญาณให้อยู่ในช่วงที่เราสนใจเท่านั้น (ต่ำกว่า $f_{\max}$) นั่นคือเราต้องการตัวกรองผ่านความถี่ต่ำที่มีความถี่ตัดที่ $f_{\max}$ ดังแสดงผลตอบสนองเชิงความถี่แบบอุดมคติของตัวกรองนี้ ในรูปซ้ายมือของรูปที่ 2.15

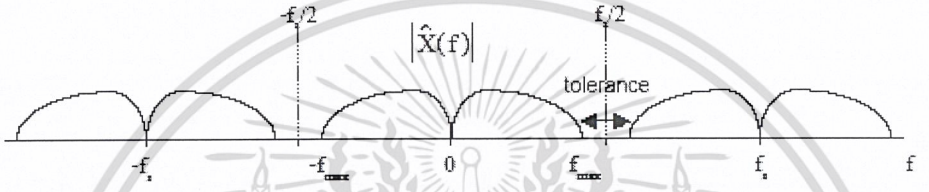


รูปที่ 2.15 ผลตอบสนองเชิงความถี่ของตัวกรองป้องกัน aliasing

อย่างไรก็ตามในชีวิตจริงตัวกรองแบบอุดมคติที่ต้องการไม่สามารถทำได้ และถ้าต้องการสร้างตัวกรองที่ใกล้เคียงกับอุดมคติก็จะทำให้ต้นทุนสูง ตัวกรองที่สามารถทำได้ในทางปฏิบัติมีรูปร่างประมาณ

ในรูปขวามือของรูปที่ 2.15 คือ มีความชันของการตัดความถี่ไม่คมเท่ากับตัวกรองอุดมคติจึงทำให้อาจมี aliasing บางส่วนเกิดขึ้นได้ถ้าเราใช้ความถี่ในการสุ่มพอดีเท่ากับ $2f_{\max}$

2. การสุ่มโดยใช้ความถี่เกินกว่า $2f_{\max}$ มากๆ หรือเรียกว่าการทำ Oversampling การสุ่มด้วยความถี่สูงขึ้นเป็นวิธีที่ใช้ป้องกัน aliasing เสริมจากวิธีแรก เช่นเดียวกันถ้าพิจารณาในเชิงความถี่จะเห็นว่าเมื่อ f_s มีค่าสูงขึ้น จะมีช่วงความถี่ที่เผื่อให้เกิด aliasing ได้กว้างขึ้น (ช่วงเผื่อนี้คือ ย่าน tolerance ที่เขียนอยู่ในรูปที่ 2.16 เป็นย่านที่มีความถี่ของสัญญาณที่เราสนใจอยู่) ช่วงเผื่อนี้เป็นย่านความถี่ที่ยอมให้มีความถี่ของสัญญาณที่ไม่ต้องการหลุดรอดเข้ามาในระบบได้บ้าง ดังนั้นการสุ่มด้วยความถี่สูงกว่าอัตราไนควิสท์จึงทำให้เราสามารถใช้ตัวกรองป้องกัน aliasing ที่ไม่ต้องมีคุณสมบัติคมมากนักได้



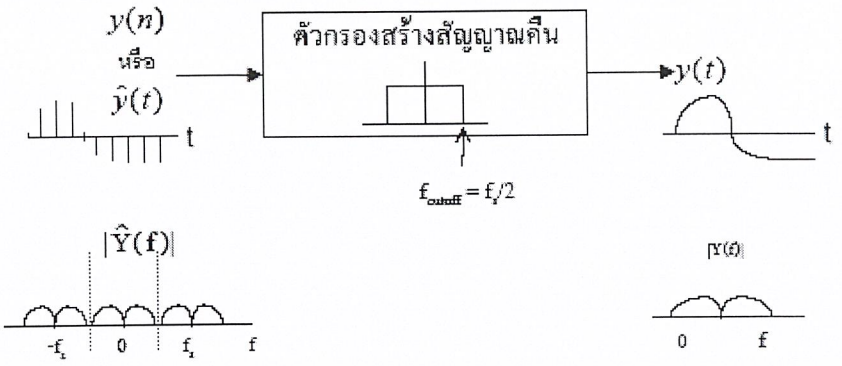
รูปที่ 2.16 สเปกตรัมของสัญญาณที่เกิดขึ้น เมื่อสุ่มด้วยความถี่สูงกว่า $2f_{\max}$ มาก ๆ

ในทางปฏิบัติจึงมักใช้ $f_s \geq 2.5 f_{\max}$ เพื่อชดเชยผลของการที่ตัวกรอง anti-aliasing ไม่เป็นอุดมคติ สำหรับ f_s สูงสุดที่เราจะสุ่มได้โดยทั่วไปก็ขึ้นอยู่กับขีดจำกัดด้านความเร็วของตัวแปลงแอนะลอกเป็นดิจิตอลและตัวประมวลผลที่เลือกใช้ ถ้าความถี่ f_s สูงขึ้นก็หมายถึงว่าต้องใช้วงจรแปลงสัญญาณที่เร็วขึ้นและใช้ปริมาณการประมวณผลที่มากขึ้นเพราะอัตราข้อมูลสูงขึ้น โดยช่วงเวลาที่ใช้ประมวลผลเพื่อให้ได้ค่าแต่ละค่าของสัญญาณขาออก (Tproc) ต้องมีค่าน้อยกว่าคาบของการสุ่ม ดังนี้

$$T_{proc} < T \tag{2.11.9}$$

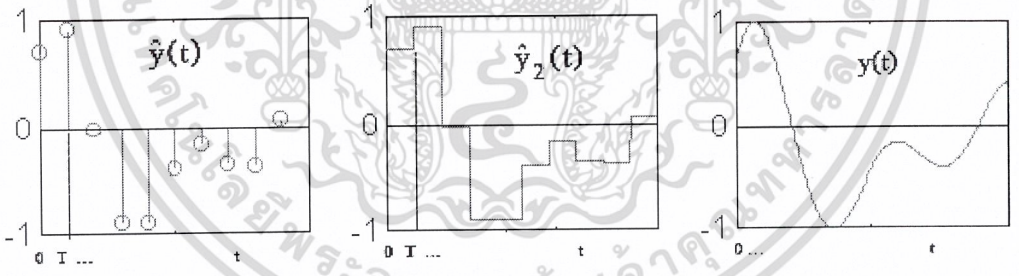
2.11.2 การสร้างสัญญาณคืน (Analog Reconstruction)

การสร้างสัญญาณคืนนั้นในทางทฤษฎีทำได้โดยผ่านสัญญาณแบบไม่ต่อเนื่อง เข้าไปยังตัวกรอง (แอนะลอก) แบบผ่านความถี่ต่ำที่มีความถี่ตัดที่ $f_s/2$ ตัวกรองนี้บางครั้งเรียกว่าตัวกรองสร้างสัญญาณคืน (reconstruction filter) ตัวกรองจะผ่านเฉพาะสัญญาณในช่วงความถี่ระหว่าง $-f_s/2$ ถึง $f_s/2$ หรือช่วงไนควิสท์ออกมา ผลที่ได้ก็คือเราจะได้สำเนาของสเปกตรัมที่อยู่รอบความถี่ 0 ออกมาเป็นสัญญาณขาออก ซึ่งมันก็คือสัญญาณแอนะลอกที่มีรูปร่างเป็นขอบของสัญญาณแบบไม่ต่อเนื่องก่อนสร้างกลับนั่นเอง ดังแสดงในรูปที่ 2.17



รูปที่ 2.17 สัญญาณขาเข้าและออกของตัวกรองสร้างสัญญาณคืน

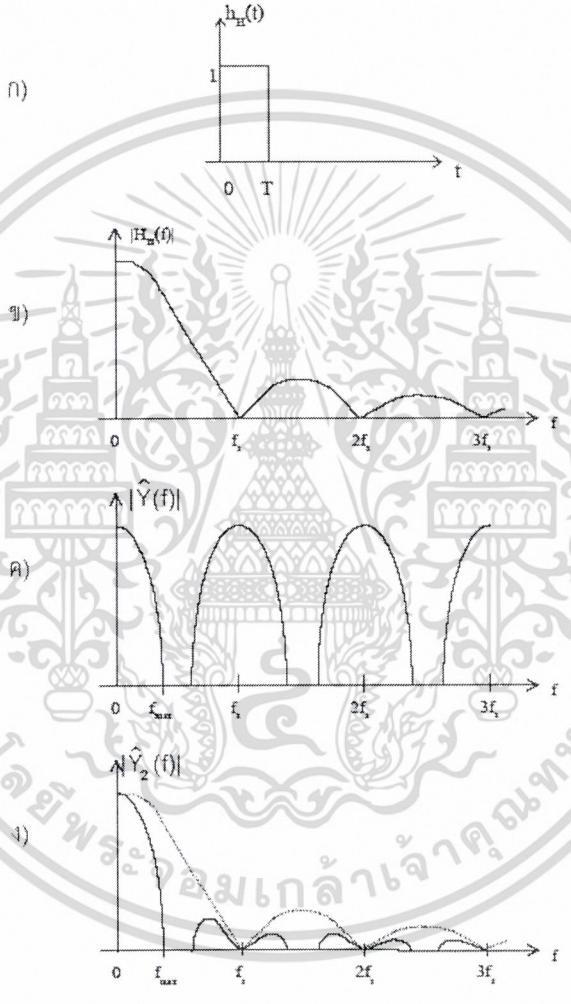
ในทางปฏิบัติเราไม่ใช้ $\hat{y}(t)$ เป็นสัญญาณขาเข้าของตัวกรองสร้างสัญญาณคืน เนื่องจากอย่างที่กล่าวมาแล้วว่าเรามองเห็นสัญญาณไม่ต่อเนื่องในลักษณะเป็นลำดับของข้อมูล และสัญญาณที่มองแบบแอนะลอกคือ $\hat{y}(t)$ เป็นสัญญาณอุดมคติที่ใช้พิสูจน์ในทางคณิตศาสตร์เท่านั้น ดังนั้นการแปลงสัญญาณดิจิตอลเป็นแอนะลอกในชีวิตจริงเราจะนำสัญญาณ $y(n)$ มาผ่านวงจรคงค่า (hold) ที่ทำงานเข้าจังหวะกับอัตราสุ่มของสัญญาณ $y(n)$ เพื่อสร้างเป็นสัญญาณลักษณะขั้นบันไดดังในรูปที่ 2.18 ก่อน จากนั้นจึงค่อยใช้สัญญาณขั้นบันไดนี้สร้างเป็นสัญญาณขาออก โดยส่งมันผ่านตัวกรองผ่านต่ำเพื่อสร้างสัญญาณคืนอีกทีหนึ่ง



รูปที่ 2.18 สัญญาณขาเข้าและขาออกของวงจรคงค่าสัญญาณ

บางคนอาจสงสัยว่าแล้วสัญญาณขั้นบันได $\hat{y}_2(t)$ ที่สร้างขึ้นมานี้ มีลักษณะของสเปกตรัมเหมือนหรือแตกต่างจากสเปกตรัมของสัญญาณ $\hat{y}(t)$ อย่างไร เราสามารถหาสเปกตรัมของสัญญาณ $\hat{y}_2(t)$ ได้ง่ายๆ โดยมองวงจรคงค่าสัญญาณเป็นระบบแอนะลอกแบบเชิงเส้นอันหนึ่ง ซึ่งมีผลตอบสนองสัญญาณอิมพัลส์คือ $h_H(t)$ ดังแสดงในรูป 2.19 ก นั่นคือวงจรนี้จะเปลี่ยนอิมพัลส์ของสัญญาณขาเข้าเป็นพัลส์สี่เหลี่ยมที่มีความกว้างเท่ากับคาบของอัตราการสุ่ม เมื่อทำการแปลงฟูริเยร์ผลตอบสนองอิมพัลส์นี้จะได้ผลตอบสนองเชิงความถี่ของระบบมีลักษณะเป็นฟังก์ชั่นซิงค์ ($sync$) ดังรูป 2.19 ข (สัญญาณพัลส์แปลงฟูริเยร์ได้สัญญาณซิงค์และสัญญาณซิงค์แปลงฟูริเยร์ได้สัญญาณพัลส์)

สเปกตรัมของสัญญาณขาออกของวงจรคงค่านี้ หาได้จากผลคูณของสเปกตรัมของสัญญาณขาเข้าและผลตอบสนองเชิงความถี่ของระบบ ซึ่งได้ผลลัพธ์ดังแสดงในรูปที่ 2.19ง สังเกตได้ว่าสเปกตรัมของสัญญาณขั้นบันไดก็ยังคงมีสำเนาของสัญญาณแอนะลอกที่ความถี่ $0, f_s, 2f_s, \ldots$ อยู่ครบถ้วน เพียงแต่สำเนาแต่ละตัวถูกกดขนาดลงไปด้วยการคูณของฟังก์ชั่นซิงค์ ซึ่งสำเนาเหล่านี้จะต้องถูกกำจัดทิ้งโดยตัวกรองสร้างสัญญาณคืนที่มีความถี่ตัดที่ $f_s/2$ เพื่อให้ได้สัญญาณแอนะลอกขาออกมีรูปร่างที่เรียบสมบูรณ์ สรุปว่าถึงแม้จะใช้วงจรคงค่าซึ่งให้ผลตอบสนองดูใกล้เคียงสัญญาณแอนะลอกที่ต้องการแล้ว เราก็ยังคงต้องใช้ตัวกรองสร้างสัญญาณคืนร่วมด้วย



รูปที่ 2.19 ก) ผลตอบสนองสัญญาณอิมพัลส์ของวงจรคงค่า, ข) ผลตอบสนองเชิงความถี่ของวงจรคงค่า, ค) สเปกตรัมของสัญญาณ $\hat{y}(t)$, ง) สเปกตรัมของสัญญาณ $\hat{y}_2(t)$

เช่นเดียวกันกับตัวกรองป้องกัน aliasing คือ เราไม่สามารถทำตัวกรองอุดมคติสำหรับกู้สัญญาณคืนได้ ซึ่งผลข้างเคียงก็คือจะทำให้สัญญาณที่อยู่ในช่วงความถี่สูงกว่า $f_s/2$ หลุดลอดออกมาที่สัญญาณขาออกด้วยกลายเป็นความผิดเพี้ยนของสัญญาณขาออกไป อย่างไรก็ตามถ้าหากใช้ความถี่ในการสุ่มสูงๆหรือ oversampling เพื่อให้มีระยะห่างระหว่างสำเนาความถี่แต่ละตัวมากขึ้น นอกจากจะช่วยแก้ปัญหาของการ

ใช้ตัวกรองป้องกัน aliasing ที่ไม่เป็นอุดมคติดังที่ได้กล่าวในหัวข้อก่อนหน้านี้แล้ว ก็ยังจะช่วยแก้ปัญหาของการใช้ตัวกรองสร้างสัญญาณคืนที่ไม่เป็นอุดมคติได้ในทำนองเดียวกันอีกด้วย

ขอสรุปเรื่องการสุ่มสัญญาณและการสร้างสัญญาณคืนด้วยทฤษฎีการสุ่มสัญญาณว่า "ถ้าเราสุ่มสัญญาณแอนะลอกด้วยความถี่ f_s ที่มากกว่า $2f_{\max}$ เราจะสามารถสร้างสัญญาณแอนะลอกคืนมาได้โดยสมบูรณ์" นั่นก็คือสัญญาณไม่ต่อเนื่องที่เกิดจากการสุ่มจะเป็นตัวแทนที่สมบูรณ์ของสัญญาณต้นฉบับโดยไม่มีการผิดเพี้ยนเลย อย่าลืมหมายเหตุไว้ด้วยความเข้าใจเองด้วยว่าในทางปฏิบัติ f_s ควรมากกว่า $2f_{\max}$ พอประมาณเพื่อชดเชยการที่เราหาตัวกรองอุดมคติไม่ได้

ในการทำการทดลองของเรา เราใช้ตัวกรองชนิด IIR ในการสังเคราะห์สัญญาณคืนกลับมาดังนั้นเราจะกล่าวถึง ทฤษฎีของตัวกรองสัญญาณชนิด IIR (Infinite impulse response)

2.12 ตัวกรองความถี่แบบ IIR (Infinite impulse response)

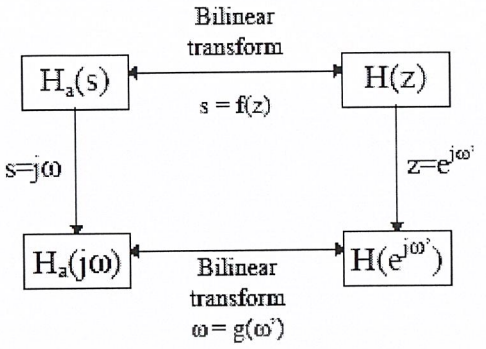
ตัวกรองแบบ IIR เป็นระบบที่มีโพลและมีความยาวของผลตอบสนองต่ออิมพัลส์ไม่จำกัดในการออกแบบตัวกรอง IIR เราจะไม่มุ่งเป้าที่การหา $h(n)$ เหมือนอย่างการออกแบบตัวกรอง FIR แต่เราจะมุ่งไปที่การหาฟังก์ชั่นถ่ายโอน $H(z)$ ของระบบ และเราจะได้เห็นอีกว่าการสร้างตัวกรอง IIR สามารถกระทำได้โดยตรงจากพารามิเตอร์ใน $H(z)$ ทันทีโดยไม่ต้องสนใจ $h(n)$ เลย

การออกแบบโดยอิงตัวกรองแอนะลอกต้นแบบ

วิธีการออกแบบตัวกรองดิจิตอลที่นิยมมากวิธีหนึ่งก็คือ การออกแบบโดยอิงตัวกรองแอนะลอกต้นแบบ ซึ่งได้แก่ตัวกรองแบบ Butterworth, Chebychev, Elliptic, Bessel, และอื่นๆ ตัวกรองแอนะลอกเหล่านี้เป็นพื้นฐานที่ถูกศึกษาและพัฒนามาถึงจุดที่ค่อนข้างสมบูรณ์แล้ว ถ้าเราสามารถหาฟังก์ชั่นหรือการแปลงอย่างใดอย่างหนึ่งที่สามารถแปลงฟังก์ชั่นถ่ายโอนของระบบแบบแอนะลอก มาเป็นระบบแบบไม่ต่อเนื่องได้ เราก็อาจจะสามารถนำตัวกรองในระบบแอนะลอกมาใช้ในระบบไม่ต่อเนื่องได้ทันที

การแปลงดังกล่าวไม่ยากอย่างที่คิด เนื่องจากหลักการของระบบต่อเนื่องและระบบไม่ต่อเนื่องมีลักษณะคล้ายคลึงกัน ดังที่เราได้เห็นการแปลง z ที่มีลักษณะการใช้งานเช่นเดียวกับการแปลงลาปลาซในระบบแบบต่อเนื่องมาแล้ว ในรูปที่ 2.20 แสดงความสัมพันธ์กันของระบบทั้งสอง จะเห็นได้ว่าจากฟังก์ชั่นถ่ายโอนของระบบต่อเนื่อง ($H_a(s)$ ซึ่งเป็นฟังก์ชั่นของ s) เราสามารถหาผลตอบสนองเชิงความถี่ได้โดยแทน $s = j\omega$ และสำหรับฟังก์ชั่นถ่ายโอนของระบบไม่ต่อเนื่อง ($H(z)$ ซึ่งเป็นฟังก์ชั่นของ z) ก็สามารถหาผลตอบสนองความถี่ได้โดยแทน $z = e^{j\omega'}$

สำหรับการแปลงระหว่างฟังก์ชั่นโอนย้ายของสองระบบคือ จาก $H_a(s)$ ไปเป็น $H(z)$ หรือจาก $H_a(\omega)$ ไปเป็น $H(e^{j\omega'})$ เราต้องการฟังก์ชั่นพิเศษในการแปลงจากโดเมน s เป็นโดเมน z ซึ่งการแปลงนี้เรียกว่า การแปลงไบลิเนียร์ (Bilinear Transform)



รูปที่2.20 ความสัมพันธ์กันของระบบแบบต่อเนื่องและไม่ต่อเนื่อง

เราจะลองศึกษาการแปลงไบลิเนียร์สำหรับแปลงระบบผ่านต่ำแอนะลอก มาเป็นระบบผ่านต่ำไม่ต่อเนื่องก่อน ซึ่งพบว่าฟังก์ชั่นสำหรับแปลงจากโดเมน s เป็นโดเมน z อยู่ในรูปดังนี้

$$s = K(\frac{z-1}{z+1}) \qquad (2.12.1)$$

โดยที่ K เป็นค่าคงที่สำหรับการแปลง ซึ่งผลที่เกิดจากการแปลงไบลิเนียร์สามารถมองได้เป็น 2 จุดใหญ่ๆ คือ

1. เกิดการแปลงโพลและศูนย์บน s-plane ของระบบต่อเนื่องไปเป็นโพลและศูนย์บน z-plane ของระบบไม่ต่อเนื่อง

ซึ่งจุดที่เราให้ความสนใจเป็นพิเศษคือ เกิดการดึงพื้นที่ในซีกซ้ายของ s-plane ไปยังพื้นที่ภายใต้วงกลมขนาด 1 หน่วยของ z-plane ถ้าระบบแบบแอนะลอกมีโพลอยู่ในซีกซ้ายของ s-plane เมื่อแปลงเป็นระบบดิจิตอลโพลนั้นก็จะอยู่ภายใต้วงกลมขนาด 1 หน่วยของ z-plane ดังในรูปที่ 2.21 นั่นหมายถึงว่าถ้าระบบแอนะลอกที่เป็นต้นแบบมีเสถียรภาพและคอซัล เมื่อแปลงเป็นระบบแบบไม่ต่อเนื่องก็จะได้ระบบที่มีเสถียรภาพและเป็นคอซัลด้วย

ถ้าส่วนจริงของ s มีค่าน้อยกว่าศูนย์ (โพลของระบบแอนะลอกอยู่ซีกซ้าย) เมื่อแปลงจะได้ขนาดของ z มีค่าน้อยกว่า 1 (โพลของระบบดิจิตอลอยู่ภายในวงกลมหนึ่งหน่วย) เริ่มต้นจากส่วนจริงของ s ซึ่งสามารถเขียนได้เป็นผลบวกของ s และ s^* ดังนี้

$$\mathrm{Re}\{s\} = \frac{(s+s^*)}{2} \qquad (2.12.2)$$

แทนค่า s ด้วยความสัมพันธ์ตามสมการที่ 2.12.1 จะได้

$$\mathrm{Re}\{s\} = \frac{K}{2}\left[\frac{z-1}{z+1} + \frac{z^{*}-1}{z^{*}+1}\right]$$

$$= \frac{K}{2}\left[\frac{(z-1)(z^{*}+1)+(z+1)(z^{*}-1)}{(z+1)(z^{*}+1)}\right]$$

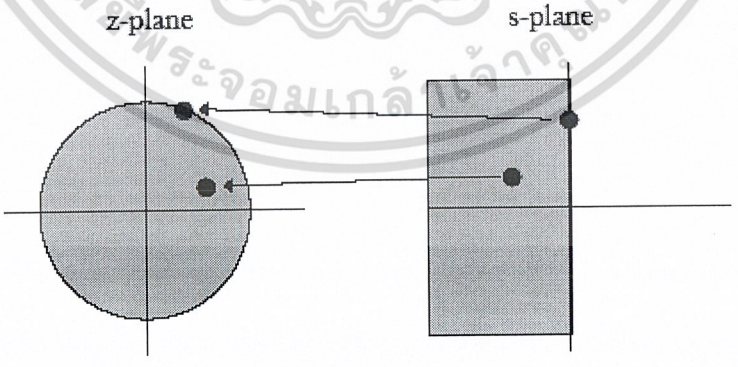
$$= \frac{K}{2}\left[\frac{2zz^{*}-2}{(z+1)(z^{*}+1)}\right]$$

$$= \frac{K(zz^{*}-1)}{(z+1)(z+1)^{*}}$$

ใช้ความสัมพันธ์ว่า zz^{*} มีค่าเท่ากับ $|z|^{2}$ เราจะสามารถเขียนสมการนี้เป็นฟังก์ชั่นของขนาดของ z ได้ดังนี้

$$\mathrm{Re}\{s\} = \frac{K(|z|^{2}-1)}{|z+1|^{2}} \qquad (2.12.3)$$

จากสมการนี้จะเห็นได้ว่า $\mathrm{Re}\{s\} = 0$ เกิดขึ้นเมื่อ $|z| = 1$ นั่นคือเส้นแบ่งระหว่างซีกซ้ายและขวาของ s-plane ถูกดึงมาที่เส้นวงกลมหนึ่งหน่วยของ z-plane ซึ่งเส้นนี้คือเส้นแบ่งเงื่อนไขความมีเสถียรภาพของระบบทั้งสอง และจะเห็นได้ว่าเมื่อ $\mathrm{Re}\{s\} = 0$ จะได้ $|z| < 1$ ตามที่ต้องการพิสูจน์



รูปที่ 2.21 การแปลงระหว่างโพลใน s-plane ไปยังโพลใน z-plane

2. เกิดการแปลงระหว่างความถี่แอนะลอกไปเป็นความถี่ดิจิตอล

ถ้าเราแทนค่า $s = j\omega$ และแทน $z = e^{j\omega'}$ ลงในสมการที่ 2.12.1 จะได้ฟังก์ชั่นที่เป็นการแปลงระหว่างความถี่แอนะลอกไปเป็นความถี่ดิจิตอลดังนี้

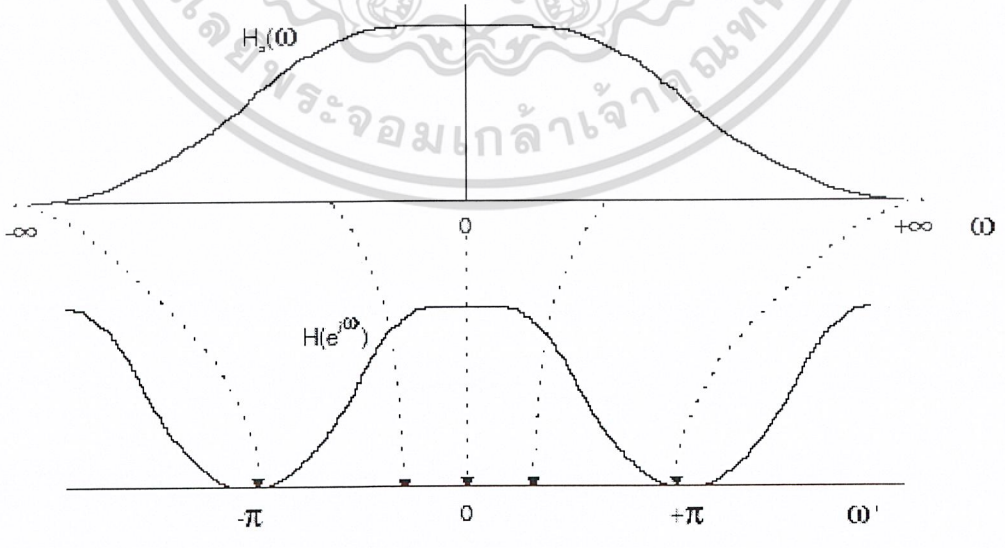
$$j\omega = K(\frac{e^{j\omega'} - 1}{e^{j\omega'} + 1})$$

$$= K[\frac{e^{j\omega'/2}(e^{j\omega'/2} - e^{-j\omega'/2})}{e^{j\omega'/2}(e^{j\omega'/2} + e^{-j\omega'/2})}]$$

$$= K(\frac{j\sin(\omega')/2}{\cos(\omega')/2})$$

$$\omega = K\tan(\frac{\omega'}{2}) \qquad (2.12.4)$$

เมื่อพิจารณาสมการนี้จะได้ว่าที่ $\omega = \infty$ จะได้ $\omega' = \pi$ และที่ $\omega = 0$ จะได้ $\omega' = 0$ นั่นคือความถี่ทั้งหมดของแอนะลอกในช่วง 0 ถึง ∞ ถูกดึงเข้ามาอยู่ในช่วง 0 ถึง π ของความถี่ดิจิตอล ซึ่งก็คือช่วงทำงานของความถี่ดิจิตอลนั่นเอง ผลของการแปลงความถี่โดยรวมเกิดขึ้นดังในรูปที่ 2.22 ซึ่งเห็นได้ชัดว่าเป็นการแปลงจากตัวกรองผ่านต่ำแอนะลอก เป็นตัวกรองผ่านต่ำดิจิตอล ถ้าลองสมมติให้ $K = 1$ และวาดกราฟระหว่าง ω และ ω' จะได้ดังรูปที่ 2.23

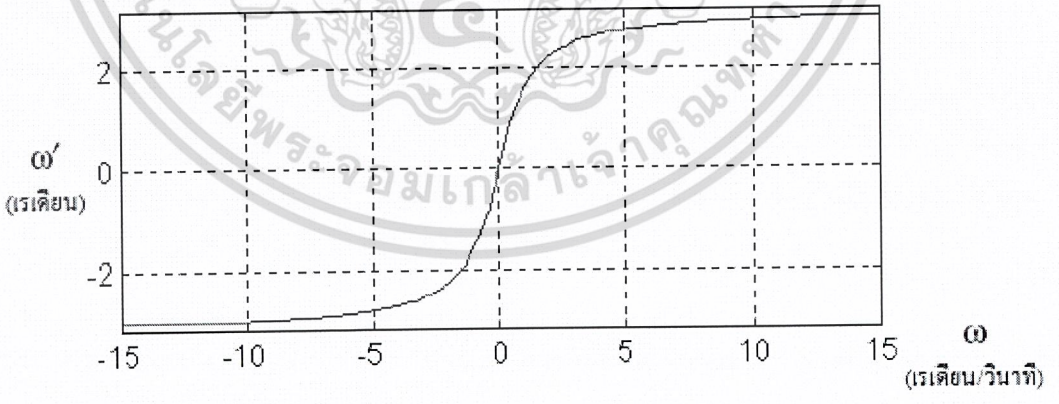


รูปที่ 2.22 การแปลงระหว่างความถี่แอนะลอกและดิจิตอลกับผลที่เกิดขึ้นกับผลตอบสนองเชิงความถี่ในการแปลงไบลิเนียร์

จากรูปที่ 2.23 จะสังเกตได้ว่า การแปลงความถี่ที่เกิดขึ้นไม่ได้มีลักษณะเป็นเชิงเส้น ซึ่งฟังก์ชันเชิงเส้นไม่สามารถนำมาใช้ในการแปลงความถี่ได้ เพราะความถี่แอนะลอกมีช่วงตั้งแต่ 0 ถึงอนันต์ แต่ช่วงทำงานของความถี่ดิจิตอลมีช่วงแค่ 0 ถึง π เท่านั้น ดังนั้น ถ้าใช้ฟังก์ชันเชิงเส้นก็คงจะได้ช่วงแถบผ่านลดเหลือเล็กนิดเดียว

การใช้การแปลงไบลิเนียร์ ทำให้ฟังก์ชันการแปลงความถี่มีลักษณะคล้ายเป็นเชิงเส้นแบบสองช่วง คือในช่วงความถี่ต้นๆ (ω ประมาณ 0 ถึง 2 เรเดียน/วินาที) กราฟจะมีความชันมาก และช่วงความถี่หลังๆ กราฟจะมีความชันน้อยลงมาก ซึ่งโดยปกติเราสามารถเลือกค่า K ให้กราฟมีจุดแบ่งของสองช่วงนี้ที่ประมาณ ณ ความถี่ตัดของตัวกรอง ซึ่งก็จะทำให้ในช่วงแถบผ่านมีการแปลงความถี่อย่างช้าๆ และในช่วงแถบหยุดมีการแปลงความถี่อย่างเร็ว ซึ่งก็จะดึงให้ความถี่แอนะลอกทั้งหมด (ยาวจนถึงอนันต์) ในแถบหยุด ถูกดึงมาจำกัดอยู่ในความถี่ π ของตัวกรองดิจิตอลได้ ปรากฏการณ์ที่ความถี่ถูกดึงหดเข้ามานี้เรียกว่า frequency warping

ผลของ frequency warping ทำให้รูปร่างของผลตอบสนองเชิงความถี่ของตัวกรองดิจิตอลที่ได้มีความแตกต่างจากตัวกรองแอนะลอกต้นแบบบ้าง แต่ก็ไม่เป็นผลสำคัญอะไร เพราะลักษณะสำคัญของตัวกรองแอนะลอกต้นแบบได้ถูกถ่ายทอดมายังตัวกรองดิจิตอลแล้ว เช่น ลักษณะของความคมและความพลิ้วของตัวกรอง และที่สำคัญคือ ได้ความถี่ตัดของตัวกรองดิจิตอลตามที่ต้องการด้วยเนื่องจากฟังก์ชั่น tan มีลักษณะเป็นคาบ กราฟที่แสดงในรูปที่ 2.23 นี้แสดงเฉพาะช่วงความถี่ ω' เท่ากับ $-\pi$ ถึง π เท่านั้น ในช่วงอื่น ๆ ก็จะมีลักษณะเป็นคาบทุก ๆ 2π เช่นเดียวกับกราฟนี้ ซึ่งก็จะมีผลทำให้ได้ผลตอบสนองเชิงความถี่ของระบบดิจิตอล มีลักษณะเป็นคาบทุก ๆ 2π



รูปที่ 2.23 กราฟความสัมพันธ์ระหว่างความถี่อนาลอกและดิจิตอลในการทำการแปลงไบลิเนียร์

สรุปความว่า ถ้ามีฟังก์ชั่นถ่ายโอน $H_a(s)$ ของตัวกรองแอนะลอกผ่านต่ำ เราจะสามารถแปลงฟังก์ชั่นนี้เป็นฟังก์ชั่นถ่ายโอน $H(z)$ ของตัวกรองดิจิตอลได้ทันที โดยใช้การแปลงไบลิเนียร์ คือแทนค่า s ใน $H_a(s)$ ตามสมการที่ 2.12.1 คือ $s = K(\frac{z-1}{z+1})$ หรือ เขียนได้ว่า

$$H(z) = [H_a(s)]_{s=K(\frac{z-1}{z+1})} \tag{2.12.5}$$

ซึ่งผลที่เกิดขึ้น คือ เราได้ตัวกรองดิจิตอลแบบ IIR ซึ่งทำหน้าที่เป็นตัวกรองผ่านต่ำในลักษณะเดียวกับตัวกรองแอนะลอกที่เป็นต้นแบบ และมีเสถียรภาพเหมือนตัวกรองต้นแบบเช่นกัน

การออกแบบโดยวิธีวางโพล และศูนย์

การออกแบบโดยวิธีวางโพลและศูนย์ เป็นวิธีอย่างง่ายๆ โดยอาศัยความเข้าใจในเรื่องผลของโพลและศูนย์ที่มีต่อผลตอบสนองเชิงความถี่ของระบบ สามารถนำไปใช้ออกแบบตัวกรองที่มีอันดับต่ำๆ และตัวกรองประเภทผ่านความถี่เดียว (peaking filter) หรือตัดความถี่เดียว (notching filter) ได้ก่อนอื่นต้องมีความเข้าใจเรื่องผลของโพล และศูนย์ที่มีต่อผลตอบสนองเชิงความถี่ก่อน โดยลองพิจารณาระบบที่มีโพลอยู่ที่ p_1 และมีศูนย์อยู่ที่ q_1 ดังนี้

$$H(z) = A(\frac{z-q_1}{z-p_1}), \text{ โดยที่ } |p_1|, |q_1| \le 1 \tag{2.12.6}$$

ถ้าพิจารณาผลตอบสนองความถี่ทางขนาดจะได้ว่า $|H(z)|$ ประกอบด้วยเทอม $|z-q_i|$ เป็นตัวคูณและเทอม $|z-p_i|$ เป็นตัวหาร ซึ่งเป็นจริงสำหรับจำนวนโพล และศูนย์ใดๆ ในที่นี้มีโพล และศูนย์อย่างละหนึ่งตัว ซึ่งจะได้

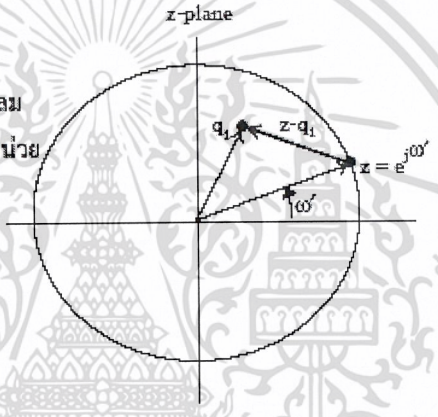
$$|H(z)| = |A| \frac{|z-q_1|}{|z-p_1|} \tag{2.12.7}$$

ถ้าเขียน p_1 , q_1 ให้อยู่ในรูปเชิงขั้ว ดังนี้ $p_1 = |p_1|e^{j\theta_p}, q_1 = |q_1|e^{j\theta_q}$ จะได้ว่า

1. ลองพิจารณาจากเวคเตอร์ $z-q_1$ ซึ่งเป็นเวคเตอร์ที่วาดสีเข้มในรูปที่ 2.4 เมื่อความถี่เปลี่ยนไป z ก็จะหมุนไปรอบๆ วงกลมหนึ่งหน่วย และ $z-q_1$ ก็จะเปลี่ยนไปด้วย และจะเห็นได้ชัดว่าขนาดของเวคเตอร์ $z-q_1$ จะมีค่าน้อยที่สุดเมื่อ z หมุนมาทับกับ q_1 พอดี หรือคือจุดที่ความถี่ ω' = มุมของ q_1 สรุปได้ว่าที่ $\omega' =$ มุมของศูนย์ $= \theta_q$ หรือ $z = e^{j\theta_q}$ จะได้ $|z-q_1|$ มีค่าน้อยที่สุดซึ่งเทอมนี้เป็นตัวคูณอยู่ในผลตอบสนองเชิงความถี่โดยรวม ดังนั้น ถ้าพิจารณาผลของศูนย์ตัวเดียว (ในกรณีที่ไม่มีผลของโพลและศูนย์อื่นรบกวน) จะพบว่า ขนาดของผลตอบสนองเชิงความถี่จะถูกดึงให้ต่ำสุดที่ความถี่ตรงกับมุม

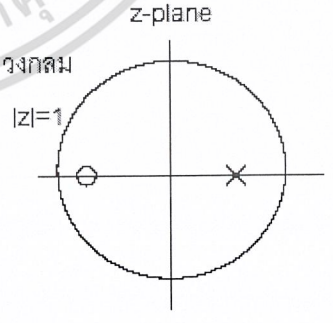
ของศูนย์นั้นๆ ยิ่งขนาดของศูนย์ $|q_1|$ มีค่าใหญ่ (ใกล้ 1) มากขึ้นเท่าใด ผลของศูนย์ก็จะแรงมากขึ้นเท่านั้น โดยผลแรงที่สุดเกิดขึ้นเมื่อ ถ้า $|q_1| = 1$ ซึ่งจะได้ $|H| = 0$ ที่ความถี่ θ_q นี้

2. ที่ ω' = มุมของโพล = θ_p หรือ $z = e^{j\theta p}$ จะได้ $|z - p_1|$ มีค่าเป็นจุดต่ำสุด แต่เนื่องจากเทอมนี้เป็นตัวหารอยู่ในผลตอบสนองเชิงความถี่ ดังนั้น มันจะส่งผลให้ $|H|$ มีค่าเป็นจุดสูงสุด (ในกรณีที่ไม่มีผลของโพล และศูนย์อื่นรบกวน) และในทำนองเดียวกัน คือ ถ้าขนาดของโพล $|p_1|$ มีค่าใหญ่ (ใกล้ 1)มากขึ้น ก็จะทำให้ผลของโพลแรงขึ้นที่ความถี่นั้น ถ้า $|p_1|$ =1 จะได้ $|H| = \infty$ ซึ่งเป็นสภาวะที่เกือบเสถียร ในระบบปกติเราจะไม่ใช้ขนาดของโพลเท่ากับ 1 ยกเว้น ระบบที่ต้องการออกแบบเป็นตัวกำเนิดสัญญาณไซน์



รูปที่ 2.24 เวคเตอร์ $z - q_1$ ใน **z-plane**

ตัวอย่างของการพิจารณาผลของโพล และศูนย์ เช่น



$$H(z) = A(\frac{z + 0.8}{z - 0.6})$$

มีโพล $= 0.6(\omega' = 0)$

มีศูนย์ $= -0.8 = 0.8e^{j\pi}$ $(\omega' = \pi)$

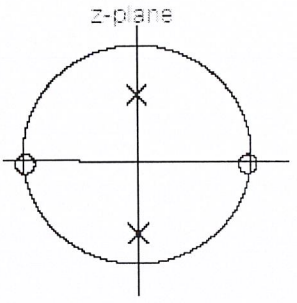
เมื่อพิจารณาในช่วงความถี่ 0 ถึง π ระบบนี้ควรมี

ขนาดของผลตอบสนองความถี่สูงสุดที่ความถี่ 0

และต่ำสุดที่ความถี่ π ดังนั้น ระบบนี้เป็นวงจรผ่านต่ำแผนภาพที่แสดงในด้านขวานี้ เรียกว่า

แผนภาพโพล-ศูนย์ ซึ่ง o แทนตำแหน่งของศูนย์ และ X แทนตำแหน่งของโพล

$$H(z) = A\frac{(z - 0.1)(z + 1)}{(z - re^{j\pi/2})(z - re^{-j\pi/2})}$$



มีโพลที่ $\omega' = \pi/2$ และ $-\pi/2$,
มีศูนย์ที่ $\omega' = 0$ และ π ดังนั้น ควรเป็นวงจรผ่านแบนด์ ที่มีความถี่ศูนย์กลางของแบนด์อยู่ที่ $\omega' = \pi/2$

การออกแบบโดยวางโพล และศูนย์ยังสามารถใช้ออกแบบ ตัวกรองประเภทกรองความถี่เดี่ยว โดยการวางโพล และศูนย์ที่มีขนาดใกล้เคียงกันไว้ความถี่เดียวกัน ถ้าให้ขนาดของโพลมีขนาดใหญ่กว่าศูนย์จะได้เป็นวงจรขยายความถี่เดียว เนื่องจากผลของโพล และศูนย์จะชดเชยกันที่ความถี่อื่นๆแต่ที่ความถี่ที่ตรงกับมุมของมัน โพลจะมีผลแรงกว่าศูนย์จึงทำให้เกิดยอดขึ้นมาที่ความถี่นั้น ในทางตรงกันข้าม ถ้าขนาดโพลมีขนาดเล็กกว่าศูนย์ ก็จะได้เป็นวงจรตัดความถี่เดี่ยวตัวกรองขยาย หรือตัดความถี่เดี่ยวจึงมีโพล และศูนย์อย่างละหนึ่งคู่ และได้เป็นฟังก์ชั่นถ่ายโอนอันดับสอง ดังนี้

$$H(z) = K\frac{(z - r_z e^{j\omega'_0})(z - r_z e^{-j\omega'_0})}{(z - r_p e^{j\omega'_0})(z - r_p e^{-j\omega'_0})} \qquad (2.12.8)$$

ฟังก์ชั่นถ่ายโอนนี้สามารถคำนวณโดยใช้ Matlab ตามรูปที่ 2.25 ซึ่งจะรับค่าความถี่ที่นอร์แมลไลซ์ด้วย $f_s/2$ เหมือนฟังก์ชั่นที่คำนวณสัมประสิทธิ์อื่นๆ ใน DSP Toolbox และรับค่า r_z, r_p, และ K ตามสมการที่ 2.12.8 จากนั้นคำนวณหาค่าสัมประสิทธิ์ของโพลิโนเมียลเศษและส่วน โดยใช้ฟังก์ชั่น zp2tf ซึ่งเป็นฟังก์ชั่นภายใน DSP Toolbox

```
function [a,b] = singfreq(fn,rz,rp,k)
w=fn*pi;
z=[rz*exp(j*w); rz*exp(-j*w)];
p=[rp*exp(j*w); rp*exp(-j*w)];
[a,b]=zp2tf(z,p,k);
```

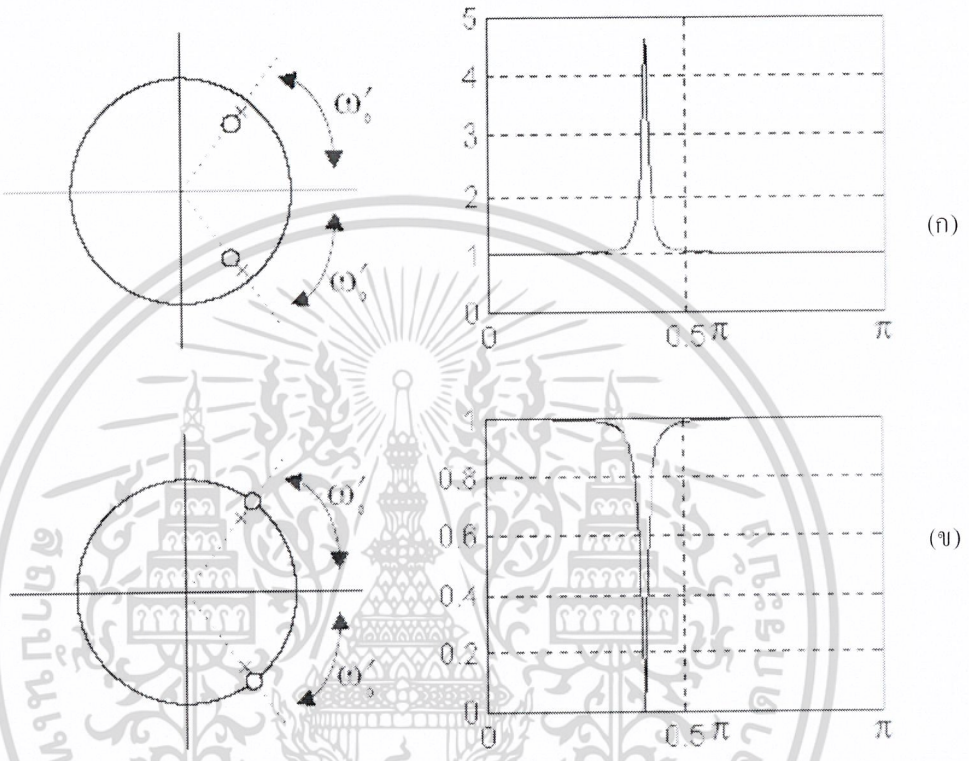
รูปที่ 2.25 ฟังก์ชัน singfreq.m สำหรับคำนวณฟังก์ชันถ่ายโอนของตัวกรองความถี่เดียว

ตัวอย่างผลตอบสนองเชิงความถี่ที่ได้จากตัวกรองแบบนี้แสดงอยู่ในรูปที่ 2.26 โดยทั้งสองรูปใช้ $\omega'_0 = 0.4\pi$

รูปที่ 2.26 ก) ใช้ $r_z = 0.91, r_p = 0.98, K = 1.07$

รูปที่ 2.26 ข) ใช้ $r_z = 1.0, r_p = 0.95, K = 0.95$

ตัวกรองความถี่เดี่ยวนี้ยังสามารถประยุกต์ทำเป็นตัวกรองที่ขยาย หรือตัดหลายๆค่าความถี่ได้ โดยการนำเอาฟังก์ชั่นถ่ายโอนอันดับสองที่ตัดความถี่เดี่ยวมาอนุกรมกันเป็นอันดับที่สูงขึ้น ดังตัวอย่างของตัวกรองที่ได้ เช่น ตัวกรองตัด หรือขยายสัญญาณฮาร์มอนิกส์ ซึ่งสัญญาณฮาร์มอนิกส์ประกอบด้วยความถี่เดี่ยวหลายๆ ความถี่ $(f_0, 2f_0, 3f_0, 4f_0, ...)$ รวมกันอยู่



รูปที่ 2.26 แผนภาพโพลและศูนย์และผลตอบสนองเชิงความถี่ของวงจรกรองความถี่เดี่ยว

การสร้างตัวกรอง IIR (IIR Filter Realization)

ตัวกรองแบบ IIR สามารถนำไปสร้างใช้งานได้โดยมองจาก $H(z)$ ได้โดยตรง เราจะดูวิธีเขียนแผนภาพการสร้างตัวกรองจาก $H(z)$ โดยเริ่มจากการจัด $H(z)$ ให้อยู่ในรูปดังต่อไปนี้ (เพื่อให้ง่ายต่อความเข้าใจ ขอแสดงโดยการใช้อันดับ = 2 โดยอันดับที่สูงขึ้นสามารถทำได้โดยง่ายเมื่อเข้าใจวิธีทำแล้ว)

$$H(z) = \frac{a_0 z^2 + a_1 z + a_2}{z^2 + b_1 z + b_2} \qquad \text{หรือ} \quad H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \qquad (2.12.9)$$

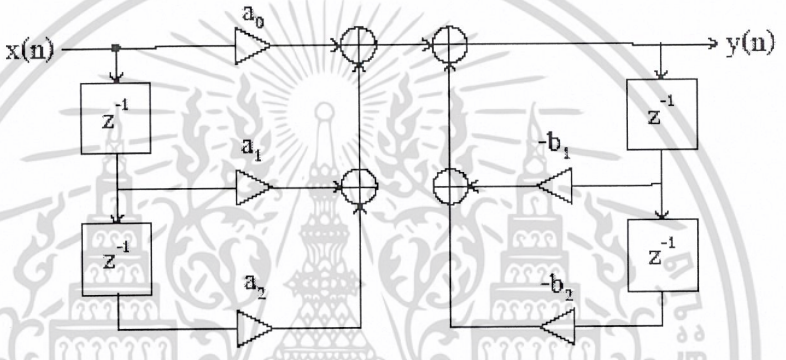
แผนภาพรูปแบบแรก คือ แบบ direct form 1 ดังแสดงในรูปที่ 2.27 รูปแบบนี้เกิดโดยตรงจากสมการผลต่างของระบบ ถ้าเราแทน $H(z) = Y(z)/X(z)$ และแก้สมการเพื่อหาสมการผลต่างของระบบ จะได้ดังนี้

$$Y(z)\{1+b_1z^{-1}+b_2z^{-2}\} = X(z)\{a_0+a_1z^{-1}+a_2z^{-2}\}$$

$$Y(z) = X(z)\{a_0+a_1z^{-1}+a_2z^{-2}\} - Y(z)\{b_1z^{-1}+b_2z^{-2}\}$$

$$y(n) = a_0x(n)+a_1x(n-1)+a_2x(n-2)-b_1y(n-1)+b_2y(n-2) \qquad (2.12.10)$$

ซึ่งจากสมการ 2.12.10 พบว่า $y(n)$ เกิดจากส่วนที่มาจากสัญญาณขาเข้าที่เวลาปัจจุบันและอดีต คือ $a_0x(n)+a_1x(n-1)+a_2x(n-2)$ แล้วทำการลบด้วยส่วนที่มาจากสัญญาณขาออกในอดีต คือ $b_1y(n-1)+b_2y(n-2)$ ซึ่งก็ตรงกับที่แสดงในแผนภาพ



รูปที่ 2.27 แผนภาพการสร้างตัวกรอง IIR (direct form 1)

อีกรูปแบบหนึ่งของการสร้างตัวกรอง IIR คือ แบบ direct form 2 หรือ canonical form ดังแสดงในรูปที่ 2.28 การพิสูจน์ที่มาของแผนภาพนี้ทำได้โดย สมมติสัญญาณขึ้นใหม่สัญญาณหนึ่งเรียกว่า $w(n)$ ซึ่งเกิดการเอา $x(n)$ ผ่านตัวส่วนของฟังก์ชั่นถ่ายโอน หรือเขียนเป็นรูปแบบฟังก์ชั่นโอนย้ายได้ว่า

$$\frac{W(z)}{X(z)} = \frac{1}{1+b_1z^{-1}+b_2z^{-2}} \qquad (2.12.11)$$

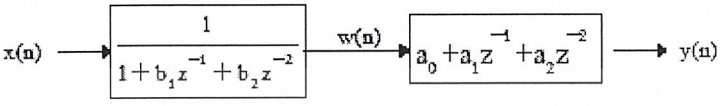
ซึ่งเมื่อทำการแปลง z ผกผันก็จะได้ว่า $w(n) = x(n)-b_1w(n-1)-b_2w(n-2)$ ซึ่งเมื่อแสดง $w(n)$ ในแผนภาพ และมองดูเฉพาะส่วนครึ่งซ้ายมือของแผนภาพ ซึ่งเหมือนเป็นส่วนที่สร้างสัญญาณ $w(n)$ ขึ้น ก็จะพบว่าตรงกับสมการที่เราเขียนขึ้น

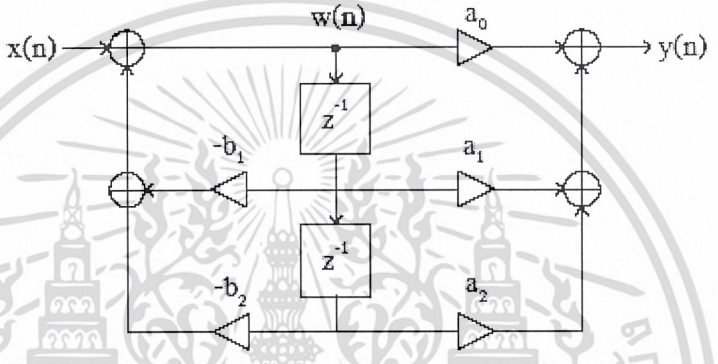
จากนั้น $y(n)$ จะหาได้จากการเอาสัญญาณ $w(n)$ ผ่านตัวเศษของฟังก์ชั่นถ่ายโอน ซึ่งเขียนในรูปแบบของฟังก์ชั่นถ่ายโอนได้ว่า

$$\frac{Y(z)}{W(z)} = a_0+a_1z^{-1}+a_2z^{-2} \qquad (2.12.12)$$

ซึ่งเมื่อทำการแปลง z ผกผันแล้วก็จะได้ว่า $y(n) = a_0 w(n) + a_1 w(n-1) + a_2 w(n-2)$ ซึ่งสมการนี้ก็คือส่วนขวามือของแผนภาพนั่นเอง ฉะนั้น โดยรวมแล้วก็จะได้

$$\frac{Y(z)}{X(z)} = \frac{W(z)}{X(z)}\frac{Y(z)}{W(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}$$ เป็นฟังก์ชั่นโดยรวมตามที่ต้องการ





รูปที่ 2.28 แผนภาพการสร้าง IIR (direct form 2)

ในกรณีที่ตัวกรองมีอันดับสูงๆ การสร้างตัวกรองโดยใช้ direct form 1 หรือ 2 ก็สามารถทำได้ แต่อาจส่งผลกระทบเรื่องความคลาดเคลื่อนของระบบ และเสถียรภาพ โดยทั่วไปมักนิยมแตกให้ $H(z)$ อยู่ในรูปของฟังก์ชั่นที่อันดับต่ำๆ แล้วใช้รูปแบบการสร้างตัวกรองแบบอนุกรม (cascade form) หรือขนาน (parallel form) เข้ามาช่วย

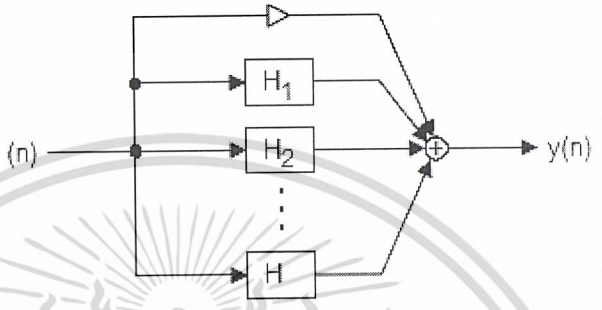
รูปแบบอนุกรมแสดงอยู่ในรูปที่ 2.29 ทำได้โดยกระจาย $H(z)$ ให้อยู่ในรูปของผลคูณของ $H_i(z)$ ก่อน ซึ่งนิยมใช้ $H_i(z)$ เป็นแบบ อันดับ 2 ยกเว้นถ้าอันดับรวมเป็นเลขคี่ก็จะมี $H_i(z)$ ตัวหนึ่งที่เป็นอันดับ 1 ส่วนที่เป็น $H_i(z)$ ในแผนภาพก็สามารถใช้ direct form 1 หรือ 2 ในการสร้างได้

$$H(z) = cH_1(z)H_2(z)....H_m(z) \qquad (2.12.13)$$

รูปที่ 2.29 แผนภาพการสร้าง IIR โดยใช้โครงสร้างแบบอนุกรม

เช่นเดียวกัน รูปแบบขนานแสดงอยู่ในรูปที่ 2.30 ทำได้โดยกระจาย $H(z)$ ให้อยู่ในรูปของผลบวกของ $H_i(z)$ ก่อน ซึ่งนิยมใช้ $H_i(z)$ เป็นแบบอันดับ 2 ยกเว้นถ้าอันดับรวมเป็นเลขคี่ก็จะมี $H_i(z)$ ตัวหนึ่งที่มีอันดับ 1

$$H(z) = c + H_1(z) + H_2(z) + ... + H_m(z) \qquad (2.12.14)$$



รูปที่ 2.30 แผนภาพการสร้างตัวกรอง IIR โดยใช้โครงสร้างแบบขนาน

เปรียบเทียบตัวกรอง FIR กับ IIR

การเลือกใช้ตัวกรอง FIR หรือ IIR สำหรับงานหนึ่งๆ เราต้องพิจารณาถึงข้อดี และข้อเสียของตัวกรองทั้งสองแบบ ซึ่งสามารถกล่าวโดยสรุปได้ดังนี้

ข้อดีของตัวกรอง FIR เมื่อเทียบกับตัวกรอง IIR

1. ให้ผลตอบสนองเชิงความถี่ที่มีเฟสแบบเชิงเส้นโดยสมบูรณ์ ในตลอดช่วงแถบผ่านซึ่งข้อนี้ไม่สามารถทำได้โดยตัวกรอง IIR หรือแม้แต่ตัวกรองแอนะลอกใด ๆ (ทำได้แต่ไม่เชิงเส้นสมบูรณ์)

2. ตัวกรอง FIR ไม่มีการป้อนกลับ หรือไม่มีโพล ทำให้มีเสถียรภาพเสมอ

3. ตัวกรอง FIR มีความทนทานที่ดีกว่าในด้านความคลาดเคลื่อนจากการประมาณค่าสัมประสิทธิ์และความคลาดเคลื่อนจากการคำนวณ ความคลาดเคลื่อนในที่นี้เกิดมาจากการใช้เลขฐานสองที่มีความยาวจำกัด (finite word length) ในการแทนค่าเลขจริงๆ ความคลาดเคลื่อนนี้ส่งผลถึงลักษณะต่างๆ ของระบบจากที่ออกแบบไว้ เช่น ความถี่ตัด, รูปร่างของผลตอบสนองความถี่, และรวมถึงเสถียรภาพของระบบด้วย

ข้อดีของตัวกรอง IIR เมื่อเทียบกับตัวกรอง FIR

1. ให้ผลตอบสนองเชิงความถี่ที่ดีกว่ามากในด้านความคม (ขนาดของแถบเปลี่ยนเล็กกว่า และความพลิ้วต่ำกว่า) ที่ขนาดอันดับเท่าๆ กับตัวกรอง FIR นั่นหมายถึงว่า โดยทั่วไปเราไม่จำเป็นต้องใช้ตัวกรอง IIR ที่มีอันดับสูงๆเหมือน FIR ดังนั้นตัวกรอง IIR จึงมีความต้องการด้านความเร็วของตัวประมวลผลที่น้อยกว่า

2. สามารถออกแบบโดยอิงจากตัวกรองต้นแบบแอนะลอกได้ ถ้ามีตัวกรองแบบแอนะลอกที่เคยใช้งานอยู่แล้ว

2.13 ทฤษฎี Chip DSP TMS320C6X

Chip DSP ในตระกูล Fixed-Point ของบริษัท Texas Instrument มีการพัฒนามาหลายรุ่นซึ่งทั้งหมดจะมีชุดคำสั่งซึ่งสามารถใช้ร่วมกันได้ โดยที่คำสั่งของรุ่นเก่าจะยังคงใช้ได้กับ Chip ตัวใหม่ซึ่งสามารถทำให้ผู้ใช้ศึกษาเพิ่มเติมไม่มาก ก็จะทำให้สามารถใช้ Chip รุ่นใหม่ๆได้ไม่ยาก และหนึ่งในนั้น คือ Chip ในตระกูล C6X จะมีหลายรุ่นซึ่งจะแตกต่างกันในส่วนของ Memory ที่บรรจุมาภายในเพื่อให้ผู้ใช้เลือกนำไปใช้งานในลักษณะต่างๆกันแต่ในส่วนหลักๆของการทำงานภายในจะเหมือนกัน ในโครงงานนี้จะนำ Chip รุ่น C6211 มาใช้งานแต่ในส่วนของทฤษฎีในบทนี้จะเรียกรวมๆว่า C6X

Chip ในรุ่น TMS320C62X นั้นถูกสร้างขึ้นในปี 1997 ซึ่ง Chip ตัวแรกของรุ่น C6X ก็คือ TMS320C6201 โดยในรุ่น C62X นี้มีสถาปัตยกรรมอยู่บนพื้นฐานแบบ very-long-instruction-word (VLIW) (ซึ่งต่างจากรุ่นก่อนหน้านี้ทั้ง C1X, C2X, และ C5X) และยังคงมีหน่วยความจำที่หลากหลายสำหรับชุดคำสั่งและข้อมูล ตามสถาปัตยกรรมแบบ Harvard สถาปัตยกรรมแบบ VLIW นั้นแม้จะมีชุดคำสั่งที่ง่ายแต่ส่วนใหญ่ก็จำเป็นสำหรับการใช้งานมากกว่าสถาปัตยกรรมแบบ DSP

Chip ในรุ่น C62X นั้นจะมีรหัสที่ไม่สอดคล้องกับรุ่นก่อนหน้านี้ และต่อมาก็มีการสร้าง Chip TMS320C6701 ในตระกูล floating point (C67X) ซึ่งถือเป็นหนึ่งในรุ่น C6X โดยที่ชุดคำสั่งของรุ่น C62X นั้นเป็นชุดคำสั่งย่อยของรุ่น C67X ส่วนรุ่นล่าสุดของ C6X นั้นก็คือ รุ่น C64X ซึ่งเป็นแบบ fixed-point Processor ในตระกูล Fixed-point นั้นมีผลดีต่ออุปกรณ์ที่ใช้แบตเตอรี่ เพราะว่ามันใช้พลังงานน้อยกว่าตระกูล equivalent floating-point

ในรุ่น C1X, C2X, และ C5X เป็น processors แบบ 16-bit และมี dynamic range และความแม่นยำที่จำกัด ส่วนรุ่น C6X นั้นเป็น processor แบบ 32-bit และมีการพัฒนาทั้ง dynamic range และในเรื่องของความแม่นยำด้วย

2.13.1 สถาปัตยกรรมของ TMS320C6X

สถาปัตยกรรมของ TMS320C6X เป็นแบบ VLIW หน่วยความจำภายในประกอบด้วย cache 2 level คือ 4kB of level 1 program cache (L1P), 4kB of level 1 data cache (L1D) และ 64 kB of RAM or level 2 cache สำหรับข้อมูล/โปรแกรม (L2) มันมีการ interface โดยตรงไปยัง synchronous memories ทั้ง 2 (SDRAM และ SBSRAM) และ Asyn memories (SRAM และ EPROM) Synchronous memory นั้นต้องการสัญญาณ clock และต้องมีการเตรียมการระหว่าง static SRAM และ dynamic SDRAM เพราะว่า SRAM จะเร็วกว่าแต่ก็แพงกว่า DRAM

On-chip peripherals จะประกอบไปด้วย 2 multichannel buffered serial ports (McBSPs), 2 timers, 16-bit host port interface (HPI) และ 32-bit external memory interface (EMIF) ซึ่งต้องใช้ไฟ 3.3 V สำหรับ I/O และ 1.8 V สำหรับ core (internal) บัสภายในประกอบด้วย 32-bit program address

bus,256-bit program data bus สำหรับ 8 ชุดคำสั่งแบบ 32-bit,2 data address buses แบบ 32-bit,2 data buses แบบ 64-bit และ 2 store data buses แบบ 64-bit และจาก 32-bit address bus นี้จึงมีพื้นที่รวมทั้งหมดของหน่วยความจำเป็น 2^{32}= 4GB ซึ่งรวมถึง external memory ทั้ง 4 ตัวคือ CE0, CE1, CE2, และ CE3 โดยรูปที่ 2.31 แสดงถึง function block diagram ของ TMS320C6X

Independent memory banks บน C6X นั้น ไว้สำหรับให้หน่วยความจำ 2 ตัว access ภายใน 1 รอบชุดคำสั่ง 2 independent memory banks นั้นสามารถ access ได้โดยใช้ 2 independent buses และเนื่องจากหน่วยความจำภายในจะถูกจัดการอยู่ภายใน memory banks 2 loads หรือ 2 stores instructions จึงสามารถแสดงในรูปแบบ parallel ได้ และถ้า data accessed อยู่ใน memory banks ที่ต่อกันก็ไม่มีผลแต่อย่างใด bus ต่างๆที่ไว้สำหรับโปรแกรมข้อมูลและ direct memory access (DMA) นั้นจะอนุญาตให้ C6X แสดงโปรแกรม fetches, data read and write และระบบปฏิบัติการ DMA และจากการที่ข้อมูลและคำสั่งต่างๆ อยู่ในพื้นที่หน่วยความจำที่ต่างกัน จึงทำให้การ access ของหน่วยความจำที่ร่วมกันมีความเป็นไปได้ ในรุ่น C6X นั้นมีพื้นที่สำหรับหน่วยความจำของ byte-addressable หน่วยความจำภายในจะถูกจัดการเหมือนโปรแกรมต่างๆและพื้นที่หน่วยความจำของข้อมูล ซึ่งมี 2 internal ports แบบ 32-bit .ในการ access หน่วยความจำภายใน

MEMORY BLOCK DESCRIPTION	BLOCK SIZE (BYTES)	HEX ADDRESS RANGE
Internal RAM (L2)	64K	0000 0000 – 0000 FFFF
Reserved	24M – 64K	0001 0000 – 017F FFFF
External Memory Interface (EMIF) Registers	256K	0180 0000 – 0183 FFFF
L2 Registers	256K	0184 0000 – 0187 FFFF
HPI Registers	256K	0188 0000 – 018B FFFF
McBSP 0 Registers	256K	018C 0000 – 018F FFFF
McBSP 1 Registers	256K	0190 0000 – 0193 FFFF
Timer 0 Registers	256K	0194 0000 – 0197 FFFF
Timer 1 Registers	256K	0198 0000 – 019B FFFF
Interrupt Selector Registers	256K	019C 0000 – 019F FFFF
EDMA RAM and EDMA Registers	256K	01A0 0000 – 01A3 FFFF
Reserved	6M – 256K	01A4 0000 – 01FF FFFF
QDMA Registers	52	0200 0000 – 0200 0033
Reserved	738M – 52	0200 0034 – 2FFF FFFF
McBSP 0/1 Data	256M	3000 0000 – 3FFF FFFF
Reserved	1G	4000 0000 – 7FFF FFFF
EMIF CE0†	256M	8000 0000 – 8FFF FFFF
EMIF CE1†	256M	9000 0000 – 9FFF FFFF
EMIF CE2†	256M	A000 0000 – AFFF FFFF
EMIF CE3†	256M	B000 0000 – BFFF FFFF
Reserved	1G	C000 0000 – FFFF FFFF

† The number of EMIF address pins (EA[21:2]) limits the maximum addressable memory (SDRAM) to 128MB per CE space. To get 256MB of addressable memory, additional general-purpose output pin or external logic is required.

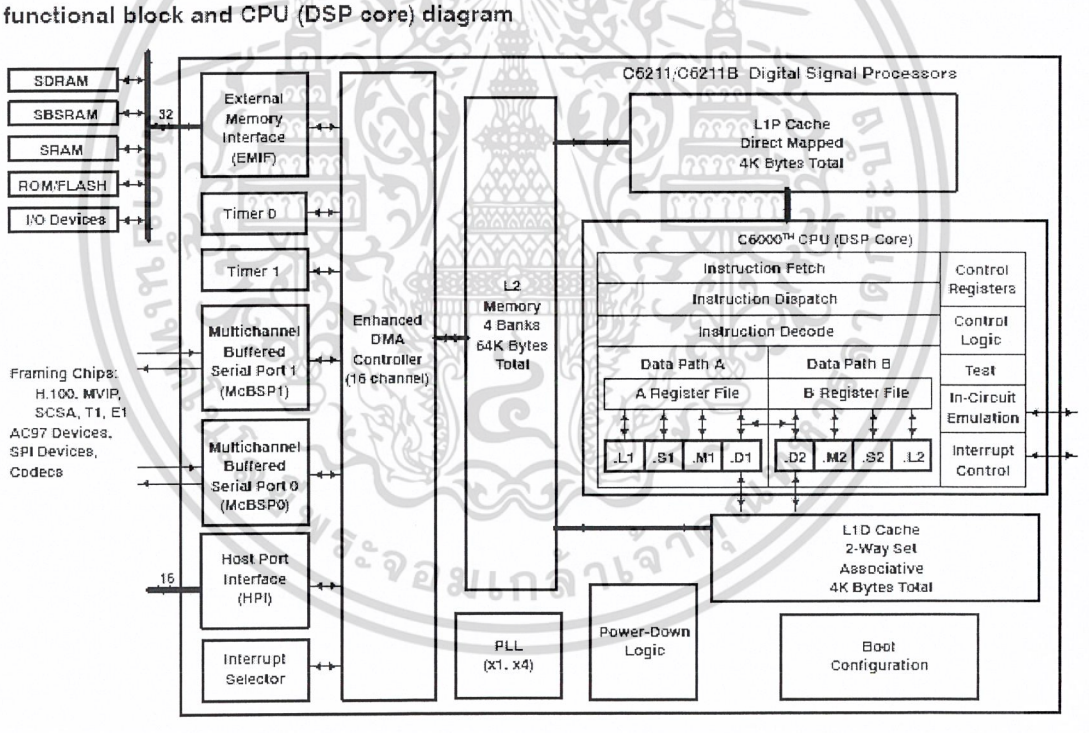
ตารางที่ 2.1 ผังตำแหน่งหน่วยความจำ

2.13.2 หน่วยของฟังก์ชั่นต่างๆ (Functional Units)

CPU จะประกอบด้วย 8 independent function units ซึ่งแบ่งออกเป็น 2 ส่วน A และ B แต่ละส่วนก็จะมีหน่วยสำหรับการคูณ (.M) สำหรับ logical and arithmetic operations (.L) สำหรับ branch, การจัดการบิทและ arithmetic operations (.S) และสำหรับ loading/storing and arithmetic operations (.D) โดย

หน่วย .S และ .L ใช้สำหรับ arithmetic, logical, และ branch instructions ส่วนการโยกย้ายข้อมูลทั้งหมดจะใช้หน่วย .D หน่วยส่วนใหญ่สามารถแสดง arithmetic operations เช่น ลบหรือบวก (SUB หรือ ADD) ได้ ยกเว้น หน่วย .M หน่วยของฟังก์ชั่นทั้ง 8 หน่วย ประกอบด้วย 4 floating/fixed-point ALUS (2 .L และ 2 .S) 2 fixed-point ALUS (หน่วย .D) และ 2 floating/fixed-point multipliers (หน่วย .M) แต่ละหน่วยของฟังก์ชั่นนั้นสามารถจะอ่านหรือเขียนได้โดยตรงกับ รีจิสเตอร์ไฟล์ภายในส่วนของมันเอง ในแต่ละส่วนก็จะรวมทั้งเซ็ทของ 16 รีจิสเตอร์แบบ 32-bit, A0 ถึง A15 และ B0 ถึง B15 หน่วยที่ลงท้ายด้วย 1 จะเขียนในรีจิสเตอร์ไฟล์ A และหน่วยที่ลงท้ายด้วย 2 จะเขียนในรีจิสเตอร์ไฟล์ B

two cross-paths (1X และ 2X) จะอนุญาตให้หน่วยของฟังก์ชั่นต่างๆจากส่วนหนึ่งสามารถ access 32-bit operand ได้จากรีจิสเตอร์ไฟล์บนด้านตรงข้าม มันจะมี 2 cross-path source ที่มากที่สุดที่อ่านต่อ 1 ไซเคิล และในแต่ละฝั่งของ functional unit ก็สามารถ access ข้อมูลจากรีจิสเตอร์บนด้านตรงข้ามโดยใช้ cross-path (functional unit บนด้านหนึ่งสามารถ access รีจิสเตอร์เซ็ทจากด้านอื่นได้) ส่วนรีจิสเตอร์ทั่วไปทั้ง 32 รีจิสเตอร์นั่น บางตัวก็ใช้สำหรับระบุตำแหน่งพิเศษหรือใช้สำหรับ conditional instruction



รูปที่ 2.31 บล็อกไดอะแกรมของ TMS320C6X

2.13.3 รีจิสเตอร์

รีจิสเตอร์ไฟล์จะมี 2 เซ็ท แต่ละเซ็ทมี 16 รีจิสเตอร์ คือ A0 ถึง A15 และ B0 ถึง B15 รีจิสเตอร์ A0, A1, B0, B1 และ B2 จะถูกใช้เป็น conditional registers รีจิสเตอร์ A4 ถึง A7 และ B4 ถึง B7 จะใช้เป็น circular addressing รีจิสเตอร์ A0 ถึง A9 และ B0 ถึง B9 (ยกเว้น B3) เป็นรีจิสเตอร์ชั่วคราว ส่วน A10 ถึง A15 และ B10 ถึง B15 จะใช้สำหรับเก็บและสำหรับการดึงข้อมูลมาใช้ก่อนที่จะกลับมาจาก subroutine

ค่าของข้อมูล 40 บิท สามารถบรรจุใน register pair ได้ โดย 32 least significant bits (LSBs) จะถูกเก็บในรีจิสเตอร์คู่ (เช่น A2) และที่เหลือ 8 บิท ถูกเก็บใน 8LSBs ของ next-upper register หรือรีจิสเตอร์คี่ (A3) และในระบบที่เหมือนกันนี้ก็ใช้เก็บค่าของ 64-bit double-precision ภายในทั้งคู่ของรีจิสเตอร์(คู่และคี่)

รีจิสเตอร์ทั้ง 32 นี้ จะถูกพิจารณาเป็นรีจิสเตอร์ทั่วไป ส่วนการ control และการ interrupts จะใช้รีจิสเตอร์เฉพาะอย่างเช่น address mode register (AMR) ใช้สำหรับ circular addressing และ interrupt control register

2.13.4 Interrupts

Interrupts จะมีทั้ง interrupt ภายในและ interrupt ภายนอก หน้าที่ของ interrupt นั้นจะหยุดการประมวลผลของ CPU และจะเปลี่ยนไปประมวลผลงานที่เกิด interrupt (ISR:Interrupt Service Routine) แหล่งกำเนิด interrupt มีทั้งหมด 16 ชนิด ประกอบด้วย timer 2 timer และ interrupt ภายนอกอีก 4 ชนิด, 4 McBSP interrupts, และ 4 DMA interrupts โดยจะเน้นที่ CPU interrupt ซึ่งมี 12 ชนิด และใช้ interrupt selector ในการเลือกใช้ interrupt ทั้ง12

Interrupt Control Registers

1. CSR(control status register): ประกอบด้วย GIE(global interrupt enable)bit และcontrol/status bits
2. IER(interrupt enable register):enables/disables individual interrupts
3. IFR(interrupt flag register):displays status of interrupts
4. ISR(interrupt set register):sets pending interrupts
5. ICR(interrupt clear register):clears pending interrupts
6. ISTP(interrupt service table pointer):locates an ISR
7. IRP(interrupt return pointer)
8. NRP(nonmaskable interrupt return pointer)

Interrupts ก็จะถูกจัดอันดับความสำคัญ โดยจะทำอันดับที่มีความสำคัญสูงก่อน และจะทำ interrupts มีความสำคัญรองลงมา

TABLE 3.5 Interrupt Service Table

Interrupt	Offset
RESET	000h
NMI	020h
Reserved	040h
Reserved	060h
INT4	080h
INT5	0A0h
INT6	0C0h
INT7	0E0h
INT8	100h
INT9	120h
INT10	140h
INT11	160h
INT12	180h
INT13	1A0h
INT14	1C0h
INT15	1E0h

ตารางที่ 2.2 การ interrupt

2.13.5 DMA(Direct memory access)

ยอมข้อมูลส่งผ่านจากอุปกรณ์ภายนอกเข้าสู่ internal memory โดยตรงโดยไม่มี CPU เข้ามายุ่งเกี่ยว DMA 4 channel จะถูกจัดโครงสร้างให้เป็นอิสระสำหรับการส่งผ่านข้อมูล ส่วน Addition channel มีความสำคัญกับ DMA ในส่วนของ HPI (host part interface) DMA สามารถเข้าถึงหน่วยความจำโดยตรงและสามารถเข้าถึง EMIF (external memory interface) ส่วนจำนวนบิทข้อมูลที่สามารถส่งผ่าน มีขนาด 8-bit, 16-bit, 32-bit จำนวนรีจิสเตอร์ถูกใช้ในการจัดโครงสร้างของ DMA เช่น Address, index, count reload ฯลฯ แอดแดรสต้นทางและปลายทางสามารถนำมาจาก internal program memory, internal data memory, external memory interface, และ internal peripheral bus การส่งผ่าน DMA สามารถทริกโดยอินเทอร์รัพจาก internal peripherals หรือจาก external pins

สำหรับแต่ละแหล่งกำเนิด, แต่ละ DMA channel สามารถโปรแกรมเพื่อลำดับความสำคัญด้วย CPU ระหว่าง 4 DMA channels, channel 0 มีความสำคัญสูงสุด และ channel 3 มีความสำคัญต่ำสุด แต่ละ DMA channel สามารถนำไปสร้าง initiating block transfer ของข้อมูลอย่างอิสระ บล็อกสามารถบรรจุจำนวนเฟรม ภายในแต่ละเฟรมสามารถมีหลายองค์ประกอบ แต่ละองค์ประกอบคือ single data value ส่วน DMA count reload register จะใช้เก็บค่าที่ใช้ในการระบุ frame count (16 MSBs) และ (16 LSBs) Enhanced DMA (EDMA) ดีกว่าตรงที่ใช้ 16 channels ในการโปรแกรมอย่างอิสระ

2.13.6 คุณลักษณะทางด้านซอฟต์แวร์

ซอฟต์แวร์ที่ใช้งานกับบอร์ดประมวลผลสัญญาณดิจิตอล C6000 คือ Code Composer Studio (CCS) โดยทางผู้ใช้งานจะต้องทำการติดตั้งโปรแกรมนี้ลงบนเครื่องไมโครคอมพิวเตอร์ที่เชื่อมต่ออยู่กับ

บอร์ดประมวลผลสัญญาณดิจิตอล CCS นี้ประกอบด้วย 3 ส่วนหลักๆได้แก่ Code Generation Tools (CGT), Integrated Development Environment (IDE) และส่วนวิเคราะห์การทำงานของโปรแกรม

CGT ประกอบด้วยคอมไพเลอร์ภาษาซี (C compiler) แอสเซมเบลอร์ (Assembler) และลิงค์เกอร์ (Linker) ผู้เขียนโปรแกรมสามารถพัฒนาโปรแกรมขึ้นมาด้วยตัวเองให้อยู่ในรูปโปรแกรม execute เพื่อถ่ายโปรแกรมดังกล่าวเข้าสู่บอร์ดประมวลผลสัญญาณดิจิตอลได้ โดยเริ่มจากผู้พัฒนาโปรแกรมเขียนโปรแกรม source file ขึ้นมาด้วยภาษาซี ตัวคอมไพล์เลอร์ภาษาซีจะทำการแปลให้เป็นภาษาแอสเซมบลี จากนั้นตัวแอสเซมเบลอร์จะทำการแปลให้เป็นภาษาเครื่องในรูปไฟล์ object และตัวลิงค์เกอร์จะทำการนำไฟล์ที่แปลได้ไปเชื่อมกับ library มาตรฐานและ library ที่ผู้เขียนโปรแกรมสร้างขึ้นแล้วสร้างไฟล์โปรแกรม execute ขึ้น

ผู้เขียนโปรแกรมสามารถพัฒนาโปรแกรมขึ้นมาในรูปของภาษาแอสเซมบลี (Linear Assembly) ได้ ซึ่งตัว Assembly Optimizer จะทำการปรับโปรแกรมให้อยู่ในรูปแบบเหมาะสมที่สุดก่อนจะทำการแปลเป็นภาษาเครื่องแล้วทำการเชื่อมกับ library และสร้างเป็นไฟล์โปรแกรม execute ส่วน IDE ทำหน้าที่ 3 ประการได้แก่ ให้ผู้ใช้งานสามารถแก้ไขโปรแกรม (Edit) สร้างโปรแกรม (build) และตรวจสอบการทำงาน (Debug) ของโปรแกรม ผู้เขียนโปรแกรมสามารถสร้างโปรแกรม execute ได้ด้วยการนำไฟล์ส่วนต่างๆมาผนวกกันในไฟล์โปรเจค (project file) โดยการใช้คำสั่ง Addfile ในเมนูหลักชื่อ Project ซึ่งประกอบด้วยไฟล์ source มีนามสกุล .c และ .asm ไฟล์ libraries มีนามสกุล .lib และไฟล์ linker command มีนามสกุล .cmd ตามลำดับ จากนั้นทำการสร้างโปรแกรมไฟล์ execute ที่มีนามสกุล .out ด้วยการใช้คำสั่ง Rebuild All ที่อยู่ในเมนูหลัก Project ซึ่ง CCS จะทำการแปลไฟล์ source ที่ผู้พัฒนาโปรแกรมเขียนขึ้นพร้อมทั้งทำการเชื่อมกับ libraries แล้วสร้างเป็นโปรแกรม execute นามสกุล .out ขึ้นมาผู้เขียนโปรแกรมสามารถถ่ายโปรแกรมที่ได้นี้เข้าสู่บอร์ดประมวลผลสัญญาณดิจิตอลให้ทำงานตามต้องการด้วยคำสั่ง Load Program ซึ่งอยู่ในเมนูหลักชื่อ File

2.14 โปรแกรม Code Composer Studio (CCS)

โปรแกรมนี้เป็นการ โปรแกรมที่พัฒนามาใช้งานบนวินโดว์ซึ่งเป็นตัวคอมไพล์เลอร์ภาษาซี แล้วใช้การเชื่อมโยงกับภาษาแอสเซมบลี โดยโปรแกรมนี้สามารถเขียนได้ทั้งภาษาซีและแอสเซมบลีและโปรแกรมนี้ยังสามารถพล็อตกราฟเป็นแบบกราฟฟิคได้

โปรแกรมนี้ทำการติดต่อกับบอร์ดผ่านพอร์ต LPT1 คือโปรแกรมจะทำการ โปรแกรมข้อมูลที่เราเขียนทั้งหมดลงบอร์ดและยังสามารถคอนโทรลผ่านพอร์ตได้อีกโดยการใช้บอร์ดกับโปรแกรมวิธีดังนี้

Hardware Setup → Code & Build → Program → Test

2.14.1 Hardware Setup

2.14.1.1 ลงโปรแกรม CCS

2.14.1.2 ทำการ Setup CCS โดยแน่ใจว่าบอร์ดนั้นเสียบกับพอร์ต LPT 1

อยู่แล้วให้เช็คที่ Bios ว่าเซตพอร์ตเป็นแบบ ECP หรือ EPP หรือไม่ และต่อมาทำการ Run โปรแกรม Setup CCS ทำการเลือกพอร์ตให้ตรงกับ Bios

2.14.1.3 ทำการ reset บอร์ดด้วยการกดปุ่ม reset ที่บอร์ดรอเสียงหรือไฟ Stand by ต้องเป็น ไบนารี่ 1-7

2.14.1.4 ทำการ Run CCS

2.14.2 Code & Build

2.14.2.1 ไฟล์แต่ละชนิด

- file.mak เป็นไฟล์โปรเจคที่เราสร้างขึ้น
- file.c เป็นไฟล์ C code ที่เราเขียน
- file.asm เป็นไฟล์ assembly code ที่เราเขียน หรือเกิดจากคอมไพล์จาก C code
- file.h เป็นไฟล์ header ที่ไว้#includeใน C
- file.lib เป็นไฟล์ ที่เราต้องแอดเพราะเป็น library ที่ไว้สนับสนุน ในบอร์ดรุ่นนี้ต้องโหลดไฟล์ rts6201.lib
- file.cmd เป็น link ไฟล์ ที่เราต้องเขียนเพื่อทำการ map memory
- file.obj เป็นobjectไฟล์ ที่สร้างโดย Assembly code
- file.out เป็นไฟล์สุดท้ายเมื่อทำการ build ไฟล์ทั้งหมดเป็นไฟล์ที่ไว้โปรแกรม
- file.gel General Extension Language เขียนคล้ายกับภาษาซีเขียน เพื่อเป็นการปรับเปลี่ยนค่าแอมปลิจูดต่างๆด้วยสไลด์บาร์

2.14.2.2 การสร้างโปรแกรม การคอมไพล์ และการ Build

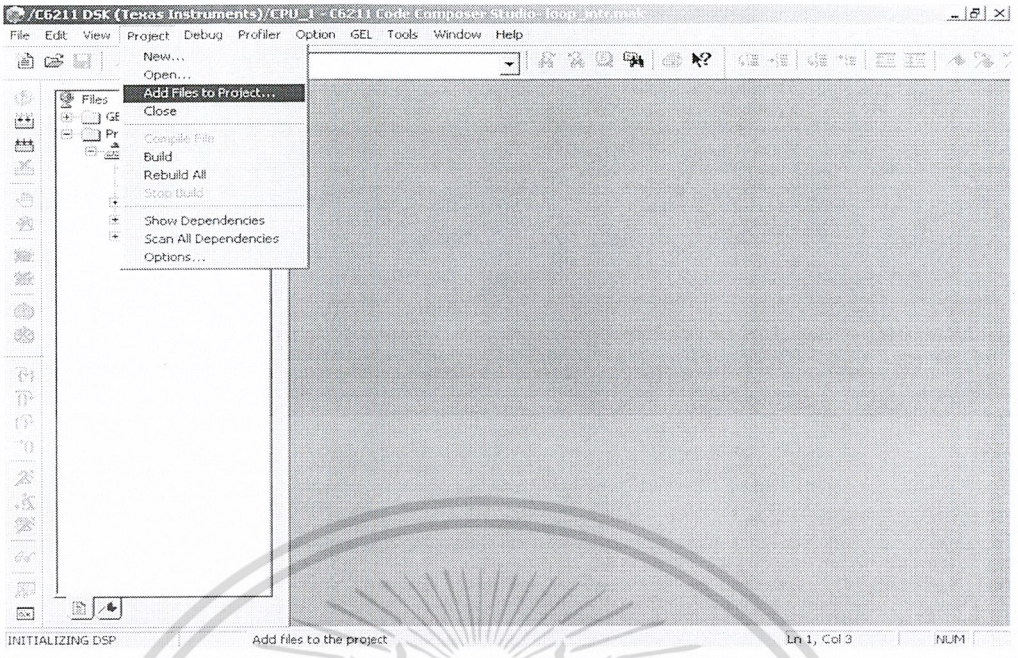
2.14.2.2.1 ทำการสร้างโปรเจคโดยควรสร้าง Subfolder รอไว้ก่อนแล้ว ไปที่ Project → New ตั้งชื่อ save ลง Subfolder ที่ตั้งไว้

2.14.2.2.2 ทำการ add Library โดยไปที่ Add file to Project ไปที่Folder C:\ti\c6000\cgtools\lib\rts6201.lib

2.14.2.2.3 ทำการ add file header โดยไปที่ Add file to Project ไปที่Folder C:\ti\c6000\cgtools\include\c6x.h

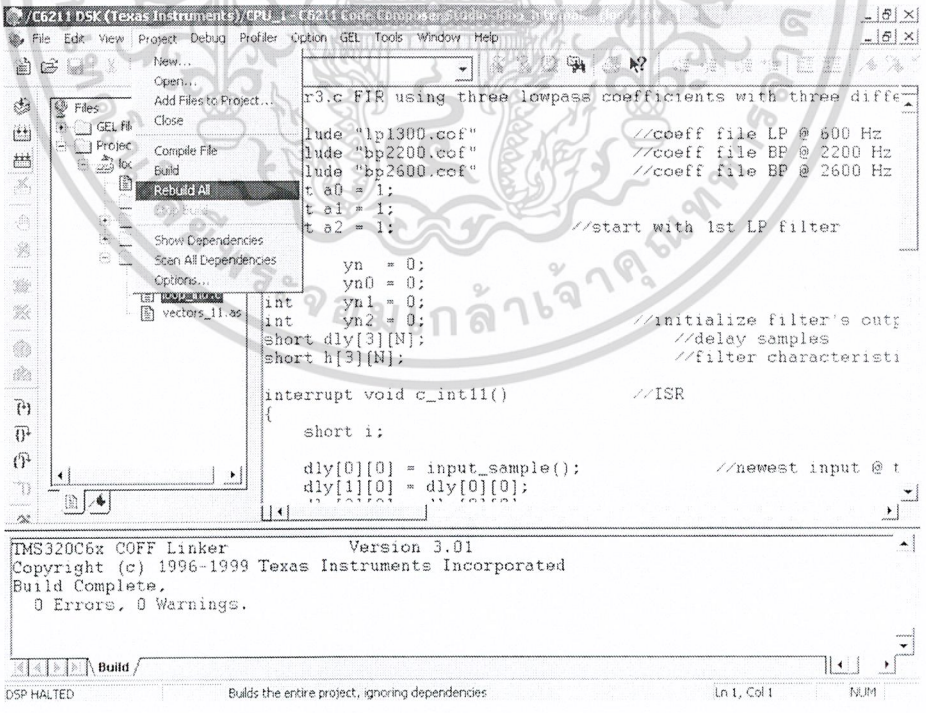
2.14.2.2.4 ทำการเขียน link file file.cmd เป็นไฟล์ที่ทำการแม็บเมโมรี่โดยการเขียนเป็นภาษาแอซเซมบลี แล้วทำการ add file เข้าโปรเจค

2.14.2.2.5 ต่อไปก็ทำการเขียน File assembly และ C เพื่อให้ทำงานตามที่เราออกแบบ แล้วทำการ add file เข้าโปรเจกต์



รูปที่ 2.32 การ Add file to project

2.14.2.2.6 ทำการ Build all เพื่อสร้างไฟล์.out โดยไปที่ Project→Rebuild all เพื่อ Build ทุกไฟล์หรือ click

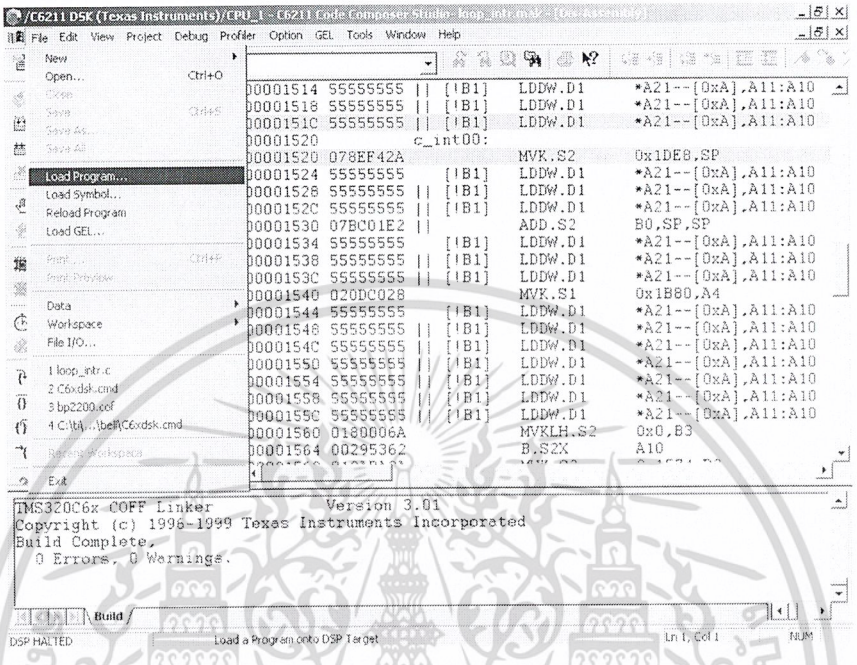


รูปที่ 2.33 การ Rebuild all

2.14.3 การโปรแกรม (Program)

เป็นการนำไฟล์ที่ได้เป็นไฟล์สุดท้ายนั้นโปรแกรมลงบอร์ดดังนี้

นำ file.out ไปทำการโปรแกรม โดยไปที่ file → Load Program



รูปที่ 2.34 การโปรแกรม

2.14.4 Test

ทำการทดสอบโดยการป้อนสัญญาณ ที่ Input Mic ของบอร์ด (JP7) และวัดสัญญาณ output speaker (JP6)

บทที่ 3
การคำนวณและการสร้าง

ในบทนี้จะแบ่งการคำนวณและการสร้างออกเป็น 3 ส่วน คือ

ส่วนที่ 1 การเขียนภาษา C ในโปรแกรม Code Composer Studio โดยใช้หลักการของ LPC เพื่อให้ได้ไฟล์ .out ออกมา ซึ่งจะมีหลักการคำนวณดังนี้

การคำนวณค่าสัมประสิทธิ์ของ LPC และอัตราการขยาย

ในการหาค่าสัมประสิทธิ์ LPC (α) และอัตราการขยาย (G) เราจำเป็นต้องมีการเตรียมข้อมูลในการวิเคราะห์ก่อน ซึ่งขั้นตอนในการเตรียมสัญญาณเข้าเป็นดังรูป 3.1

การวิเคราะห์สามารถแบ่งเป็นขั้นตอนต่าง ๆ ดังนี้

3.1 การพรีเอมฟาซิส (Pre-emphasis)

3.2 การแบ่งช่วงสัญญาณ (Frame blocking)

3.3 การวินโดว์ (Windowing)

3.4 การหาออโตคอร์รีเลชั่น (Autocorrelation analysis)

3.5 การหาสัมประสิทธิ์ α และค่าอัตราการขยาย G

3.1 การพรีเอมฟาซิส

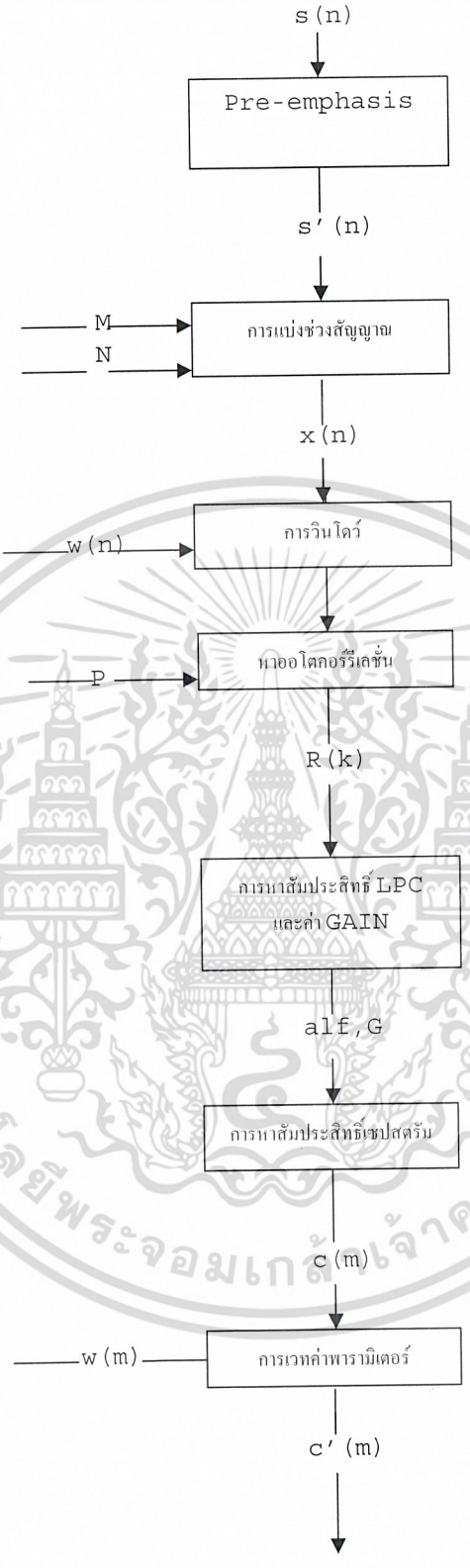
เนื่องจากสัญญาณเสียงพูดของมนุษย์ จะมีองค์ประกอบส่วนใหญ่อยู่บริเวณความถี่ต่ำ เมื่อเทียบกับแถบความถี่ที่ปฏิบัติงาน (Bandwidth) ไม่เกิน 5 กิโลเฮิรตซ์ ดังนั้นเพื่อให้อัตราส่วนสัญญาณเสียงต่อสัญญาณรบกวน (signal to noise ratio : SNR) มีค่าค่อนข้างคงที่ ตลอดช่วงความถี่ที่ปฏิบัติงานนี้ เราจึงต้องมีการพรีเอมฟาซิส โดยเน้นความถี่สูงให้มีขนาดสูงขึ้น นั่นคือ การพรีเอมฟาซิสก็คือการกรองสัญญาณด้วยวงจรกรองความถี่สูงผ่าน (high pass filter) ซึ่งมักนิยมใช้วงจรกรองอันดับหนึ่ง มีฟังก์ชั่นถ่ายโอนเป็น

$$H(z) = 1 - az^{-1} \tag{3.1.1}$$

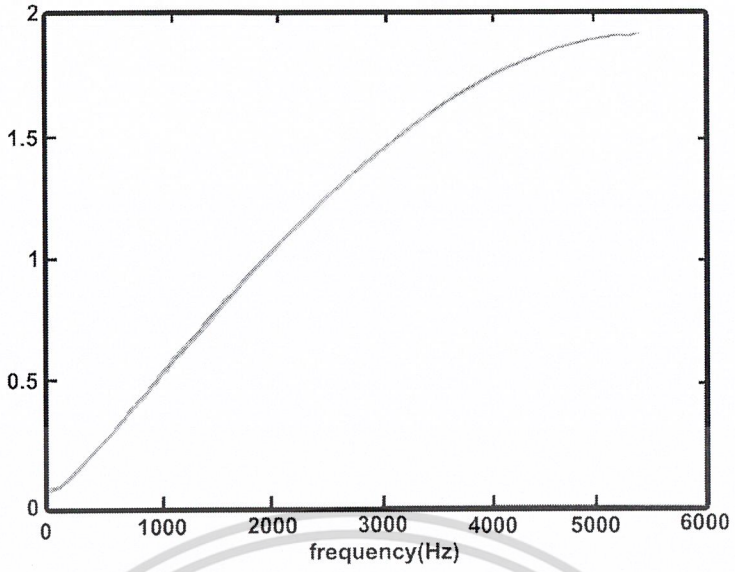
โดยที่ $0.9 < a < 1.0$ เราจะได้ว่า

$$s'(n) = s(n) - as(n-1) \tag{3.1.2}$$

ยิ่งค่า a ใกล้ 1 เท่าใด ความถี่สูงก็จะถูกขยายมากขึ้นเท่านั้น ค่า a ที่นิยมสำหรับใช้ในการหาพารามิเตอร์ของ LPC คือ 15/16 = 0.9375 เมื่อนำฟังก์ชั่นถ่ายโอนมาพล็อตกราฟของขนาดเทียบกับความถี่จะได้ดังรูปที่ 3.2



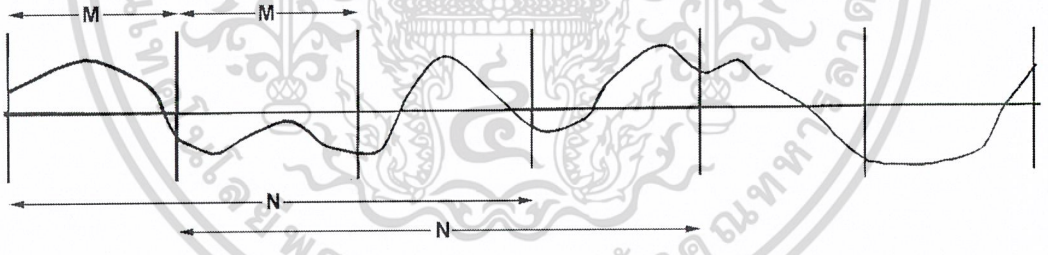
รูปที่ 3.1 ขั้นตอนการเตรียมสัญญาณในการวิเคราะห์



รูปที่ 3.2 ขนาดสเปคตรัมของฟังก์ชันถ่ายโอนของการพรีเอมฟาซิส

3.2 การแบ่งช่วงสัญญาณ

สัญญาณที่ผ่านการพรีเอมฟาซิสแล้ว $s'(n)$ จะถูกตัดแบ่งออกเป็นช่วงๆ หรือ เฟรม ช่วงละ N ตัวอย่างสัญญาณ การวิเคราะห์จะวิเคราะห์ทีละช่วงของแต่ละ N ตัวอย่างสัญญาณ ดังรูปที่ 3.3



รูปที่ 3.3 การแบ่งช่วงของสัญญาณที่ใช้ในการวิเคราะห์

โดยช่วงในการวิเคราะห์แต่ละช่วงจะถูกเลื่อนไปเป็นระยะ M ช่วงสัญญาณ จะเห็นได้ว่า ถ้าค่า M โตกว่าค่า N ในการเลื่อนของช่วงในการวิเคราะห์จะทำให้บางสัญญาณไม่ถูกใช้ในการวิเคราะห์ ก็จะเป็นการสูญเสียส่วนหนึ่งทำให้ผลที่ได้ไม่ถูกต้องเท่าที่ควร ถ้าค่า M เล็กกว่าค่า N ก็จะทำให้ตัวอย่างสัญญาณทุกตัวถูกนำมาวิเคราะห์ ยิ่งค่า M เล็กเท่าใด ความแม่นยำในการวิเคราะห์ก็จะยิ่งสูงขึ้นเท่านั้น แต่ก็จะทำให้การคำนวณช้าลง

หลักการกำหนดขนาดของวินโดว์

3.2.1 วินโดว์จะต้องสั้นพอที่คุณสมบัติของเสียงที่เราสนใจจะวิเคราะห์นั้น ยังไม่เปลี่ยนแปลงในวินโดว์

3.2.2 วินโดว์จะต้องยาวพอที่จำนวนตัวอย่างสัญญาณในวินโดว์ นั้นยังสามารถนำมาคำนวณหาคุณสมบัติที่ต้องการได้

3.2.3 วินโดว์ที่ติดกัน ไม่ควรจะสั้นจนกระโดดข้ามข้อมูลบางส่วนไปแต่ควรเลื่อนวินโดว์ให้น้อยกว่าขนาดของเฟรม

เนื่องจากเราใช้ความถี่ในการสุ่มสัญญาณ 11.025 กิโลเฮิรตซ์ ในการวิเคราะห์นี้ เราเลือกใช้ค่า N=300 และค่า M=100 นั่นคือ ช่วงในการวิเคราะห์ คือ 27.21 มิลลิวินาที และระยะในการเลื่อนเฟรมประมาณ 9 วินาที

3.3 การวินโดว์

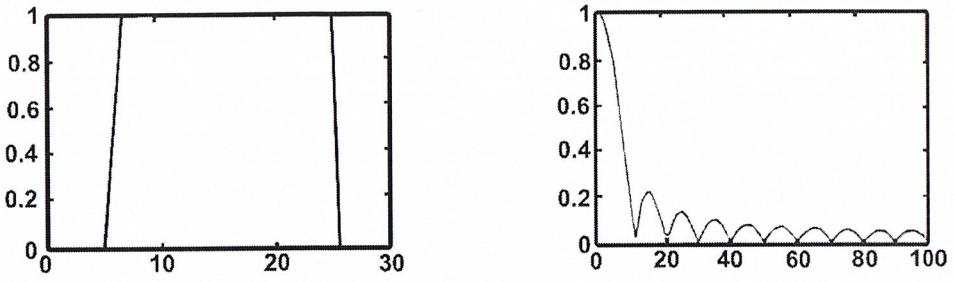
พิจารณาในช่วงสัญญาณ N ตัวอย่างสัญญาณของช่วงใด ๆ ที่ตัดมาวิเคราะห์ (รูปที่ 3.4) จะเห็นว่าที่ขอบของเฟรมที่ตัดมานี้มีความไม่ต่อเนื่องของสัญญาณ ถ้ามองในโดเมนของความถี่สูง ก็จะมีความถี่สูงเกิดขึ้น ดังนั้นเพื่อที่จะลดองค์ประกอบทางความถี่ที่สูงเหล่านี้ เราจะคูณด้วยฟังก์ชันวินโดว์เพื่อลดความไม่ต่อเนื่องของสัญญาณที่ขอบ และไม่ทำให้สเปคตรัมของสัญญาณในช่วงความถี่ต่ำเปลี่ยนไปมากนัก ในที่นี้จะใช้ฟังก์ชันวินโดว์แฮมมิ่ง (Hamming window function) ซึ่งนิยามโดยสมการดังนี้ คือ

$$w(n) = 0.54 - 0.46\cos(2\pi n/(N-1)) \qquad \text{เมื่อ } n = 0,1,2,\ldots,N-1 \qquad (3.3.1)$$

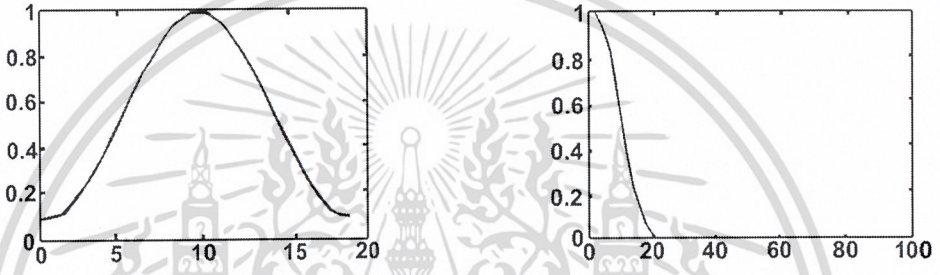
รูปที่ 3.5 และ 3.6 แสดงองค์ประกอบทางเวลา และทางความถี่ของฟังก์ชันวินโดว์แบบสี่เหลี่ยม และแฮมมิ่ง ตามลำดับ จะเห็นว่าสเปคตรัมของวินโดว์แฮมมิ่งมีริพเพิล ripple น้อยกว่าของวินโดว์สี่เหลี่ยม ดังนั้นจากรูปที่ 3.1 เราจะได้ว่า

$$x'(n) = x(n)w(n) \qquad (3.3.2)$$

รูปที่ 3.4 ส่วนของสัญญาณที่ตัดมาวิเคราะห์



รูปที่ 3.5 วินโดว์สี่เหลี่ยม (ก) ในโดเมนเวลา (ข) ในโดเมนความถี่



รูปที่ 3.6 วินโดว์แบบแฮมมิ่ง (ก) ในโดเมนเวลา (ข) ในโดเมนความถี่

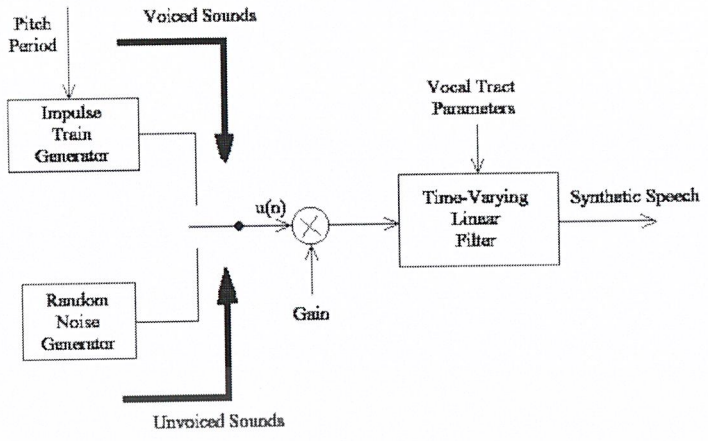
3.4 การคำนวณออโตคอร์รีเลชัน

จากหลักการพื้นฐานของการประมาณเชิงเส้นคือ การประมาณค่าสัญญาณจากผลรวมเชิงเส้นของสัญญาณก่อนหน้านี้ สมมติว่าสัญญาณเดิมเป็น $s(n)$ การประมาณค่าสัญญาณเป็น $s'(n)$ ดังนั้นเราสามารถอธิบายการประมาณเชิงเส้นได้ด้วยสมการต่อไปนี้

$$s'(n) = \sum_{k=1}^{p} \alpha_k s(n-k) \qquad (3.4.1)$$

เมื่อ α_k เป็นค่าคงที่ เรียกวิธีการนี้ว่าการประมาณเชิงเส้นอันดับ p โดยมีเงื่อนไขว่า ค่า α_k ที่ใช้ในการประมาณจะต้องทำให้ ผลรวมของกำลังสองของความคลาดเคลื่อน $\{s(n)-s'(n)\}^2$ มีค่าน้อยที่สุดนั่นคือ $\sum e^2 = \sum \{s(n)-s'(n)\}^2$ มีค่าต่ำที่สุด ซึ่งจะใช้การประมาณเชิงเส้นวิธีออโตคอร์รีเลชัน (Autocorreiation Method) หรือ วิธีอัตตสัมพันธ์

จากหลักการพื้นฐานของการประมาณเชิงเส้นและแบบจำลองระบบสร้างสัญญาณเสียง เราจะสามารถเขียนบล็อกไดอะแกรมการทำการประมาณเชิงเส้นมาสร้างสัญญาณเสียงพูดได้ดังรูปที่ 3.7



รูปที่ 3.7 บล็อกไดอะแกรมแสดงโมเดลการสร้างสัญญาณเสียงพูดอย่างง่าย

จากรูปที่ 3.7 สามารถเขียนสมการได้เป็น

$$s(n) = G * u(n) + \sum_{k=1}^{p} \alpha_k s(n-k) \tag{3.4.2}$$

การประมาณเชิงเส้นโดยใช้สัมประสิทธิ์ $\{\alpha_k\}$ คือ

$$s'(n) = \sum_{k=1}^{p} \alpha_k s(n-k) \tag{3.4.3}$$

ดังนั้น ความคลาดเคลื่อน คือ

$$\begin{aligned} e(n) &= s(n) - s'(n) \\ &= s(n) - \sum_{k=1}^{p} \alpha_k s(n-k) \end{aligned} \tag{3.4.4}$$

ฟังก์ชันถ่ายโอนระหว่าง $e(n)$ และ $s(n)$ คือ

$$\begin{aligned} A(z) &= E(z)/S(z) \\ &= 1 - \sum_{k=1}^{p} \alpha_k z^{-1} \end{aligned} \tag{3.4.5}$$

จากสมการ (3.4.2) - (3.4.4) จะเห็นได้ว่า

$$e(n) = G * u(n) \tag{3.4.6}$$

ดังนั้น ค่าผลรวมของกำลังสองของความคลาดเคลื่อน

$$E_n = \sum_m e_n^2(m)$$
$$E_n = \sum_m [s(m) - s'(m)]^2 \tag{3.4.7}$$

โดยที่ n คือ ช่วงที่ n ของสัญญาณที่ใช้คำนวณ เพราะฉะนั้นเพื่อให้ได้ค่า E_n ต่ำที่สุดจะต้องมีเงื่อนไขว่า

$$\frac{\partial \mathrm{E}_\mathrm{n}}{\partial \alpha_\mathrm{i}} = 0 \qquad \text{เมื่อ } i = 1,2,\ldots, p$$

จากสมการ (3.4.7)

$$\frac{\partial \mathrm{E}_\mathrm{n}}{\partial \alpha_\mathrm{i}} = -2 s_n(m-i) \sum_m [s_n(m) - \sum_{\mathrm{k=1}}^{\mathrm{p}} \alpha_k s_n(m-k)] \qquad \text{เมื่อ } i = 1,2,\ldots, p$$
$$= -2[\sum_m s(m)s(m-i) - \sum_{k=1}^{p} \sum_m \alpha_k s(m-k)s(m-i)]$$

$\frac{\partial \mathrm{E}_\mathrm{n}}{\partial \alpha_\mathrm{i}} = 0$ ก็ต่อเมื่อ

$$\sum_{k=1}^{p} \alpha_k \sum_m s_n(m-k)s_n(m-i) = \sum_m s_n(m)s_n(m-i) \qquad \text{เมื่อ } i = 1,2,\ldots, p \tag{3.4.8}$$

ถ้าเรากำหนดให้ $\phi_n(i,k) = \sum s_n(m-k)s_n(m-i)$ เพราะฉะนั้น

$$\sum_{k=1}^{p} \alpha_k \phi_n(i,k) = \phi_n(i,0) \tag{3.4.9}$$

โดยสมการ (3.4.7)-(3.4.8) จะได้ว่า

$$E_n = \sum_m s_n^2(m) - \sum_{k=1}^{p} \alpha_k \sum_m s_n(m)s_n(m-k)$$

และจาก $\phi_n(i,k) = \sum s_n(m-k)s(m-i)$

$$E_n = \phi_n(0,0) - \sum_{k=1}^{p} \alpha_k \phi_n(0,k) \qquad (3.4.10)$$

สมมติว่าใน 1 เฟรม ของสัญญาณ ที่ตัดมาคำนวณมี N ตัวอย่าง คือ $s_n(0), s_n(1), s_n(2), ..., s_n(N-1)$ ในที่นี้เราให้ $s_n(m) = 0$ เมื่อ $m > N-1$ เพราะฉะนั้น

$$\phi_n(i,k) = \sum_{m} s_n(m-k)s_n(m-i)$$

$$= \sum_{m=0}^{N-1-(i-k)} s_n(m)s_n(m+i-k) \qquad 0 \le k \le p, 1 \le i \le p$$

ให้ $$R_n(k) = \sum_{m}^{N-1-k} s_n(m)s_n(m+k) \text{ เมื่อ } k = 0,1,2,...,p \qquad (3.4.11)$$

ดังนั้น จากสมการ (3.3.2) และ (3.4.11) จะได้ว่า

$$R_n(k) = \sum_{m=0}^{N-1-k} x'(m)x'(m+k) \qquad (3.4.12)$$

จากสมการที่ (3.4.9) จะได้ว่า

$$\sum_{k=1}^{p} \alpha_k R_n(|i-k|) = R_n(i) \qquad \text{เมื่อ } i = 1,2,...,p \qquad (3.4.13)$$

จากสมการที่ (3.4.13) เขียนให้อยู่ในรูปเมตริกซ์ได้เป็น

$$\begin{bmatrix} R_n(0) & R_n(1) & \cdots & R_n(p-1) \\ R_n(1) & R_n(0) & \cdots & R_n(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_n(p-1) & R_n(p-2) & \cdots & R_n(0) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \vdots \\ R_n(p) \end{bmatrix} \qquad (3.4.14)$$

หรือ $$R_n * \alpha = r_n \qquad (3.4.15)$$

เมื่อ $R_n = \begin{bmatrix} R_n(0) & R_n(1) & \cdots & R_n(p-1) \\ R_n(1) & R_n(0) & \cdots & R_n(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_n(p-1) & R_n(p-2) & \cdots & R_n(0) \end{bmatrix}, \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix}$ และ $r_n = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \vdots \\ R_n(p) \end{bmatrix}$

3.5 การหาสัมประสิทธิ์ α และอัตราการขยาย G

เมื่อได้ค่า $R_n(0), R_n(1), R_n(2), ..., R_n(p)$ แล้วก็สามารถหาค่า α ได้จากสมการ (3.4.15) นั่นคือ

$$\alpha = R_n^{-1} * \mathrm{r_n} \tag{3.5.1}$$

และจากสมการ (3.5.1) จะได้ว่า

$$e(n) = G * u(n)$$

$$E_n = \sum_{m=0}^{N-1} e^2(m) = G^2 \sum_{m=0}^{N-1} u^2(m)$$

จากสมการที่ (3.4.10) จะได้ว่า

$$E_n = \phi_n(0,0) - \sum_{k=1}^{p} \alpha_k \phi_n(0,k)$$

$$= R_n(0) - \sum_{k=1}^{p} \alpha_k R_n(k) \tag{3.5.2}$$

และจากสมการที่ (3.5.2) เราสามารถหาค่า G โดยตรงจาก

$$G^2 = \frac{R_n(0) - \sum_{k=1}^{p} \alpha_k R_n(k)}{\sum_{m=0}^{N-1} u^2(m)} \tag{3.5.3}$$

3.6 การหาค่า pitch period

เมื่อได้ค่าสัมประสิทธิ์แล้ว ในกรณีที่อินพุทเป็นเสียง เราก็จะต้องหาค่า pitch period ซึ่งหาได้จาก

$$r_c(n) = \sum_{k=1}^{p} r_a(k) r_{ss}(n-k) \tag{3.5.4}$$

โดยที่ค่า r_a หาได้จาก

$$r_a(n) = \sum_{i=1}^{p} a_p(i) a_p(i+k) \tag{3.5.5}$$

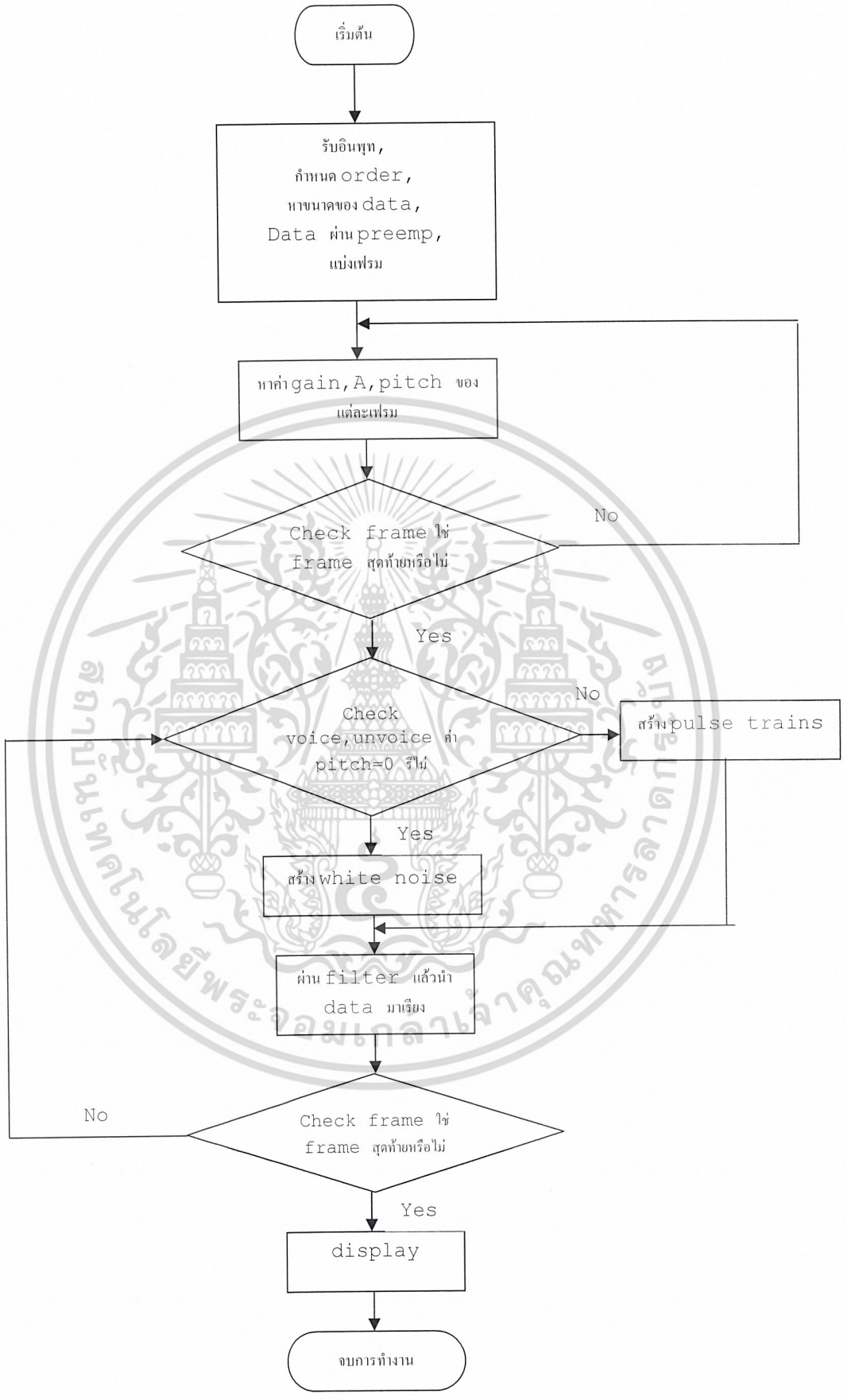
โดยที่ค่า pitch period หาได้จากค่าสูงสุดของลำดับ $\frac{r_e(n)}{r_e(0)}$ ซึ่งถ้าค่าสูงสุดมีค่าอย่างน้อยเท่ากับ 0.25 เฟรมนั้นก็จะถูกพิจารณาว่าเป็น voiced โดยมี pitch period เท่ากับค่าของ $n = N_p$ ซึ่งทำให้ $\frac{r_e(n)}{r_e(0)}$ มีค่ามากสุด แต่ถ้าค่าสูงสุดมีค่าน้อยกว่า 0.25 แล้ว เฟรมนั้นจะถูกพิจารณาว่าเป็น unvoiced และค่า pitch ก็จะเป็น 0

3.7 All-pole filter

ในโครงงานชิ้นนี้ระบบที่เราสนใจ คือ Filter ที่ใช้ในการสังเคราะห์สัญญาณเสียง ในที่นี้เราจะใช้ filter แบบ All-pole filter โดยจะสามารถสร้างมาจากการนำค่าสัมประสิทธิ์ และค่า Gain ที่ได้จากการวิเคราะห์สัญญาณเสียงแล้วเราจะนำค่าพารามิเตอร์เหล่านี้มาใส่ในฟิลเตอร์ที่มี transfer function คือ

$$H(z) = \frac{G}{1 + \sum_{k=1}^{p} \alpha_k z^{-k}} \tag{3.5.6}$$

โดยหลังจากที่นำค่าพารามิเตอร์ต่างๆ มาใส่ใน Filter แล้ว Filter ก็จะทำการสังเคราะห์สัญญาณเสียงออกมาดังรูปที่ 3.7



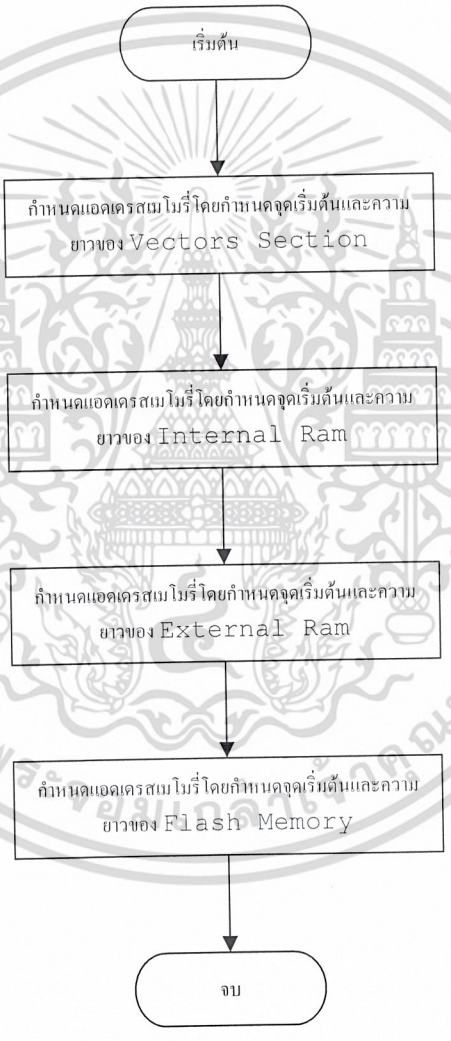
รูปที่ 3.8 แผนภาพการทำงานของโปรแกรม

3.8 การเขียนโปรแกรม Code Composer Studio

ในที่นี้การเขียนโปรแกรมในส่วนต่างๆทุกสมการจะต้องมีไฟล์พื้นฐานในการอินเทอร์รัพแม็บเมโมรี่ การจัดรีจิสเตอร์ฟังก์ชั่นต่างๆที่เหมือนกัน เราจึงจำเป็นต้องสร้างไฟล์พื้นฐาน โดยไฟล์ที่สร้างขึ้นมีดังต่อไปนี้

3.8.1 Linker command file (file.cmd)

เป็นไฟล์ที่ทำการแม็บเมโมรี่ มีแผนภาพการทำงานดังนี้



รูปที่ 3.9 แผนภาพการทำงานโปรแกรม linker command file

3.8.2 Interrupt vector File

การอินเทอร์รัพเป็นชื่อเรียกกระบวนการที่เข้ามาขัดจังหวะการทำงานโดยปกติของ CPU เมื่อมีการอินเทอร์รัพเกิดขึ้นและมีการ enable การตอบสนองอินเทอร์รัพไว้ จะทำการกระโดดไปยัง address ในหน่วยความจำที่กำหนดไว้ เรียกตำแหน่งนั้นว่า interrupt vector address ดังนั้นจะต้องมีการเขียนโปรแกรมย่อย interrupt service routine (ISR) ไว้ที่ interrupt vector

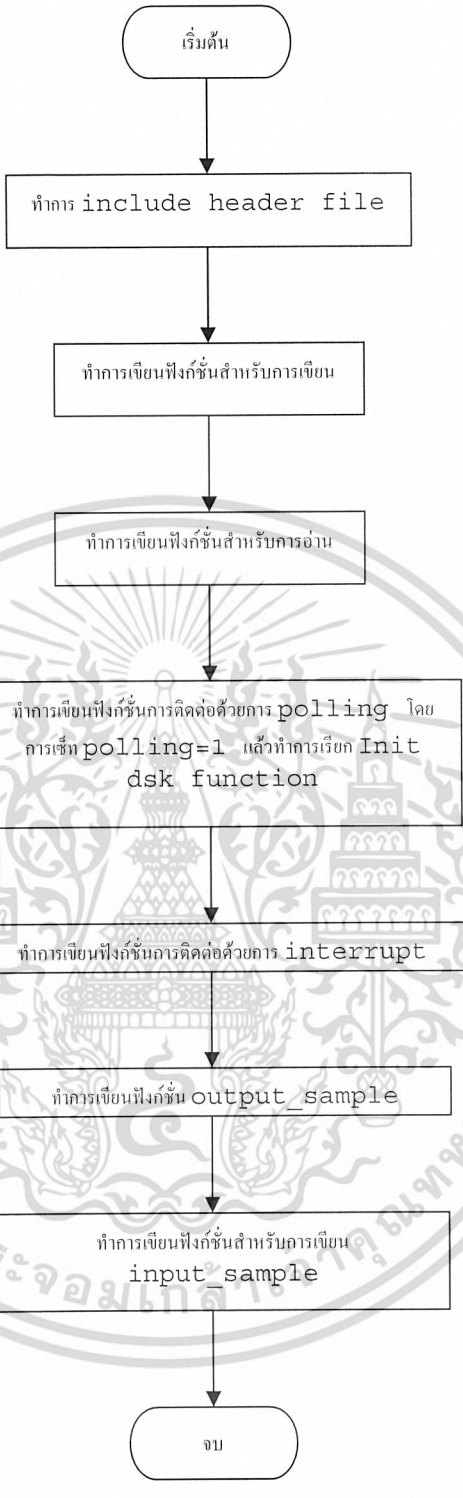
ดังนั้นไฟล์ interrupt vector file นี้เป็นไฟล์ที่เลือกตัวอินเทอร์รัพเพื่อเชื่อมกับตัว ISR (interrupt service routine) โดยอินเทอร์รัพต่างๆนั้นจะมีตาราง address ดังตารางที่ 2.5

3.8.3 Initialization/Communication File (C6xdskinit.c)

ไฟล์นี้เป็นไฟล์คอมไพล์ร่วมเก็บฟังก์ชั่นต่างๆไว้ ได้แก่ ฟังก์ชั่นอินเทอร์รัพ (comm.intr) การโพล (comm.poll) และฟังก์ชั่นการรับแซมเปิ้ลและส่งแซมเปิ้ลออก (Input_sample, Output_sample) รวมทั้งฟังก์ชั่น Mcbsp0_read, Mcbsp0_write ฟังก์ชั่นเหล่านี้จะทำการติดต่อกับ DSK บอร์ดและยังมีการ include header file ทั้งหมดสี่ไฟล์ดังนี้

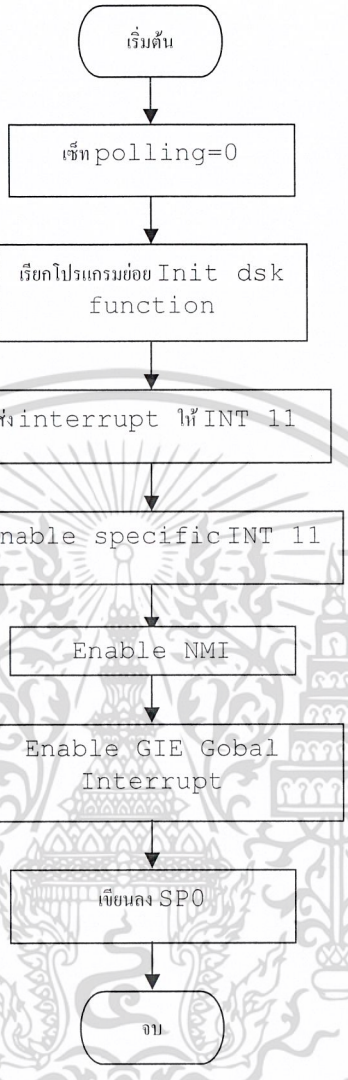
- C6x.h เป็นไฟล์ที่โปรแกรม CCS มีไว้ สามารถนำมาเรียกใช้ได้เลยซึ่งไฟล์นี้เป็นการเชื่อมโยงภาษาซีให้เข้ากับภาษาแอสเซมบลี โดยมีแผนผังการทำงานดังรูปที่ 3.10
- C6xdsk.h เป็นไฟล์ที่ทำการกำหนดค่าต่างๆให้ภาษาซีนั้นรู้จักกับรีจิสเตอร์ต่างๆ
- C6xdskinit.h เป็นไฟล์ที่กำหนดชนิดของฟังก์ชั่นที่เราเขียนในโปรแกรมหลักว่าให้เป็นชนิดตัวแปรชนิดใด
- C6xinterrupts.h เป็นไฟล์กำหนดค่าเริ่มต้นของตัวอินเทอร์รัพของบอร์ดที่เราทำการเลือก

โดยโปรแกรมหลักมีแผนผังการเขียนโปรแกรมดังนี้



รูปที่ 3.10 แผนภาพการทำงานโปรแกรม Initialization/Communication File

โปรแกรมย่อยฟังก์ชั่นการติดต่อด้วยการอินเทอร์รัพ



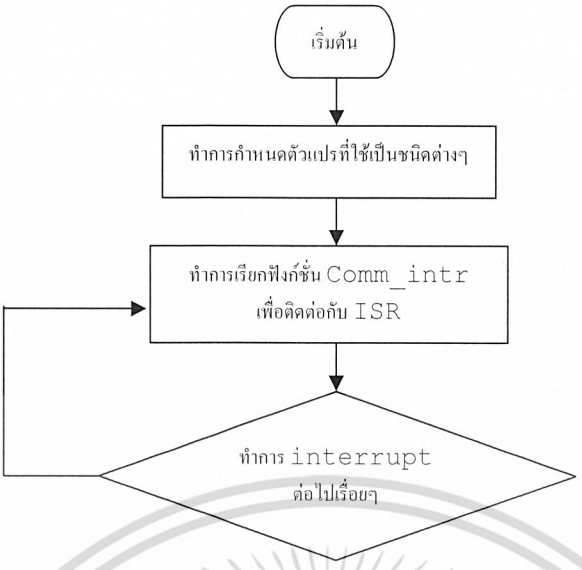
รูปที่ 3.11 แผนภาพการทำงานโปรแกรมย่อยฟังก์ชั่นการติดต่อด้วยการอินเทอร์รัพ

3.8.4 โปรแกรมหลัก

โปรแกรมนี้จะเป็นโปรแกรมหลักคือ เราจะทำการเขียนสมการต่างๆที่ตัวโปรแกรมนี้และการเขียนโปรแกรมรับค่าอินพุทนั้นจะสามารถทำได้สองวิธีคือ

- แบบอินเทอร์รัพ
- แบบโพล

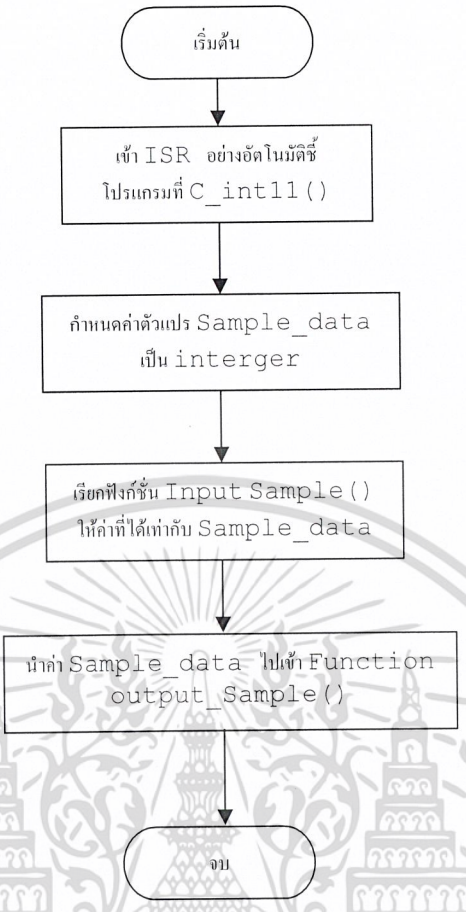
ในชิ้นงานนี้ขอเสนอแบบอินเทอร์รัพเพียงแบบเดียวเท่านั้น ซึ่งเมื่อเกิดการอินเทอร์รัพแล้วระบบจะทำการติดต่อ ISR (interrupt service routine) โดยอัตโนมัติเพราะฉะนั้น การอินเทอร์รัพหนึ่งครั้งจะนำเข้า ISR ไปและก็จับเข้าทำการประมวลผล (Process) ตามสมการต่างๆ โปรแกรมหลักมีการทำงานดังนี้



รูปที่ 3.12 แผนภาพการทำงานโปรแกรมหลัก

ต่อไปเมื่อเข้าฟังก์ชั่น comm_intr โปรแกรมก็จะถูกเรียกเข้า ISR ที่ชี้ค่าเป็น c_int11 แล้วทำการรับส่วนของ input sample เข้ามาทำการประมวลผลต่างๆ เพราะฉะนั้นทุกโปรแกรมที่เขียนต่อไปก็ทำการเปลี่ยนแปลงโปรแกรมแค่ส่วน c_int11 เท่านั้น

โดยเริ่มแรกเราจะทำการเขียนโปรแกรมให้รับค่าเข้ามาแล้วเก็บค่าไว้ที่ตัวแปรหนึ่ง แล้วก็นำค่าออกไปเลย เราเรียกโปรแกรมนี้ว่า loop_intr โปรแกรมนี้ก็จะเป็นโปรแกรมที่ทำงานรับอินพุทแล้วส่งออกเลย ทำให้อินพุทจะเท่ากับเอาท์พุทมีการทำงานดังนี้



รูปที่ 3.13 แผนภาพการทำงานฟังก์ชั่น comm_intr (ติดต่อ ISR) หรือ c_int11

เพราะฉะนั้นต่อไปเราก็จะทำการเขียนโปรแกรมต่อไปเราจะแก้ไขในส่วนที่เป็น comm_intr หรือส่วนที่เป็น c_int11 เท่านั้น

ตัวอย่างโปรแกรม Loop_intr

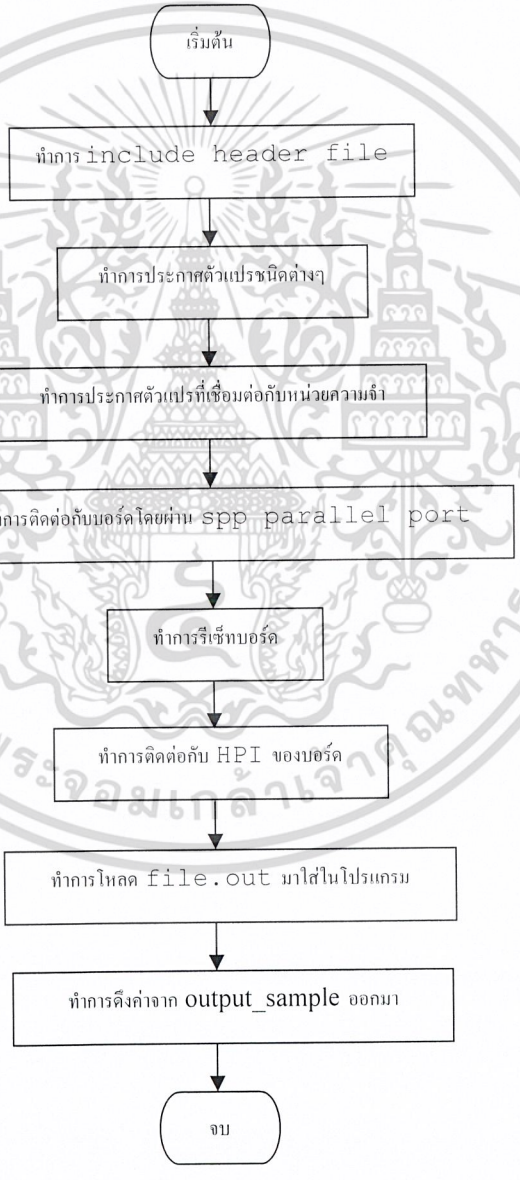
```
interrupt void c_int11()    //interrupt service routine
{
   int sample_data;

   sample_data = input_sample(); //input
   output_sample(sample_data);   //output
   return;
}

void main()
{
  comm_intr();

  while(1);
}
```

เปลี่ยนเฉพาะส่วนนี้

ส่วนที่ 2 การนำไฟล์ .out ที่ได้จากส่วนที่ 1 มาเขียนลงในโปรแกรม Visual C++ เพื่อทำการติดต่อกับชิพ DSP TMS320C6211 โดยแบ่งการทำงานออกเป็นดังนี้

3.9 การเขียนโปรแกรมติดต่อกับ ชิพ DSP TMS320C6211

เป็นการเขียนโปรแกรมภาษา C ที่ใช้ในการติดต่อกับบอร์ด โดยไม่ต้องผ่านโปรแกรม Code Composer Studio ซึ่งจะใช้หลักการของ HPI (Host Post Interface) ในการติดต่อกับชิพ DSP แล้วนำไฟล์ .out จากส่วนที่ 1 มาใส่ลงในโปรแกรม ซึ่งก็คือการนำหลักการ LPC ที่คำนวณได้มาทำการประมวลผลลงในชิพ DSP นั่นเอง



รูปที่ 3.14 แผนภาพการทำงานของโปรแกรมที่ติดต่อกับบอร์ด

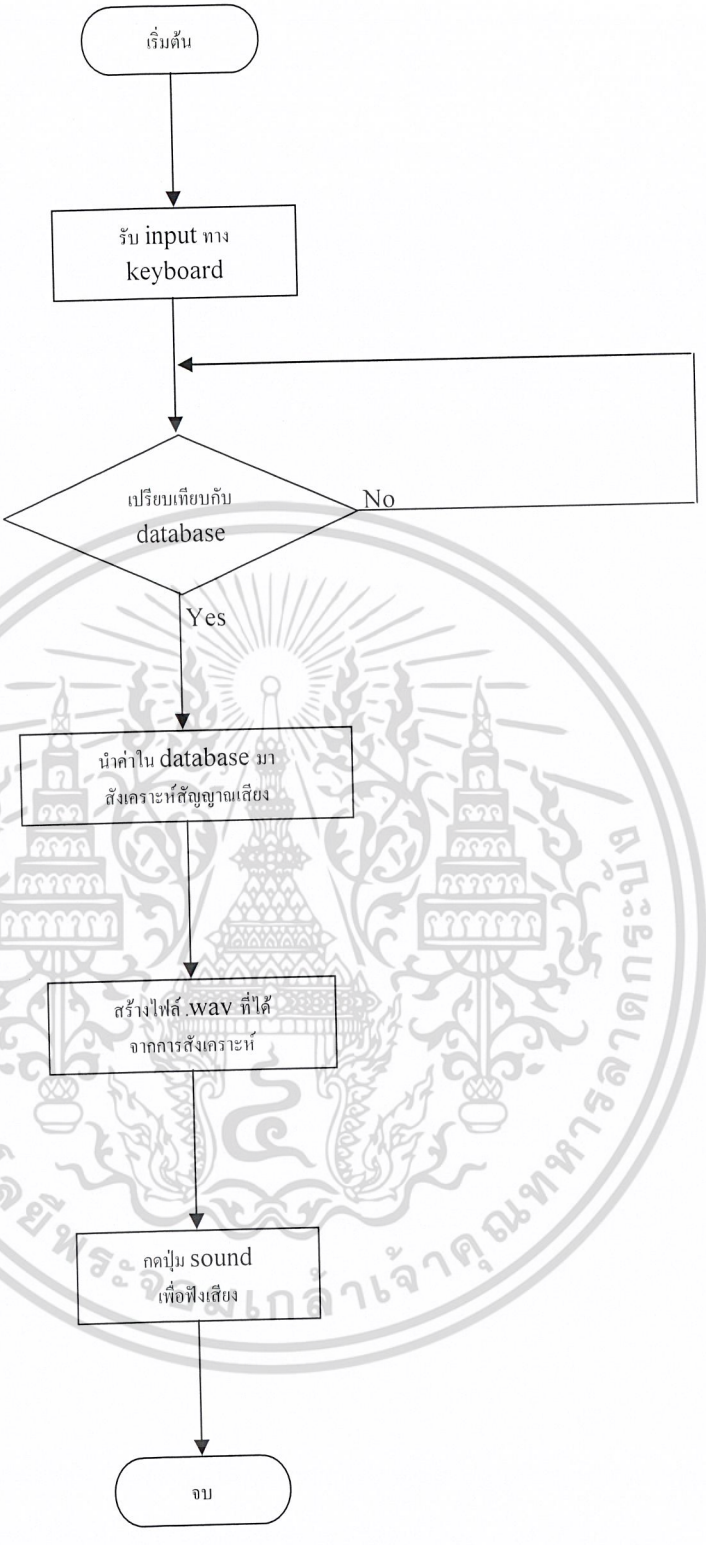
ส่วนที่ 3 การประยุกต์และนำไปใช้งาน โดยจะนำหลักการ LPC ที่คำนวณได้นี้ มาใช้ในการสังเคราะห์เสียงของตัวอักษรต่างๆ คือ จะทำการสร้างหน้าต่างขึ้นมาในการรับข้อมูลที่เป็นตัวอักษรจากผู้ใช้ จากนั้นจะนำตัวอักษรเหล่านี้เทียบกับค่า parameter ที่ได้หาไว้แล้ว และนำเสียงของตัวอักษรออกมา รวมทั้งแสดงค่า parameter ต่างๆออกมาด้วย โดยแบ่งการทำงานออกเป็นส่วนๆดังนี้

3.10 การสร้างฐานข้อมูล

เป็นการสร้างฐานข้อมูลของตัวอักษรต่างๆซึ่งจะประกอบด้วย ค่าสัมประสิทธิ์ ค่า gain และค่า pitch period ของแต่ละเฟรมข้อมูล โดยที่ค่าต่างๆเหล่านี้จะมาจากการคำนวณโดยหลักการ LPC ซึ่งจะใช้โปรแกรม Microsoft Access ในการสร้างฐานข้อมูลและเชื่อมต่อกับหน้าต่างที่สร้างขึ้นมาจากโปรแกรม Visual C++

3.11 การนำเสียงของตัวอักษรออกมา

เป็นการนำ output ที่ได้จากการสังเคราะห์สัญญาณเสียงโดยใช้หลักการของ lpc มาเขียนให้เป็นไฟล์ .wav และเล่นเป็นเสียงออกมา รวมทั้งสร้างหน้าต่างเพื่อแสดงค่าพารามิเตอร์ต่างๆของการคำนวณออกมา



รูปที่ 3.15 แผนภาพการทำงานในส่วนการประยุกต์และนำไปใช้งาน

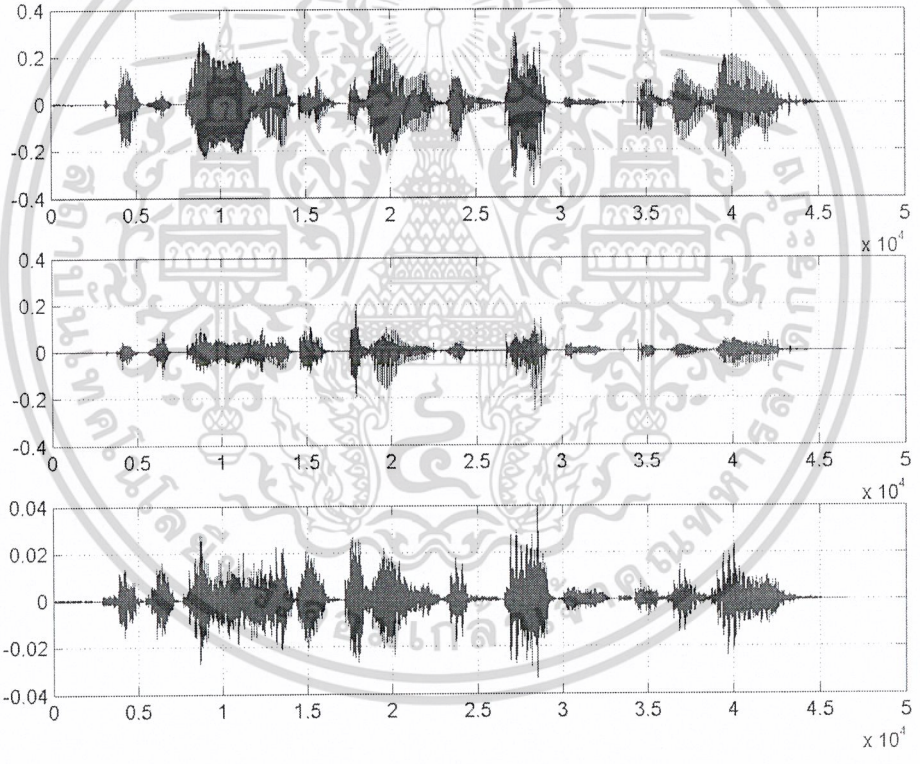
บทที่ 4

การทดลองและผลการทดลอง

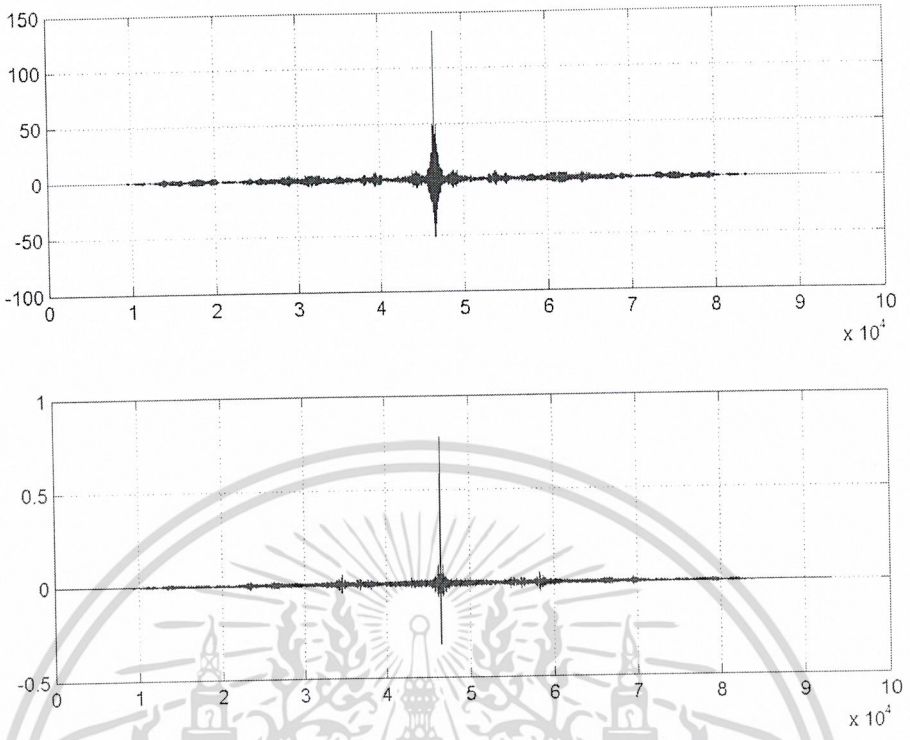
ผลการทดลองส่วนที่ 1

เป็นการทดลองประมวลผลการสังเคราะห์สัญญาณเสียง โดยใช้หลักการ LPC (Linear Predictive Coding) ด้วยโปรแกรม matlab ในที่นี้เราใช้สัญญาณ input 4 ตัว คือ s1ofwb, s2ofwb, s2omwb, w1 โดยหลักการนี้เราจะสามารถหาค่าสัมประสิทธิ์ ค่าGain และ ค่า Pitch ได้

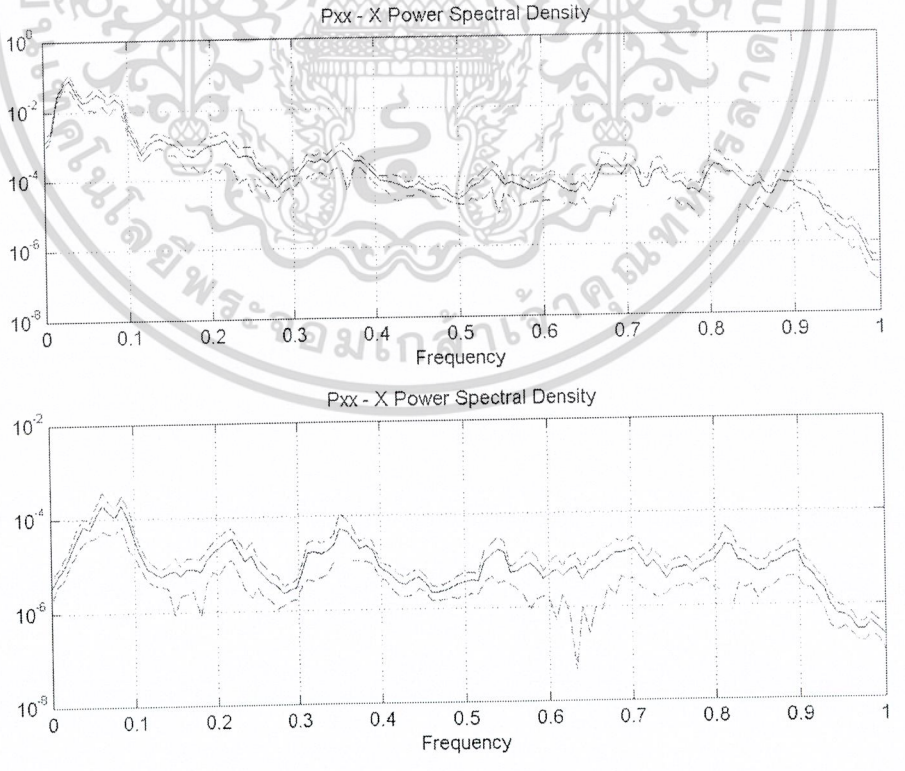
ผลการทดลองเป็นการเปรียบเทียบระหว่างสัญญาณ input และสัญญาณ output โดยเราจะทำการแสดงผล 3 อย่าง คือ กราฟแสดงสัญญาณ,กราฟแสดงค่าการหา autocorrelation และกราฟแสดงเสปกตรัมของสัญญาณ รวมไปถึงแสดงตารางค่าสัมประสิทธิ์ ค่า Gain และค่า Pitch ที่หาได้ แต่ในที่นี้เราจะนำเสนอค่า parameter ต่างๆที่เราหามาได้เพียง 2-3 เฟรมเท่านั้น



รูปที่ 4.1 กราฟเปรียบเทียบระหว่างสัญญาณ input (s1ofwb) (กราฟที่ 1) เทียบกับสัญญาณ output (กราฟที่ 3) ซึ่งได้จากการสังเคราะห์สัญญาณเสียงโดยใช้หลักการของ LPC โดยใช้โปรแกรม matlab ในการประมวลผล ส่วนกราฟที่ 2 จะแสดงสัญญาณ input เมื่อผ่านการทำ pre-emphasis



รูปที่ 4.2 กราฟผลการทำ **Autocorrelation** ระหว่างสัญญาณ **input**(บน) เทียบกับสัญญาณ **output**(ล่าง)



รูปที่ 4.3 กราฟผลการหา **spectrum** ของสัญญาณ **input**(บน) เทียบกับสัญญาณ **output**(ล่าง)

ตารางที่ 4.1 ค่าสัมประสิทธิ์ ที่วิเคราะห์ได้จากโปรแกรม Matlabในที่นี้เรายกมาแค่สาม Frame (order 16)

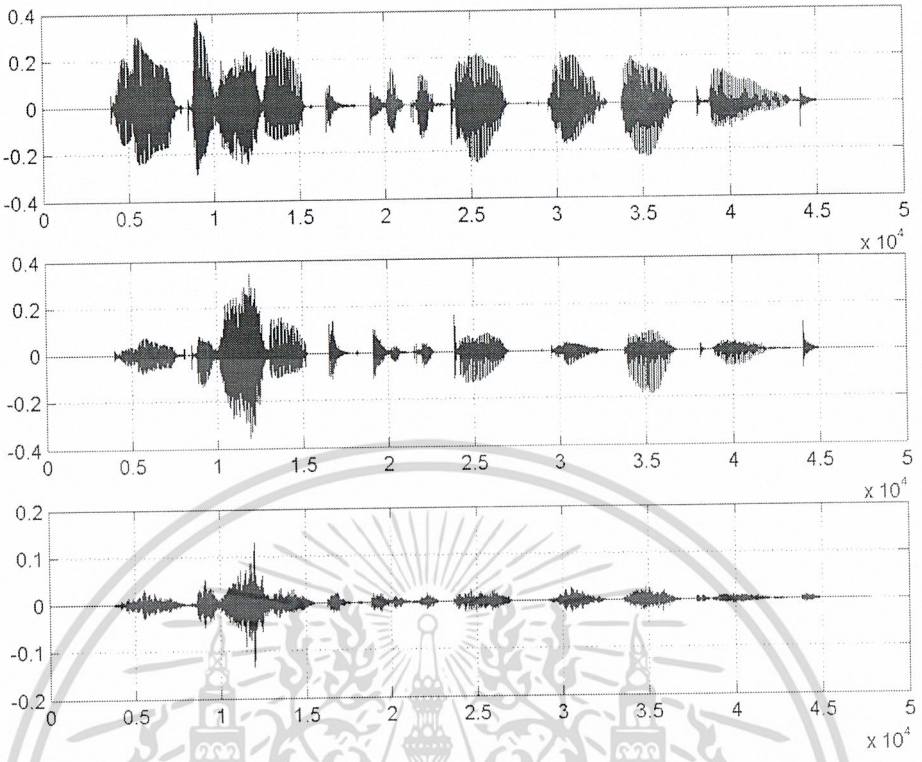
index	frame1	frame2	frame3
1	-0.70965	-0.71539	-0.5906
2	1.0249	1.052	0.86333
3	-0.19225	-0.22529	-0.089749
4	0.31343	0.43511	0.2531
5	-0.14036	-0.28226	-0.19989
6	0.27206	0.43044	0.20755
7	-0.12004	-0.30807	-0.22329
8	0.17231	0.36871	0.19379
9	-0.013798	-0.20824	-0.23255
10	0.13236	0.26597	0.24677
11	-0.0057057	-0.073668	-0.19318
12	0.18261	0.20845	0.1896
13	-0.10298	-0.1294	-0.24402
14	0.28811	0.2024	0.23511
15	-0.12379	-0.1092	-0.11098
16	0.12808	0.096002	0.033995

ตารางที่ 4.2 ค่า Gain ที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่สาม frame

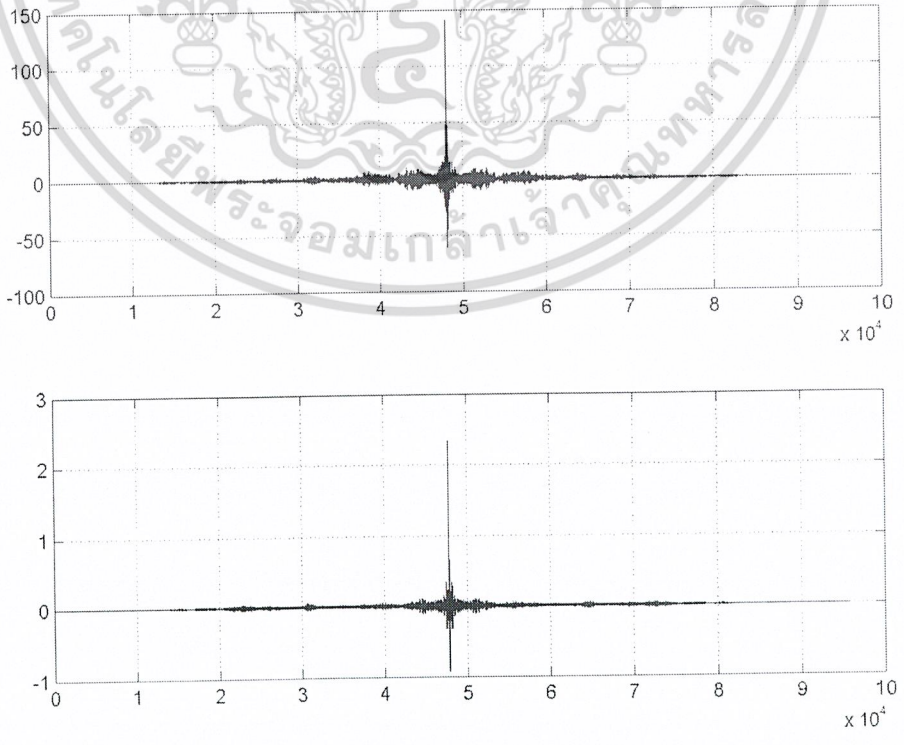
frame	1	2	3
gain	0.0078736	0.008069	0.0074867

ตารางที่ 4.3 ค่า Pitch period ที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่ 10 frame

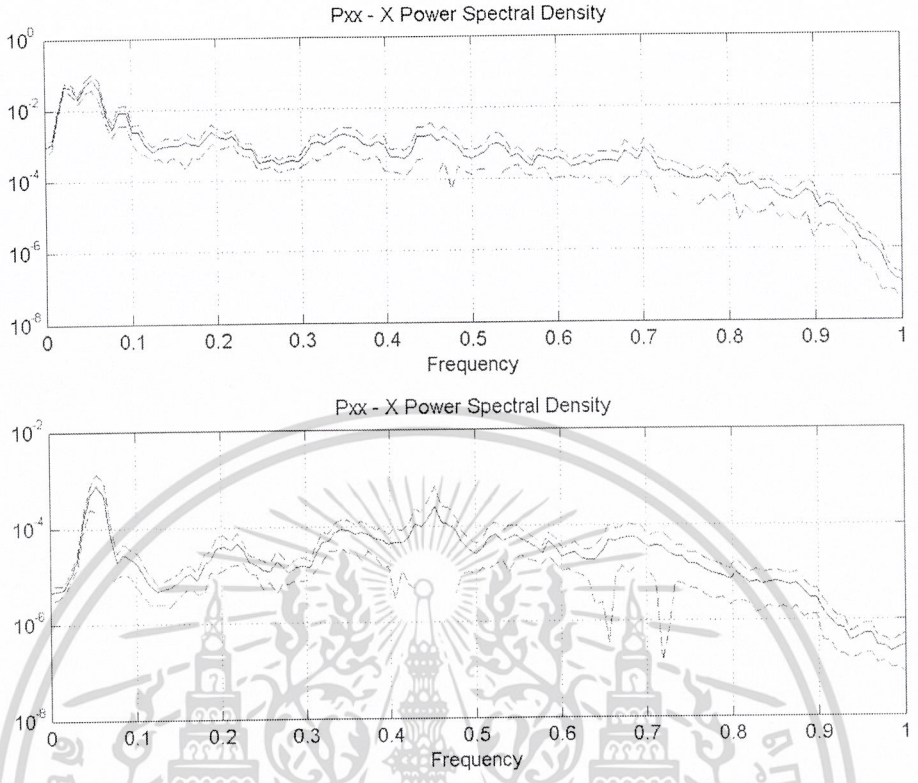
frame	1	2	3	4	5	6	7	8	9	10
pitch	1	1	1	1	1	1	1	3	5	1



รูปที่ 4.4 กราฟเปรียบเทียบระหว่างสัญญาณ input (s2ofwb) (กราฟที่ 1) เทียบกับสัญญาณ output (กราฟที่ 3) ซึ่งได้จากการสังเคราะห์สัญญาณเสียงโดยใช้หลักการของ LPC โดยใช้โปรแกรม matlab ในการประมวลผล ส่วนกราฟที่ 2 จะแสดงสัญญาณ input เมื่อผ่านการทำ pre-emphasis



รูปที่ 4.5 กราฟผลการทำ Autocorrelation ระหว่างสัญญาณ input(บน) เทียบกับสัญญาณ output(ล่าง)



รูปที่ 4.6 กราฟผลการหา **spectrum** ของสัญญาณ **input**(บน) เทียบกับสัญญาณ **output**(ล่าง)

ตารางที่ 4.4 ค่าสัมประสิทธิ์ ที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่สาม Frame (order 16)

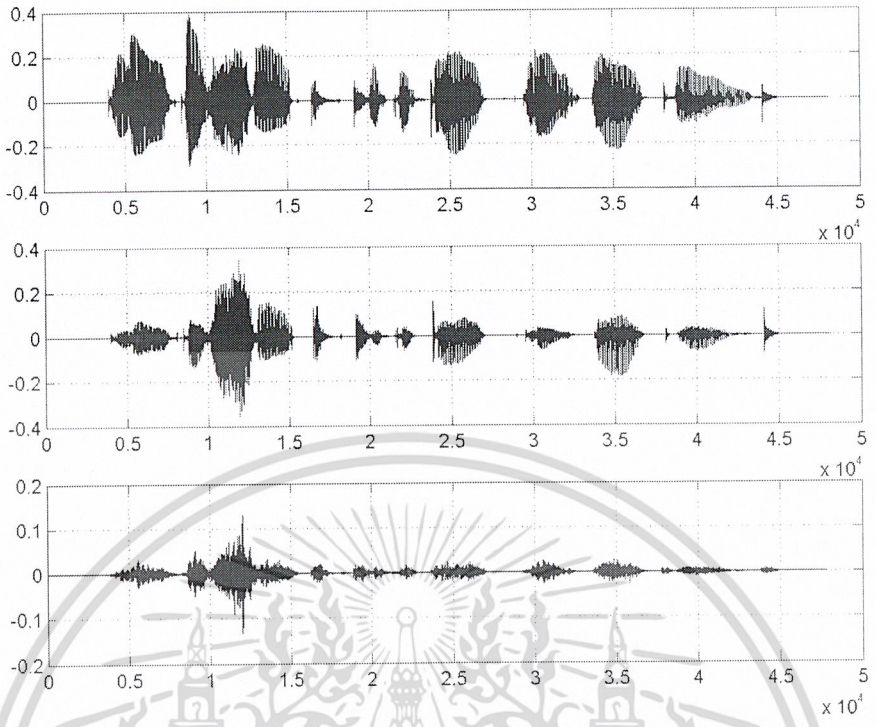
index	frame1	frame2	frame3
1	0.23013	0.2066	0.22054
2	0.29242	0.29599	0.2743
3	-0.034865	0.010451	-0.071032
4	0.16131	0.091183	0.1499
5	-0.041823	-0.037578	-0.04174
6	0.017842	0.074175	0.080943
7	-0.19124	-0.077597	-0.028195
8	-0.0020004	0.15458	0.14969
9	-0.12624	-0.074701	-0.055633
10	-0.049497	0.018957	0.025828
11	-0.13753	-0.15488	-0.13405
12	-0.034753	0.067071	0.076615
13	-0.10673	-0.14577	-0.085555
14	0.032723	0.0062442	0.00040134
15	-0.050291	-0.061549	-0.052543
16	0.070369	0.044879	0.026554

ตารางที่ 4.5 ค่า Gainที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่สาม frame

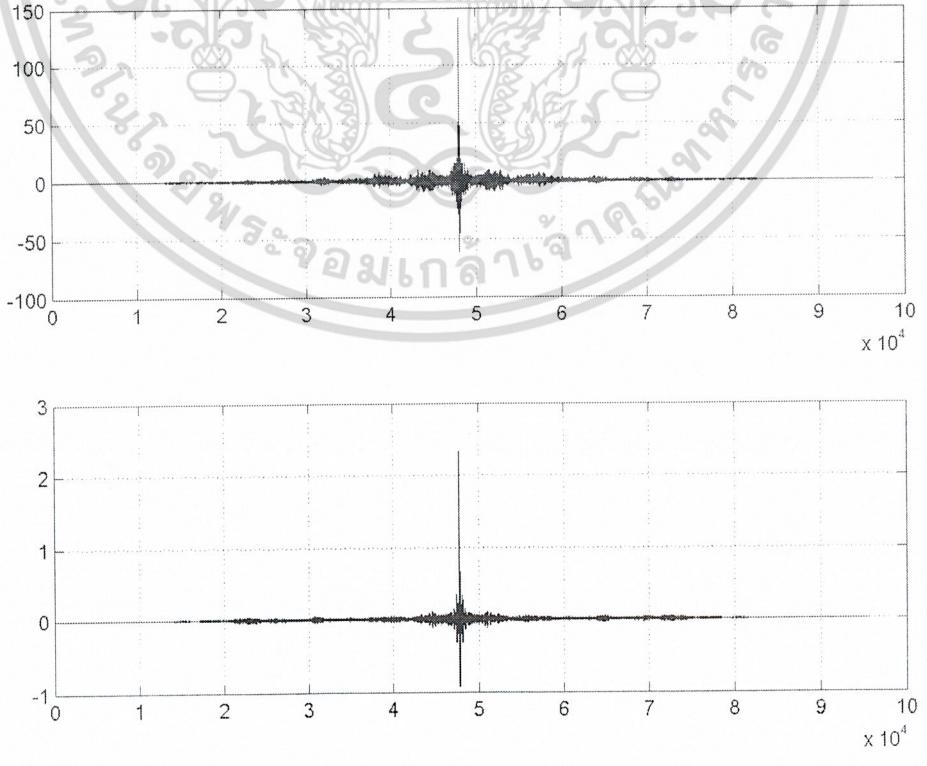
frame	1	2	3
gain	0.002057	0.0024438	0.0023218

ตารางที่ 4.6 ค่า Pitch period ที่วิเคราะห์ได้จากโปรแกรม Matlabในที่นี้เรายกมาแค่ 10 frame

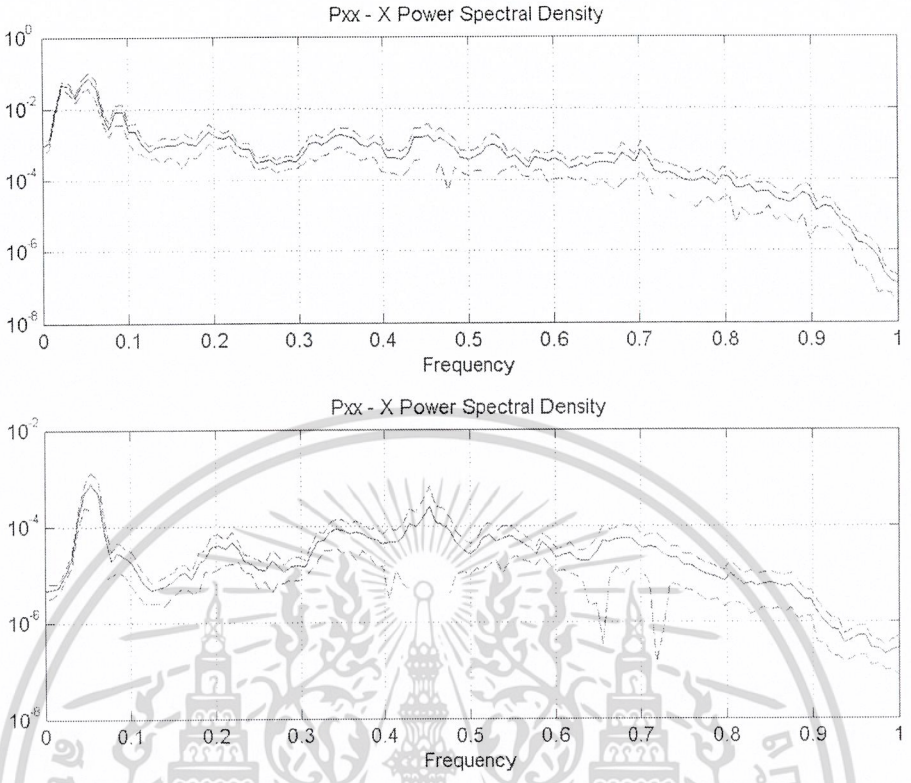
frame	1	2	3	4	5	6	7	8	9	10
pitch	5	3	3	5	3	3	13	1	1	5



รูปที่ 4.7 กราฟเปรียบเทียบระหว่างสัญญาณ input (s2omwb) (กราฟที่ 1) เทียบกับสัญญาณ output (กราฟที่ 3) ซึ่งได้จากการสังเคราะห์สัญญาณเสียงโดยใช้หลักการของ LPC โดยใช้โปรแกรม **matlab** ในการประมวลผล ส่วนกราฟที่ 2 จะแสดงสัญญาณ **input** เมื่อผ่านการทำ **pre-emphasis**



รูปที่ 4.8 กราฟผลการทำ Autocorrelation ระหว่างสัญญาณ input(บน) เทียบกับสัญญาณ output(ล่าง)



รูปที่ 4.9 กราฟผลการหา spectrum ของสัญญาณ input(บน) เทียบกับสัญญาณ output(ล่าง)

ตารางที่ 4.7 ค่าสัมประสิทธิ์ ที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่สาม Frame (order 16)

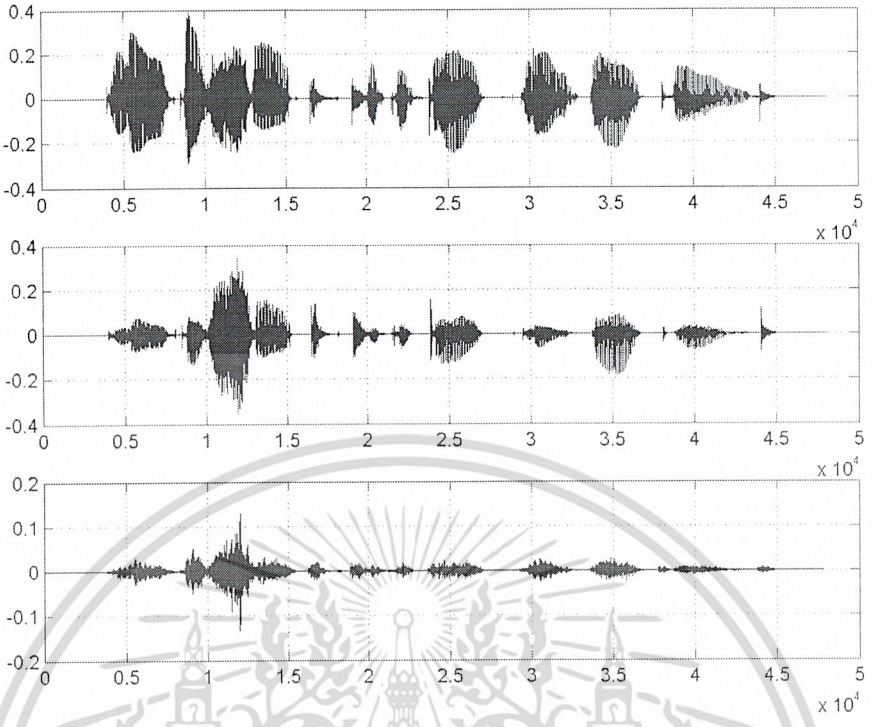
index	frame1	frame2	frame3
1	0.1034	0.074369	0.12585
2	0.18356	0.18238	0.25224
3	-0.096228	-0.072222	-0.030054
4	0.15213	0.13115	0.22237
5	-0.15268	-0.17698	-0.1595
6	-0.019147	-0.040664	0.022079
7	-0.23017	-0.25955	-0.22976
8	-0.039268	-0.015563	-0.053974
9	-0.14391	-0.15646	-0.20295
10	0.069189	0.057325	-0.080136
11	-0.12227	-0.12275	-0.1517
12	0.062608	0.077031	0.0052906
13	-0.058901	-0.040171	-0.096181
14	0.016077	-0.068236	-0.053044
15	-0.031102	0.010819	-0.035722
16	0.02986	-0.018518	0.041095

ตารางที่ 4.8 ค่า Gain ที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่สาม frame

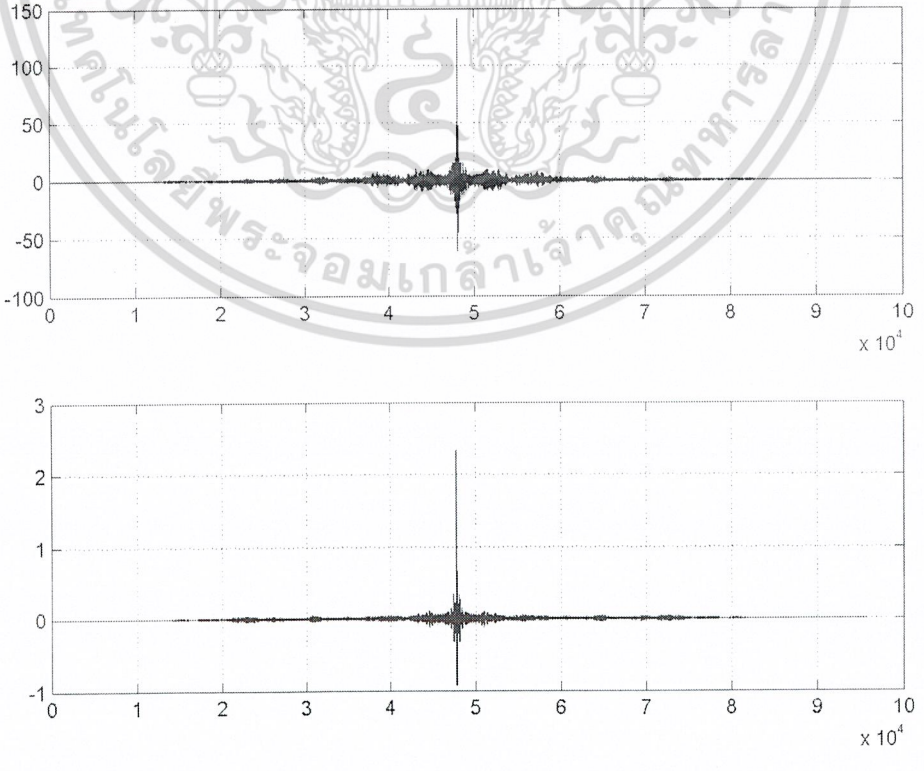
frame	1	2	3
gain	0.0028454	0.0029156	0.0030887

ตารางที่ 4.9 ค่า Pitch period ที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่ 10 frame

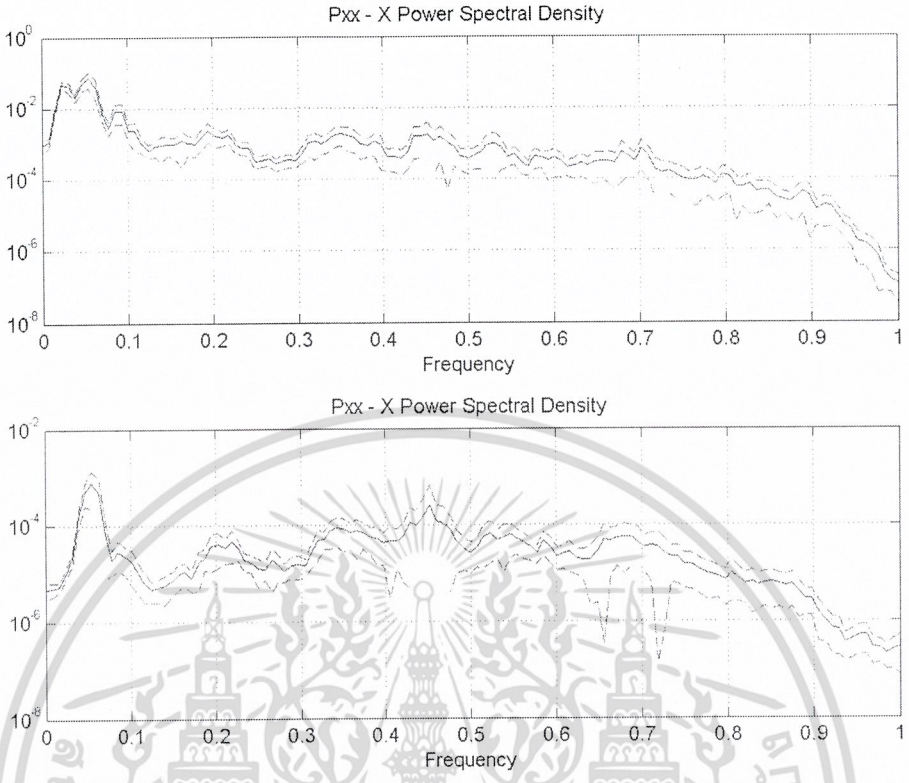
frame	1	2	3	4	5	6	7	8	9	10
pitch	3	5	5	5	3	5	5	5	3	5



รูปที่ 4.10 กราฟเปรียบเทียบระหว่างสัญญาณ input (w1) (กราฟที่ 1) เทียบกับสัญญาณ output (กราฟที่ 3) ซึ่งได้จากการสังเคราะห์สัญญาณเสียงโดยใช้หลักการของ LPC โดยใช้โปรแกรม matlab ในการประมวลผล ส่วนกราฟที่ 2 จะแสดงสัญญาณ input เมื่อผ่านการทำ pre-emphasis



รูปที่ 4.11 กราฟผลการทำ Autocorrelation ระหว่างสัญญาณ input(บน) เทียบกับสัญญาณ output(ล่าง)



รูปที่ 4.12 กราฟผลการหา spectrum ของสัญญาณ input(บน) เทียบกับสัญญาณ output(ล่าง)

ตารางที่ 4.10 ค่าสัมประสิทธิ์ ที่วิเคราะห์ได้จากโปรแกรม Matlabในที่นี้เรายกมาแค่สาม Frame (order 22)

index	frame1	frame2	frame3
1	-1.4295	-1.7943	-1.3805
2	0.4444	1.0952	0.48779
3	0.2674	-0.18153	0.064077
4	-0.10602	0.036476	-0.022045
5	0.10737	-0.098837	0.061824
6	0.030996	0.26653	0.038305
7	-0.14247	-0.25232	-0.020503
8	0.040739	0.070831	-0.065964
9	0.175	0.02598	0.11702
10	0.054094	0.28776	0.074006
11	-0.19345	-0.4558	-0.076599
12	0.036745	0.20536	-0.0049961
13	0.2761	0.21631	0.1563
14	-0.078448	-0.11347	-0.014553
15	-0.12177	-0.20317	-0.15999
16	-0.11193	0.085253	-0.04676
17	0.12465	0.00022886	0.054267
18	0.045287	-0.019977	-0.034902
19	-0.0087603	0.074047	0.05686
20	0.10827	0.16679	0.06426
21	-0.087121	-0.21794	0.058206
22	0.00049481	0.051731	-0.081745

ตารางที่ 4.11 ค่า Gainที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่สาม frame

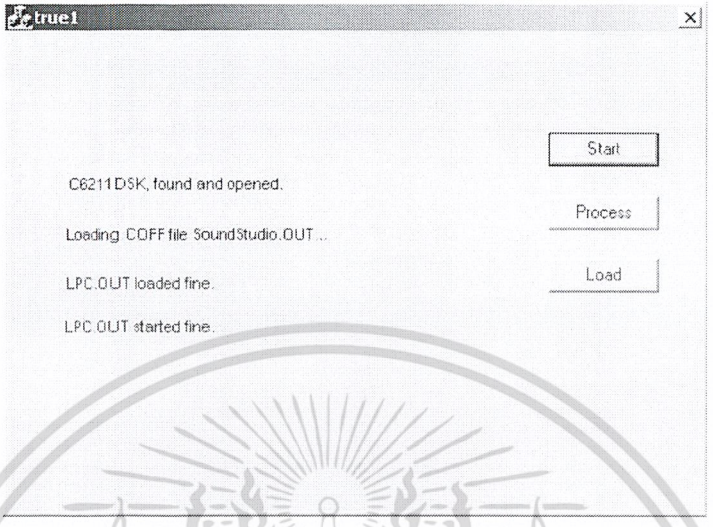
frame	1	2	3
gain	0.02627	0.025451	0.030504

ตารางที่ 4.12 ค่า Pitch period ที่วิเคราะห์ได้จากโปรแกรม Matlab ในที่นี้เรายกมาแค่ 10 frame

frame	1	2	3	4	5	6	7	8	9	10
pitch	1	1	2	2	2	3	2	2	1	2

ผลการทดลองส่วนที่ 2

เป็นผลการทดลองในส่วนของการติดต่อกับบอร์ด TMS320C6211 โดยจะสร้างหน้าต่างเพื่อควบคุมการทำงาน

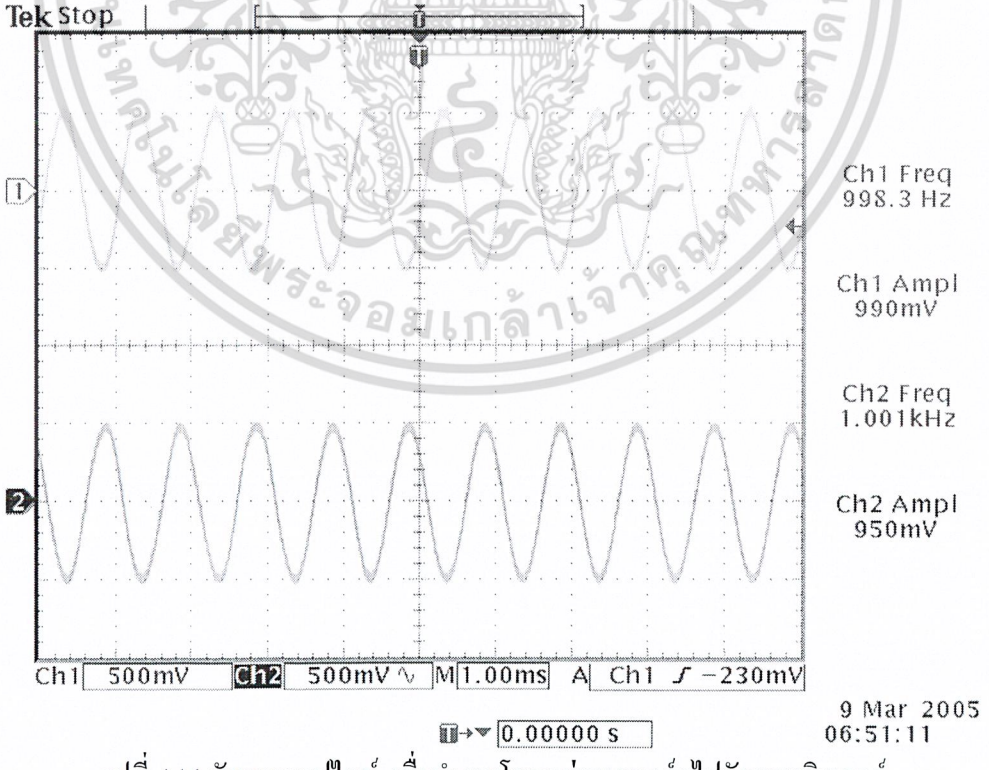


รูปที่ 4.13 หน้าต่างควบคุมการทำงาน

ซึ่งจะแบ่งการทดลองออกเป็น 3 ส่วน คือ

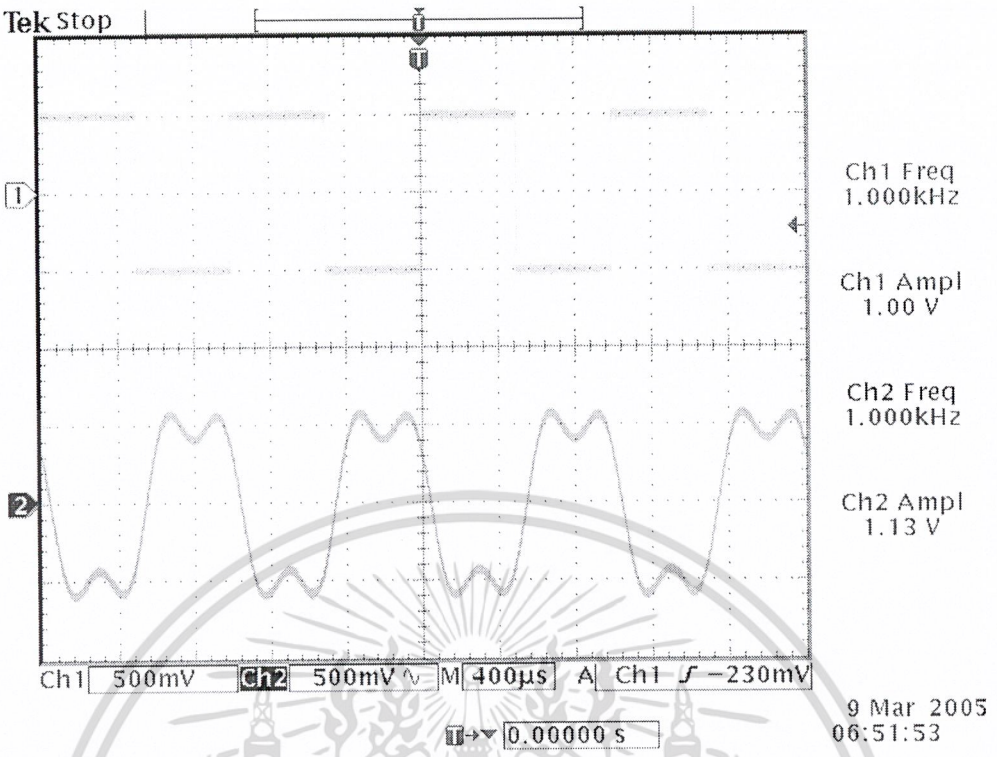
<u>ส่วนโหลดค่าจากบอร์ดไปยังคอมพิวเตอร์</u>

โดยจะทำการทดลองสัญญาณ 3 ตัว คือ สัญญาณรูปไซน์ สัญญาณรูปสี่เหลี่ยม และ สัญญาณฟันเลื่อย

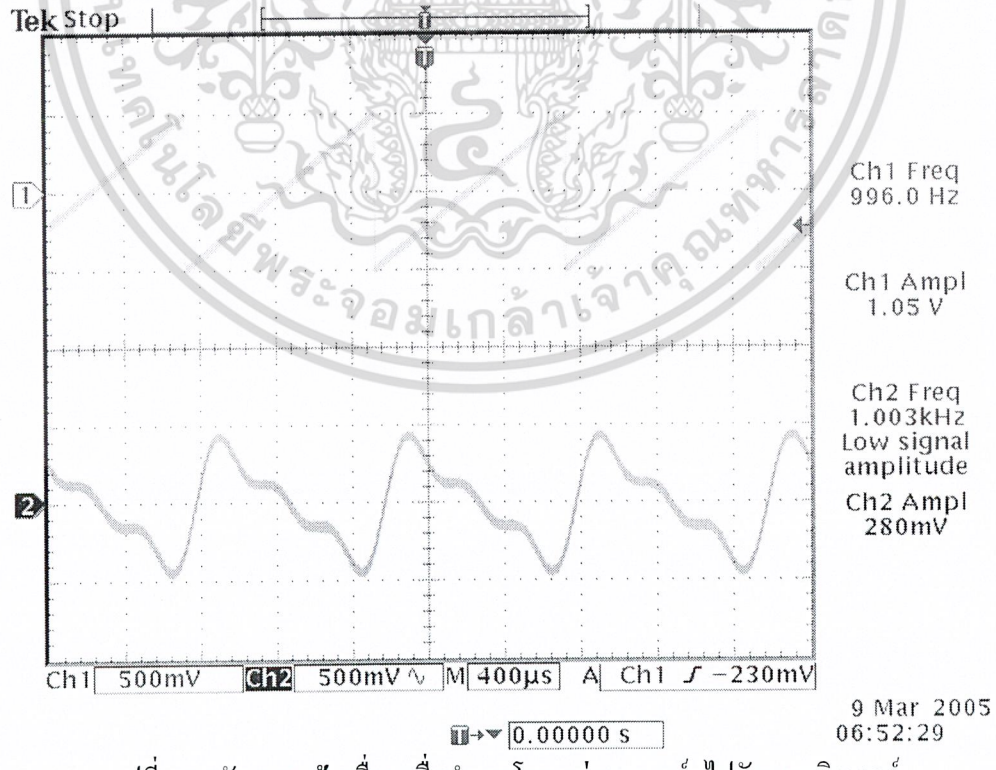


รูปที่ 4.14 สัญญาณรูปไซน์ เมื่อทำการโหลดค่าจากบอร์ดไปยังคอมพิวเตอร์

โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท



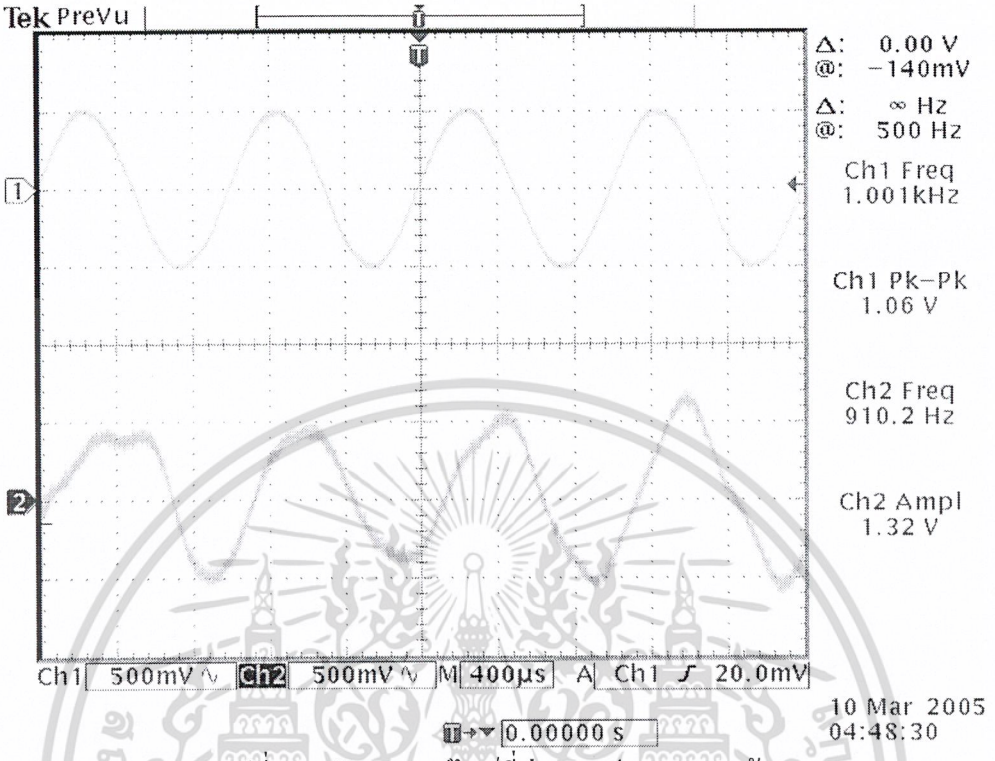
รูปที่ 4.15 สัญญาณรูปสี่เหลี่ยม เมื่อทำการโหลดค่าจากบอร์ดไปยังคอมพิวเตอร์
โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท



รูปที่ 4.16 สัญญาณฟันเลื่อย เมื่อทำการโหลดค่าจากบอร์ดไปยังคอมพิวเตอร์
โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท

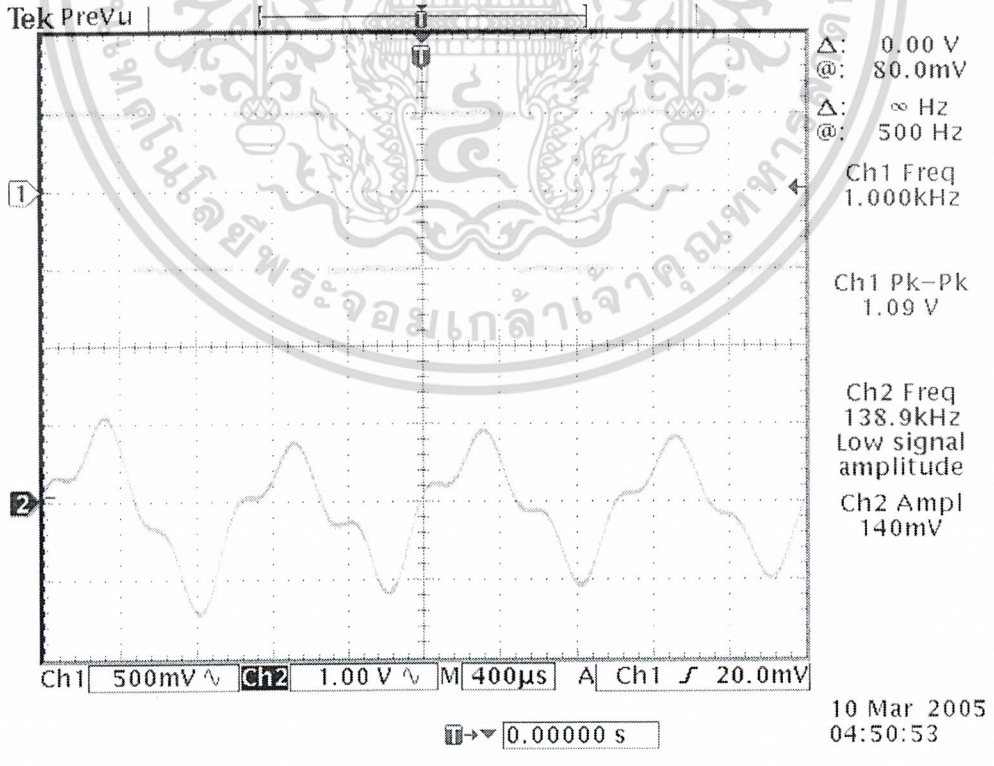
ส่วนการประมวลผล

โดยจะทำการทดลองโดยใช้สัญญาณรูปไซน์ และ สัญญาณรูปสี่เหลี่ยม



รูปที่ 4.17 สัญญาณรูปไซน์ที่ผ่านการประมวลผลแล้ว

โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท

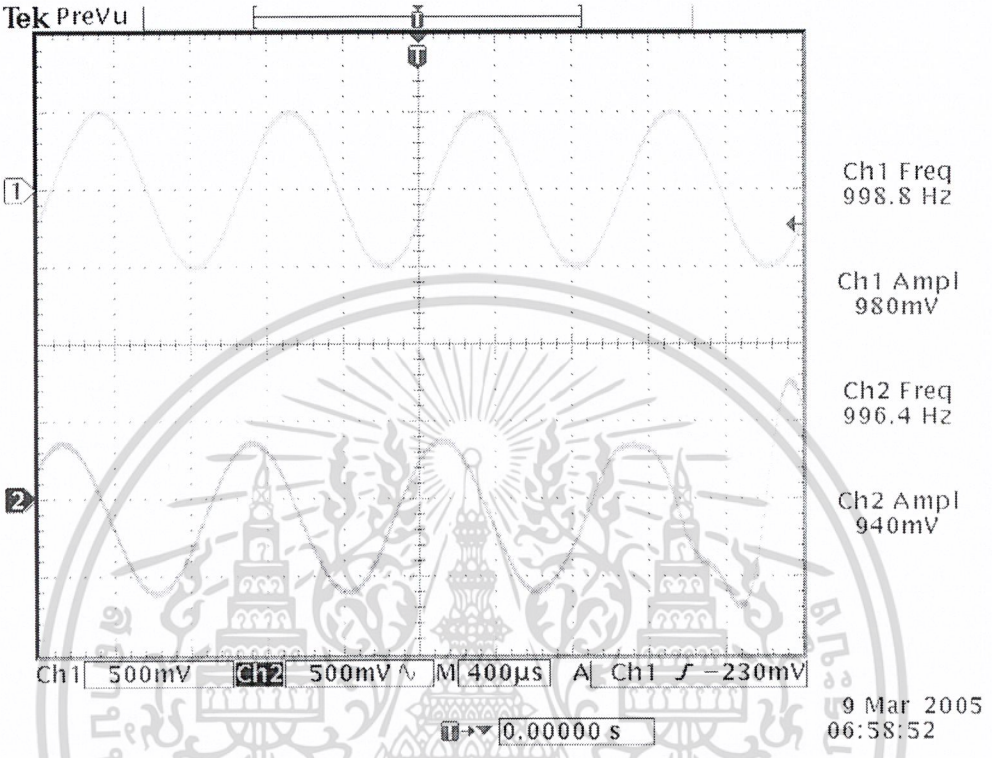


รูปที่ 4.18 สัญญาณรูปสี่เหลี่ยมที่ผ่านการประมวลผลแล้ว

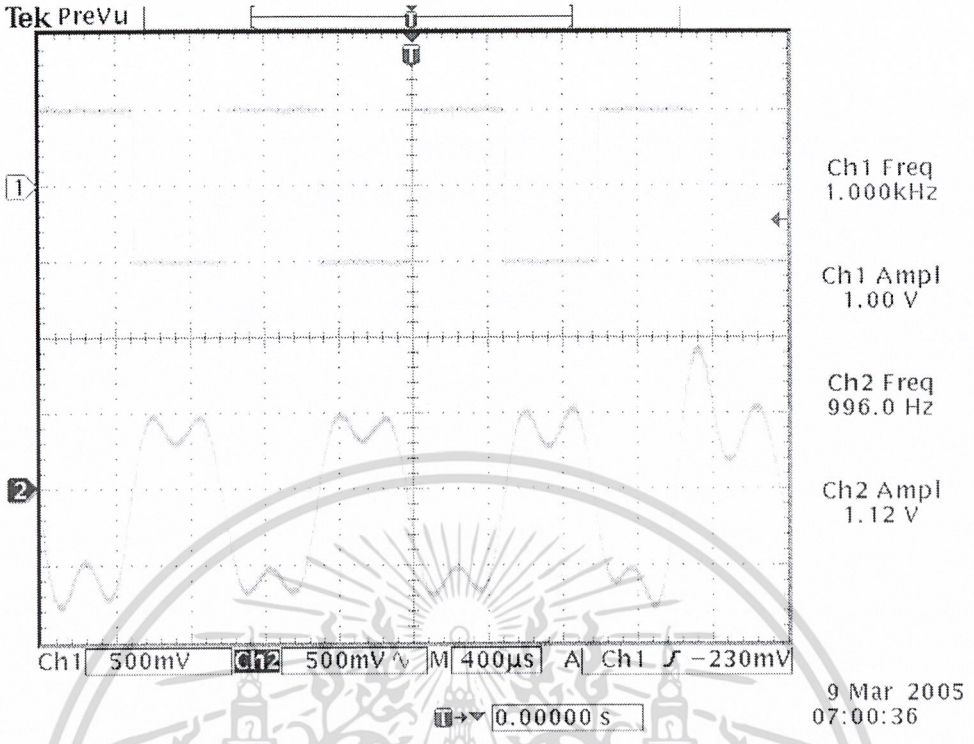
โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท

<u>ส่วนโหลดค่าจากคอมพิวเตอร์มายังบอร์ด</u>

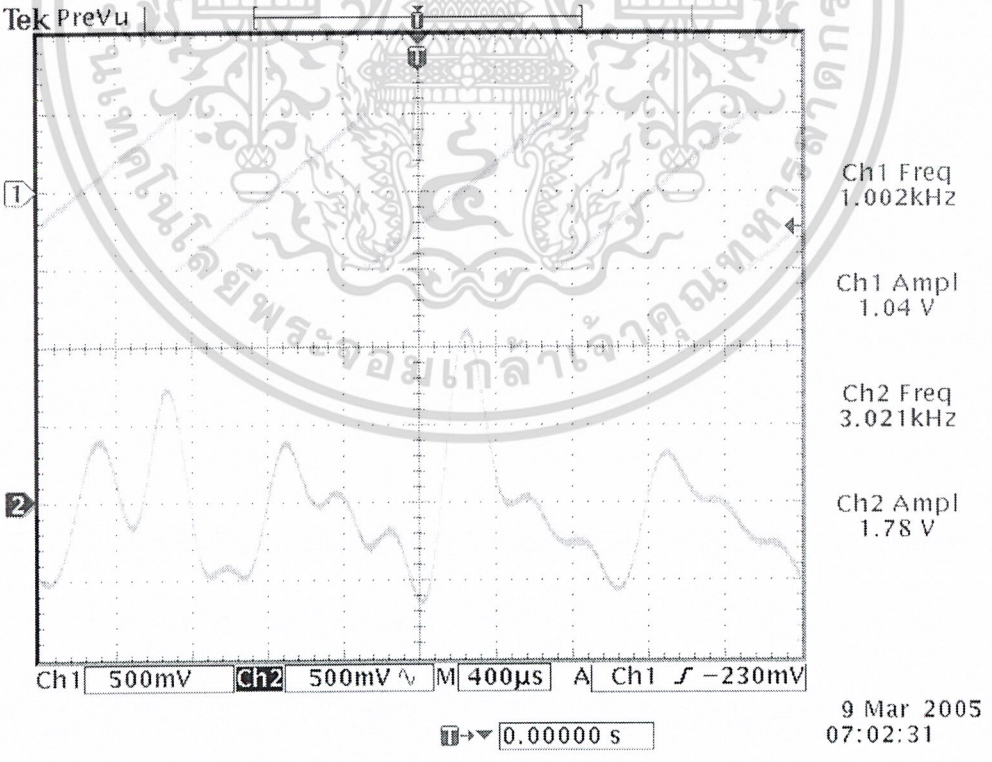
โดยจะทำการทดลองสัญญาณ 3 ตัว คือ สัญญาณรูปไซน์ สัญญาณรูปสี่เหลี่ยม และ สัญญาณฟันเลื่อย



รูปที่ 4.19 สัญญาณรูปไซน์ เมื่อทำการโหลดค่าจากคอมพิวเตอร์มายังบอร์ด
โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท



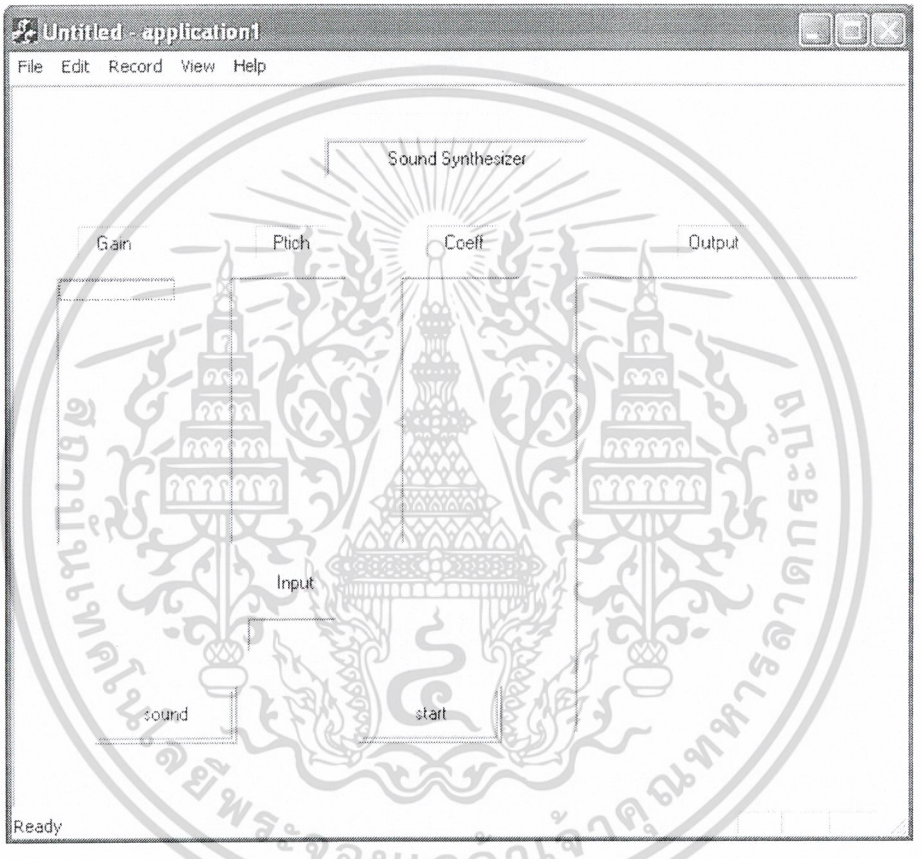
รูปที่ 4.20 สัญญาณรูปสี่เหลี่ยม เมื่อทำการโหลดค่าจากคอมพิวเตอร์มายังบอร์ด
โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท



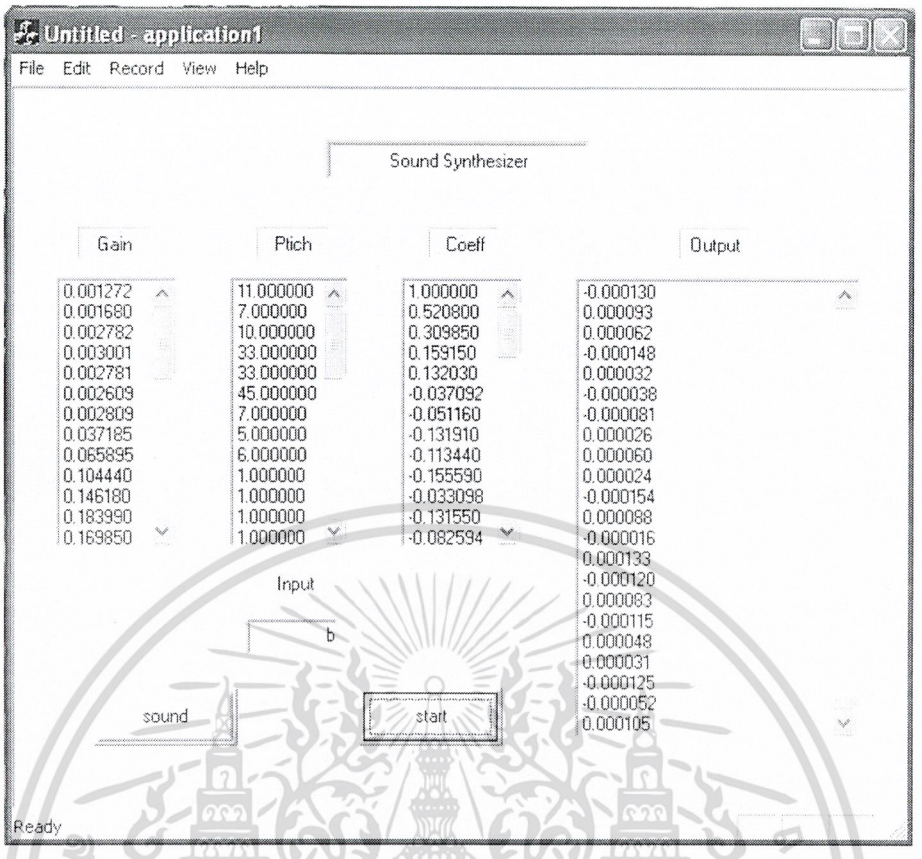
รูปที่ 4.21 สัญญาณฟันเลื่อยเมื่อทำการโหลดค่าจากคอมพิวเตอร์มายังบอร์ด
โดยรูปบนคือสัญญาณอินพุท รูปล่างคือสัญญาณเอาท์พุท

ผลการทดลองส่วนที่ 3

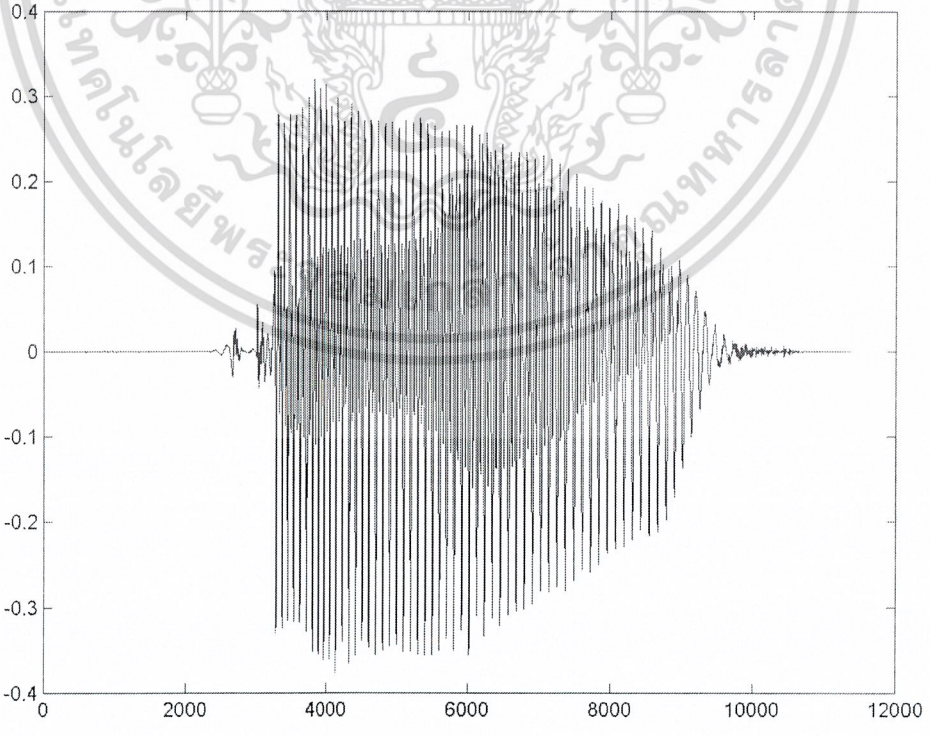
เป็นผลการทดลองในส่วนของการประยุกต์และนำไปใช้งาน ซึ่งเป็นการนำค่า parameter ต่างๆ มาสร้างฐานข้อมูล และจะนำไปเปรียบเทียบกับตัวอักษร (อินพุท) ที่รับเข้ามา เมื่อตรวจสอบว่าตรงกับค่าของตัวอักษรตัวใดแล้ว ก็จะนำค่า parameter ของตัวอักษรนั้นไปทำการสังเคราะห์เสียงของตัวอักษรตัวนั้นออกมา โดยจะทำการสร้างหน้าต่างรับตัวอักษร รวมทั้งแสดงค่า parameter ต่างๆออกมา โดยในที่นี้เราจะทำการทดลองตัวอักษร b, c, d ตามลำดับ



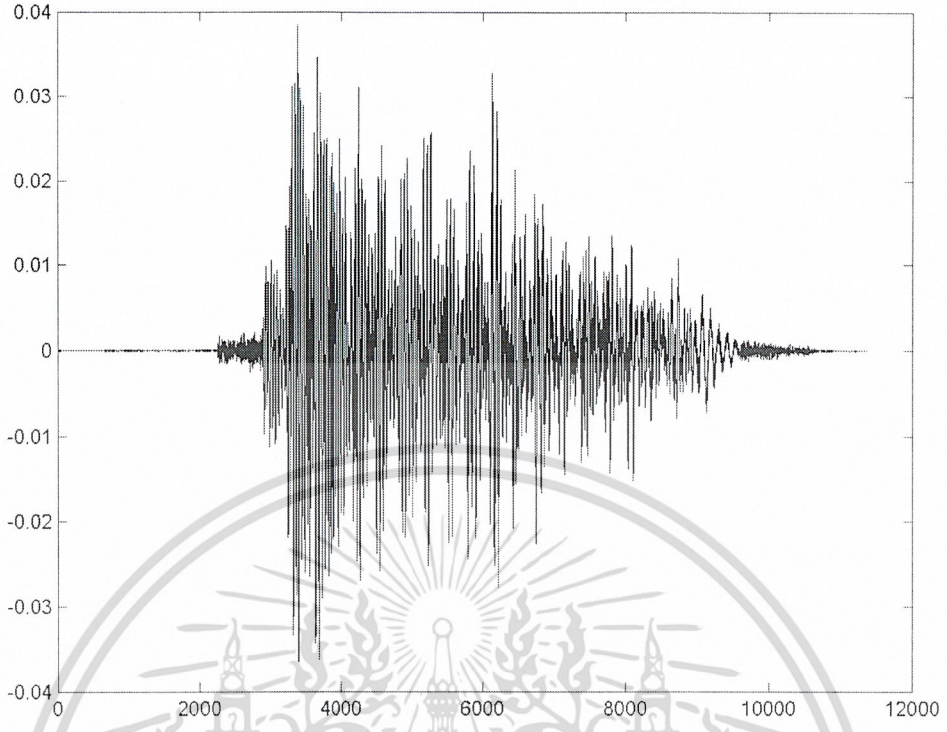
รูปที่ 4.22 หน้าต่างที่ใช้ในการรับค่าและแสดงค่า parameter ต่างๆ



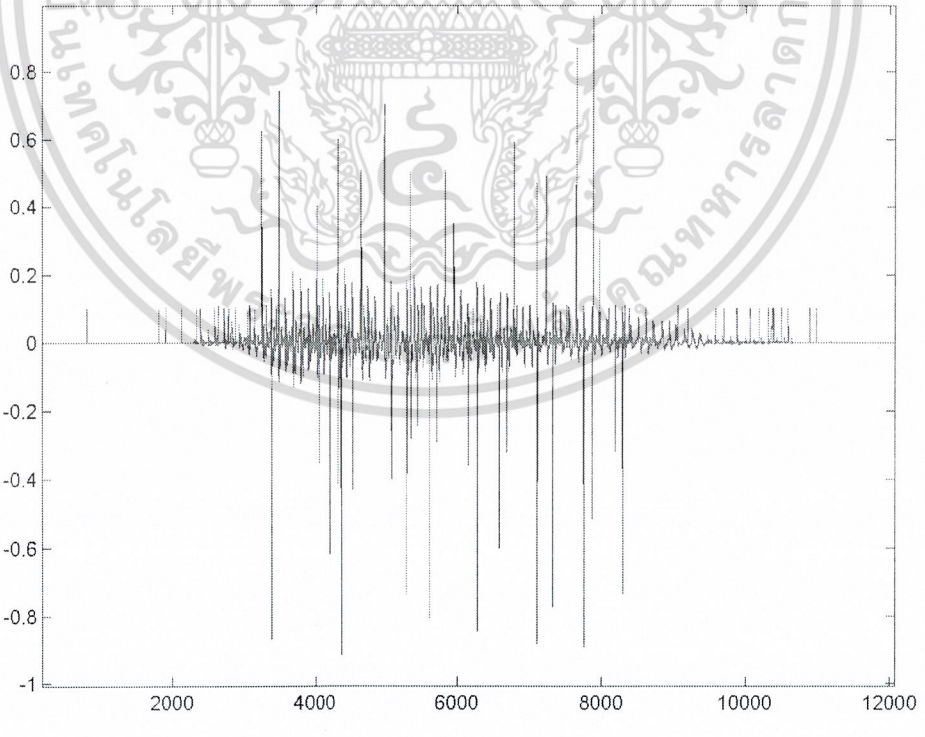
รูปที่ 4.23 ค่า parameter ของตัวอักษร b



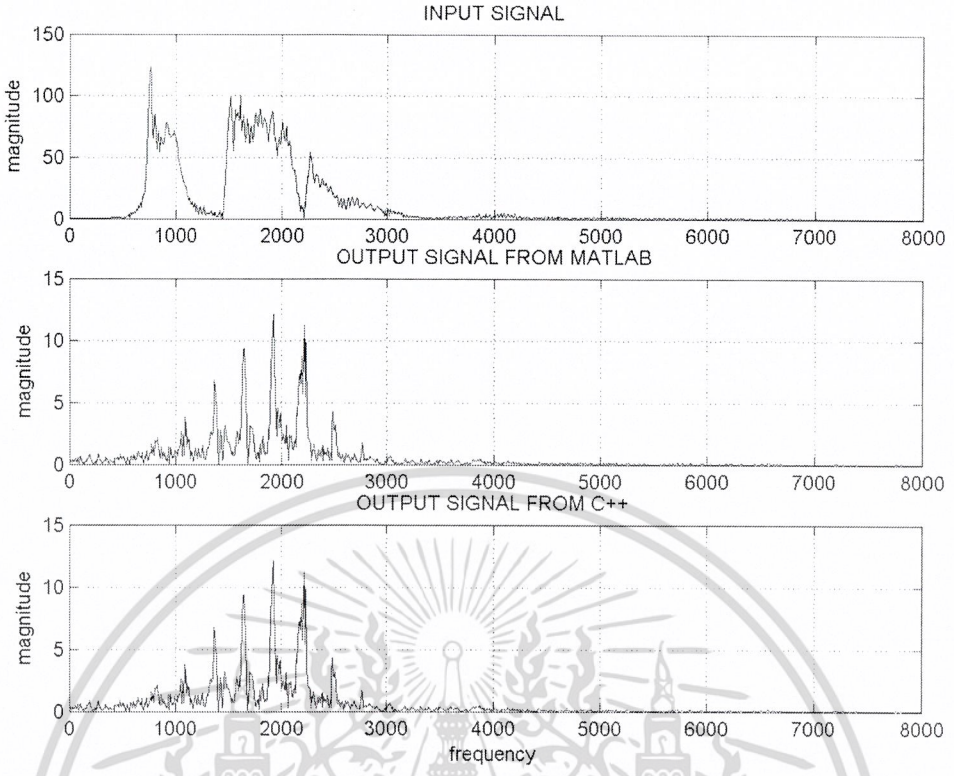
รูปที่ 4.24 กราฟสัญญาณเสียงอินพุท (ตัวอักษร b)



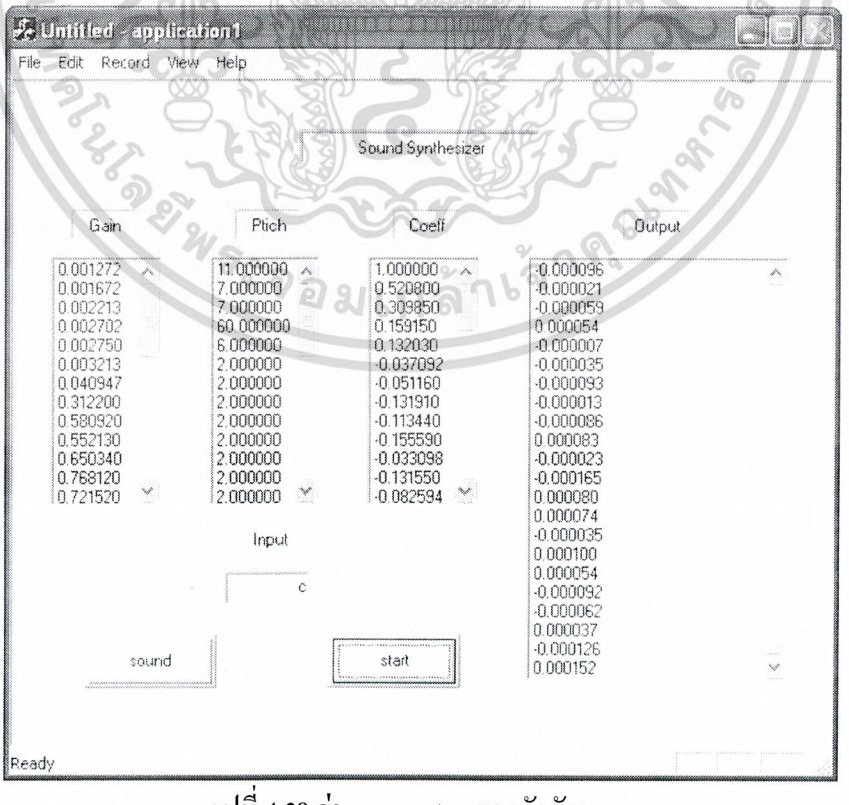
รูปที่ 4.25 กราฟสัญญาณเอาท์พุท (ตัวอักษร b) โดยใช้โปรแกรม matlab



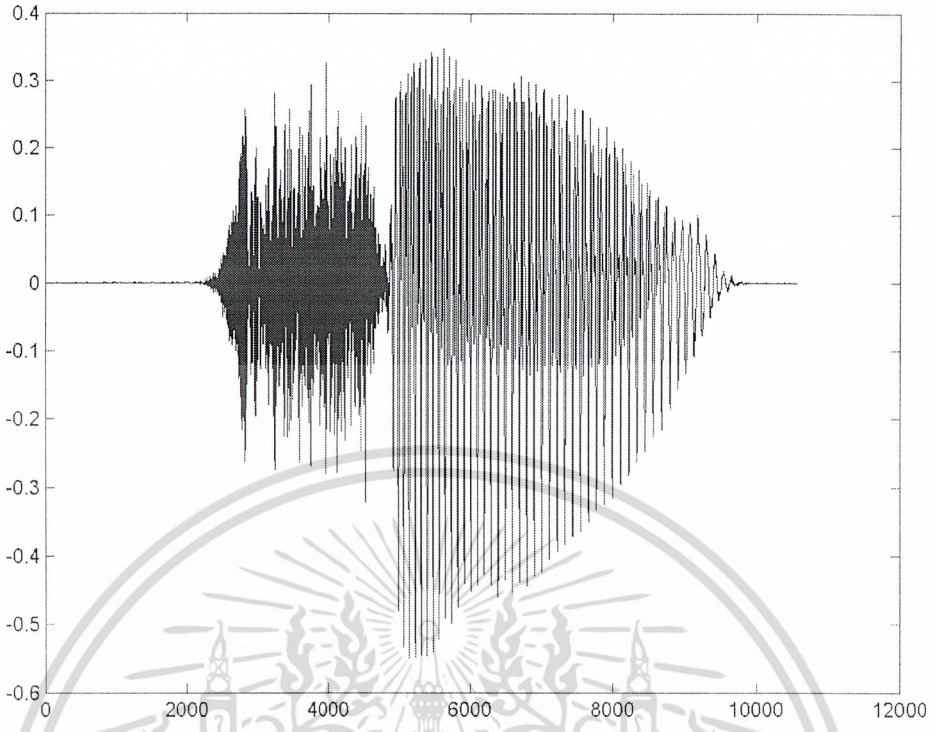
รูปที่ 4.26 กราฟสัญญาณเอาท์พุท (ตัวอักษร b) โดยใช้โปรแกรม visual c++



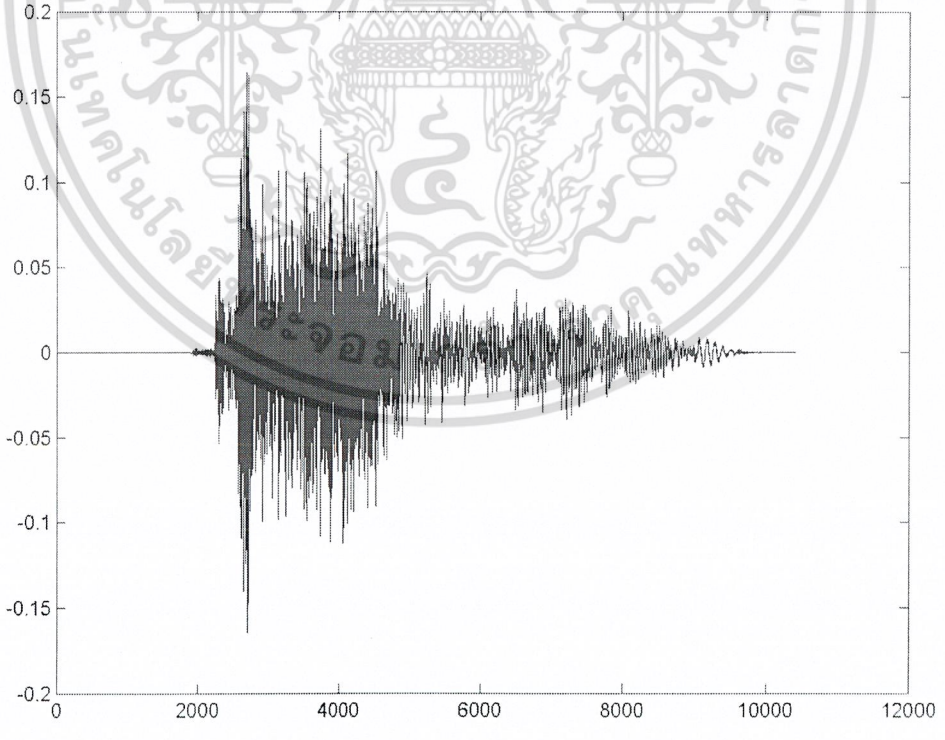
รูปที่ 4.27 กราฟสเปกตรัมจากการทำ Fast Fourier Transform ของสัญญาณอินพุท (ตัวอักษร b) เทียบกับ สัญญาณเอาท์พุทจากโปรแกรม matlab และ สัญญาณเอาท์พุทจากโปรแกรม visual c++



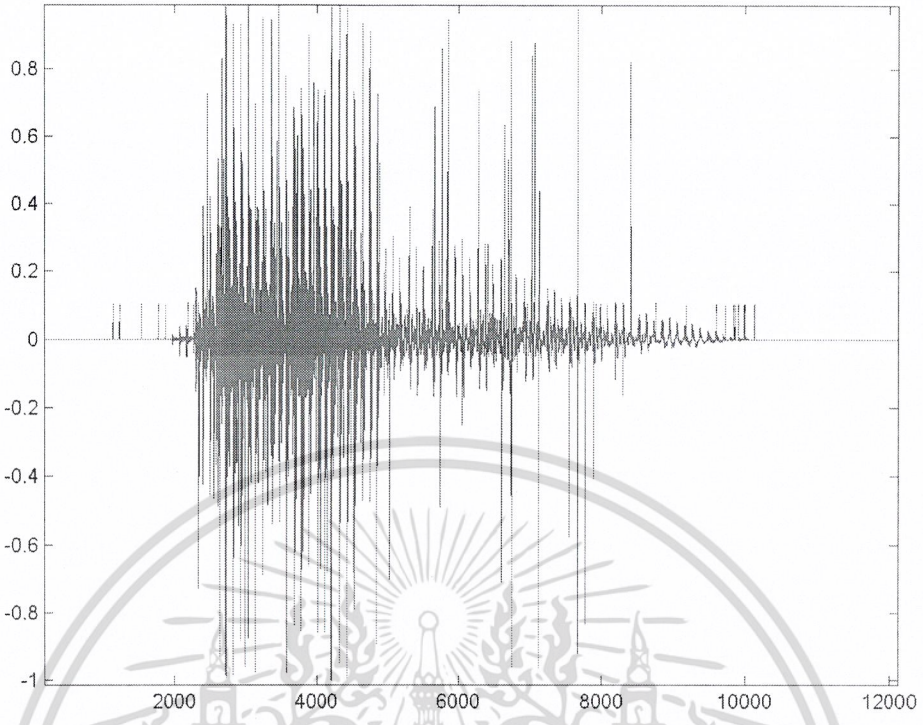
รูปที่ 4.28 ค่า parameter ของตัวอักษร c



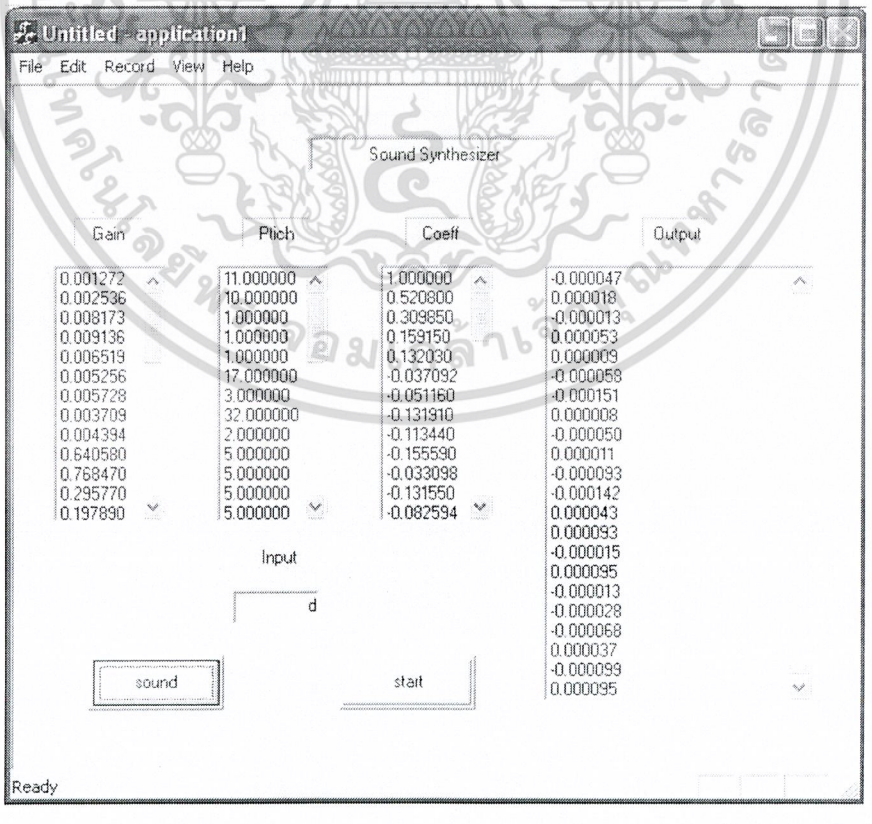
รูปที่ 4.29 กราฟสัญญาณอินพุท (ตัวอักษร c)



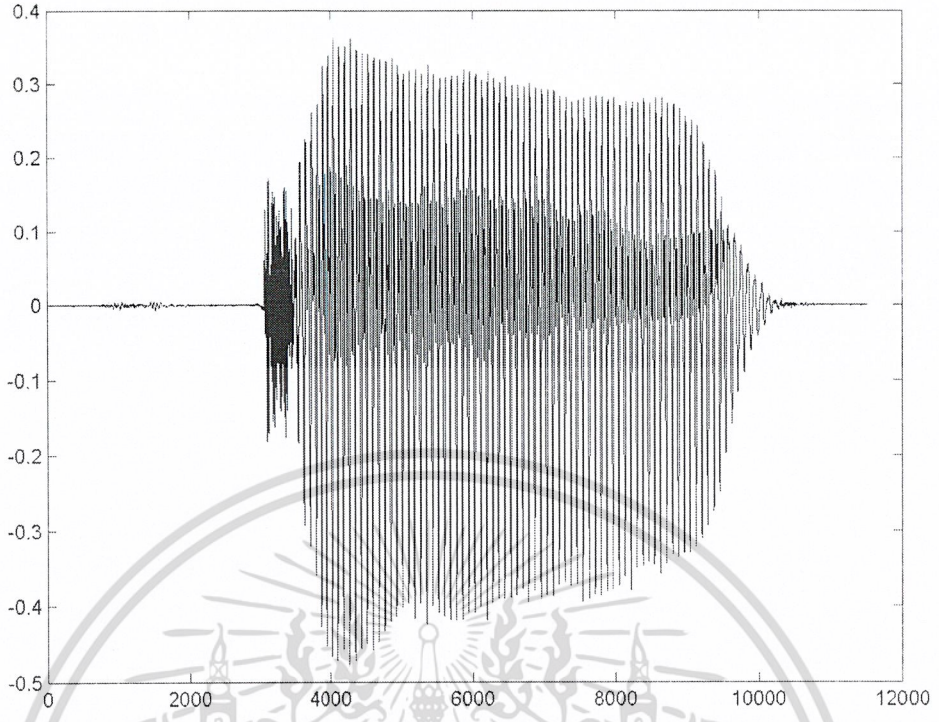
รูปที่ 4.30 กราฟสัญญาณเอาท์พุท (ตัวอักษร c) โดยใช้โปรแกรม matlab



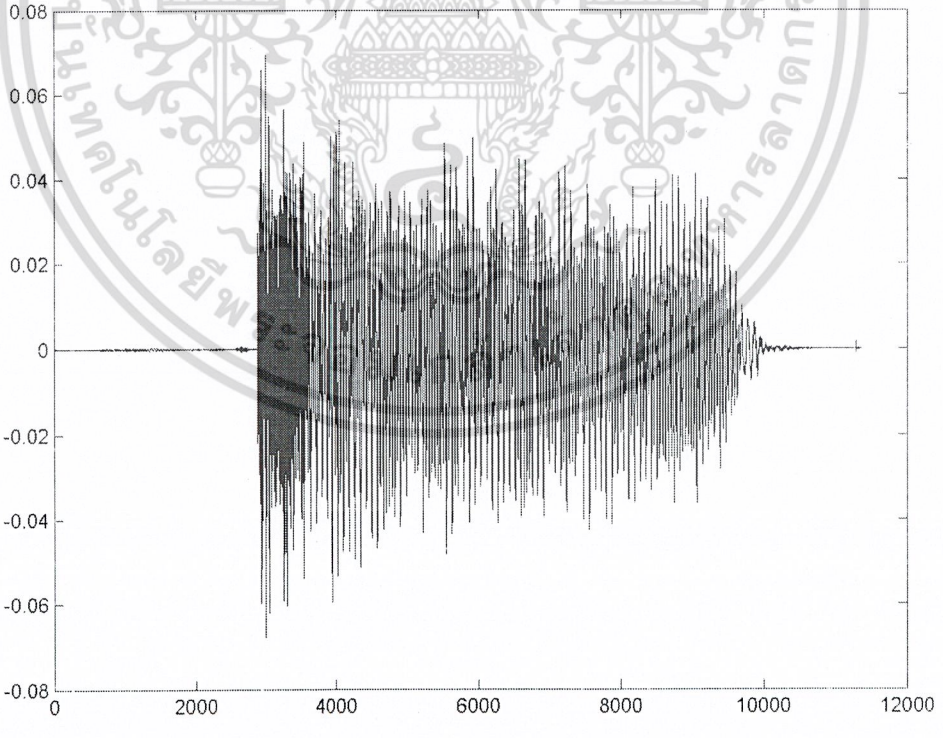
รูปที่ 4.31 กราฟสัญญาณเอาท์พุท (ตัวอักษร c) โดยใช้โปรแกรม visual c++



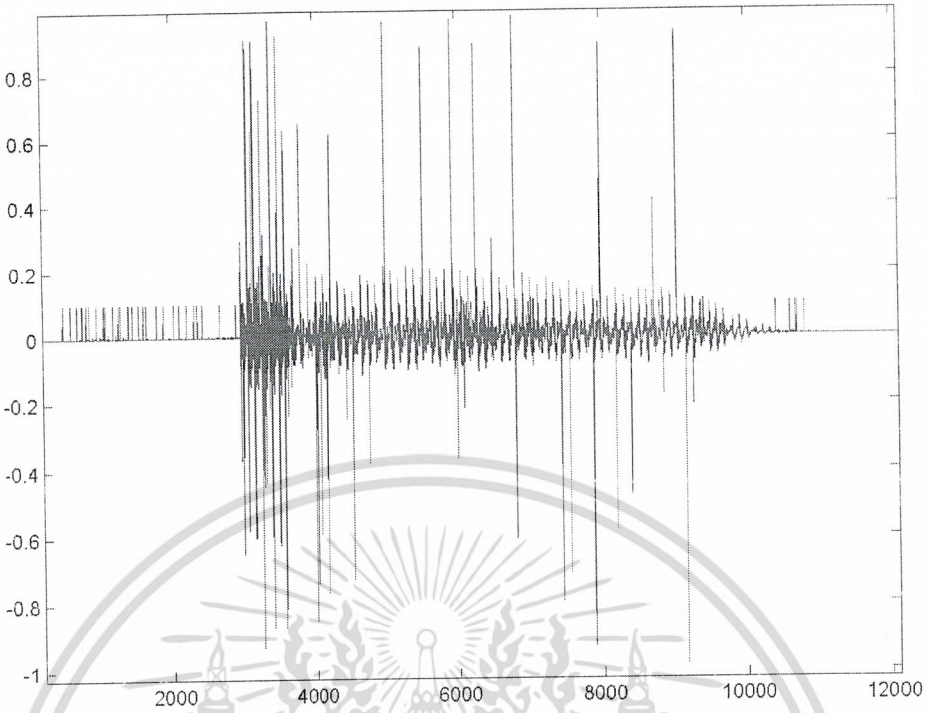
รูปที่4.32 ค่า parameter ของตัวอักษร d



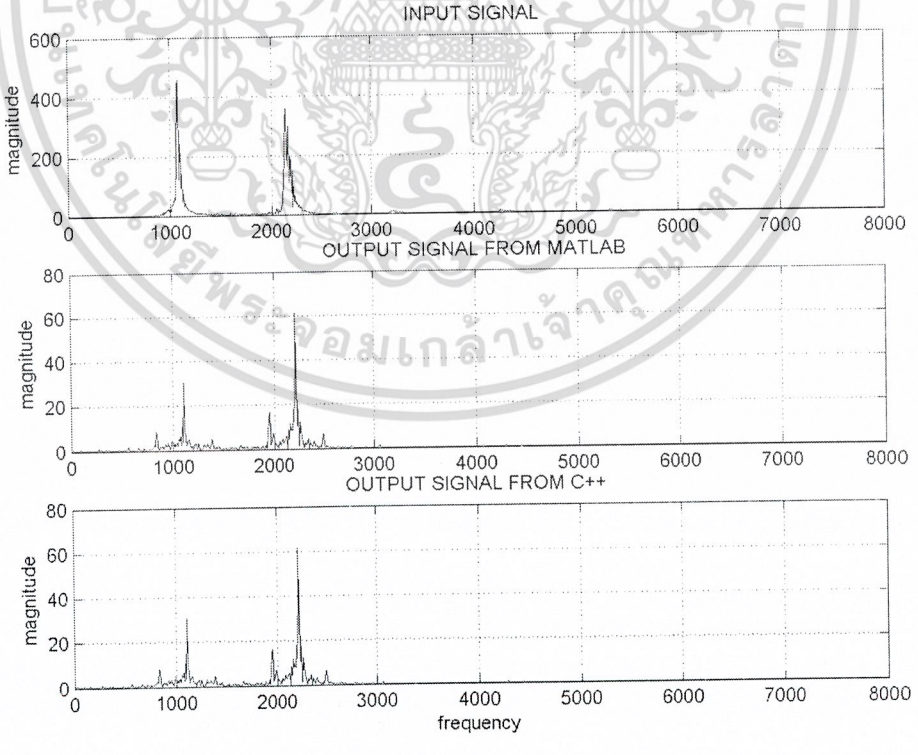
รูปที่ 4.33 กราฟสัญญาณอินพุท (ตัวอักษร d)



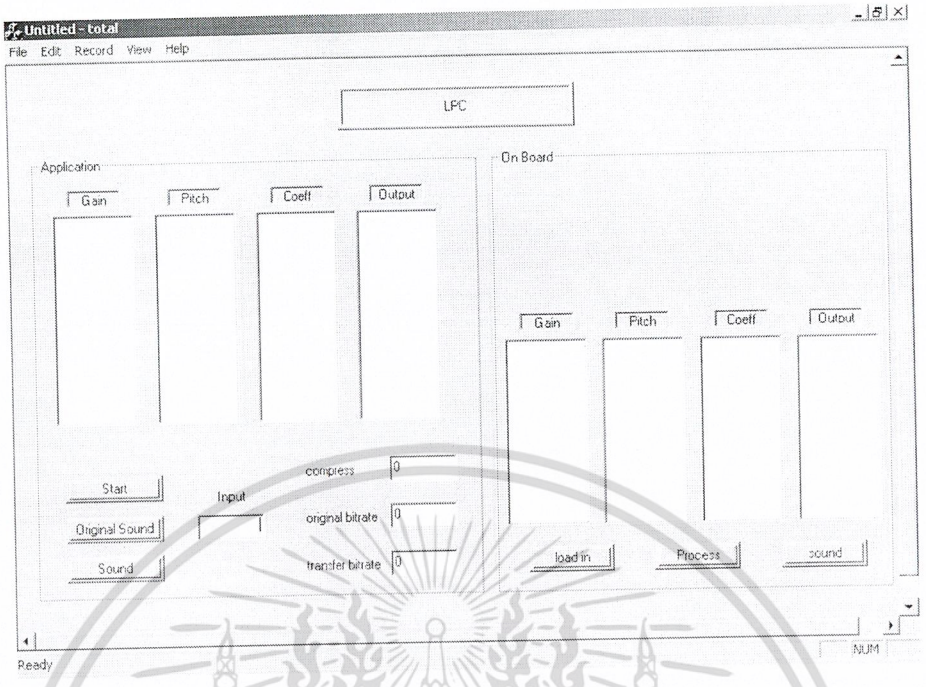
รูปที่ 4.34 กราฟสัญญาณเอาท์พุท (ตัวอักษร d) โดยใช้โปรแกรม matlab



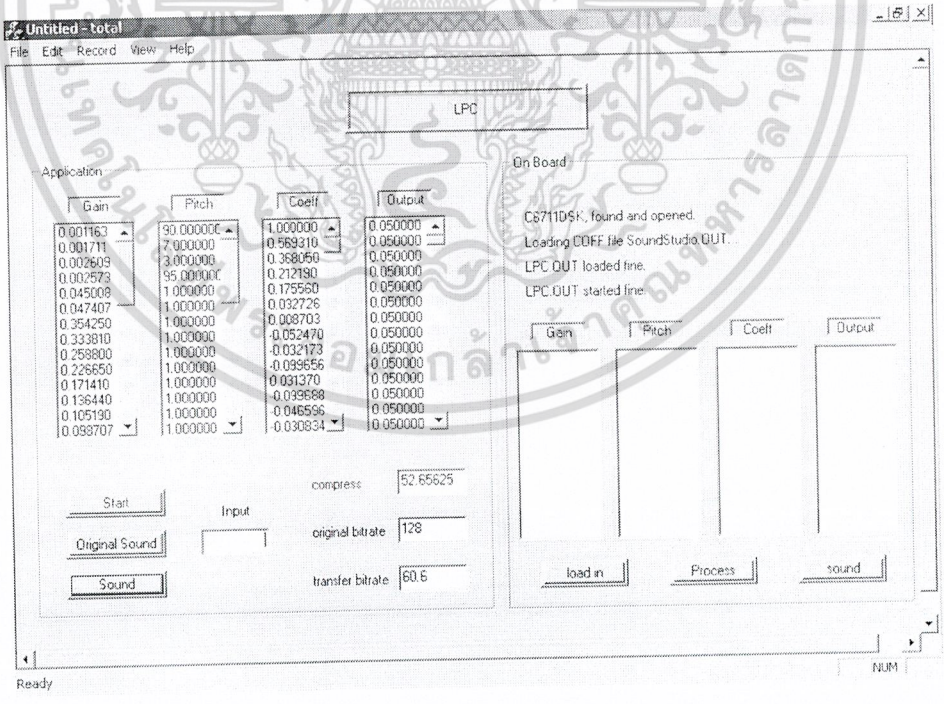
รูปที่ 4.35 กราฟสัญญาณเอาท์พุท (ตัวอักษร d) โดยใช้โปรแกรม visual c++



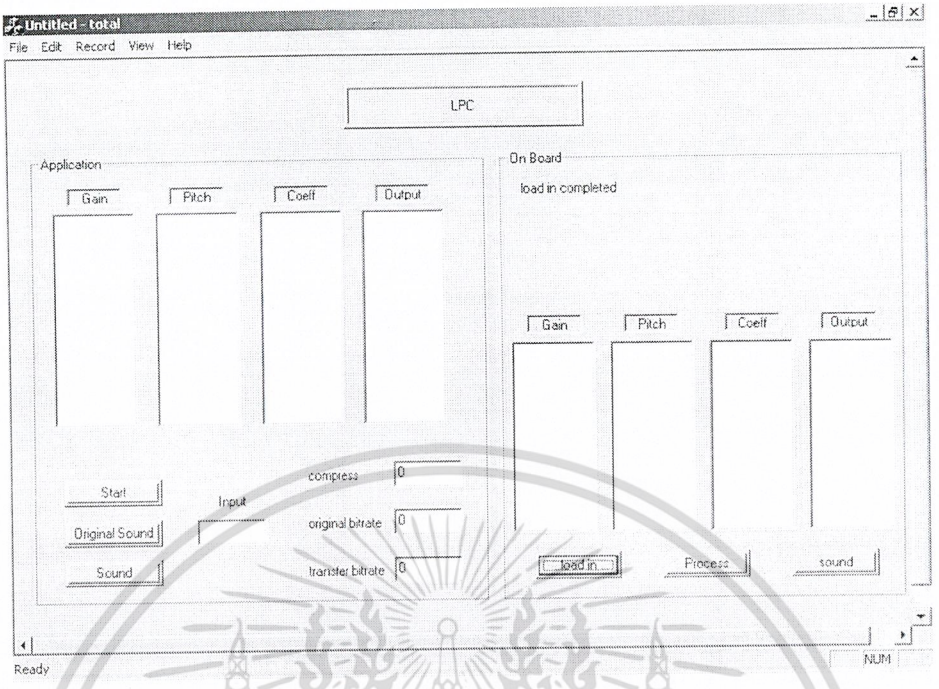
รูปที่ 4.36 กราฟสเปกตรัมจากการทำ Fast Fourier Transform ของสัญญาณอินพุท (ตัวอักษร d) เทียบกับ สัญญาณเอาท์พุทจากโปรแกรม matlab และ สัญญาณเอาท์พุทจากโปรแกรม visual c++



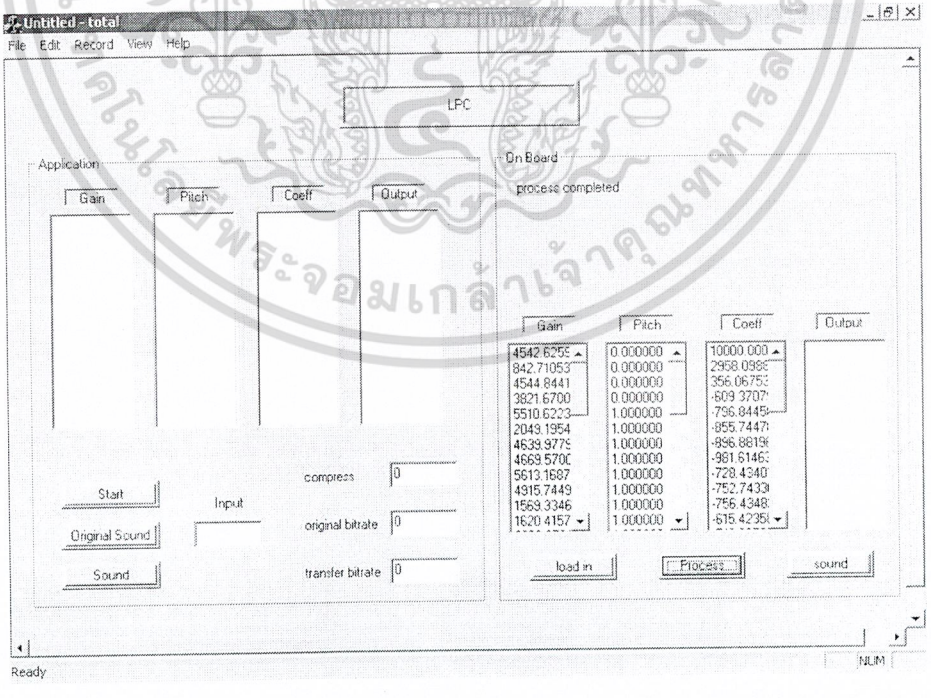
รูปที่ 4.37 หน้าต่างรวมการควบคุมการทำงานผ่านบอร์ดและการประยุกต์นำไปใช้งาน



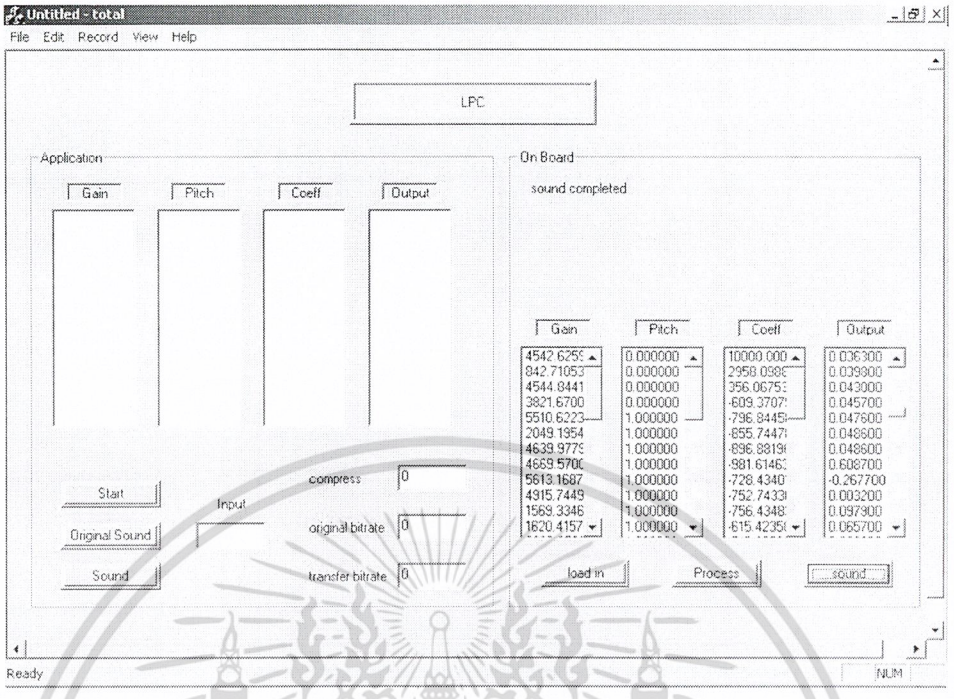
รูปที่ 4.38 หน้าต่างรวมการควบคุมการทำงานผ่านบอร์ดและการประยุกต์นำไปใช้งาน (เมื่อทำเฉพาะการประยุกต์การใช้งาน)



รูปที่ 4.39 หน้าต่างรวมการควบคุมการทำงานผ่านบอร์ดและการประยุกต์นำไปใช้งาน (เมื่อเริ่มทำการโหลดเสียงผ่านบอร์ดไปยังคอมพิวเตอร์)



รูปที่ 4.40 หน้าต่างรวมการควบคุมการทำงานผ่านบอร์ดและการประยุกต์นำไปใช้งาน (เมื่อทำการประมวลผลโดยใช้บอร์ด)



รูปที่ 4.41 หน้าต่างรวมการควบคุมการทำงานผ่านบอร์ดและการประยุกต์นำไปใช้งาน (เมื่อทำการสังเคราะห์สัญญาณเสียงโดยใช้บอร์ด)

บทที่ 5
วิจารณ์และสรุปผล

จากการทดลองสังเคราะห์สัญญาณเสียงโดยใช้หลักการ LPC ด้วยโปรแกรม matlab ในเทอมแรกนั้นทำให้เราได้ศึกษาเกี่ยวกับหลักการ LPC และการใช้โปรแกรม matlab ได้ดียิ่งขึ้น และทราบถึงปัญหาของการประมวลผลของโปรแกรม matlab คือ โปรแกรม matlab จะใช้เวลาในการประมวลผลค่อนข้างมาก ดังนั้นทางผู้จัดทำจึงทำการเปลี่ยนจากโปรแกรม Matlab มาเขียนโปรแกรม Visual C++ เพื่อทำการประมวลผลร่วมกับบอร์ด TMS320C6211 ซึ่งจะใช้เวลาน้อยกว่าโปรแกรม Matlab รวมทั้งจะทำการสร้างหน้าต่างเพื่อแสดงค่า parameter ต่างๆ ของเสียงแต่ละเสียงด้วย

นอกจากนี้ก็ยังมีส่วนของโปรแกรมที่เป็นการประยุกต์การใช้งานของโปรเจกต์นี้ คือ จะทำการสร้างหน้าต่างโดยมีช่องให้รับค่าเป็นตัวอักษรทางหน้าจอ และเมื่อกดปุ่มก็จะแสดงค่า parameter ต่างๆ ออกมา รวมทั้งมีเสียงของตัวอักษรนั้นออกมาด้วย

<u>ปัญหาที่พบ</u>

เนื่องจากหน่วยความจำของบอร์ด TMS320C6211 มีจำกัดจึงทำให้ไม่สามารถประมวลภายในบอร์ดได้โดยตรงทั้งหมด จึงต้องทำการแบ่งโปรแกรมเพื่อทำการประมวลผลในคอมพิวเตอร์ส่วนหนึ่งแล้วจึงนำค่าที่ได้มาทำการสังเคราะห์สัญญาณเสียงออกมาทางบอร์ด แต่คุณภาพเสียงที่ได้นั้นก็จะไม่ดีเท่าที่ควรเนื่องจากปัญหาของหน่วยความจำของบอร์ดทำให้เลือกใช้ order ได้ค่อนข้างน้อย ส่วนในเรื่องของคุณภาพเสียงของโปรแกรมที่เป็นการประยุกต์การใช้งานนั้นถ้าใช้ order ที่มากขึ้นเสียงก็จะชัดขึ้นแต่ก็จะทำให้ใช้เวลาในการประมวลผลนานขึ้นไปด้วย

เอกสารอ้างอิง

[1] C.Britten Rorabaugh,"Digital Filter Designer's Handbook",U.S.A.,1993

[2] J.D.Markel A.H.Gray,Jr,"Linear Prediction of Speech",Springer-Verlag Berlin Heidelberg New York,1976

[3] L.R.Rabinear/R.W.Schafer,"Digital Processing of Speech Signals",Englewood Clifts,New Jersey,1978

[4] Rulph,"DSP Applications Using C and the TMS320C6X DSK",A Wiley-Interscience Publication,2002

[5] T.W.Parks/C.S.Burrus,"Digital Filter Design",Texas,1987

[6] "TMS320C6000,Programmer's Guide",Dallas,Texas,2000

[7] พรชัย ภววงษ์ศักดิ์,"การประมวลผลสัญญาณดิจิตอลเบื้องต้น",1999

ภาคผนวก ก

Main

```
clc; % clear the command line
clear all; % clear the workspace
%tt=[1:500];
%xx=cos(2*3.14*400/16000*tt);
%start program**********
InputFilename = 'd.wav';%รับค่าinput
[data, sr, bit] = wavread(InputFilename); % read the wavefile
%data=xx';
%sr=16000;
%order=round(sr/1000);
order=49;
[row1,co]=size(data);
if co > 1
    data=data(:,1);
end
[aCoeff,pitch,gain,outcompresslpc,rm,speech,aaa] =
compresscoder(data,order);
taCoef=aCoeff';
tgain=gain';
tpitch=pitch';


%outcompresslpc2 = compresscoder2(data);

% display the results GRAPH OF SPEECH, COMPRESSLPC
figure(1);
subplot(3,1,1);
plot(data);
grid;
subplot(3,1,2);
plot(speech);
grid;
subplot(3,1,3);
plot(outcompresslpc);
grid;
%subplot(3,1,3);
%plot(outcompresslpc2);
%grid;
%****************************************
%display cross-correlation
auto1= xcorr(data,outcompresslpc);
[g,h]=max(auto1);
auto11=auto1/g;
figure(2);
plot(auto11)
grid;

%****************************************

%figure(3);
%subplot(2,1,1);
%spectrum(data)
%grid;
%subplot(2,1,2);
%spectrum(outcompresslpc)
```

```
%grid;

%*****************************************
disp('PLAY THE SOUND');
disp('Press a key to play the original sound!');
pause
soundsc(data,sr);

disp('Press a key to play the sound that be pre-emphasis!');
pause
soundsc(speech,sr);

disp('Press a key to play the LPC compressed sound!');
pause;
soundsc(outcompresslpc,sr);
disp('Root mean square error');
rms1=rm;
rms1
wavwrite(outcompresslpc,sr,'wavefile.wav');

%disp('Press a key to play the voice-excited LPC compressed sound!');
%pause;
%soundsc(outcompresslpc2,sr);
```

Compresscoder

```
function [aCoeff,pitch,gain,outcompresslpc,rm,speech,aaa] =
compresscoder( data,order)

% system constants
% ----------------
sr = 16000; % sampling rate in Hertz (Hz)

% encoded the speech using LPC, compute the parameter wiht lpc method
[aCoeff, pitch, gain,speech] = processlpc(data, sr, order);

% decode/synthesize speech using LPC and impulse-trains as excitation
[outcompresslpc,rm,aaa] = synlpc(aCoeff, pitch, sr, gain,data);
```

Processlpc

```
function [aCoeff,pitch,gain,speech] = processlpc(data,sr,order)
%*****************************************************************
%*****************************************************************
%set parameter
L = order;
fr = 20; %time of M ms
fs = 30; %time of N ms
preemp = .9378; %coefficient of preemphasize

%************************ transpost data**********
[row col] = size(data);
if col==1 data=data'; end
nframe = 0;
%************************

%find length of N(samples in each frame of data must be sepperated),
%M(samples in each frame must be computed)
msfr = round(sr/1000*fr); % Convert ms to samples (M)
msfs = round(sr/1000*fs); % Convert ms to samples (N)
duration = length(data);  % Find length of data(speech)

%************************Process precmphasize data(speech)*******
speech = filter([1 -preemp], 1, data)'; % Preemphasize speech
%speech=data;
%****************** comfuse************
msoverlap = msfs - msfr;%samples N-M
ramp = [0:1/(msoverlap-1):1]'; % Compute part of window

%display input,and input(after pre emphasis)
%if 0
%subplot(3,1,1);
%plot(data);
%subplot(3,1,2);
%plot(speech);
%end
%*************************************

for frameIndex=1:msfr:duration-msfs+1 % frame rate=20ms M (ค่าต่อไปที่
framIndex คือค่า msfr+1)
frameData = speech(frameIndex:(frameIndex+msfs-1)); % frame size=30ms N
nframe = nframe+1;% count nframe
frameData1(:,nframe)=frameData;
autoCor = xcorr(frameData); % Compute the cross correlation
%*********************************
autoCorVec = autoCor(msfs+[0:L]);% นำค่า autoCor มาใช้เป็นจำนวน L+1 ค่า รวมค่าที่ rss0ด้วย
ass=autoCor(msfs+[1:msfs-1]);% ตั้งแต่ค่ากลางจนถึงค่าสุดท้าย(-1) คือ ค่าautoCor เป็นจำนวนคี่ตั้งนั้นจึงต้องลบ
ออกหนึ่งไม่เอาค่าrss0
rss=autoCor(msfs+[1:L]);% ไม่รวมค่า rss0

% find coefficient use Levinson's method******
%if 0
%AA=levinson(autoCorVec,L);
%aaa(:,nframe)=AA';
%end
%**************************
```

```
%brig autocorelation to find matrix Rss
err(1) = autoCorVec(1);
for m=1:L
    for k=1:L
        if m==k
            Rss(m,k)=err(1);
        else
            Rss(m,k)=rss(abs(m-k));
        end
    end
end
Rssi=inv(Rss);
%************************* find lpc coefficient **********************
A=Rssi*rss;
aCoeff(:,nframe)=[1;A*(-1)];
C=[1;A*(-1)];
%*****************************
E=[A*(-1)];
%find gain*******************************
i=0;temp=0;
for i=1:L
    temp=temp+A(i)*rss(i);
end
temp=err(1)-temp;
gain(nframe)=sqrt(temp);
%************************************
%************************* find re,ra ********************************
%if 0
for k=1:L;
    for i=1:L-k
        temp=temp+E(i)*E(i+k);
    end
    ra(k)=temp;
    temp=0;
end
%find re0**
re0=0;
for i=1:L
    re0=re0+ra(i)*rss(i);
end
%find re
temp = 0;
for n = 1:length(frameData)
    for k = 1:L
        if (n == k)
            temp = temp + ra(k)*err(1);
        else
            temp = temp + ra(k)*ass(abs(n-k));
        end
    end
    re(n) = temp;
    temp = 0;
end
%*********************************************************
condition=re/re0;
for i = 1:length(condition)
```

```
    if condition(i) == max(condition)
        Np = i;
 end
 end
if condition(Np) >= 0.01
    pitch(nframe) = Np;
    exitation = 'voice';
else
    exitation = 'unvoice';
    pitch(nframe)=0;
end
%*************************************************************
%end
if 0
errSig = filter([1 (A*(-1))'],1,frameData);
autoCorErr = xcorr(errSig);
[B,I] = sort(autoCorErr);
num = length(I);
if B(num-1) > .01*B(num)
pitch(nframe) = abs(I(num) - I(num-1));
else
pitch(nframe) = 0;
end
end
%************************** investigate (vioce or unvoice)*************
end

%stream = filter(1, [1 -preemp], stream)';
```

Synlpc

```
function [synWave,rm,aaa] = synlpc(aCoeff,pitch,sr,gain,data)

aaa=[];
%set constant***************
fr = 20;
fs = 30;
preemp = .9378;
%convert ms to samples*******
msfs = round(sr*fs/1000); % framesize in samples
msfr = round(sr*fr/1000); % framerate in samples
msoverlap = msfs - msfr;
ramp = [0:1/(msoverlap-1):1]';
[L1 nframe] = size(aCoeff); % L1 = 1+number of LPC coeffs
%****************************
for frameIndex=1:nframe
A = aCoeff(:,frameIndex);
% first check if it is voiced or unvoiced sound:
if ( pitch(frameIndex) ~= 0 )
t = 0 : 1/sr : fs*10^(-3); % sr sample freq. for fr ms
d = 0 : 1/pitch(frameIndex) : 1; % 1/pitchfreq. repetition freq.
residFrame = (pulstran(t, d, 'tripuls', 0.00000000001))'; % sawtooth
width of 0.001s
residFrame = residFrame + 0.01*randn(msfs+1,1);
else
residFrame = [];
for m = 1:msfs
residFrame = [residFrame; randn];
end % for
end;
%88888888888888888888888888888888888888888888888888888

bb=residFrame;
if(frameIndex==1) % add synthesize frames using a trapezoidal window
aaa = bb(1:msfr);
else
aaa = [aaa; lap+bb(1:msoverlap).*ramp; ...
bb(msoverlap+1:msfr)];
end
if(frameIndex==nframe)
aaa = [aaa; bb(msfr+1:msfs)];
else
lap = bb(msfr+1:msfs).*flipud(ramp);
end

%8888888888888888888888888888888888888888888888888888
synFrame = filter(gain(frameIndex), A', residFrame); % synthesize
speech from LPC coeffs
if(frameIndex==1) % add synthesize frames using a trapezoidal window
synWave = synFrame(1:msfr);
else
synWave = [synWave; overlap+synFrame(1:msoverlap).*ramp; ...
synFrame(msoverlap+1:msfr)];
end
```

```
if(frameIndex==nframe)
synWave = [synWave; synFrame(msfr+1:msfs)];
else
overlap = synFrame(msfr+1:msfs).*flipud(ramp);
end
end;
%synWave = filter(1, [1 -preemp], synWave);
%*****************************************
%*******find root mean square*************
ms=0;
rm=0;
if length(data)>length(synWave)
    W=length(synWave);
else W=length(data);
end

% find rms*****************************************
[v,x]=max(data);
data1=data/v;
[y,z]=max(synWave);
synWave1=synWave/y;
for i= 1:W
    ms =  ms+(data1(i)-synWave1(i)')^2;
end .
ms=ms/W;
rm = sqrt(ms);
```

ภาคผนวก ข

```
/************************************************************************
*********
* FILENAME
*   dsk6x11hpi.h
*
* DESCRIPTION
*   The public header file for dsk6x11hpi.dll
*
************************************************************************
********/
#include <windows.h>

/* Enumeration used with dsk_board_type */
typedef enum {
               TYPE_UNKNOWN = 0,
               TYPE_6211_DSK,
               TYPE_6711_DSK
             } dskDLL_BOARD_TYPE, *PdskDLL_BOARD_TYPE;
/* End of Enumeration used with dsk6x_board_type */

/* DSK Public Handle declaration */
typedef  void * dskHANDLE;
/* End of DSK Public Handle declaration */



/* DSK Function Prototypes */
#ifdef __cplusplus
extern "C" {
#endif

extern BOOL  dsk6x_open(char *, dskHANDLE*);
extern BOOL  dsk6x_close(dskHANDLE );
extern BOOL  dsk6x_board_type( dskHANDLE , PdskDLL_BOARD_TYPE, unsigned
 short *);
extern BOOL  dsk6x_hpi_open(dskHANDLE );
extern BOOL  dsk6x_hpi_close(dskHANDLE );
extern BOOL  dsk6x_reset_dsp(dskHANDLE,unsigned char,unsigned char);
extern BOOL  dsk6x_reset_board(dskHANDLE);
extern BOOL  dsk6x_hpi_read(dskHANDLE ,unsigned long *, unsigned long *
, unsigned long);
extern BOOL  dsk6x_hpi_write(dskHANDLE ,unsigned long *,unsigned long *
, unsigned long);
extern BOOL  dsk6x_hpi_fill(dskHANDLE ,unsigned long,unsigned long *,un
signed long);
extern BOOL  dsk6x_hpi_generate_int(dskHANDLE );
extern int   dsk6x_coff_load(dskHANDLE ,char *, BOOL, BOOL, BOOL);

#ifdef __cplusplus
}
#endif

/* End of DSK Function Prototypes */
```

```
#ifndef RAND_MAX
#define RAND_MAX 32767
#endif

double uniform()
{
    return ((double)(rand()& RAND_MAX) / RAND_MAX-0.5);
}
```

```
// totalView.cpp : implementation of the CTotalView class
//

#include "stdafx.h"
#include "total.h"



#include "totalSet.h"
#include "totalDoc.h"
#include "totalView.h"
#include <fstream>
#include <cmath>
#include "mmsystem.h"
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "dsk6x11hpi.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#include "uniform.h"
#include "math.h"

char ip;

//============================================================================
const int AppTimerID = 1234567;

/* Declare variable for use with DSK-PC communication */
BOOL           verbose_flag=0, clear_bss_flag=0, dump_flag=0;
unsigned char  little_endian=0,big_endian=1,HPI_boot=1,ROM_boot=0;
char           *pfname="dsk6x11spp378.cfg", *input_file=

"input.out", *output_file="loop.out", *sound_file="loop_intr.out";

unsigned short  rev_id;
dskHANDLE          handle;
dskDLL_BOARD_TYPE  board_type;

/* Declare DSK interface memory variables. */
/* In HOST_SIN we have declare a variable area for PC interface at 0x00
000220 */

unsigned long   DSK_mem_addr1=0x80000000, DSK_mem1[1], DSK_mem_length=1
;
unsigned long   DSK_mem_addr2=0x80000004, DSK_mem2[1];//, DSK_mem_lengt
h2=1;
unsigned long   DSK_mem_addr3=0x80000008, DSK_mem3[1];

int coff_ret,input[16500],outpro[16500],ccoeff[50][101],cgain[50];
double voice2[32][500];
//============================================================================
////////////////////////////////////////////////////////////////////////////
```

```
//////
// CTotalView

IMPLEMENT_DYNCREATE(CTotalView, CRecordView)

BEGIN_MESSAGE_MAP(CTotalView, CRecordView)
    //{{AFX_MSG_MAP(CTotalView)
    ON_BN_CLICKED(IDC_BUT_START, OnButStart)
    ON_BN_CLICKED(IDC_BUT_ORI, OnButOri)
    ON_BN_CLICKED(IDC_BUT_SOUND, OnButSound)
    ON_BN_CLICKED(IDC_BUT_LOAD, OnButLoad)
    ON_BN_CLICKED(IDC_BUT_PROCESS, OnButProcess)
    ON_BN_CLICKED(IDC_BUT_BOARD_SOUND, OnButBoardSound)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CTotalView construction/destruction

CTotalView::CTotalView()
    : CRecordView(CTotalView::IDD)
{
    //{{AFX_DATA_INIT(CTotalView)
    m_pSet = NULL;
    m_comp = 0.0;
    m_oribit = 0.0;
    m_transbit = 0.0;
    m_input = _T("");
    m_status = _T("");
    m_status2 = _T("");
    m_status3 = _T("");
    m_display = _T("");
    m_display2 = _T("");
    //}}AFX_DATA_INIT
    // TODO: add construction code here

}

CTotalView::~CTotalView()
{
}

void CTotalView::DoDataExchange(CDataExchange* pDX)
{
    CRecordView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTotalView)
    DDX_Control(pDX, IDC_LIST_BOARD_P, m_boardp);
    DDX_Control(pDX, IDC_LIST_BOARD_O, m_boardo);
    DDX_Control(pDX, IDC_LIST_BOARD_G, m_boardg);
    DDX_Control(pDX, IDC_LIST_BOARD_C, m_boardc);
    DDX_Control(pDX, IDC_LIST_PITCH, m_xpitch);
    DDX_Control(pDX, IDC_LIST_OUT, m_xout);
    DDX_Control(pDX, IDC_LIST_GAIN, m_xgain);
    DDX_Control(pDX, IDC_LIST_COEFF, m_xcoeff);
    DDX_Text(pDX, IDC_EDIT_COMP, m_comp);
    DDX_Text(pDX, IDC_EDIT_ORIBIT, m_oribit);
    DDX_Text(pDX, IDC_EDIT_TRANSBIT, m_transbit);
```

```
    DDX_Text(pDX, IDC_EDIT1, m_input);
    DDV_MaxChars(pDX, m_input, 1);
    DDX_Text(pDX, IDC_STATUS, m_status);
    DDX_Text(pDX, IDC_STATUS2, m_status2);
    DDX_Text(pDX, IDC_STATUS3, m_status3);
    DDX_Text(pDX, IDC_display, m_display);
    DDX_Text(pDX, IDC_display2, m_display2);
    //}}AFX_DATA_MAP
}

BOOL CTotalView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CRecordView::PreCreateWindow(cs);
}

void CTotalView::OnInitialUpdate()
{
    m_pSet = &GetDocument()->m_totalSet;
    CRecordView::OnInitialUpdate();
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit();

}

/////////////////////////////////////////////////////////////////////////////
//////
// CTotalView diagnostics

#ifdef _DEBUG
void CTotalView::AssertValid() const
{
    CRecordView::AssertValid();
}

void CTotalView::Dump(CDumpContext& dc) const
{
    CRecordView::Dump(dc);
}

CTotalDoc* CTotalView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTotalDoc)));
    return (CTotalDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
//////
// CTotalView database support
CRecordset* CTotalView::OnGetRecordset()
{
    return m_pSet;
}
```

```
//////////////////////////////////////////////////////////////////////////
//////
// CTotalView message handlers

double gaussian()
{
    static int ready=0;
    static double gstore;
    double v1,v2,r,fac,gaus;

    double uniform();
    // dddd
    if (ready==0){
        do{
            v1=2.*uniform();
            v2=2.*uniform();
            r=v1*v1+v2*v2;
        }while(r>1.0);
        //aaa
        fac=sqrt(-2.*log(r)/r);
        gstore=v1*fac;
        gaus=v2*fac;
        ready=1;
    }
    else {
        ready=0;
        gaus=gstore;
    }
    return(gaus);
    }

//-------------------------------
using namespace std;
void write_lsb (ofstream &st, unsigned int x, int len = 4) {
for (int i = 0; i < len; ++i) {
unsigned int r = x & 0xFF;
x >>= 8;
st.put(r );
}
}
int sample_rate;
const int  sample_bits = 16, num_channels = 2;//sample_rate =16000,
void init_wav_file (ofstream &st, const char *name) {
st << "RIFF";
write_lsb (st, 0); // Packet length unknown
st << "WAVEfmt ";
write_lsb (st, 16); // Format chunk length
write_lsb (st, 1, 2); // PCM data
write_lsb (st, num_channels, 2);
write_lsb (st, sample_rate);
write_lsb (st, sample_rate * num_channels *(sample_bits/8));//data rate
write_lsb (st, num_channels * sample_bits / 8.0, 2);//align = number of
 byte per time
write_lsb (st, sample_bits, 2);
st << "data";
write_lsb (st, 0); // data length
}
```

```
void write_sample (ofstream &st, float amp) {
int uamp = static_cast<int>(amp / 2 * (1<< sample_bits));
// Write the sample on all channels
for (int i = 0; i < num_channels; ++i)
write_lsb (st, uamp, sample_bits/8);
}


const int package_len_pos = 4;
const int data_len_pos = 40;
void close_wav_file (ofstream &st) {
// Find the file size
int len = st.tellp ();
st.seekp (package_len_pos);
write_lsb (st, len - 8);
st.seekp (data_len_pos);
write_lsb (st, len - 44);
st.close ();
}


void CTotalView::OnButStart()
{
    // TODO: Add your control notification handler code here
    m_xpitch.ResetContent();
    m_xcoeff.ResetContent();
    m_xgain.ResetContent();
    m_xout.ResetContent();

    //check sound
    UpdateData(TRUE);
    m_pSet->MoveFirst();
    while (m_input!=m_pSet->m_sound){
    m_pSet->MoveNext();
    }
    int nframe=m_pSet->m_nframe;// get nframe
    ip==m_input;

    // initial value
    double ccoeff[150][152],cgain[1000],cpitch[1000],voice[500];// (rea
l order=49)
    int index,i,j,count=0,length=481,order=151,count1=151,impulse[500],
noise[500];
    double  y1=0,y2=0,y3=0,y4=0,y5=0,y6=0,y7=0,y8=0,y9=0,y10=0;
    double y11=0,y12=0,y13=0,y14=0,y15=0,y16=0,y17=0,y18=0,y19=0,y20=0;
    double y21=0,y22=0,y23=0,y24=0,y25=0,y26=0,y27=0,y28=0,y29=0,y30=0;
    double y31=0,y32=0,y33=0,y34=0,y35=0,y36=0,y37=0,y38=0,y39=0,y40=0;
    double y41=0,y42=0,y43=0,y44=0,y45=0,y46=0,y47=0,y48=0,y49=0,y50=0;
    double y51=0,y52=0,y53=0,y54=0,y55=0,y56=0,y57=0,y58=0,y59=0,y60=0;
    double y61=0,y62=0,y63=0,y64=0,y65=0,y66=0,y67=0,y68=0,y69=0,y70=0;
    double y71=0,y72=0,y73=0,y74=0,y75=0,y76=0,y77=0,y78=0,y79=0,y80=0;
    double y81=0,y82=0,y83=0,y84=0,y85=0,y86=0,y87=0,y88=0,y89=0,y90=0;
    double y91=0,y92=0,y93=0,y94=0,y95=0,y96=0,y97=0,y98=0,y99=0,y100=0
;

    double y101=0,y102=0,y103=0,y104=0,y105=0,y106=0,y107=0,y108=0,y109
=0,y110=0;
```

```
    double y111=0,y112=0,y113=0,y114=0,y115=0,y116=0,y117=0,y118=0,y119
=0,y120=0;
    double y121=0,y122=0,y123=0,y124=0,y125=0,y126=0,y127=0,y128=0,y129
=0,y130=0;
    double y131=0,y132=0,y133=0,y134=0,y135=0,y136=0,y137=0,y138=0,y139
=0,y140=0;
    double y141=0,y142=0,y143=0,y144=0,y145=0,y146=0,y147=0,y148=0,y149
=0,y150=0;
    int msfr=320,msfs=480,msoverlap,m=1,k=1,size;
    double ramp[161],synwave[50000],overlap[161],ramp1[161];
    ramp[1]=0;
    size=msfr;
    msoverlap=msfs-msfr;
    for (j=2;j<=msoverlap;j++){
        ramp[j]=ramp[j-1]+0.00628931;//1/(msoverlap-1);

    }

    for ( index=1;index<nframe+1;index++){
        //check fram and sound
            UpdateData(TRUE);
            m_pSet->MoveFirst();
            while(m_input!=m_pSet->m_sound||index!=m_pSet->m_frame){
                m_pSet->MoveNext();
            }
        //get pitch coeff and gain
            cpitch[index]=m_pSet->m_pitch;
            cgain[index]=m_pSet->m_gain;
            ccoeff[index][1]=m_pSet->m_coeff1;
            ccoeff[index][2]=m_pSet->m_coeff2;
            ccoeff[index][3]=m_pSet->m_coeff3;
            ccoeff[index][4]=m_pSet->m_coeff4;
            ccoeff[index][5]=m_pSet->m_coeff5;
            ccoeff[index][6]=m_pSet->m_coeff6;
            ccoeff[index][7]=m_pSet->m_coeff7;
            ccoeff[index][8]=m_pSet->m_coeff8;
            ccoeff[index][9]=m_pSet->m_coeff9;
            ccoeff[index][10]=m_pSet->m_coeff10;
            ccoeff[index][11]=m_pSet->m_coeff11;
            ccoeff[index][12]=m_pSet->m_coeff12;
            ccoeff[index][13]=m_pSet->m_coeff13;
            ccoeff[index][14]=m_pSet->m_coeff14;
            ccoeff[index][15]=m_pSet->m_coeff15;
            ccoeff[index][16]=m_pSet->m_coeff16;
            ccoeff[index][17]=m_pSet->m_coeff17;
            ccoeff[index][18]=m_pSet->m_coeff18;
            ccoeff[index][19]=m_pSet->m_coeff19;
            ccoeff[index][20]=m_pSet->m_coeff20;
            ccoeff[index][21]=m_pSet->m_coeff21;
            ccoeff[index][22]=m_pSet->m_coeff22;
            ccoeff[index][23]=m_pSet->m_coeff23;
            ccoeff[index][24]=m_pSet->m_coeff24;
            ccoeff[index][25]=m_pSet->m_coeff25;
            ccoeff[index][26]=m_pSet->m_coeff26;
            ccoeff[index][27]=m_pSet->m_coeff27;
            ccoeff[index][28]=m_pSet->m_coeff28;
            ccoeff[index][29]=m_pSet->m_coeff29;
            ccoeff[index][30]=m_pSet->m_coeff30;
```

```
ccoeff[index][31]=m_pSet->m_coeff31;
ccoeff[index][32]=m_pSet->m_coeff32;
ccoeff[index][33]=m_pSet->m_coeff33;
ccoeff[index][34]=m_pSet->m_coeff34;
ccoeff[index][35]=m_pSet->m_coeff35;
ccoeff[index][36]=m_pSet->m_coeff36;
ccoeff[index][37]=m_pSet->m_coeff37;
ccoeff[index][38]=m_pSet->m_coeff38;
ccoeff[index][39]=m_pSet->m_coeff39;
ccoeff[index][40]=m_pSet->m_coeff40;
ccoeff[index][41]=m_pSet->m_coeff41;
ccoeff[index][42]=m_pSet->m_coeff42;
ccoeff[index][43]=m_pSet->m_coeff43;
ccoeff[index][44]=m_pSet->m_coeff44;
ccoeff[index][45]=m_pSet->m_coeff45;
ccoeff[index][46]=m_pSet->m_coeff46;
ccoeff[index][47]=m_pSet->m_coeff47;
ccoeff[index][48]=m_pSet->m_coeff48;
ccoeff[index][49]=m_pSet->m_coeff49;
ccoeff[index][50]=m_pSet->m_coeff50;


ccoeff[index][51]=m_pSet->m_coeff51;
ccoeff[index][52]=m_pSet->m_coeff52;
ccoeff[index][53]=m_pSet->m_coeff53;
ccoeff[index][54]=m_pSet->m_coeff54;
ccoeff[index][55]=m_pSet->m_coeff55;
ccoeff[index][56]=m_pSet->m_coeff56;
ccoeff[index][57]=m_pSet->m_coeff57;
ccoeff[index][58]=m_pSet->m_coeff58;
ccoeff[index][59]=m_pSet->m_coeff59;
ccoeff[index][60]=m_pSet->m_coeff60;

ccoeff[index][61]=m_pSet->m_coeff61;
ccoeff[index][62]=m_pSet->m_coeff62;
ccoeff[index][63]=m_pSet->m_coeff63;
ccoeff[index][64]=m_pSet->m_coeff64;
ccoeff[index][65]=m_pSet->m_coeff65;
ccoeff[index][66]=m_pSet->m_coeff66;
ccoeff[index][67]=m_pSet->m_coeff67;
ccoeff[index][68]=m_pSet->m_coeff68;
ccoeff[index][69]=m_pSet->m_coeff69;
ccoeff[index][70]=m_pSet->m_coeff70;

ccoeff[index][71]=m_pSet->m_coeff71;
ccoeff[index][72]=m_pSet->m_coeff72;
ccoeff[index][73]=m_pSet->m_coeff73;
ccoeff[index][74]=m_pSet->m_coeff74;
ccoeff[index][75]=m_pSet->m_coeff75;
ccoeff[index][76]=m_pSet->m_coeff76;
ccoeff[index][77]=m_pSet->m_coeff77;
ccoeff[index][78]=m_pSet->m_coeff78;
ccoeff[index][79]=m_pSet->m_coeff79;
ccoeff[index][80]=m_pSet->m_coeff80;

ccoeff[index][81]=m_pSet->m_coeff81;
ccoeff[index][82]=m_pSet->m_coeff82;
ccoeff[index][83]=m_pSet->m_coeff83;
```

```
ccoeff[index][84]=m_pSet->m_coeff84;
ccoeff[index][85]=m_pSet->m_coeff85;
ccoeff[index][86]=m_pSet->m_coeff86;
ccoeff[index][87]=m_pSet->m_coeff87;
ccoeff[index][88]=m_pSet->m_coeff88;
ccoeff[index][89]=m_pSet->m_coeff89;
ccoeff[index][90]=m_pSet->m_coeff90;

ccoeff[index][91]=m_pSet->m_coeff91;
ccoeff[index][92]=m_pSet->m_coeff92;
ccoeff[index][93]=m_pSet->m_coeff93;
ccoeff[index][94]=m_pSet->m_coeff94;
ccoeff[index][95]=m_pSet->m_coeff95;
ccoeff[index][96]=m_pSet->m_coeff96;
ccoeff[index][97]=m_pSet->m_coeff97;
ccoeff[index][98]=m_pSet->m_coeff98;
ccoeff[index][99]=m_pSet->m_coeff99;

ccoeff[index][100]=m_pSet->m_coeff100;
ccoeff[index][101]=m_pSet->m_coeff101;
ccoeff[index][102]=m_pSet->m_coeff102;
ccoeff[index][103]=m_pSet->m_coeff103;
ccoeff[index][104]=m_pSet->m_coeff104;
ccoeff[index][105]=m_pSet->m_coeff105;
ccoeff[index][106]=m_pSet->m_coeff106;
ccoeff[index][107]=m_pSet->m_coeff107;
ccoeff[index][108]=m_pSet->m_coeff108;
ccoeff[index][109]=m_pSet->m_coeff109;
ccoeff[index][110]=m_pSet->m_coeff110;

ccoeff[index][111]=m_pSet->m_coeff111;
ccoeff[index][112]=m_pSet->m_coeff112;
ccoeff[index][113]=m_pSet->m_coeff113;
ccoeff[index][114]=m_pSet->m_coeff114;
ccoeff[index][115]=m_pSet->m_coeff115;
ccoeff[index][116]=m_pSet->m_coeff116;
ccoeff[index][117]=m_pSet->m_coeff117;
ccoeff[index][118]=m_pSet->m_coeff118;
ccoeff[index][119]=m_pSet->m_coeff119;
ccoeff[index][120]=m_pSet->m_coeff120;

ccoeff[index][121]=m_pSet->m_coeff121;
ccoeff[index][122]=m_pSet->m_coeff122;
ccoeff[index][123]=m_pSet->m_coeff123;
ccoeff[index][124]=m_pSet->m_coeff124;
ccoeff[index][125]=m_pSet->m_coeff125;
ccoeff[index][126]=m_pSet->m_coeff126;
ccoeff[index][127]=m_pSet->m_coeff127;
ccoeff[index][128]=m_pSet->m_coeff128;
ccoeff[index][129]=m_pSet->m_coeff129;
ccoeff[index][130]=m_pSet->m_coeff130;
ccoeff[index][131]=m_pSet->m_coeff131;
ccoeff[index][132]=m_pSet->m_coeff132;
ccoeff[index][133]=m_pSet->m_coeff133;
ccoeff[index][134]=m_pSet->m_coeff134;
ccoeff[index][135]=m_pSet->m_coeff135;
ccoeff[index][136]=m_pSet->m_coeff136;
```

```
            ccoeff[index][137]=m_pSet->m_coeff137;
            ccoeff[index][138]=m_pSet->m_coeff138;
            ccoeff[index][139]=m_pSet->m_coeff139;
            ccoeff[index][140]=m_pSet->m_coeff140;
            ccoeff[index][141]=m_pSet->m_coeff141;
            ccoeff[index][142]=m_pSet->m_coeff142;
            ccoeff[index][143]=m_pSet->m_coeff143;
            ccoeff[index][144]=m_pSet->m_coeff144;
            ccoeff[index][145]=m_pSet->m_coeff145;
            ccoeff[index][146]=m_pSet->m_coeff146;
            ccoeff[index][147]=m_pSet->m_coeff147;
            ccoeff[index][148]=m_pSet->m_coeff148;
            ccoeff[index][149]=m_pSet->m_coeff149;
            ccoeff[index][150]=m_pSet->m_coeff150;
            ccoeff[index][151]=m_pSet->m_coeff151;


    // filter calculate
    //*************************************************************
*
    //--------------------check for voice or un voice and filter------
----------
            if (cpitch[index]!=0) {
    //-----------find impulse train-------------

    for (i=order;i<length+1;i++){
        if(i==count1){
            impulse[i]=1;
            count1=count1+250;
        }
        else
        {
            impulse[i]=0;
        }

    }
    count1=152;
    }else{
    //-----------------noise--------------------
    double gaussian();
    srand( (unsigned)time( NULL ) );
    for (i=1;i<481;i++){
    noise[i]=gaussian();
    }

    }
//*********************************************************
    for (i=1;i<order;i++){// initial voice (y(n))
        voice[i]=0.01;
    }
    for (i=order;i<length+1;i++){
        y1=ccoeff[index][2]*voice[i-1];y2=ccoeff[index][3]*voice[i-2];y
3=ccoeff[index][4]*voice[i-3];
        y4=ccoeff[index][5]*voice[i-4];y5=ccoeff[index][6]*voice[i-5];y
6=ccoeff[index][7]*voice[i-6];
        y7=ccoeff[index][8]*voice[i-7];y8=ccoeff[index][9]*voice[i-8];y
9=ccoeff[index][10]*voice[i-9];
        y10=ccoeff[index][11]*voice[i-10];y11=ccoeff[index][12]*voice[i
```

```
-11];y12=ccoeff[index][13]*voice[i-12];
        y13=ccoeff[index][14]*voice[i-13];y14=ccoeff[index][15]*voice[i
-14];y15=ccoeff[index][16]*voice[i-15];
        y16=ccoeff[index][17]*voice[i-16];y17=ccoeff[index][18]*voice[i
-17];y18=ccoeff[index][19]*voice[i-18];
        y19=ccoeff[index][20]*voice[i-19];y20=ccoeff[index][21]*voice[i
-20];y21=ccoeff[index][22]*voice[i-21];
        y22=ccoeff[index][23]*voice[i-22];y23=ccoeff[index][24]*voice[i
-23];y24=ccoeff[index][25]*voice[i-24];
        y25=ccoeff[index][26]*voice[i-25];y26=ccoeff[index][27]*voice[i
-26];y27=ccoeff[index][28]*voice[i-27];
        y28=ccoeff[index][29]*voice[i-28];y29=ccoeff[index][30]*voice[i
-29];y30=ccoeff[index][31]*voice[i-30];
        y31=ccoeff[index][32]*voice[i-31];y32=ccoeff[index][33]*voice[i
-32];y33=ccoeff[index][34]*voice[i-33];
        y34=ccoeff[index][35]*voice[i-34];y35=ccoeff[index][36]*voice[i
-35];y36=ccoeff[index][37]*voice[i-36];
        y37=ccoeff[index][38]*voice[i-37];y38=ccoeff[index][39]*voice[i
-38];y39=ccoeff[index][40]*voice[i-39];
        y40=ccoeff[index][41]*voice[i-40];y41=ccoeff[index][42]*voice[i
-41];y42=ccoeff[index][43]*voice[i-42];
        y43=ccoeff[index][44]*voice[i-43];y44=ccoeff[index][45]*voice[i
-44];y45=ccoeff[index][46]*voice[i-45];
        y46=ccoeff[index][47]*voice[i-46];y47=ccoeff[index][48]*voice[i
-47];y48=ccoeff[index][49]*voice[i-48];
        y49=ccoeff[index][50]*voice[i-49];y50=ccoeff[index][51]*voice[i
-50];y51=ccoeff[index][52]*voice[i-51];

        y52=ccoeff[index][53]*voice[i-52];y53=ccoeff[index][54]*voice[i
-53];y54=ccoeff[index][55]*voice[i-54];
        y55=ccoeff[index][56]*voice[i-55];y56=ccoeff[index][57]*voice[i
-56];y57=ccoeff[index][58]*voice[i-57];
        y58=ccoeff[index][59]*voice[i-58];y59=ccoeff[index][60]*voice[i
-59];y60=ccoeff[index][61]*voice[i-60];
        y61=ccoeff[index][62]*voice[i-61];y62=ccoeff[index][63]*voice[i
-62];y63=ccoeff[index][64]*voice[i-63];
        y64=ccoeff[index][65]*voice[i-64];y65=ccoeff[index][66]*voice[i
-65];y66=ccoeff[index][67]*voice[i-66];
        y67=ccoeff[index][68]*voice[i-67];y68=ccoeff[index][69]*voice[i
-68];y69=ccoeff[index][70]*voice[i-69];
        y70=ccoeff[index][71]*voice[i-70];y71=ccoeff[index][72]*voice[i
-71];y72=ccoeff[index][73]*voice[i-72];
        y73=ccoeff[index][74]*voice[i-73];y74=ccoeff[index][75]*voice[i
-74];y75=ccoeff[index][76]*voice[i-75];
        y76=ccoeff[index][77]*voice[i-76];y77=ccoeff[index][78]*voice[i
-77];y78=ccoeff[index][79]*voice[i-78];
        y79=ccoeff[index][80]*voice[i-79];y80=ccoeff[index][81]*voice[i
-80];y81=ccoeff[index][82]*voice[i-81];
        y82=ccoeff[index][83]*voice[i-82];y83=ccoeff[index][84]*voice[i
-83];y84=ccoeff[index][85]*voice[i-84];
        y85=ccoeff[index][86]*voice[i-85];y86=ccoeff[index][87]*voice[i
-86];y87=ccoeff[index][88]*voice[i-87];
        y88=ccoeff[index][89]*voice[i-88];y89=ccoeff[index][90]*voice[i
-89];y90=ccoeff[index][91]*voice[i-90];
        y91=ccoeff[index][92]*voice[i-91];y92=ccoeff[index][93]*voice[i
-92];y93=ccoeff[index][94]*voice[i-93];

        y94=ccoeff[index][95]*voice[i-94];y95=ccoeff[index][96]*voice[i
```

```
-95];y96=ccoeff[index][97]*voice[i-96];
        y97=ccoeff[index][98]*voice[i-97];y98=ccoeff[index][99]*voice[i
-98];y99=ccoeff[index][100]*voice[i-99];
        y100=ccoeff[index][101]*voice[i-100];y101=ccoeff[index][102]*vo
ice[i-101];y102=ccoeff[index][103]*voice[i-102];
        y103=ccoeff[index][104]*voice[i-103];y104=ccoeff[index][105]*vo
ice[i-104];y105=ccoeff[index][106]*voice[i-105];
        y106=ccoeff[index][107]*voice[i-106];y107=ccoeff[index][108]*vo
ice[i-107];y108=ccoeff[index][109]*voice[i-108];
        y109=ccoeff[index][110]*voice[i-109];y110=ccoeff[index][111]*vo
ice[i-110];y111=ccoeff[index][112]*voice[i-111];
        y112=ccoeff[index][113]*voice[i-112];y113=ccoeff[index][114]*vo
ice[i-113];y114=ccoeff[index][115]*voice[i-114];
        y115=ccoeff[index][116]*voice[i-115];y116=ccoeff[index][117]*vo
ice[i-116];y117=ccoeff[index][118]*voice[i-117];
        y118=ccoeff[index][119]*voice[i-118];y119=ccoeff[index][120]*vo
ice[i-119];y120=ccoeff[index][121]*voice[i-120];
        y121=ccoeff[index][122]*voice[i-121];y122=ccoeff[index][123]*vo
ice[i-122];y123=ccoeff[index][124]*voice[i-123];
        y124=ccoeff[index][125]*voice[i-124];y125=ccoeff[index][126]*vo
ice[i-125];y126=ccoeff[index][127]*voice[i-126];
        y127=ccoeff[index][128]*voice[i-127];y128=ccoeff[index][129]*vo
ice[i-128];y129=ccoeff[index][130]*voice[i-129];
        y130=ccoeff[index][131]*voice[i-130];y131=ccoeff[index][132]*vo
ice[i-131];y132=ccoeff[index][133]*voice[i-132];
        y133=ccoeff[index][134]*voice[i-133];y134=ccoeff[index][135]*vo
ice[i-134];y135=ccoeff[index][136]*voice[i-135];

        y136=ccoeff[index][137]*voice[i-136];y137=ccoeff[index][138]*vo
ice[i-137];y138=ccoeff[index][139]*voice[i-138];
        y139=ccoeff[index][140]*voice[i-139];y140=ccoeff[index][141]*vo
ice[i-140];y141=ccoeff[index][142]*voice[i-141];
        y142=ccoeff[index][143]*voice[i-142];y143=ccoeff[index][144]*vo
ice[i-143];y144=ccoeff[index][145]*voice[i-144];
        y145=ccoeff[index][146]*voice[i-145];y146=ccoeff[index][147]*vo
ice[i-146];y147=ccoeff[index][148]*voice[i-147];
        y148=ccoeff[index][149]*voice[i-148];y149=ccoeff[index][150]*vo
ice[i-149];y150=ccoeff[index][151]*voice[i-150];


    if (cpitch[index]!=0){
            voice[i]=cgain[index]*impulse[i]-y1-y2-y3-y4-y5-y6-y7-y8-y9
-y10
                -y11-y12-y13-y14-y15-y16-y17-y18-y19-y20-y21-y22-y23-y2
4-y25-y26-y27-y28-y29-y30
                -y31-y32-y33-y34-y35-y36-y37-y38-y39-y40-y41-y42-y43-y4
4-y45-y46-y47-y48-y49-y50
                -y51-y52-y53-y54-y55-y56-y57-y58-y59-y60-y61-y62-y63-y6
4-y65-y66-y67-y68-y69-y70
                -y71-y72-y73-y74-y75-y76-y77-y78-y79-y80-y81-y82-y83-y8
4-y85-y86-y87-y88-y89-y90
                -y91-y92-y93-y94-y95-y96-y97-y98-y99-y100-y101-y102-y10
3-y104-y105-y106-y107
                -y108-y109-y110-y111-y112-y113-y114-y115-y116-y117-y118
-y119-y120-y121-y122-y123
                -y124-y125-y126-y127-y128-y129-y130-y131-y132-y133-y134
-y135-y136-y137-y138-y139
                -y140-y141-y142-y143-y144-y145-y146-y147-y148-y149-y150
```

```
;


    }else{
            voice[i]=cgain[index]*noise[i]-y1-y2-y3-y4-y5-y6-y7-y8-y9-y
10
                -y11-y12-y13-y14-y15-y16-y17-y18-y19-y20-y21-y22-y23-y2
4-y25-y26-y27-y28-y29-y30
                -y31-y32-y33-y34-y35-y36-y37-y38-y39-y40-y41-y42-y43-y4
4-y45-y46-y47-y48-y49-y50
                -y51-y52-y53-y54-y55-y56-y57-y58-y59-y60-y61-y62-y63-y6
4-y65-y66-y67-y68-y69-y70
                -y71-y72-y73-y74-y75-y76-y77-y78-y79-y80-y81-y82-y83-y8
4-y85-y86-y87-y88-y89-y90
                -y91-y92-y93-y94-y95-y96-y97-y98-y99-y100-y101-y102-y10
3-y104-y105-y106-y107
                -y108-y109-y110-y111-y112-y113-y114-y115-y116-y117-y118
-y119-y120-y121-y122-y123
                -y124-y125-y126-y127-y128-y129-y130-y131-y132-y133-y134
-y135-y136-y137-y138-y139
                -y140-y141-y142-y143-y144-y145-y146-y147-y148-y149-y150
;
    }

    y1=0;y2=0;y3=0;y4=0;y5=0;y6=0;y7=0;y8=0;y9=0;y10=0;
    y11=0;y12=0;y13=0;y14=0;y15=0;y16=0;y17=0;y18=0;y19=0;y20=0;
    y21=0;y22=0;y23=0;y24=0;y25=0;y26=0;y27=0;y28=0;y29=0;y30=0;
    y31=0;y32=0;y33=0;y34=0;y35=0;y36=0;y37=0;y38=0;y39=0;y40=0;
    y41=0;y42=0;y43=0;y44=0;y45=0;y46=0;y47=0;y48=0;y49=0;y50=0;

    y51=0,y52=0,y53=0,y54=0,y55=0,y56=0,y57=0,y58=0,y59=0,y60=0;
    y61=0,y62=0,y63=0,y64=0,y65=0,y66=0,y67=0,y68=0,y69=0,y70=0;
    y71=0,y72=0,y73=0,y74=0,y75=0,y76=0,y77=0,y78=0,y79=0,y80=0;
    y81=0,y82=0,y83=0,y84=0,y85=0,y86=0,y87=0,y88=0,y89=0,y90=0;
    y91=0,y92=0,y93=0,y94=0,y95=0,y96=0,y97=0,y98=0,y99=0,y100=0;

    y101=0,y102=0,y103=0,y104=0,y105=0,y106=0,y107=0,y108=0,y109=0,y110
=0;
    y111=0,y112=0,y113=0,y114=0,y115=0,y116=0,y117=0,y118=0,y119=0,y120
=0;
    y121=0,y122=0,y123=0,y124=0,y125=0,y126=0,y127=0,y128=0,y129=0,y130
=0;
    y131=0,y132=0,y133=0,y134=0,y135=0,y136=0,y137=0,y138=0,y139=0,y140
=0;
    y141=0,y142=0,y143=0,y144=0,y145=0,y146=0,y147=0,y148=0,y149=0;y150
=0;

    }

//=================================================================



    //================================================

    if(index==1){
        for(j=1;j<=320;j++){//msfr
            synwave[j]=voice[j];
```

```
        }
        }else{
            for (j=1;j<=160;j++){        //msoverlap          //middle
                synwave[size+j]=overlap[j]+voice[j]*ramp[j];
            }
            for (j=160+1;j<=320;j++){//msoverlap msfr        //final
                synwave[size+j]=voice[j];//msoverlap
            }
            // combine
            //size=msoverlap+msfr-msoverlap+size;
            size=320+size;
        }

        if (index==nframe){
            for (j=320+1;j<=480;j++){//msfr msfs
            synwave[size+j-320]=voice[j];
            }
        }else{
            //=======flipud
            for (j=1;j<=160;j++){//msoverlap
            ramp1[j]=ramp[160-j+1];//msoverlap
            }
            //========
            for(j=320+1;j<=480;j++){//msfr msfs
            overlap[k]=voice[j]*ramp1[k];
            k++;
            }
            k=1;
        }
    //================
    }
    for (j=1;j<=size+msoverlap;j++){
        synwave[j]=5*synwave[j];
    }
//  m_test=size;
    UpdateData(FALSE);
    char buffer[1000] ;
    // display output (listbox)
    for( j=1;j<=size+msoverlap;j++){//size+msoverlap
    sprintf(buffer,"%f \n ",synwave[j] );
        m_xout.AddString(buffer);
    }
// display coeff (listbox)
    for(  j=1;j<=151;j++){
    sprintf(buffer,"%f \n ",ccoeff[1][j]);
        m_xcoeff.AddString(buffer);
    }
// display gain (listbox)
    for( j=1;j<nframe+1;j++){
    sprintf(buffer,"%f \n ",cgain[j] );
        m_xgain.AddString(buffer);
    }
// display pitch (listbox)
    for( j=1;j<nframe+1;j++){
    sprintf(buffer,"%f \n ",cpitch[j] );
        m_xpitch.AddString(buffer);
    }
```

```
int L=10000;
    if (m_input=="1")
        L=44100;
    if (m_input=="2")
        L=44100;

    //-------------------------------------
    sample_rate=16000;
    ofstream st("Test.wav");
    init_wav_file(st, "Test.wav");
//float t = 0;
    for ( i = 0; i < L ; ++i) {//sample_rate
    write_sample (st,synwave[i]);
//t += 1.0/sample_rate;
    }
    close_wav_file (st);
    //-------------------------------------
    m_oribit=16000*8/1000 ; //samplingrate 16000
    m_transbit=60.6;//(order*8+12)*50/1000=60.6 (order=150)
    m_comp=(1-m_transbit/m_oribit)*100;
    UpdateData(FALSE);

}

void CTotalView::OnButOri()
{
    // TODO: Add your control notification handler code here
    if(!dsk6x_open(pfname,&handle))
    {
    m_display="Can't open driver, DSK board is not connected.";
    UpdateData(FALSE);
    return ;
    }

    // Reset the C6x DSP on the open board
    if(!dsk6x_reset_dsp(handle, little_endian, HPI_boot))
    {
    m_display="Can't reset DSP. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Open the Host Port Interface on the open board
    if(!dsk6x_hpi_open(handle))
    {
    m_display="Can't open HPI. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Get the board information from the open board
    if(!dsk6x_board_type(handle,&board_type,&rev_id) )
    {
    m_display="Can't get board information. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
```

```
    return ;
    }

    // Display the board type and revision number
    switch (board_type)
    {
    case 0:
    m_display2="Unknown board, found and opened.";
    UpdateData(FALSE);
        break;
    case 1:
    m_display2="C6211DSK, found and opened.";
    UpdateData(FALSE);
        break;
    case 2:
    m_display2="C6711DSK, found and opened.";
    UpdateData(FALSE);
        break;
    }
    m_status="Loading COFF file SoundStudio.OUT...";
    UpdateData(FALSE);

    // Load the COFF file HOST_SIN.OUT to run the sin generator
    coff_ret=dsk6x_coff_load(handle,sound_file,verbose_flag,clear_bss_f
lag,dump_flag);
    if(coff_ret)
    {
        AfxMessageBox("Can't load HOST_SIN.");
        dsk6x_close(handle);
        return ;
    }
    m_status2="LPC.OUT loaded fine.";
    UpdateData(FALSE);

    // Get the board information from the open board
    if (!dsk6x_hpi_generate_int(handle))
    {
        AfxMessageBox("Can't start SounStudio.");
        return ;
    }
    m_status3="LPC.OUT started fine.";
    UpdateData(FALSE);

    if (m_input=="a")
    PlaySound("a.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="b")
    PlaySound("b.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="c")
    PlaySound("c.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="d")
    PlaySound("d.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="e")
    PlaySound("e.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="f")
    PlaySound("f.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="g")
    PlaySound("g.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="h")
```

```
    PlaySound("h.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="i")
    PlaySound("i.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="j")
    PlaySound("j.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="k")
    PlaySound("k.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="l")
    PlaySound("l.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="m")
    PlaySound("m.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="n")
    PlaySound("n.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="o")
    PlaySound("o.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="p")
    PlaySound("p.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="q")
    PlaySound("q.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="r")
    PlaySound("r.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="s")
    PlaySound("s.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="t")
    PlaySound("t.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="u")
    PlaySound("u.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="v")
    PlaySound("v.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="w")
    PlaySound("w.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="x")
    PlaySound("x.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="y")
    PlaySound("y.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="z")
    PlaySound("z.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="1")
    PlaySound("s2ofwb.wav",NULL,SND_SYNC|SND_FILENAME);
    if (m_input=="2")
    PlaySound("s1ofwb.wav",NULL,SND_SYNC|SND_FILENAME);

    dsk6x_reset_dsp(handle, little_endian, HPI_boot);
}

void CTotalView::OnButSound()
{
    // TODO: Add your control notification handler code here
    if(!dsk6x_open(pfname,&handle))
    {
    m_display="Can't open driver, DSK board is not connected.";
    UpdateData(FALSE);
    return ;
    }

    // Reset the C6x DSP on the open board
    if(!dsk6x_reset_dsp(handle, little_endian, HPI_boot))
    {
```

```
    m_display="Can't reset DSP. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Open the Host Port Interface on the open board
    if(!dsk6x_hpi_open(handle))
    {
    m_display="Can't open HPI. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Get the board information from the open board
    if(!dsk6x_board_type(handle,&board_type,&rev_id) )
    {
    m_display="Can't get board information. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Display the board type and revision number
    switch (board_type)
    {
    case 0:
    m_display2="Unknown board, found and opened.";
    UpdateData(FALSE);
        break;
    case 1:
    m_display2="C6211DSK, found and opened.";
    UpdateData(FALSE);
        break;
    case 2:
    m_display2="C6711DSK, found and opened.";
    UpdateData(FALSE);
        break;
    }
    m_status="Loading COFF file SoundStudio.OUT...";
    UpdateData(FALSE);

    // Load the COFF file HOST_SIN.OUT to run the sin generator
    coff_ret=dsk6x_coff_load(handle,sound_file,verbose_flag,clear_bss_f
lag,dump_flag);
    if(coff_ret)
    {
        AfxMessageBox("Can't load HOST_SIN.");
        dsk6x_close(handle);
        return ;
    }
    m_status2="LPC.OUT loaded fine.";
    UpdateData(FALSE);

    // Get the board information from the open board
    if (!dsk6x_hpi_generate_int(handle))
    {
```

```
        AfxMessageBox("Can't start SounStudio.");
        return ;
    }
    m_status3="LPC.OUT started fine.";
    UpdateData(FALSE);
    PlaySound("Test.wav",NULL,SND_SYNC|SND_FILENAME);
    dsk6x_reset_dsp(handle, little_endian, HPI_boot);
//  PlaySound("Test.wav",NULL,SND_SYNC|SND_FILENAME);
    CEdit *textbox;
    textbox = (CEdit*)GetDlgItem(IDC_EDIT1);
    textbox->SetSel(0,-1);
    textbox->SetFocus();
    textbox->Clear();
}

void CTotalView::OnButLoad()
{
    // TODO: Add your control notification handler code here
    m_boardc.ResetContent();
    m_boardg.ResetContent();
    m_boardp.ResetContent();
    m_boardo.ResetContent();
    /* Open a C6xDSK board on an SPP parallel port at address 0x0378 */
    if(!dsk6x_open(pfname,&handle))
    {
    m_display="Can't open driver, DSK board is not connected.";
    UpdateData(FALSE);

        return ;
    }
    /* Reset the C6x DSP on the open board */
    if(!dsk6x_reset_dsp(handle, little_endian, HPI_boot))
    {
    m_display="Can't reset DSP. 0x%lx";
    UpdateData(FALSE);
        dsk6x_close(handle);
        return ;
    }
    /* Open the Host Port Interface on the open board */
    if(!dsk6x_hpi_open(handle))
    {
    m_display="Can't open HPI. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
        return ;
    }
    /* Get the board information from the open board */
    if(!dsk6x_board_type(handle,&board_type,&rev_id) )
    {
    m_display="Can't get board information. 0x%lx";
    UpdateData(FALSE);
        dsk6x_close(handle);
        return ;
    }
    /* Display the board type and revision number */
    switch (board_type)
    {
    case 0:
```

```
    m_display2="Unknown board, found and opened.";
    UpdateData(FALSE);
        break;
    case 1:
    m_display2="C6211DSK, found and opened.";
    UpdateData(FALSE);
        break;
    case 2:
    m_display2="C6711DSK, found and opened.";
    UpdateData(FALSE);
        break;
    }
    m_status="Loading COFF file SoundStudio.OUT...";
    UpdateData(FALSE);
    /* Load the COFF file HOST_SIN.OUT to run the sin generator */
    coff_ret=dsk6x_coff_load(handle,input_file,verbose_flag,clear_bss_f
lag,dump_flag);
    if(coff_ret)
    {
        AfxMessageBox("Can't load HOST_SIN.");
        dsk6x_close(handle);
        return ;
    }
    m_status2="LPC.OUT loaded fine.";
    UpdateData(FALSE);
    /* Get the board information from the open board */
    if (!dsk6x_hpi_generate_int(handle))
    {
        AfxMessageBox("Can't start SounStudio.");
        return ;
    }
    m_status3="LPC.OUT started fine.";
    UpdateData(FALSE);
//=========================================================================
    int count=0,speech[16500],i=0,num=1;
    /* Wait for user input to change the gain of the sin wave being out
put by the DSK */
/*7999*/while (count!=15999){
    dsk6x_hpi_read(handle, DSK_mem2, &DSK_mem_length, DSK_mem_addr2);
    count=DSK_mem2[0];
    }

/*7999*/if (count==15999){
/*8000*/for (i=0;i<15999;i++){

    while (num=0){
    dsk6x_hpi_read(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3);
    num=DSK_mem3[0];     }

    while(num!=1){
    dsk6x_hpi_read(handle, DSK_mem1, &DSK_mem_length, DSK_mem_addr1);
    speech[i]=DSK_mem1[0];
    input[i]=speech[i];

    num=1;
    DSK_mem3[0]=num;
    dsk6x_hpi_write(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3);
        }
```

```
    while(num!=0){
    dsk6x_hpi_read(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3);
    num=DSK_mem3[0];
    }

    }
    m_status="";
    m_status2="";
    m_status3="";
    m_display2="";
    m_display="load in completed";
    UpdateData(FALSE);
    }
}

void CTotalView::OnButProcess()
{
    // TODO: Add your control notification handler code here
    double data1[10500],data3[10500];
    int data2[10500],i;

    for (i=1;i<=10000;i++){
        data3[i]=input[i];
        data1[i]=data3[i]/10000;
        if (data1[i]!=1){

            data2[i]=data1[i];
        }
        else{

            data2[i]=0;
        }

        data1[i]=data1[i]-data2[i];

    }

    sample_rate=8000;
    ofstream st("original.wav");
    init_wav_file(st, "original.wav");

    for (i = 0; i < 10000 ; ++i) {//sample_rate
    write_sample (st,data1[i]);

    }
    close_wav_file (st);
//=========================================================================
    //==========process=====================
    double data[10500],ramp[161],voice[500];
    double ccoeff[55][102],synwave[10500],overlap[161],ramp1[161],x[500
];
    int duration=10000,nframe=0,size,order=34;
    int msfr=320,msfs=480,msoverlap;
    int t,j,m,n,index,k=1,p=1;
    double total=0;
    int run=0;
    //inverse
```

```
    double numerator,denominator,kk[155],AA[155],err[155],A[155],gain;
    int count,g=1,l,pv=1,inv,v;
    double errs[500],cgain[55];
        numerator=0;
        denominator=0;
        count=1;
        // errs
        msoverlap=msfs-msfr;
        //test
    //.............................................

    double temp=0,autoc[500],rss[500];
    data1[0]=0;
        //================pre em
    for (index=1;index<=duration;index++){
        data[index]=data1[index]-0.9378*data1[index-1];
    }
//.....................Analysis.............................
    //++++++++++++++++++++++++++++++++++++++
    for (index=1;index<=duration-msfs+1;index=index+msfr){
        for (i=index;i<=index+msfs-1;i++){
        x[p]=data[i];   //data per frame (480 sample)
        p++;
        }
        p=1;
        nframe=nframe+1;// count frame

        //,,,,,,,,,,,,,,,find autocorrelation,,,,,,,,,,,,
            for (m=0;m<msfs;m++)    // use 49(order) but coeff =50
        {
            for(n=1;n<=msfs-m;n++)
            {
                temp=temp+x[n]*x[n+m];
            }
            autoc[m]=temp;
            rss[m]=autoc[m];
            temp=0;
        }
            //===============invese=================
                //------------------ invese matrix RSS ------------
            err[1]= rss[0];
            kk[1]=0;A[0]=1;A[1]=0;
            for (inv=1;inv<order+1;inv++){
                for(v=1;v<count;v++){
                numerator=numerator+A[v]*rss[count-pv];
                pv=pv+1;
                }
                numerator=numerator+g*rss[count];
                pv=1;
                denominator=-1*err[inv];
                kk[inv]=numerator/denominator;
            //-----flipud[A]-------
            if (v==1){
                AA[1]=0;
            }
            else{
            for(l=1;l<count;l++){
            AA[l]=A[count-l];
```

```
            }

            }
                for (j=1;j<count;j++){
                temp=temp+kk[v]*AA[j];
                A[j]=A[j]+temp;
                temp=0;
            }
            A[count]=kk[inv];

            err[inv+1]=(1-kk[inv]*kk[inv])*err[inv];
            numerator=0;
            count=count+1;
            }
            count=1;
            for(j=1;j<=order+1;j++){
            //ccoeff[nframe][1]=1;
            ccoeff[nframe][j]=A[j-1];
            }


            //..............error.....................
            for (inv=1;inv<=order;inv++){
                for (v=0;v<=run;v++){
                total=total+A[v]*x[inv-v];
                }
                run++;
                errs[inv]=total;
                total=0;
            }run=0;

            for(inv=order+1;inv<=msfs;inv++){
                errs[inv]=A[0]*x[inv]+A[1]*x[inv-1]+A[2]*x[inv-2]+A[3]*
x[inv-3]+A[4]*x[inv-4]+A[5]*x[inv-5]+A[6]*x[inv-6]+A[7]*x[inv-7]+A[8]*x
[inv-8]+A[9]*x[inv-9]+A[10]*x[inv-10]+A[11]*x[inv-11]+A[12]*x[inv-12]+A
[13]*x[inv-13]+A[14]*x[inv-14]+A[15]*x[inv-15]+A[16]*x[inv-16]+A[17]*x[
inv-17]+A[18]*x[inv-18]+A[19]*x[inv-19]+A[20]*x[inv-20]+A[21]*x[inv-21]
+A[22]*x[inv-22]+A[23]*x[inv-23]+A[24]*x[inv-24]+A[25]*x[inv-25]+A[26]*
x[inv-26]+A[27]*x[inv-27]+A[28]*x[inv-28]+A[29]*x[inv-29]+A[30]*x[inv-3
0]+A[31]*x[inv-31]+A[32]*x[inv-32]+A[33]*x[inv-33]+A[34]*x[inv-34]+A[35
]*x[inv-35]+A[36]*x[inv-36]+A[37]*x[inv-37]+A[38]*x[inv-38]+A[39]*x[inv
-39]+A[40]*x[inv-40]+A[41]*x[inv-41]+A[42]*x[inv-42]+A[43]*x[inv-43]+A[
44]*x[inv-44]+A[45]*x[inv-45]+A[46]*x[inv-46]+A[47]*x[inv-47]+A[48]*x[i
nv-48]+A[49]*x[inv-49];
            }
        //,,,,,,,,,,,,,,,,,,gain,,,,,,,,,,,,,,,,,,,,
                gain=sqrt(err[order+1]);
                cgain[nframe]=gain;
        //.....................pitch.....................
//              cpitch=1;
    }
//  int data4[950];
//  t=1;
    for (i=1;i<=nframe;i++){
        for (j=1;j<=order+1;j++){
            //if (j<=)
            ccoeff[i][j]=ccoeff[i][j]*10000;
            //data4[t]=ccoeff[i][j];
```

```
            //t++;
        }
        cgain[i]=cgain[i]*10000;
        //data4[t]=cgain[i];
        //t++;
    }


//=========================================================================
    if(!dsk6x_open(pfname,&handle))
    {
    m_display="Can't open driver, DSK board is not connected.";
    UpdateData(FALSE);
    return ;
    }

    // Reset the C6x DSP on the open board
    if(!dsk6x_reset_dsp(handle, little_endian, HPI_boot))
    {
    m_display="Can't reset DSP. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Open the Host Port Interface on the open board
    if(!dsk6x_hpi_open(handle))
    {
    m_display="Can't open HPI. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Get the board information from the open board
    if(!dsk6x_board_type(handle,&board_type,&rev_id) )
    {
    m_display="Can't get board information. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Display the board type and revision number
    switch (board_type)
    {
    case 0:
    m_display2="Unknown board, found and opened.";
    UpdateData(FALSE);
        break;
    case 1:
    m_display2="C6211DSK, found and opened.";
    UpdateData(FALSE);
        break;
    case 2:
    m_display2="C6711DSK, found and opened.";
    UpdateData(FALSE);
        break;
```

```
    }
    m_status="Loading COFF file SoundStudio.OUT...";
    UpdateData(FALSE);

    // Load the COFF file HOST_SIN.OUT to run the sin generator
    coff_ret=dsk6x_coff_load(handle,output_file,verbose_flag,clear_bss_
flag,dump_flag);
    if(coff_ret)
    {
        AfxMessageBox("Can't load HOST_SIN.");
        dsk6x_close(handle);
        return ;
    }
    m_status2="LPC.OUT loaded fine.";
    UpdateData(FALSE);

    // Get the board information from the open board
    if (!dsk6x_hpi_generate_int(handle))
    {
        AfxMessageBox("Can't start SounStudio.");
        return ;
    }
    m_status3="LPC.OUT started fine.";
    UpdateData(FALSE);
//=====================================================================
    int num=1,wait,data5[1000];

for ( nframe=1;nframe<=30;nframe++){
    for ( i=1;i<=35;i++){
    //=========================================
    while (num!=2){
    dsk6x_hpi_read(handle, DSK_mem1, &DSK_mem_length, DSK_mem_addr1);
    num=DSK_mem1[0];
    }
    wait=0;

    DSK_mem3[0]=ccoeff[nframe][i];//data4[i];
    dsk6x_hpi_write(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3);

    while (wait!=1){
    dsk6x_hpi_read(handle, DSK_mem2, &DSK_mem_length, DSK_mem_addr2);
    wait=DSK_mem2[0];
    dsk6x_hpi_read(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3);
    data5[i]=DSK_mem3[0];//fix

    }
    num=1;
    DSK_mem1[0]=num;
    dsk6x_hpi_write(handle, DSK_mem1, &DSK_mem_length, DSK_mem_addr1);
    }
    //=========================================
    while (num!=2){
    dsk6x_hpi_read(handle, DSK_mem1, &DSK_mem_length, DSK_mem_addr1);
    num=DSK_mem1[0];
    }
    wait=0;

    DSK_mem3[0]=cgain[nframe];//data4[i];
```

```
    dsk6x_hpi_write(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3);

    while (wait!=1){
    dsk6x_hpi_read(handle, DSK_mem2, &DSK_mem_length, DSK_mem_addr2);
    wait=DSK_mem2[0];
    dsk6x_hpi_read(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3);
    data5[i]=DSK_mem3[0];//fix

    }
    num=1;
    DSK_mem1[0]=num;
    dsk6x_hpi_write(handle, DSK_mem1, &DSK_mem_length, DSK_mem_addr1);
    }
    //=================================================================
    int frame=1,voice1[31][500];
    for (frame=1;frame<=30;frame++){
        for (i=1;i<=481;i++){
        while (wait!=2){
            dsk6x_hpi_read(handle, DSK_mem2, &DSK_mem_length, DSK_mem_a
ddr2);
            wait=DSK_mem2[0];
        }
        dsk6x_hpi_read(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3
);
        voice1[frame][i]=DSK_mem3[0];
        voice2[frame][i]=voice1[frame][i];
        num=3;
        DSK_mem1[0]=num;
        dsk6x_hpi_write(handle, DSK_mem1, &DSK_mem_length, DSK_mem_addr
1);
        wait=1;
        }
    }

/*  int frame=1,eff[31][50],gn[31];
    for (frame=1;frame<=30;frame++){
        for (i=1;i<=31;i++){
        while (wait!=2){
            dsk6x_hpi_read(handle, DSK_mem2, &DSK_mem_length, DSK_mem_a
ddr2);
            wait=DSK_mem2[0];
        }
        dsk6x_hpi_read(handle, DSK_mem3, &DSK_mem_length, DSK_mem_addr3
);
        eff[frame][i]=DSK_mem3[0];
        gn[frame]=eff[frame][31];
        num=3;
        DSK_mem1[0]=num;
        dsk6x_hpi_write(handle, DSK_mem1, &DSK_mem_length, DSK_mem_addr
1);
        wait=3;//1
        }
    }*/

//  dsk6x_reset_dsp(handle, little_endian, HPI_boot);

    m_status="";
    m_status2="";
```

```
    m_status3="";
    m_display2="";
    m_display="process completed";
    UpdateData(FALSE);
    char buffer[1000] ;

// display coeff (listbox)
    for(  j=1;j<=order+1;j++){
    sprintf(buffer,"%f \n ",ccoeff[1][j]);
        m_boardc.AddString(buffer);
    }
// display gain (listbox)
    for( j=1;j<order+1;j++){
    sprintf(buffer,"%f \n ",cgain[j] );
        m_boardg.AddString(buffer);
    }
    m_input="a";
/*  CEdit *textbox;
    textbox = (CEdit*)GetDlgItem(IDC_EDIT1);
    textbox->SetSel(0,-1);
    textbox->SetFocus();
    textbox->Clear();*/
    double cpitch[31];
    for ( index=1;index<nframe;index++){
    //check fram and sound
    //  UpdateData(TRUE);
        m_pSet->MoveFirst();
        while(m_input!=m_pSet->m_sound||index!=m_pSet->m_frame){
            m_pSet->MoveNext();
        }
        //get pitch coeff and gain
            cpitch[index]=m_pSet->m_pitch;
    }
// display pitch (listbox)
    for( j=1;j<order+1;j++){
    sprintf(buffer,"%f \n ",cpitch[j] );
        m_boardp.AddString(buffer);
    }
    m_input="";

}

void CTotalView::OnButBoardSound()
{
    // TODO: Add your control notification handler code here
    double voice3[32][500];
    double synwave[10500],ramp[161],overlap[161],ramp1[161];
    int voice4[32][500],index,j,k=1,frame,i;
    int size=320,msfs=480,msfr=320,msoverlap,duration=10000,nframe=30;
    msoverlap=msfs-msfr;
    ramp[1]=0;

    for (frame=1;frame<=30;frame++){
        for (i=1;i<=480;i++){
            voice3[frame][i]=voice2[frame][i]/10000;
            if(voice3[frame][i]>1){
                voice4[frame][i]=voice3[frame][i];
                voice3[frame][i]=voice3[frame][i]-voice4[frame][i];
```

```
            }
        }
        }
        for (j=2;j<=msoverlap;j++){
            ramp[j]=ramp[j-1]+0.0062893081761006289;//1/(msoverlap-1);
        }

        //=========================
        for (index=1;index<=nframe;index++){
                    if(index==1){
            for(j=1;j<=320;j++){//msfr
                synwave[j]=voice3[index][j];
            }
            }else{
                for (j=1;j<=160;j++){       //msoverlap          //middle
                    synwave[size+j]=overlap[j]+voice3[index][j]*ramp[j];
                }
                for (j=160+1;j<=320;j++){//msoverlap msfr       //final
                    synwave[size+j]=voice3[index][j];//msoverlap
                }
                // combine
                //size=msoverlap+msfr-msoverlap+size;
                size=320+size;
            }

            if (index==nframe){
                for (j=320+1;j<=480;j++){//msfr msfs
                synwave[size+j-320]=voice3[index][j];
                }
            }else{
                //=======flipud
                for (j=1;j<=160;j++){//msoverlap
                ramp1[j]=ramp[160-j+1];//msoverlap
                }
                //========
                for(j=320+1;j<=480;j++){//msfr msfs
                overlap[k]=voice3[index][j]*ramp1[k];
                k++;
                }
                k=1;
            }
        //=========================
        }
        //=================================================================
====
        if(!dsk6x_open(pfname,&handle))
        {
        m_display="Can't open driver, DSK board is not connected.";
        UpdateData(FALSE);
        return ;
        }

        // Reset the C6x DSP on the open board
        if(!dsk6x_reset_dsp(handle, little_endian, HPI_boot))
        {
        m_display="Can't reset DSP. 0x%lx";
        UpdateData(FALSE);
        dsk6x_close(handle);
```

```
    return ;
    }

    // Open the Host Port Interface on the open board
    if(!dsk6x_hpi_open(handle))
    {
    m_display="Can't open HPI. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Get the board information from the open board
    if(!dsk6x_board_type(handle,&board_type,&rev_id) )
    {
    m_display="Can't get board information. 0x%lx";
    UpdateData(FALSE);
    dsk6x_close(handle);
    return ;
    }

    // Display the board type and revision number
    switch (board_type)
    {
    case 0:
    m_display2="Unknown board, found and opened.";
    UpdateData(FALSE);
        break;
    case 1:
    m_display2="C6211DSK, found and opened.";
    UpdateData(FALSE);
        break;
    case 2:
    m_display2="C6711DSK, found and opened.";
    UpdateData(FALSE);
        break;
    }
    m_status="Loading COFF file SoundStudio.OUT...";
    UpdateData(FALSE);

    // Load the COFF file HOST_SIN.OUT to run the sin generator
    coff_ret=dsk6x_coff_load(handle,sound_file,verbose_flag,clear_bss_f
lag,dump_flag);
    if(coff_ret)
    {
        AfxMessageBox("Can't load HOST_SIN.");
        dsk6x_close(handle);
        return ;
    }
    m_status2="LPC.OUT loaded fine.";
    UpdateData(FALSE);

    // Get the board information from the open board
    if (!dsk6x_hpi_generate_int(handle))
    {
        AfxMessageBox("Can't start SounStudio.");
        return ;
    }
```

```
    m_status3="LPC.OUT started fine.";
    UpdateData(FALSE);
//=============================================================================
    sample_rate=8000;
    ofstream st("syn.wav");
    init_wav_file(st, "syn.wav");

    for ( i = 0; i < 10000 ; ++i) {//sample_rate
    write_sample (st,synwave[i]);

    }
    close_wav_file (st);
    //--------------------------------------
    PlaySound("syn.wav",NULL,SND_SYNC|SND_FILENAME);
    dsk6x_reset_dsp(handle, little_endian, HPI_boot);
    m_status="";
    m_status2="";
    m_status3="";
    m_display2="";
    m_display="sound completed";
    UpdateData(FALSE);

    char buffer[1000] ;
    // display output (listbox)
    for( j=1;j<=size+msoverlap;j++){//size+msoverlap
    sprintf(buffer,"%f \n ",synwave[j] );
        m_boardo.AddString(buffer);
    }

}
```