การพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

นาย นพดล จตุไพบูลย์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2553

DEVELOPMENT OF AN ELECTRONIC STETHOSCOPE PROTOTYPE

Mr. Noppadon Jatupaiboon

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2010

หัวข้อวิทยานิพนธ์	การพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
โดย	นาย นพดล จตุไพบูลย์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร. เศรษฐา ปานงาม
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	ดร. พศิน อิศรเสนา ณ อยุธยา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร. บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. อติวงศ์ สุชาโต)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ผู้ช่วยศาสตราจารย์ ดร. เศรษฐา ปานงาม)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม
(ดร. พศิน อิศรเสนา ณ อยุธยา)

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร. ภาณุทัต บุญประมุข)

นพดล จตุไพบูลย์ : การพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์. (DEVELOPMENT OF AN ELECTRONIC STETHOSCOPE PROTOTYPE) อ. ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร. เศรษฐา ปานงาม, อ. ที่ปรึกษาวิทยานิพนธ์ร่วม : ดร. พศิน อิศรเสนา ณ อยุธยา, 85 หน้า.

ในปัจจุบันได้มีการผลิตหูฟังแพทย์แบบอิเล็กทรอนิกส์ขึ้นมาเพื่อแก้ไขข้อจำกัดของหูฟังแพทย์แบบดั้งเดิม ข้อดีของหูฟังแพทย์แบบอิเล็กทรอนิกส์มีมากมาย เนื่องจากเสียงถูกส่งผ่านในรูปของสัญญาณไฟฟ้า ทำให้สามารถปรับปรุงคุณภาพเสียง บันทึกข้อมูลเสียง และส่งข้อมูลเสียงไร้สาย ซึ่งสิ่งเหล่านี้เป็นประโยชน์อย่างมาก ต่อการวินิจฉัยโรคในระบบโทรเวช แต่หูฟังแพทย์แบบอิเล็กทรอนิกส์ในปัจจุบันมีราคาค่อนข้างสูง งานวิจัยนี้จึงได้นำเสนอการออกแบบและพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ ที่สามารถกรองความถี่เสียงให้อยู่ในช่วงที่ใช้ตรวจฟัง สามารถลดเสียงรบกวนภายนอกโดยใช้ตัวกรองแบบปรับตัวได้ และสามารถทำการประมวลผลสัญญาณดิจิตอลแบบเวลาจริง จากผลการทดลองพบว่าสามารถลดเสียงรบกวนภายนอกได้ประมาณ 5 dB โดยทำให้เสียงที่ใช้ในการตรวจฟังเปลี่ยนแปลงไปบ้าง

ภาควิชา : .....วิศวกรรมคอมพิวเตอร์.... ลายมือชื่อนิสิต : .....
สาขาวิชา : .....วิศวกรรมคอมพิวเตอร์... ลายมือชื่อ อ. ที่ปรึกษาวิทยานิพนธ์หลัก : .....
ปีการศึกษา : .....2553.....ลายมือชื่อ อ. ที่ปรึกษาวิทยานิพนธ์ร่วม : .....

5170342621 : MAJOR COMPUTER ENINEERING

KEYWORDS : ELECTRONIC STETHOSCOPE / NOISE REDUCTION / ADAPTIVE FILTER

NOPPADON JATUPAIBOON : DEVELOPMENT OF AN ELECTRONIC STETHOSCOPE PROTOTYPE. ADVISOR : ASST.PROF. SETHA PAN-NGUM, Ph.D., CO-ADVISOR : PASIN ISARASENA, Ph.D., 85 pp.

Electronic stethoscope is developed to improve the limitations and add functions to a conventional stethoscope. In electronic stethoscope, sound is transformed into electrical signal which offers various benefits including signal quality improvement, signal record and signal transfer via wireless. Since these are all very useful for telemedicine, electronic stethoscope in the market is expensive. Therefore, this research aims to design and develop a low-cost electronic stethoscope prototype that can filter frequency, reduce external noise by adaptive filtering and make real-time digital signal processing. The results show that it can reduce external noises about 5 dB and some signal distortion is apparent.

Department : ...Computer Engineering...Student's Signature : .....
Field of Study : ..Computer Engineering... Advisor's Signature: .....
Academic Year : .....2010.....Co-Advisor's Signature : .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความอนุเคราะห์อย่างยิ่งของ ผศ.ดร. เศรษฐา ปานงาม และ ดร. พศิน อิศรเสนา ณ อยุธยา ซึ่งท่านได้ให้ความรู้ แนะนำแนวทางการวิจัย ตรวจสอบให้คำแนะนำ และสนับสนุนเป็นอย่างดี จนทำให้การวิจัยในครั้งนี้สำเร็จออกมาด้วยดี

ขอขอบคุณ นาย ภควัฒน์ ดับโศก ที่คอยแนะนำเรื่องฮาร์ดแวร์ ขอขอบคุณ นางสาว พัชรา สุวิจิตรพงษ์ ที่ให้คำแนะนำเรื่องหูฟังแพทย์ ขอขอบคุณครอบครัว ที่คอยติดตาม ให้กำลังใจและสนับสนุน รวมถึงท่านอื่น ๆ ที่มิได้กล่าวชื่อไว้ ณ ที่นี้ ที่มีส่วนช่วยให้วิทยานิพนธ์สำเร็จได้ด้วยดี

สารบัญ

หน้า

สารบัญตาราง

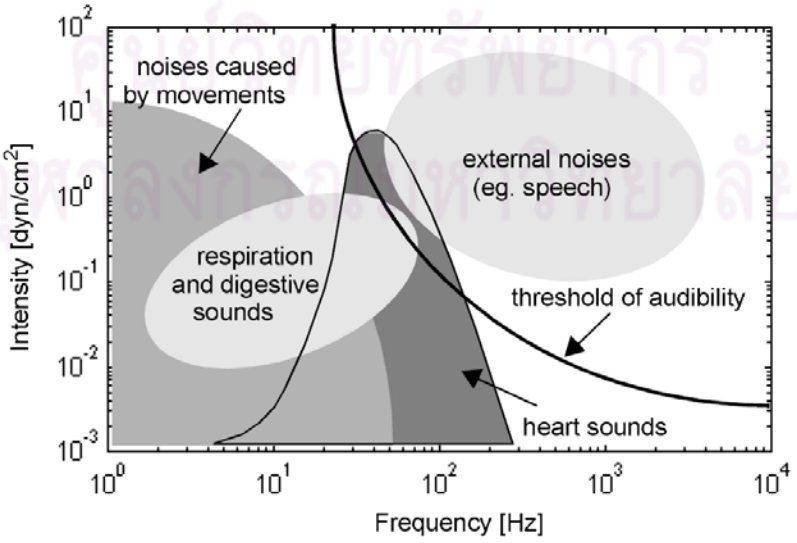
สารบัญภาพ

ภาพที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

หูฟังแพทย์ (Stethoscope) เป็นอุปกรณ์ทางการแพทย์พื้นฐานที่ใช้ในการตรวจฟัง (Auscultation) เสียงจากการทำงานของอวัยวะต่างๆ หูฟังแพทย์ที่มีขายในปัจจุบันสามารถแบ่งได้เป็น 2 ประเภท คือ หูฟังแพทย์แบบดั้งเดิม (Conventional Stethoscope) และหูฟังแพทย์แบบอิเล็กทรอนิกส์ (Electronic Stethoscope) ข้อเสียของหูฟังแพทย์แบบดั้งเดิม คือ ไม่สามารถขยายเสียงให้ดังขึ้นได้ ทำให้มีปัญหาเมื่อต้องการตรวจฟังเสียงที่เบามากๆ ในปัจจุบันได้มีการผลิตหูฟังแพทย์แบบอิเล็กทรอนิกส์ขึ้นมาเพื่อแก้ไขข้อจำกัดของหูฟังแพทย์แบบดั้งเดิม นอกจากนี้ ข้อดีของหูฟังแพทย์แบบอิเล็กทรอนิกส์ยังมีมากมาย เนื่องจากเสียงถูกส่งผ่านในรูปของสัญญาณไฟฟ้า ทำให้สามารถบันทึกข้อมูลเสียง ส่งข้อมูลไร้สาย ปรับปรุงคุณภาพสัญญาณเสียง และสามารถแสดงผลข้อมูลเสียงในรูปแบบของภาพและเสียง ซึ่งทั้งหมดนี้เป็นประโยชน์ในระบบโทรเวช (Telemedicine) และสามารถใช้ในระบบการเรียนการสอนได้ [1] แม้ว่าหูฟังแพทย์แบบอิเล็กทรอนิกส์จะมีข้อดีมากมาย แต่ปัญหาที่พบบ่อยในหูฟังแพทย์แบบอิเล็กทรอนิกส์ คือ ปัญหาเสียงรบกวน เนื่องจากหูฟังแพทย์แบบอิเล็กทรอนิกส์จะใช้ไมโครโฟนเป็นตัวรับเสียงซึ่งมีความไวต่อเสียง ทำให้มีโอกาสรับเสียงอื่นๆ ที่ไม่ใช่เสียงที่ต้องการจะตรวจฟังมาด้วย ดังแสดงในรูปที่ 1 โดยเฉพาะเสียงรบกวนภายนอก (External Noise) ซึ่งเป็นสาเหตุสำคัญที่ทำให้เสียงจากการตรวจฟังยากที่จะวินิจฉัย [2], [3]



รูปที่ 1 เสียงต่างๆ ที่มีโอกาสเกิดขึ้นขณะบันทึกเสียงหัวใจ [3]

แม้ว่าการกรองความถี่เสียงให้อยู่ในช่วงที่ใช้ตรวจฟัง จะสามารถลดเสียงรบกวนภายนอกได้ส่วนหนึ่ง แต่ก็ไม่สามารถลดเสียงรบกวนภายนอกที่มีความถี่อยู่ในช่วงที่ใช้ตรวจฟังได้ ดังนั้นจึงควรมีวิธีจำเพาะในการลดเสียงรบกวนภายนอก ในปัจจุบันมีหูฟังแพทย์แบบอิเล็กทรอนิกส์เพียงไม่กี่ยี่ห้อที่มีความสามารถในการลดเสียงรบกวนภายนอก ซึ่งได้แก่ 3M และ Thinklabs จึงเป็นแรงกระตุ้นให้ผู้ทำวิจัย ออกแบบและพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ที่มีความสามารถในการลดเสียงรบกวนภายนอก เพื่อปรับปรุงคุณภาพของเสียงที่ใช้ในการตรวจฟังให้ดีขึ้น

1.2 วัตถุประสงค์ของการวิจัย

เพื่อออกแบบและพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ ที่สามารถกรองความถี่เสียงให้อยู่ในช่วงที่ใช้ตรวจฟัง สามารถลดเสียงรบกวนภายนอกที่เกิดขึ้น และสามารถทำการประมวลสัญญาณดิจิตอลแบบเวลาจริง

1.3 ขอบเขตของการวิจัย

- ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์สามารถกรองความถี่เสียง ให้อยู่ในช่วงที่ใช้ตรวจฟัง 3 ช่วง ได้แก่ ช่วง Bell จะกรองเสียงให้อยู่ในช่วงความถี่ 0-200 Hz ช่วง Diaphragm จะกรองเสียงให้อยู่ในช่วงความถี่ 100-500 Hz และช่วง Extended จะกรองเสียงให้อยู่ในช่วงความถี่ 0-2000 Hz
- ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์สามารถลดเสียงรบกวนภายนอกที่มีความดังสม่ำเสมอได้
- ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์สามารถทำการประมวลสัญญาณดิจิตอลแบบเวลาจริงได้

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ได้ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ ที่สามารถกรองความถี่เสียงให้อยู่ในช่วงที่ใช้ตรวจฟัง สามารถลดเสียงรบกวนภายนอกที่เกิดขึ้น และสามารถทำการประมวลสัญญาณดิจิตอลแบบเวลาจริง ซึ่งสามารถนำไปพัฒนาต่อจนผลิตใช้ได้จริงในประเทศ เป็นการลดการนำเข้าเครื่องมือแพทย์จากต่างประเทศซึ่งมีราคาแพง และเป็นการพัฒนาความรู้ทางด้านวิศวกรรมการแพทย์

1.5 วิธีดำเนินการทำวิจัย

- ศึกษาข้อมูลเอกสารและงานวิจัยที่เกี่ยวข้องกับหูฟังแพทย์
- ศึกษาชิ้นส่วนอุปกรณ์ต่างๆ ที่เป็นส่วนประกอบของหูฟังแพทย์แบบอิเล็กทรอนิกส์

- ออกแบบต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
- เลือกอุปกรณ์ต่างๆ และนำมาประกอบเป็นต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
- เขียนโปรแกรมจำลองการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
- เขียนโปรแกรมเพื่อกรองความถี่เสียงให้อยู่ในช่วงที่ใช้ตรวจฟังบนต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
- เขียนโปรแกรมเพื่อลดเสียงรบกวนภายนอกบนต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
- ทดสอบประสิทธิภาพของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
- ปรับปรุงประสิทธิภาพของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์
- สรุปผลการวิจัยและจัดทำวิทยานิพนธ์

1.6 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตอบรับให้ตีพิมพ์เป็นบทความทางวิชาการในหัวข้อเรื่อง Development of an Electronic Stethoscope Prototype [4] โดย นายนพดล จตุไพบูลย์ ผศ.ดร. เศรษฐา ปานงาม และ ดร. พศิน อิศรเสนา ณ อยุธยา ในงานประชุมวิชาการ ECTI-Conference on Application Research and Development (ECTI-CARD 2010) ณ โรงแรมจอมเทียนปาล์มบีช พัทยา จังหวัดชลบุรี ประเทศไทย วันที่ 10-12 พฤษภาคม พ.ศ. 2553 และในหัวข้อเรื่อง Electronic Stethoscope Prototype with Adaptive Noise Cancellation โดย นายนพดล จตุไพบูลย์ ผศ.ดร. เศรษฐา ปานงาม และ ดร. พศิน อิศรเสนา ณ อยุธยา ในงานประชุมวิชาการ IEEE ICT & Knowledge Engineering (IEEE ICT & KE 2010) ณ มหาวิทยาลัยสยาม จังหวัดกรุงเทพมหานคร ประเทศไทย วันที่ 24-25 พฤศจิกายน พ.ศ. 2553

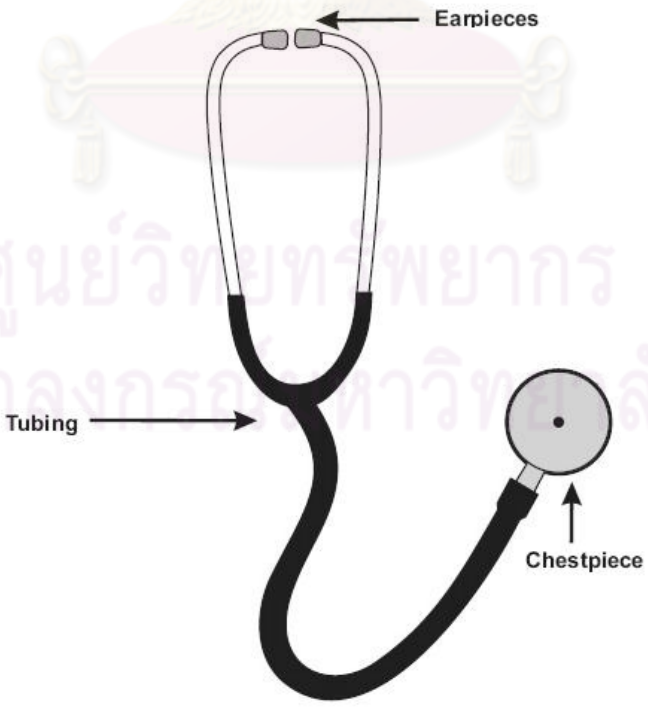
บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 หูฟังแพทย์

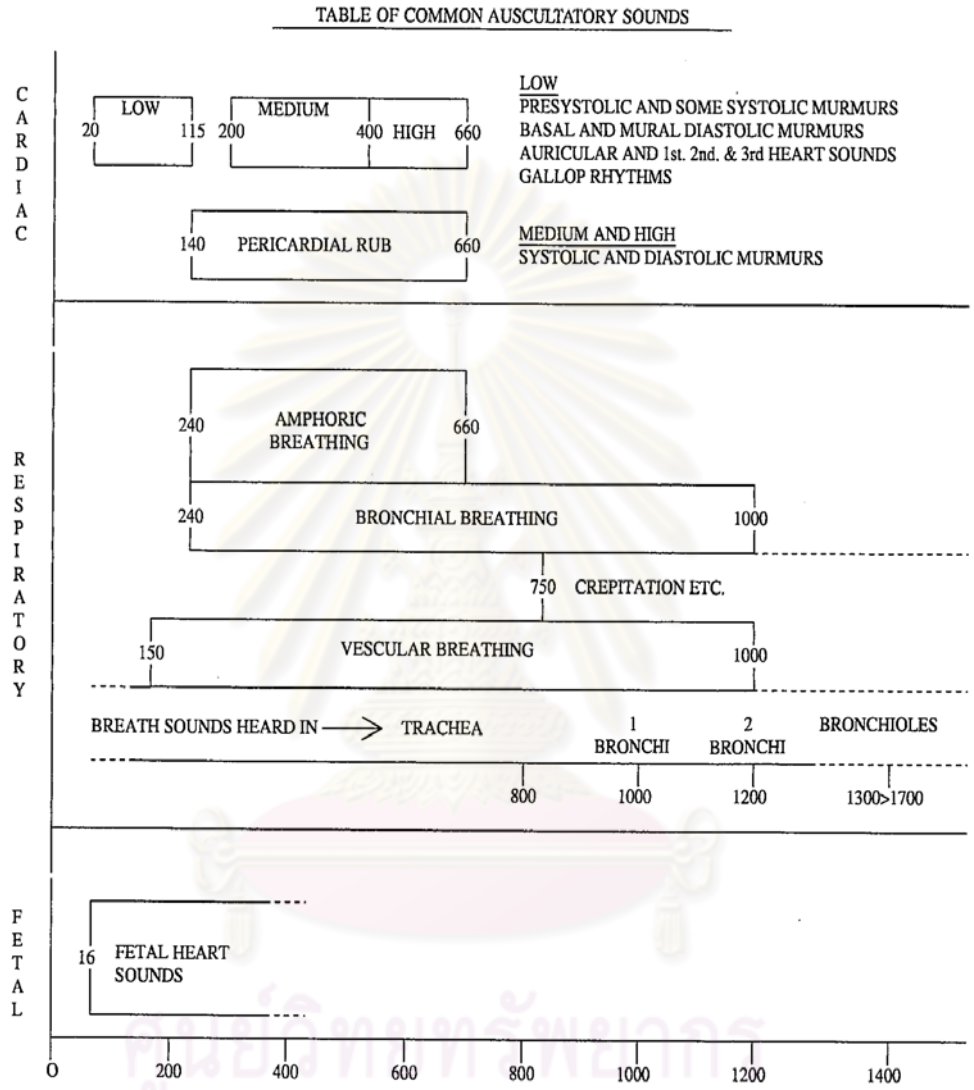
หูฟังแพทย์ (Stethoscope) เป็นอุปกรณ์ทางการแพทย์พื้นฐานที่ใช้ในการตรวจฟัง เสียงจากการทำงานของอวัยวะต่างๆ โดยเฉพาะที่ ระบบหัวใจ (Cardiac) และระบบการหายใจ (Respiratory) ส่วนประกอบของหูฟังแพทย์โดยทั่วไปจะประกอบด้วย 3 ส่วนหลัก ดังแสดงในรูปที่ 2 ดังนี้ [1]

- Earpieces ใช้ใส่หูเพื่อฟัง ควรมีขนาดเหมาะสมกับรูหู
- Chestpiece ใช้วางแนบตรงตำแหน่งที่จะตรวจฟัง ส่วนใหญ่มี 2 ด้าน คือ ด้าน Bell ซึ่งมีผลตอบสนองเสียงความถี่ต่ำได้ดี และด้าน Diaphragm ซึ่งมีผลตอบสนองเสียงความถี่สูงได้ดี
- Tubing เป็นท่อนำเสียงจากส่วน Chestpiece ไปถึงส่วน Earpieces



รูปที่ 2 ส่วนประกอบหลักของหูฟังแพทย์

จากรูปที่ 2 แสดงให้เห็นว่า เสียงที่เกี่ยวกับระบบหัวใจ มีความถี่อยู่ในช่วง 20-660 Hz และ เสียงที่เกี่ยวกับระบบการหายใจ มีความถี่อยู่ในช่วง 150-2000 Hz ดังนั้นเสียงที่ใช้ในการตรวจฟังส่วนใหญ่มีความถี่อยู่ในช่วง 20-2000 Hz



รูปที่ 3 แสดงช่วงความถี่ของเสียงที่ใช้ในการตรวจฟังโดยทั่วไป [2]

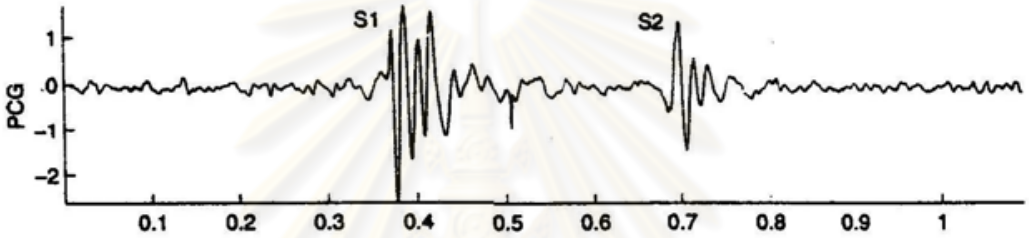
2.1.2 **เสียง**

เสียง คือ พลังงานรูปหนึ่งที่เกิดจากการสั่นสะเทือนของโมเลกุลของอากาศ ทำให้เกิด การอัดและขยายสลับกันของโมเลกุลอากาศ ความดันบรรยากาศจึงเกิดการเปลี่ยนแปลงตามการเคลื่อนที่ ของโมเลกุลอากาศ เรียกว่า คลื่นเสียง ความดังเสียงขึ้นอยู่กับความสูงหรือแอมปลิจูด (Amplitude) ส่วนความทุ้มแหลมของเสียงขึ้นกับความถี่ (Frequency) ของคลื่นเสียง [5] เสียงที่เกียวกับงานวิจัยนี้ มีดังนี้

1. <u>เสียงหัวใจ</u> [6]

เสียงหัวใจเกิดจาก การบีบตัว (Systole) คลายตัว (Diastole) ของหัวใจ การปิดเปิดของลิ้นหัวใจ และการเปลี่ยนแปลงการไหลเวียนของเลือดในหัวใจ โดยในแต่ละรอบการเต้นของหัวใจจะมีเสียงหัวใจหลักๆ ดังแสดงในรูปที่ 4 ดังนี้

- เสียงหัวใจที่หนึ่ง (S1) เป็นเสียงความถี่ต่ำ (Lubb)
- เสียงหัวใจที่สอง (S2) เป็นเสียงความถี่สูง (Dubb)
- เสียงหัวใจที่สาม (S3) เป็นเสียงที่เกิดหลัง S2 เล็กน้อย ปกติเสียงนี้จะไม่ได้ยิน
- เสียงหัวใจที่สี่ (S4) เป็นเสียงที่เกิดก่อน S1 เล็กน้อย ปกติเสียงนี้จะไม่ได้ยิน



รูปที่ 4 สัญญาณเสียงหัวใจ

นอกจากเสียงหัวใจหลักๆ ที่กล่าวมาแล้ว ยังมีเสียงหัวใจผิดปกติ ที่เรียกว่า เมอเมอ (Murmur) โดยเสียง Murmur แบ่งออกเป็น 3 ชนิด ดังนี้

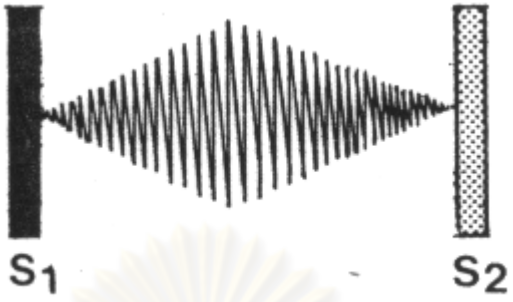
1.1. Systolic Murmur เป็นเสียงที่เกิดระหว่าง S1 กับ S2 แบ่งเป็น 2 ชนิด ดังนี้

1.1.1. Pan Systolic Murmur เป็นเสียงที่เกิดหลัง S1 และอาจดังตลอดจนถึง S2 โดยความดังของเสียงเท่ากันตลอด ดังแสดงในรูปที่ 5

รูปที่ 5 ลักษณะของเสียง Pan Systolic Murmur

1.1.2. Ejection Systolic Murmur เป็นเสียงที่เกิดหลัง S1 และอาจดังตลอดจนถึง S2 โดยความดังของเสียงจะค่อยๆ เพิ่มขึ้นจนถึงจุดสูงสุด ซึ่งมักจะ

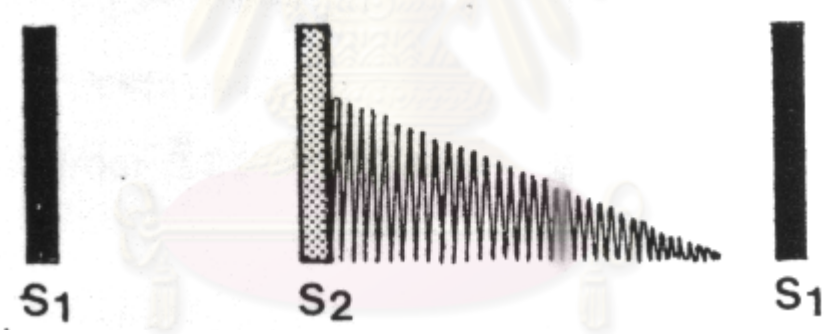
อยู่บริเวณกึ่งกลางระหว่าง S1 และ S2 แล้วค่อยๆลดลง ดังแสดงในรูปที่ 6



รูปที่ 6 ลักษณะของเสียง Ejection Systolic Murmur

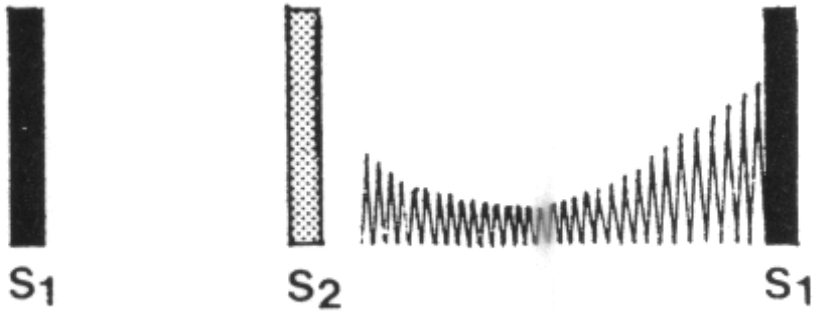
1.2. Diastolic Murmur เป็นเสียงที่เกิดระหว่าง S2 กับ S1 ในรอบถัดไป แบ่งเป็น 2 ชนิด ดังนี้

1.2.1. Diastolic Blowing Murmur เป็นเสียงที่เกิดหลัง S2 และอาจดังตลอดจนถึง S1 โดยความดังของเสียงจะค่อยๆลดลง ดังแสดงในรูปที่ 7



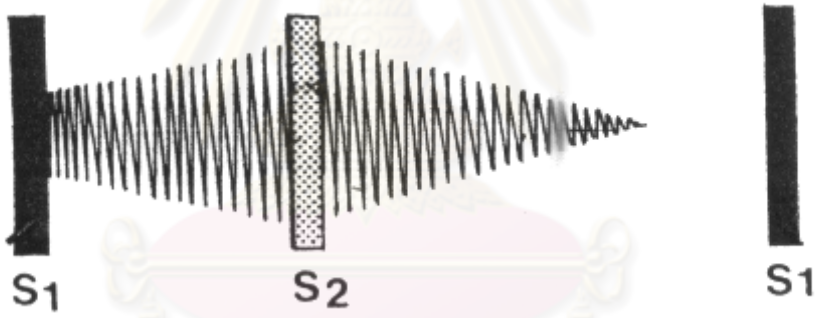
รูปที่ 7 ลักษณะของเสียง Diastolic Blowing Murmur

1.2.2. Diastolic Rumble Murmur เป็นเสียงที่เกิดหลัง S2 และอาจดังตลอดจนถึง S1 โดยความดังของเสียงจะค่อยๆลดลงจนถึงจุดต่ำสุดแล้วค่อยๆเพิ่มขึ้น ดังแสดงในรูปที่ 8



รูปที่ 8 ลักษณะของเสียง Diastolic Rumble Murmur

1.3. Continuous Murmur เป็นเสียงที่เกิดระหว่าง S1 กับ S1 ในรอบถัดไป โดยความดังของเสียงจะค่อยๆเพิ่มขึ้นจนถึงจุดสูงสุด ซึ่งมักจะอยู่บริเวณ S2 แล้วค่อยๆ ลดลง ดังแสดงในรูปที่ 9



รูปที่ 9 ลักษณะของเสียง Continuous Murmur

2. <u>เสียงรบกวน (Noise)</u> [7]

เสียงรบกวน คือ เสียง เสียงที่นอกเหนือจากที่คนสนใจ เพราะทำให้เกิดการรบกวนการรับรู้เสียงที่ต้องการหรือความเงียบ เสียงรบกวน โดยทั่วไปจะแบ่งออกได้เป็น 4 ประเภท ดังนี้

2.1. เสียงที่ดังสม่ำเสมอ (Steady State Noise) เป็นเสียงที่ต่อเนื่องที่มีลักษณะ และความเข้มของเสียงที่ค่อนข้างคงที่ คือ ไม่เปลี่ยนแปลงเกินกว่า 5 dB ใน 1 วินาที เสียงชนิดนี้ ได้แก่ เสียงเครื่องทอผ้า เสียงเครื่องจักร เสียงพัดลม เสียงเครื่องยนต์ไอพ่น

2.2. เสียงที่เปลี่ยนแปลงระดับเสมอ (Fluctuating Noise) เป็นเสียงที่มีความเข้มสูงๆ ต่ำๆ การเปลี่ยนแปลงของระดับเสียงนั้นเกินกว่า 5 dB ใน 1 วินาที เสียงชนิดนี้ ได้แก่ เสียงเลื่อยวงเดือน กบไสไม้ไฟฟ้า เสียงไซเรน เป็นต้น

2.3. เสียงกระทบ (Impulsive Noise) เป็นเสียงที่เกิดขึ้นแล้วค่อยๆ หายไป เสียงกระทบนี้จะมีระยะเวลาที่เกิดน้อยกว่า 0.5 วินาที และระดับความดังเสียงจะเปลี่ยนแปลงไปอย่างน้อย 40 dB ภายในระยะเวลานั้น เสียงกระทบอาจจะเกิดขึ้นติด ๆ กัน หรือ อาจจะเกิดขึ้นนาน ๆ ครั้งก็ได้ เสียงชนิดนี้ ได้แก่ เสียงตอกเสาเข็มในการก่อสร้าง หรือทุบโลหะ เสียงเครื่องย้ำหมุดเสียงระเบิด เป็นต้น

2.4. เสียงที่ดังเป็นระยะ (Intermittent Noise) เป็นเสียงที่ไม่ต่อเนื่อง ซึ่งจะแตกต่างจากเสียงกระทบ ในแง่ที่มีระยะเวลาที่ยาวนานกว่า และมีลักษณะที่ไม่แน่ชัด เสียงชนิดนี้ได้แก่ เสียงจากเครื่องอัดลม เสียงการจราจร เสียงเครื่องบินที่บินผ่านไปมา เป็นต้น

2.1.3 ระบบประมวลผลสัญญาณดิจิตอล [8]

ระบบประมวลผลสัญญาณดิจิตอลโดยส่วนใหญ่ ประกอบด้วยส่วนต่างๆ ดังแสดงในรูปที่ 10 ดังนี้

รูปที่ 10 ส่วนประกอบในระบบประมวลผลสัญญาณดิจิตอล

1. วงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล (Analog to Digital Converter, ADC)

วงจรประเภทนี้ก็คือ ตัวแปลงสัญญาณแอนะลอกเป็นดิจิตอล โดยทั่วไป เราจะใช้ตัวแปลงสัญญาณแอนะลอกเป็นดิจิตอลในรูปของวงจรรวมสำเร็จรูป ซึ่งสามารถแบ่งได้เป็น 2 กระบวนการย่อย คือ

1.1. วงจรสุ่มสัญญาณ (Sampler) สัญญาณขาเข้าของวงจรนี้เป็นสัญญาณต่อเนื่อง x(t) ส่วนสัญญาณขาออกเป็นสัญญาณไม่ต่อเนื่อง x(n) พารามิเตอร์วงจรสุ่มสัญญาณนี้ก็คือ ค่าอัตราการสุ่ม (Sampling Rate) ค่านี้เป็นตัวกำหนดว่า วงจรสุ่มจะสุ่มสัญญาณด้วยอัตรากี่ครั้งต่อวินาที หรือกี่เฮริตซ์ (Hz) ทฤษฎีการสุ่ม

สัญญาณ (Sampling Theorem) ระบุไว้ว่า เพื่อให้ได้สัญญาณที่สุ่มแล้วเป็นตัวแทนที่ถูกต้องของสัญญาณนี้ ความถี่ในการสุ่มจะต้องมีค่ามากกว่าสองเท่าของความถี่สูงสุดในสัญญาณซึ่งความถี่ที่นี้ มีชื่อเรียกว่า ความถี่ไนควิสท์ (Nyquist Frequency) ถ้าเราใช้ความถี่ในการสุ่มที่ต่ำกว่าค่าความถี่ไนควิสท์จะเกิดเอเลียดซิ่ง (Aliasing) ซึ่งเป็นผลให้สัญญาณขาเข้าที่สุ่มมาได้มีความผิดเพี้ยนไปก่อนที่จะเข้าไปในส่วนของการประมวลผล จึงจำเป็นอย่างยิ่งที่เราต้องพยายามไม่ให้ Aliasing เกิดขึ้น หรือให้เกิดน้อยที่สุดเท่าที่จะทำได้

1.2. วงจรแบ่งขั้นสัญญาณ (Quantizer) สัญญาณ x(n) ที่ได้จากวงจรสุ่มสัญญาณถือว่ามีความละเอียดเต็มที่ในทางขนาด ซึ่งในทางปฏิบัติเมื่อนำ ไปใช้งานจะต้องลดความละเอียดของ x(n) ลงให้สามารถแทนได้ด้วยสัญญาณดิจิตอลที่มีจำนวนบิตจำกัด กระบวนการลดความละเอียดนี้ เรียกว่า การแบ่งขั้นของสัญญาณ (Quantization) ความละเอียดที่ได้จากการแบ่งขั้นสัญญาณขึ้นอยู่กับจำนวนบิตที่จะใช้

2. <u>วงจรประมวลผลสัญญาณดิจิตอล (Digital Signal Processing, DSP)</u>

ส่วนนี้เป็นหัวใจหลัก ซึ่งทำหน้าที่ประมวลผลสัญญาณ x(n) เพื่อกระทำผลบางอย่างกับสัญญาณ เช่น เป็นวงจรกรองความถี่บางย่านออกและให้ผลลัพธ์ของการประมวลผลเป็นสัญญาณขาออก y(n) วงจรประมวลผลสัญญาณนี้ ถ้าจะพิจารณากันอย่างง่าย ๆ แท้ที่จริงก็คือตัวคำนวณนั่นเอง กล่าวได้ว่าเป็นการคำนวณหาสัญญาณขาออกจากสัญญาณขาเข้า โดยมองเห็นสัญญาณขาเข้าในลักษณะ ลำดับของค่า วงจรประเภทนี้จะมีไมโครโปรเซสเซอร์ ที่ออกแบบมาเฉพาะสำหรับงานการคำนวณทางด้านการประมวลผลสัญญาณแบบทำให้สามารถทำงานประเภทนี้ได้อย่างรวดเร็วกว่าการใช้ไมโครโปรเซสเซอร์ที่ออกแบบมาสำหรับงานทั่วๆไป

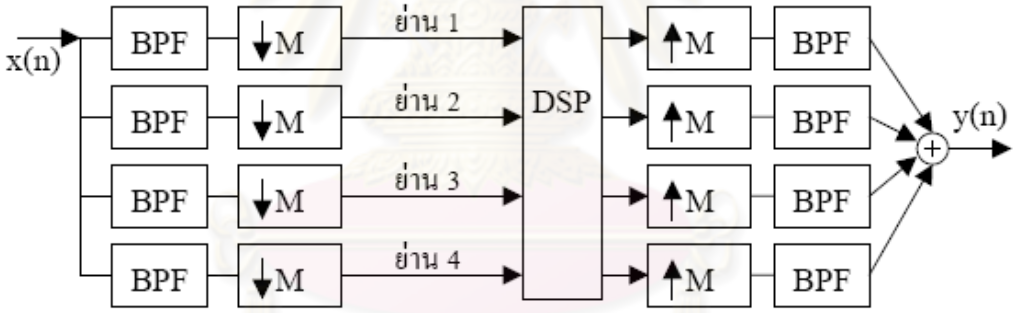
3. <u>วงจรสร้างสัญญาณคืน (Digital to Analog Converter, DAC)</u>

ใช้ในระบบที่มีสัญญาณขาออกสุดท้ายเป็นสัญญาณต่อเนื่อง (การประมวลผลสัญญาณบางอย่างต้องการสัญญาณขาออกเป็นไม่ต่อเนื่อง ก็ไม่จำเป็นต้องมีส่วนที่ 3 นี้) โดยทำหน้าที่แปลงสัญญาณไม่ต่อเนื่อง y(n) ให้กลับเป็นสัญญาณต่อเนื่อง y(t) ซึ่งจะเป็นสัญญาณขาออกสุดท้ายของระบบ วงจรประเภทนี้ก็คือ ตัวแปลงสัญญาณดิจิตอลเป็นแอนะลอกนั่นเอง ซึ่งก็มีในรูปแบบวงจรรวมสำเร็จรูปเช่นกัน

2.1.4 การประมวลผลโดยแยกย่านความถี่ย่อย [8]

การประมวลผลสัญญาณในงานบางอย่าง สามารถให้ผลที่ดีขึ้นได้ โดยแยกสัญญาณที่จะประมวลผลเป็นสัญญาณในย่านความถี่ย่อยๆ (Subband) แล้วประมวลผลในแต่ละย่านความถี่แยกจากกัน (หรืออาจใช้ผลในแต่ละย่านความถี่มาประมวลผลร่วมกันทีหลังก็ได้) ดังแสดงในรูปที่ 11 ซึ่งเป็นการแยกสัญญาณออกเป็น 4 ย่านความถี่

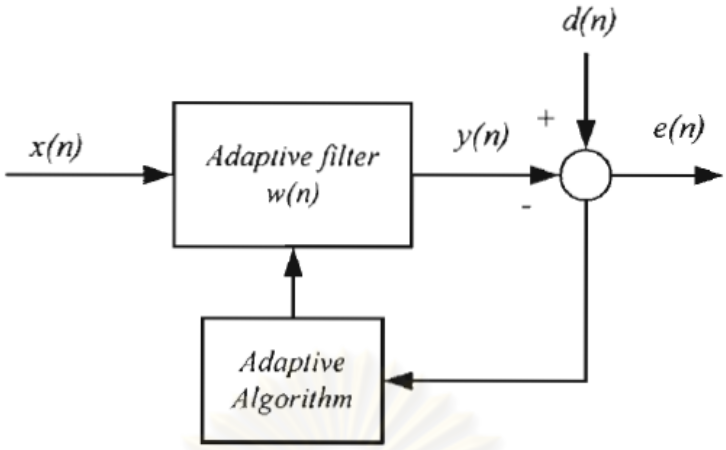
หลังจากประมวลผลแล้วก็จะนำ สัญญาณทั้งสี่มารวมกันเป็นสัญญาณขาออก เครื่องมือที่ใช้สำหรับแยกย่านความถี่ ก็คือ ตัวกรองดิจิตอล 4 ตัวที่เป็นแบบผ่านย่านความถี่ตัวละย่าน บางทีเรียกตัวกรองเหล่านี้รวมว่า แผงตัวกรอง (Filter Bank) เมื่อสัญญาณถูกแยกออกเป็น 4 ย่านย่อยแล้ว เนื่องจากแต่ละย่านจะมีแถบความถี่เป็น 1/4 ของแถบความถี่เดิม ดังนั้น เราสามารถลดอัตราการสุ่มข้อมูลของแต่ละย่านลงให้เหลือเพียง 1/4 ของอัตราเดิมได้ด้วย โดยที่ไม่เกิดเอเลียดซิ่ง ซึ่งจะทำให้อัตราการประมวลผลที่ต้องการลดลงมาก สำหรับการประมวลผลที่ต้องการสัญญาณขาออกจะก็แปลงให้แต่ละย่านมีอัตราสุ่มเท่าเดิม จากนั้นจึงนำสัญญาณแต่ละย่านมารวมกัน ก็จะได้สัญญาณขาออก



รูปที่ 11 การประมวลผลแบบแยกย่านความถี่ย่อย

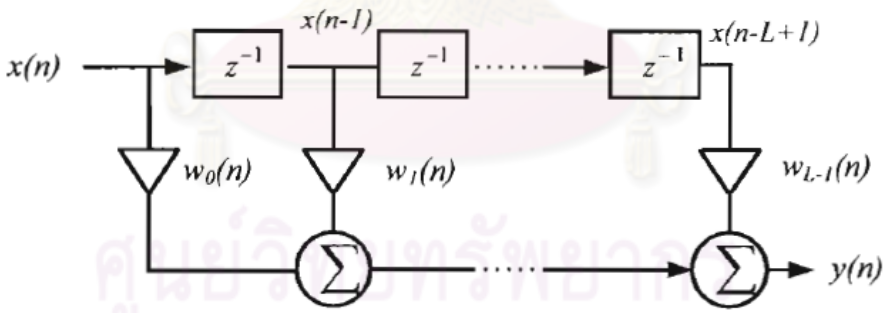
2.1.5 การลดเสียงรบกวนโดยตัวกรองแบบปรับตัวได้ [9]

ในภาวะที่สัญญาณเสียงรบกวน และสัญญาณเสียงที่เราต้องการอยู่ในย่านความถี่เดียวกัน ถ้าต้องการลดสัญญาณเสียงรบกวนส่วนใหญ่ให้หมดไป การใช้ตัวกรองแบบธรรมดา (Filter) จะทำให้สัญญาณเสียงที่ต้องการสูญหายไปด้วย เนื่องจากตัวกรองแบบธรรมดา เป็นตัวกรองแบบที่มีค่าน้ำหนักคงที่ แต่สำหรับตัวกรองแบบปรับตัวได้ (Adaptive Filter) จะเป็นตัวกรองแบบที่มีค่าน้ำหนักปรับเปลี่ยนตลอดเวลา โดยอาศัยโมเดลของสิ่งแวดล้อมที่สร้างขึ้นในการคำนวณหาค่าน้ำหนัก จึงทำให้สามารถลดสัญญาณเสียงรบกวนส่วนใหญ่ได้ โดยที่ยังคงสัญญาณเสียงที่ต้องการไว้ ซึ่งวิธีนี้มีชื่อเรียกว่า การลดเสียงรบกวนโดยตัวกรองแบบปรับตัวได้ (Adaptive Noise Cancellation, ANC)

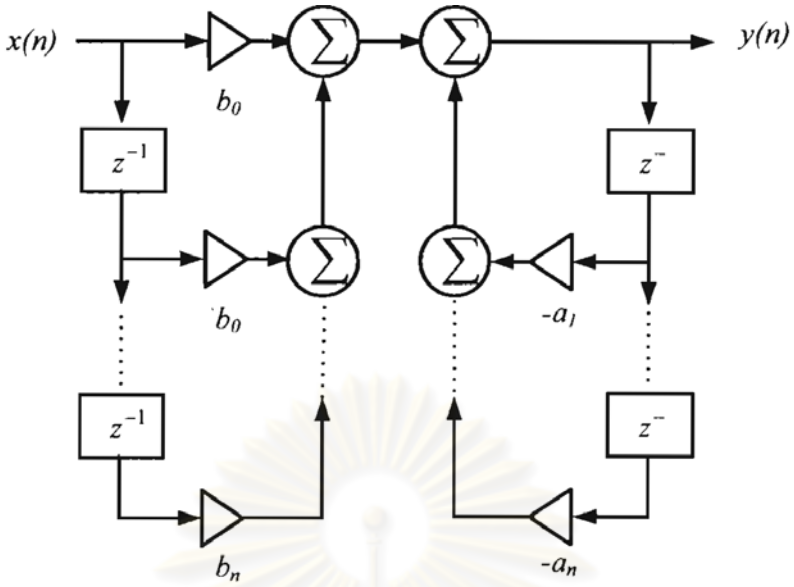


รูปที่ 12 โครงสร้างพื้นฐานของ ANC

จากรูปที่ 12 $d(n)$ คือ สัญญาณเสียงที่ต้องการที่มีสัญญาณเสียงรบกวน ณ ตำแหน่งที่สนใจ $x(n)$ คือ สัญญาณเสียงรบกวนจากแหล่งกำเนิด $y(n)$ คือ สัญญาณเสียงรบกวนขาออกที่ถูกปรับค่าให้ใกล้เคียงกับสัญญาณเสียงรบกวน ณ ตำแหน่งที่สนใจ $w(n)$ คือ ค่าน้ำหนัก และ $e(n)$ คือสัญญาณเสียงที่ผ่านการลดเสียงรบกวนแล้ว ระบบจะพยายามปรับ $e(n)$ ให้มีค่าต่ำที่สุด โดยการปรับเปลี่ยน $w(n)$ ของตัวกรองแบบปรับตัวได้ โดยรับ $w(n)$ มาจากการประมวลผลของอัลกอริทึมแบบปรับตัวได้ (Adaptive Algorithm)

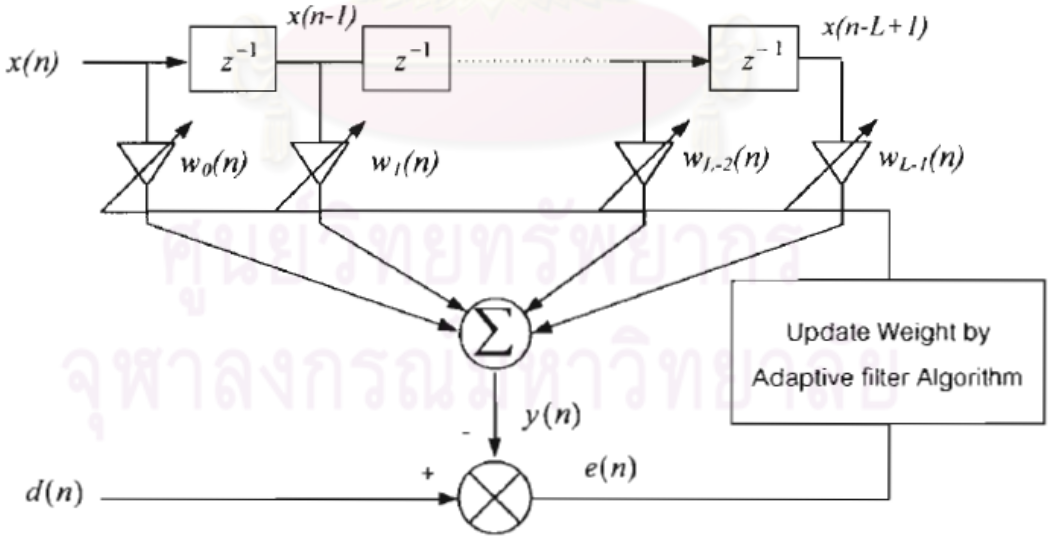


รูปที่ 13 ตัวกรองชนิด FIR



รูปที่ 14 ตัวกรองชนิด IIR

รูปแบบของตัวกรองแบบปรับตัวได้ สามารถแบ่งตามจำนวนค่าน้ำหนัก $w(n)$ ได้ คือ FIR (Finite Impulse Response) และ IIR (Infinite Impulse Response) ดังแสดงในรูปที่ 13 และ 14 ตัวกรอง FIR จะมีจำนวนค่าน้ำหนัก $w(n)$ จำกัด ส่วนตัวกรอง IIR จะมีจำนวนค่าน้ำหนัก $w(n)$ ไม่จำกัด ตัวกรอง FIR จะมีความเสถียรตลอดเวลา ส่วนตัวกรอง IIR มีโอกาสที่จะไม่เสถียร ตัวกรองแบบปรับตัวได้ ส่วนใหญ่จึงอยู่ในรูปแบบ FIR



รูปที่ 15 ตัวกรองแบบปรับตัวได้ชนิด FIR

จากรูปที่ 15 พิจารณา สัญญาณขาออก $y(n)$ จะได้ว่า

$$y(n) = \sum_{i=0}^{L-1} w_i(n)x(n-i) \qquad (1)$$

เมื่อ L คือค่าอันดับ (Order) ของอัลกอริทึมแบบปรับตัวได้

พิจารณา เวคเตอร์สัญญาณขาเข้า $\vec{x}(n)$ และ เวคเตอร์ค่าน้ำหนัก $\vec{w}(n)$ จะได้ว่า

$$\vec{x}(n) = [x(n) \quad x(n-1) \quad \cdots \quad x(n-L+1)]^T \quad (2)$$

$$\vec{w}(n) = [w_0(n) \quad w_1(n) \quad \cdots \quad w_{L-1}(n)]^T \quad (3)$$

ค่า $y(n)$ จากสมการที่ 1 สามารถเขียนใหม่ได้เป็น

$$y(n) = \vec{w}(n)^T \vec{x}(n) \quad (4)$$

พิจารณา สัญญาณความผิดพลาด $e(n)$ จะได้ว่า

$$e(n) = d(n) - y(n) \quad (5)$$

แทนค่า $y(n)$ จากสมการที่ 4 ลงในสมการที่ 5 จะได้ว่า

$$e(n) = d(n) - \vec{w}(n)^T \vec{x}(n) \quad (6)$$

อัลกอริทึมแบบปรับตัวได้ (Adaptive Algorithm) ที่ใช้ในการปรับเวคเตอร์ค่าน้ำหนัก $\vec{w}(n)$ เพื่อให้สัญญาณความผิดพลาด $e(n)$ มีค่าต่ำที่สุด มีอยู่หลายอัลกอริทึม เช่น LMS NLMS RLS Kalman ซึ่งแต่ละอัลกอริทึมก็จะมีการประมวลผล และประสิทธิภาพแตกต่างกัน งานวิจัยนี้เลือกใช้ อัลกอริทึม LMS และ NLMS เนื่องจากเป็นอัลกอริทึมที่มีความซับซ้อนในการประมวลผลต่ำ และมีความเร็วในการลู่เข้าหาคำตอบสุดท้ายที่ถูกต้องอยู่ในเกณฑ์ดี ต่อไปนี้จะอธิบายถึงวิธีการทำงานของอัลกอริทึมทั้งสองนี้

1. <u>อัลกอริทึม LMS (Least Mean Square)</u>

ในการปรับเวคเตอร์ค่าน้ำหนัก $\vec{w}(n)$ ของอัลกอริทึม LMS มีวัตถุประสงค์เพื่อให้ค่าความผิดพลาดเฉลี่ยกำลังสองเฉลี่ย $E[e^2(n)]$ มีค่าต่ำที่สุด

พิจารณา ฟังก์ชันวัตถุประสงค์ $J(n)$ จะได้ว่า

$$J(n) = E[e^2(n)] \quad (7)$$

ในทางปฏิบัติไม่สามารถหาค่า $E[e^2(n)]$ ได้จริง แต่สามารถประมาณค่าได้เป็น ค่าความผิดพลาดกำลังสอง $e^2(n)$ จะได้ว่า

$$J(n) = e^2(n) \quad (8)$$

พิจารณา ค่าเกรเดียนต์ของฟังก์ชันวัตถุประสงค์ $\nabla J(n)$ จากสมการที่ 8 จะได้ว่า

$$\nabla J(n) = 2[\nabla e(n)]e(n) \quad (9)$$

พิจารณา ค่าเกรเดียนต์ของค่าความผิดพลาด $\nabla e(n)$ จากสมการที่ 6 จะได้ว่า

$$\nabla e(n) = -\vec{x}(n) \tag{10}$$

แทนค่า $\nabla e(n)$ จากสมการที่ 10 ลงในสมการที่ 9 จะได้ว่า

$$\nabla J(n) = -2\vec{x}(n)e(n) \tag{11}$$

พิจารณา สมการ Steepest Descent

$$\vec{w}(n+1) = \vec{w}(n) - \frac{\mu}{2}\nabla J(n) \tag{12}$$

เมื่อ μ คือ ค่าอัตราการปรับตัว (Step Size)

แทนค่า $\nabla J(n)$ จากสมการที่ 11 ลงในสมการที่ 12 จะได้ว่า

$$\vec{w}(n+1) = \vec{w}(n) + \mu\vec{x}(n)e(n) \tag{13}$$

สมการที่ 13 คือ สมการปรับเวคเตอร์ค่าน้ำหนัก $\vec{w}(n)$ ของอัลกอริทึม LMS

ค่าอัตราการปรับตัว μ เป็นค่าที่ใช้ควบคุมเสถียรภาพของระบบ และความเร็วที่เวคเตอร์ค่าน้ำหนัก $\vec{w}(n)$ จะลู่เข้าสู่ค่าที่ถูกต้อง ถ้าค่า μ น้อยเกินไปจะส่งผลให้ $\vec{w}(n)$ ลู่เข้าสู่ค่าที่ถูกต้องช้า ในทางกลับกัน ถ้าค่า μ มากเกินไปก็อาจส่งผลให้ $\vec{w}(n)$ ไม่สามารถลู่เข้าสู่ค่าที่ถูกต้องได้ หรือถึงแม้จะลู่ได้แต่ก็อาจจะเปลี่ยนแปลงไปมา ไม่คงตัว ดังนั้นการเลือกค่า μ ที่เหมาะสมมีผลต่อประสิทธิผลการทำงานของระบบมาก โดยหลักการเลือกใช้ค่า μ ที่เหมาะสมสามารถแสดงได้ดังอสมการที่ 14

$$0 < \mu < \frac{2}{LP_x} \tag{14}$$

เมื่อ P_x คือ กำลังของสัญญาณขาเข้า และ L คือ ค่าอันดับ

2. <u>อัลกอริทึม NLMS (Normalized Least Mean Square)</u>

ในการปรับเวคเตอร์ค่าน้ำหนัก $\vec{w}(n)$ ของอัลกอริทึม NLMS มีวัตถุประสงค์เดียวกับอัลกอริทึม LMS แต่จะมีส่วนของการนอร์มอลไรซ์ (Normalize) สัญญาณขาเข้า

ค่า $\mu(n)$ จากสมการที่ 14 สามารถเขียนใหม่ได้เป็น

$$\mu(n) = \frac{\alpha}{LP_x(n)} \tag{15}$$

เมื่อ α คือ ค่าอัตราการปรับตัวนอร์มอลไรซ์ (Normalized Step Size) โดยที่ $0 < \alpha < 2$

กำหนดให้ความยาวของหน้าต่าง ที่ใช้ในการหาค่า $P_x(n)$ มีค่าเท่ากับ L

พิจารณา ค่า $P_x(n)$ จะได้ว่า

$$P_x(n) = \frac{\vec{x}(n)^T \vec{x}(n)}{L} \quad (16)$$

แทนค่า $P_x(n)$ จากสมการที่ 16 ลงในว่า

$$\mu(n) = \frac{\alpha}{\vec{x}(n)^T \vec{x}(n)} \quad (17)$$

ค่า $\mu(n)$ จากสมการที่ 17 สามารถเขียนใหม่ได้เป็น

$$\mu(n) = \frac{\alpha}{\delta + \vec{x}(n)^T \vec{x}(n)} \quad (18)$$

เมื่อ δ คือ ค่าคงที่ป้องกันตัวส่วนเป็นศูนย์ โดยที่ $\delta > 0$

แทนค่า $\mu(n)$ จากสมการที่ 18 ลงในสมการที่ 13 จะได้ว่า

$$\vec{w}(n+1) = \vec{w}(n) + \frac{\alpha}{\delta + \vec{x}(n)^T \vec{x}(n)} \vec{x}(n)e(n) \quad (19)$$

สมการที่ 19 คือ สมการปรับเวคเตอร์ค่าน้ำหนัก $\vec{w}(n)$ ของอัลกอริทึม NLMS

การนอร์มอลไรซ์ (Normalize) สัญญาณขาเข้า จะมีผลดีเมื่อค่ากำลังของสัญญาณขาเข้า $P_x(n)$ มีการเปลี่ยนแปลง เพราะ เมื่อค่า $P_x(n)$ เปลี่ยนแปลง จะส่งผลให้ค่า $\mu(n)$ เปลี่ยนแปลงด้วย ถ้าค่า $P_x(n)$ มากขึ้น จะส่งผลให้ค่า $\mu(n)$ ที่ถูกแทนด้วยเทอมของ $\frac{\alpha}{\delta + \vec{x}(n)^T \vec{x}(n)}$ น้อยลง ในทางกลับกัน ถ้าค่า $P_x(n)$ น้อยลง จะส่งผลให้ค่า $\mu(n)$ มากขึ้น ซึ่งตรงตามหลักการเลือกใช้ค่า $\mu(n)$ ที่เหมาะสม

2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัย [10] ได้นำเสนอวิธีการลดเสียงรบกวนในการบันทึกสัญญาณเสียงหัวใจ ผ่านการประมวลผลบนเครื่องคอมพิวเตอร์ ทำการบันทึกเสียง 2 ช่องทาง ไมโครโฟนตัวที่ 1 ถูกบรรจุในส่วน Chestpiece ของหูฟังแพทย์เพื่อรับสัญญาณเสียงหัวใจ และ ไมโครโฟนตัวที่ 2 ถูกวางไว้ด้านนอกเพื่อรับสัญญาณเสียงรบกวนภายนอก แล้วนำสัญญาณเสียงแอนะลอกทั้ง 2 มาแปลงเป็นสัญญาณเสียงดิจิตอล ที่อัตราสุ่ม 2 KHz แล้วจึงทำการลดเสียงรบกวนภายนอกโดยการทำ Adaptive Noise Cancellation Filter (ANC) โดยมีซอฟต์แวร์ที่สามารถวิเคราะห์หาอัตราการเต้นของหัวใจ ระยะเวลาของช่วง Systole และ Diastole ผลการทดสอบพบว่าสามารถลดเสียงรบกวนในการบันทึกสัญญาณเสียงหัวใจได้ประมาณ 16 dB

งานวิจัย [11] ได้นำเสนอการออกแบบและพัฒนาหูฟังแพทย์แบบอิเล็กทรอนิกส์ โดยผ่านบอร์ด AVR MCU โดยรับเสียงของอวัยวะที่ต้องการจะตรวจฟังผ่านทางไมโครโฟน แล้วนำสัญญาณเสียงมาผ่านวงจรแผงตัวกรองแอนะลอก เพื่อทำการแบ่งช่วงความถี่ที่ใช้ตรวจฟังดังนี้ คือ เสียงหัวใจ (30-500 Hz) เสียงปอด (100-1000 Hz) และเสียงหัวใจ-ปอด (30-1000 Hz) แล้วแสดงผลของเสียงผ่านทางหูฟัง หรือส่งสัญญาณเสียงผ่านทางบลูทูธไปยังเครื่องคอมพิวเตอร์ โดยมีซอฟต์แวร์ที่สามารถวิเคราะห์หาอัตราการเต้นของหัวใจ ผลการทดสอบพบว่า หูฟังแพทย์สามารถแสดงผลภาพสัญญาณเสียงหัวใจบนคอมพิวเตอร์ได้แบบ Real-time ผ่านการส่งข้อมูลเสียงแบบบลูทูธ

งานวิจัย [12] ได้นำเสนอการออกแบบและพัฒนาหูฟังแพทย์แบบอิเล็กทรอนิกส์ไร้สาย โดยรับเสียงของอวัยวะที่ต้องการจะตรวจฟังผ่านทางไมโครโฟน แล้วนำสัญญาณเสียงแอนะลอกมาผ่านวงจรขยายเสียง และวงจรกรองสัญญาณเสียงให้อยู่ในช่วงความถี่ 25-1500 Hz เพื่อใช้ในการตรวจฟังเสียงหัวใจและปอด แล้วนำสัญญาณเสียงแอนะลอกมาแปลงเป็นสัญญาณเสียงดิจิตอล ที่อัตราสุ่ม 8 kHz ก่อนจะถูกเข้ารหัสแบบ PCM (Pulse Code Modulation) แล้วส่งสัญญาณเสียงผ่านทางบลูทุธ เมื่อส่งเสร็จฝั่งรับจะทำการถอดรหัส และขยายสัญญาณเสียง แล้วแสดงผลของเสียงผ่านทางหูฟัง ผลการทดสอบพบว่า สัญญาณเสียงที่ได้พบว่าผิดเพี้ยนพอสมควร เนื่องจากวงจรกรอง

งานวิจัย [13] ได้นำเสนอการออกแบบและพัฒนาหูฟังแพทย์แบบอิเล็กทรอนิกส์เพื่อใช้ตรวจฟังเสียงหัวใจ ผ่านการประมวลบนบอร์ด DSP ทำการบันทึกเสียงผ่านสองช่องทาง ไมโครโฟนตัวที่ 1 ถูกบรรจุในส่วน Chestpiece ของหูฟังแพทย์เพื่อบันทึกสัญญาณเสียงหัวใจ และไมโครโฟนตัวที่ 2 ถูกวางไว้ด้านนอกเพื่อบันทึกสัญญาณเสียงรบกวนภายนอก แล้วนำสัญญาณเสียงแอนะลอกทั้ง 2 มาแปลงเป็นสัญญาณเสียงดิจิตอล ที่อัตราสุ่ม 8 kHz แล้วจึงทำการลดเสียงรบกวนภายนอกโดยการทำ Adaptive Noise Cancellation Filter (ANC) และ กรองเสียงให้อยู่ในช่วงความถี่ 25-500 Hz เพื่อใช้ตรวจฟังเสียงหัวใจ แล้วแสดงผลในรูปแบบของเสียงผ่านทางลำโพง และในรูปแบบของภาพสัญญาณเสียงหัวใจ นอกจากนี้ยังสามารถบันทึกสัญญาณเสียงลงในหน่วยความจำได้ ผลการทดสอบพบว่า เสียงหัวใจถูกกรองให้อยู่ในช่วงความถี่ 25-500 Hz และสามารถลดเสียงรบกวนภายนอกได้จริง

งานวิจัย [14] ได้นำเสนอหูฟังแพทย์แบบอิเล็กทรอนิกส์โดย ผ่านการประมวลบนบอร์ด DSP โดยรับเสียงของอวัยวะที่ต้องการจะตรวจฟังผ่านทางไมโครโฟน แล้วนำสัญญาณเสียงแอนะลอกมาแปลงเป็นสัญญาณเสียงดิจิตอล ที่อัตราสุ่ม 8 kHz แล้วนำมาผ่านแผงตัวกรอง (Filterbank) เพื่อแยกย่านความถี่ย่อยออกเป็น 16 ย่าน ย่านละ 250 Hz ทำให้สามารถปรับระดับ

เสียงในแต่ละย่านได้อิสระ เพื่อให้ได้ยินเสียงในช่วงความถี่ที่สนใจจะตรวจฟังได้ชัดเจนยิ่งขึ้น โดยแบ่งช่วงความถี่ดังนี้ คือ ช่วง Bell (0-500 Hz) ช่วง Diaphragm (0-1000 Hz) และช่วง Extend (0-1500 Hz) นอกจากนี้ยังสามารถบันทึกเสียงลงในหน่วยความจำและสามารถแสดงผลของเสียงในอัตราความเร็วปกติและความเร็วลดลงครึ่งหนึ่งได้ ผลการทดสอบพบว่า หูฟังแพทย์มี Dynamic Range ที่ประมาณ 75 dB และการขยายเสียงโดย DSP ในแต่ละช่วงความถี่มีค่าประมาณ 21 dB

จากงานวิจัยที่เกี่ยวข้องกับหูฟังแพทย์แบบอิเล็กทรอนิกส์ สามารถสรุปได้ว่า ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ ควรมีคุณสมบัติพื้นฐาน ดังนี้

- สามารถกรองความถี่เสียงให้อยู่ในช่วงที่ใช้ตรวจฟัง เพื่อลดเสียงรบกวนที่มีความถี่ไม่อยู่ช่วงที่ใช้ตรวจฟัง การกรองความถี่เสียงในหูฟังแพทย์แบบอิเล็กทรอนิกส์ สามารถทำได้โดยใช้ตัวกรองแอนะลอก พบได้ในงานวิจัย [11], [12] หรือ ตัวกรองดิจิตอล พบได้ในงานวิจัย [10], [13], [14]
- สามารถลดเสียงรบกวนภายนอก การใช้หูฟังแพทย์แบบอิเล็กทรอนิกส์ตรวจฟัง มักมีเสียงรบกวนจากภายนอกรวมเข้ามาด้วย ซึ่งบางส่วนมีความถี่อยู่ในช่วงเดียวกับที่ใช้ในการตรวจฟัง ทำให้ไม่สามารถใช้ตัวกรองธรรมดา ลดเสียงรบกวนภายนอกเหล่านี้ได้ การลดเสียงรบกวนภายนอกในหูฟังแพทย์แบบอิเล็กทรอนิกส์ สามารถทำได้โดยวิธี Adaptive Noise Cancellation (ANC) พบได้ในงานวิจัย [10] ซึ่งประมวลผลบนคอมพิวเตอร์ และ [13] ซึ่งทำการประมวลผลแบบเวลาจริงบนบอร์ด DSP

บทที่ 3
วิธีดำเนินการวิจัย

3.1 การออกแบบการทดลอง

ออกแบบการทดลองเป็น 2 ขั้นตอน คือ การทดลองบนโปรแกรมจำลองการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ และการทดลองบนต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

1. <u>การทดลองบนโปรแกรมจำลองการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์</u>

ในการทดลองนี้จะทำการวิเคราะห์ผลของการเปลี่ยนแปลงพารามิเตอร์ของอัลกอริทึมแบบปรับตัวได้ (Adaptive Algorithm) วิเคราะห์คุณภาพของสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกโดยอัลกอริทึมแบบปรับตัวได้ และทำการเปรียบการใช้ข้อมูลเสียงจำลอง (Simulated Data) และข้อมูลเสียงจริง (Actual Data)

2. <u>การทดลองบนต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์</u>

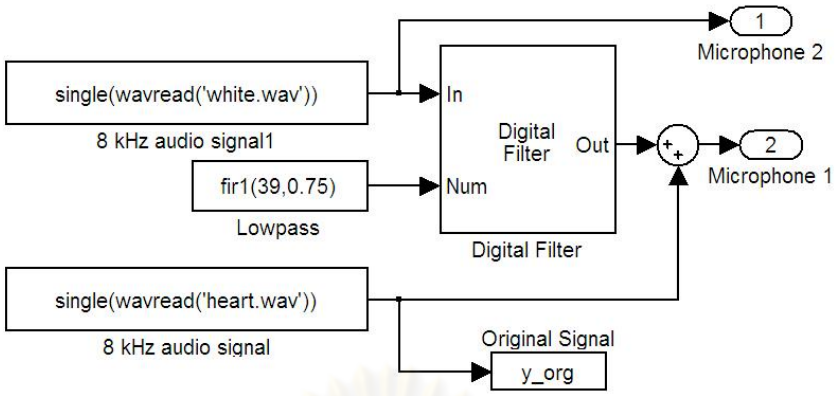
ในการทดลองนี้จะทำการวิเคราะห์คุณภาพของสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอก โดยการใช้ตัวกรองแบบปรับตัวได้ (Adaptive Filter)

3.2 โปรแกรมจำลองการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

งานวิจัยนี้เลือกใช้โปรแกรม Simulink ใน MATLAB โดยใช้ Blocksets ในส่วนของการประมวลผลสัญญาณ เพื่อสร้างข้อมูลเสียงจำลอง และทดสอบการทำงานของอัลกอริทึมแบบปรับตัวได้ (Adaptive Algorithm) 4 ชนิด คือ LMS (Least Mean Square), NLMS (Normalized Least Mean Square), S-LMS (Subband - Least Mean Square) และ S-NLMS (Subband - Normalized Least Mean Square)

1. <u>โปรแกรมสร้างข้อมูลเสียงจำลอง (Simulated Data)</u>

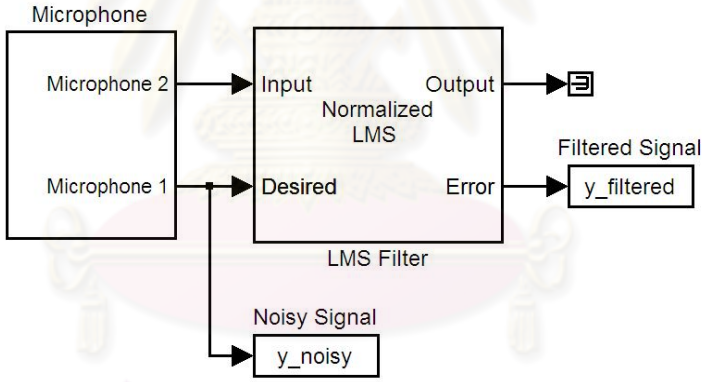
นำสัญญาณเสียงหัวใจมารวมกับสัญญาณเสียงรบกวนภายนอกที่ผ่านวงจรกรองความถี่ต่ำผ่านที่ 3000 Hz สัญญาณที่ได้ คือ สัญญาณเสียงของไมโครโฟนตัวที่ 1 ส่วนสัญญาณเสียงรบกวนภายนอก คือ สัญญาณเสียงของไมโครโฟนตัวที่ 2 ดังแสดงในรูปที่ 16



รูปที่ 16 โปรแกรมสร้างข้อมูลเสียงจำลอง

2. <u>โปรแกรมทดสอบการทำงานของอัลกอริทึม LMS และ NLMS</u>

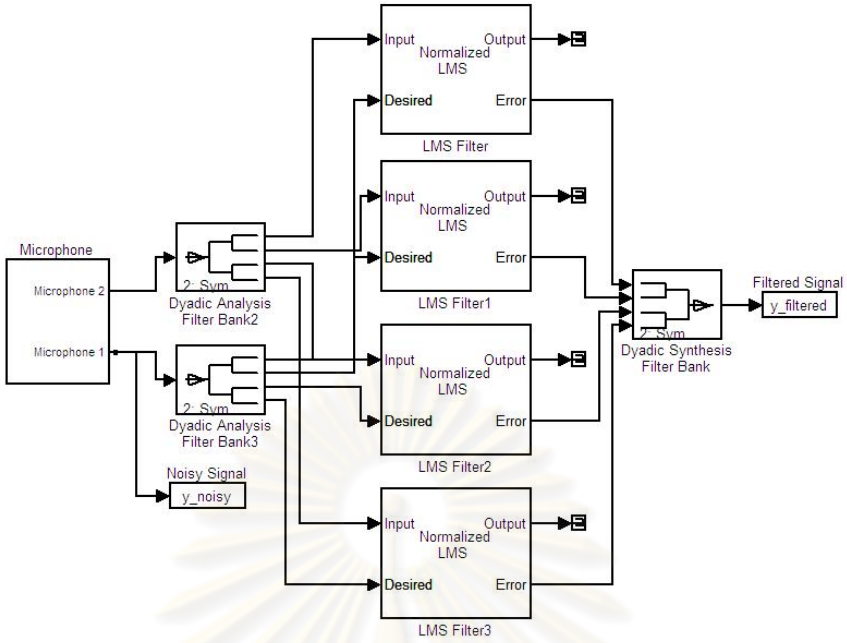
นำสัญญาณเสียงของไมโครโฟนตัวที่ 1 และ 2 มาเข้าบล๊อค LMS Filter โดยให้เข้าในช่องสัญญาณที่ต้องการ (Desired) และ สัญญาณขาเข้า (Input) ตามลำดับ สัญญาณที่ได้ คือ สัญญาณความผิดพลาด (Error) หรือ สัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอก ดังแสดงในรูปที่ 17



รูปที่ 17 โปรแกรมทดสอบการทำงานของอัลกอริทึม LMS และ NLMS

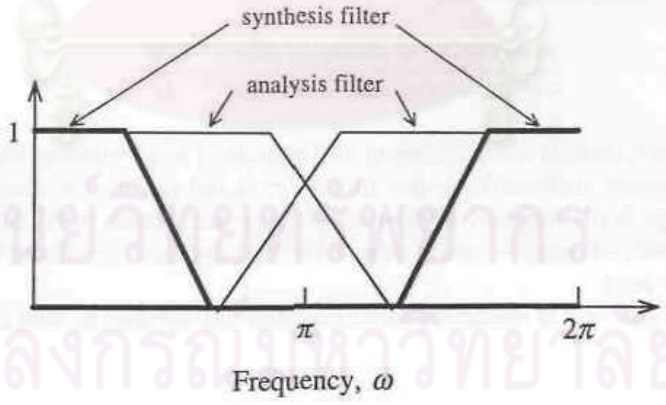
3. <u>โปรแกรมทดสอบการทำงานของอัลกอริทึม S-LMS และ S-NLMS</u>

นำสัญญาณเสียงของไมโครโฟนตัวที่ 1 และ 2 มาเข้าบล๊อค Dyadic Analysis Filter Bank แบบ Symmetric เพื่อแยกย่านความถี่ย่อยของแต่ละสัญญาณออกเป็น 4 ย่าน นำสัญญาณแต่ละย่านของไมโครโฟนตัวที่ 1 และ 2 มาเข้าบล๊อค LMS Filter โดยให้เข้าในช่อง สัญญาณที่ต้องการ (Desired) และ สัญญาณขาเข้า (Input) ตามลำดับ หลังจากนั้นนำสัญญาณความผิดพลาด (Error) ของแต่ละย่านมาเข้าบล๊อค Dyadic Synthesis Filter Bank แบบ Symmetric เพื่อรวมย่านความถี่ย่อย สัญญาณที่ได้ คือ สัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอก ดังแสดงในรูปที่ 18



รูปที่ 18 โปรแกรมทดสอบการทำงานของอัลกอริทึม S-LMS และ S-NLMS

เนื่องจากสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกที่ได้นั้น มีเสียงผิดปกติซึ่งเกิดจากเอเลียดซิ่งระหว่างย่านความถี่ย่อย (Interband Aliasing) จึงต้องทำการออกแบบตัวกรองของ Dyadic Analysis Filter Bank และ Dyadic Synthesis Filter Bank เพื่อลดการเกิดเอเลียดซิ่งระหว่างย่านความถี่ย่อย ดังแสดงในรูปที่ 19



รูปที่ 19 ผลตอบสนองทางความถี่ของตัวกรองเพื่อลดการเกิดเอเลียดซิ่งระหว่างย่าน [15]

เมื่อพิจารณา ค่าอัตราส่วนความซับซ้อนของการประมวลผล แบบเต็มย่าน ต่อ แบบแยกย่าน ดังแสดงในสมการที่ 20 โดยที่ M คือ จำนวนย่านความถี่ย่อย และ D คือ จำนวนเท่าในการลดอัตราสุ่ม เนื่องจากอัลกอริทึม S-LMS และ S-NLMS ทำการแยกย่านความถี่ย่อยออกเป็น 4 ย่าน จะได้ว่า M = 4 และ ทำการลดอัตราสุ่มเท่ากับจำนวนย่านความถี่ย่อย (Critical Sampling) จะได้ว่า D = M = 4 ดังนั้น ค่าอัตราส่วนความซับซ้อนของการประมวลผลของอัลกอริทึมแบบแยก

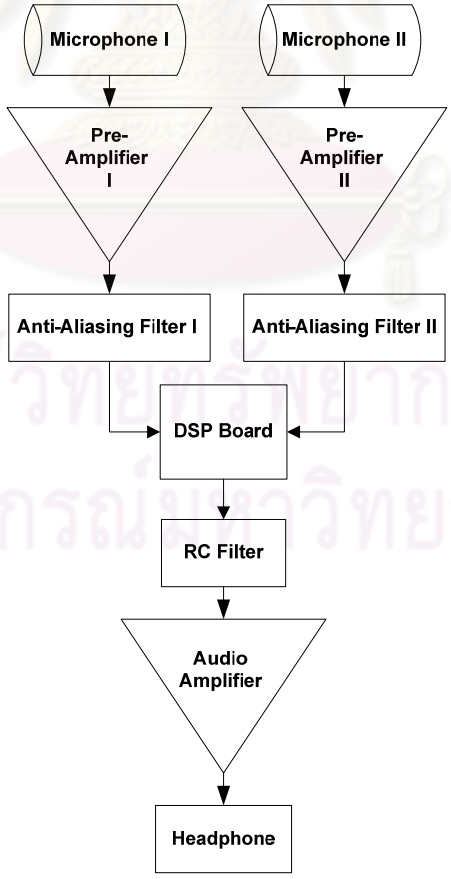
ย่าน ต่อ แบบเต็มย่าน มีค่าประมาณ 0.5 ซึ่งอัตราส่วนนี้ยังไม่รวมการประมวลผลในส่วนของตัวกรองที่ใช้ในการแยกย่านและรวมย่านความถี่ [15]

$$\frac{Subband}{Fullband} = \frac{2M}{D^2} \qquad (20)$$

แม้ว่าการประมวลผลในส่วนของอัลกอริทึมแบบแยกย่านจะน้อยกว่า แต่เมื่อพิจารณาการประมวลผลในส่วนของตัวกรองที่ต้องใช้ในการแยกย่านและรวมย่านความถี่จะพบว่ามีค่าสูงมาก ดังนั้น เพื่อเป็นการลดภาระการประมวลผลในตัวประมวลผลซึ่งมีจำกัด ผู้ทำวิจัยจะเลือกใช้อัลกอริทึมแบบเต็มย่านเท่านั้น ในต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

3.3 ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

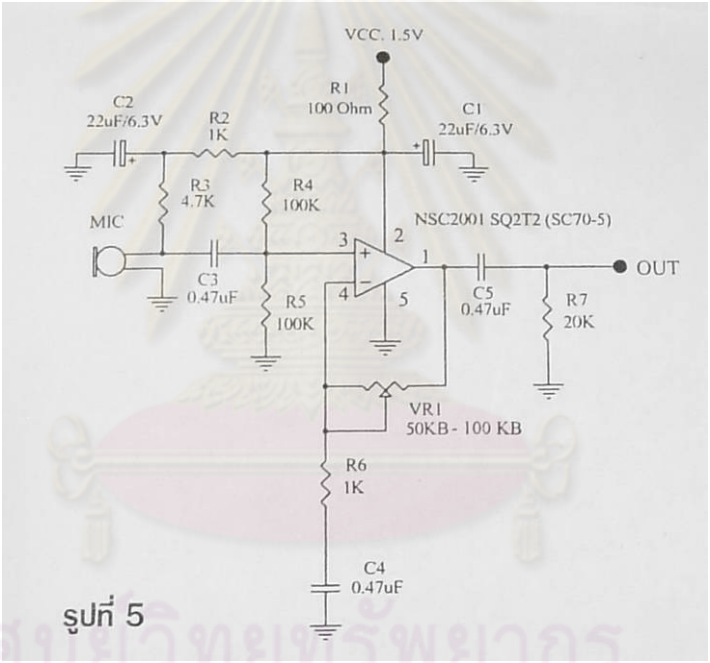
หลักการทำงานของหูฟังแพทย์แบบอิเล็กทรอนิกส์ จะรับเสียงที่สนใจจะตรวจฟังผ่านทางไมโครโฟน นำมาผ่านเครื่องขยายเสียง เพื่อขยายเสียงให้ดังเพียงพอก่อนจะมาถึงผู้ตรวจฟัง แต่หูฟังแพทย์แบบอิเล็กทรอนิกส์มักจะมีปัญหาเสียงรบกวนภายนอกต่างๆ จึงควรทำการลดเสียงรบกวนภายนอก เพื่อเพิ่มประสิทธิภาพในการตรวจฟัง ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ มีส่วนประกอบหลักดังแสดงในรูปที่ 20 ดังนี้



รูปที่ 20 ส่วนประกอบหลักของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

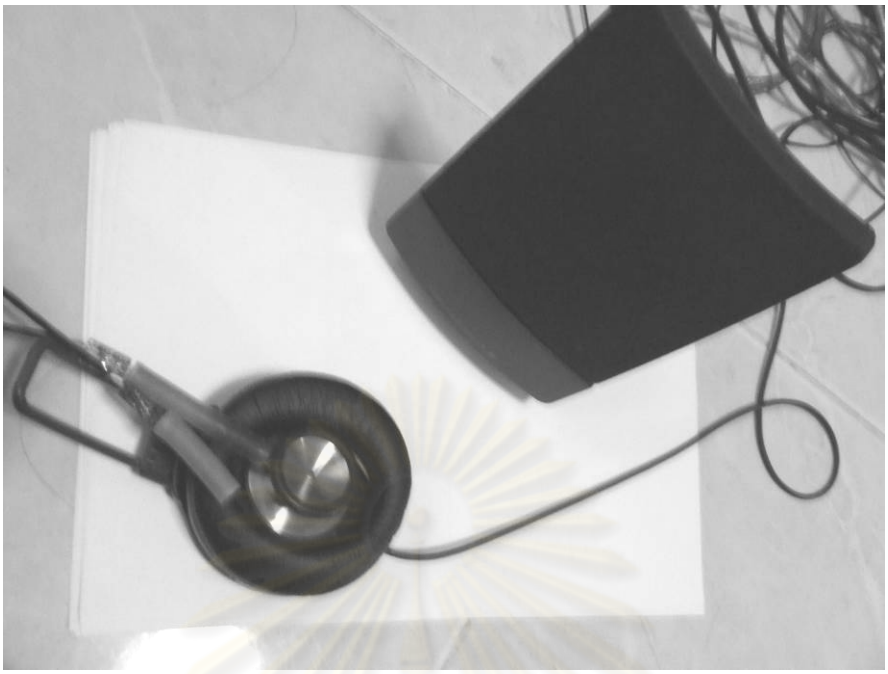
1. Microphone

งานวิจัยนี้เลือกใช้วงจร Microphone จำนวน 2 ตัว เบอร์ A06 ของบริษัท XYCON ทำงานที่แรงดัน 0.9–6.0 โวลต์ วงจรแสดงในรูปที่ 21 โดยไมโครโฟนตัวที่ 1 ทำหน้าที่รับเสียงจากอวัยวะที่ต้องการจะตรวจฟัง โดยนำไมโครโฟนสอดไว้ในท่อนำเสียงของหูฟังแพทย์แบบดั้งเดิม ซึ่งงานวิจัยนี้เลือกใช้หูฟังแพทย์แบบดั้งเดิม รุ่น 3M Littmann Classic II S.E. ซึ่งเป็นรุ่นที่นิสิต นักศึกษาแพทย์นิยมใช้กัน เนื่องจากให้คุณภาพเสียงอยู่ในเกณฑ์ดี และ มีราคาไม่แพง แต่เลือกใช้ท่อยางแทนท่อนำเสียงของหูฟังแพทย์แบบดั้งเดิม เพราะไม่ต้องการตัดท่อนำเสียงของหูฟังแพทย์แบบดั้งเดิม ส่วนไมโครโฟนตัวที่ 2 ทำหน้าที่รับเสียงรบกวนภายนอก โดยสอดไว้ในท่อยางที่มีความยาวใกล้เคียงกับท่อยางของไมโครโฟนตัวที่ 1 เพื่อให้เสียงรบกวนภายนอกที่ไมโครโฟนทั้งสองตัวได้รับมีความสัมพันธ์กันมากที่สุด



รูปที่ 21 วงจร Microphone เบอร์ A06

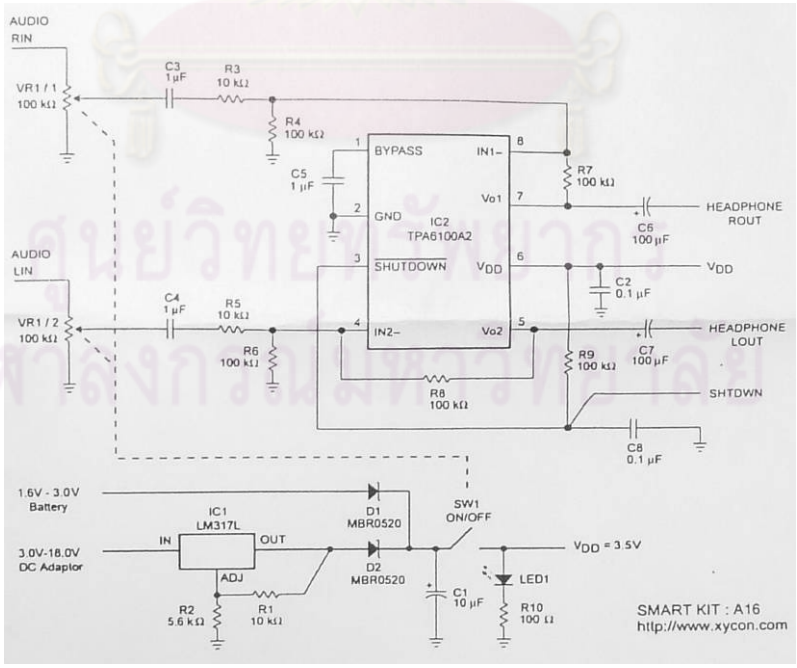
ข้อมูลเสียงจริง (Actual Data) ได้มาจาก การอัดเสียงหัวใจในสภาวะที่มีเสียงรบกวนภายนอก โดยการเล่นไฟล์เสียงหัวใจที่หูฟัง และเล่นไฟล์เสียงรบกวนภายนอกที่ลำโพง นำหัว Chestpiece ซึ่งมีไมโครโฟนตัวที่ 1 สอดอยู่ในท่อนำเสียง ไปวางไปบนหูฟังเพื่อรับเสียงหัวใจ และวางไมโครโฟนตัวที่ 2 ไว้ที่ตำแหน่งใกล้ๆ กัน เพื่อรับเสียงรบกวนภายนอก ดังแสดงในรูปที่ 22



รูปที่ 22 การอัดเสียงข้อมูลเสียงจริง

2. Pre Amplifier

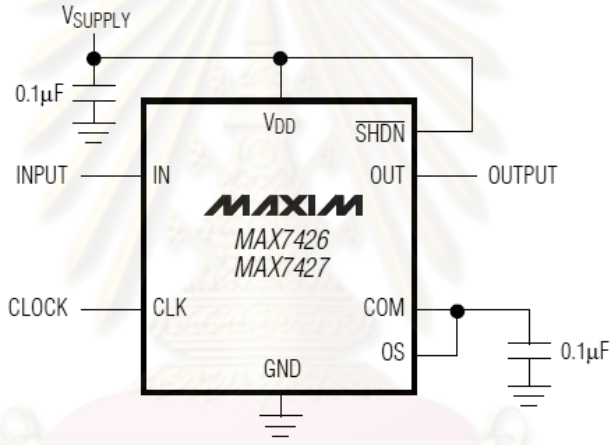
งานวิจัยนี้เลือกใช้วงจร Amplifier จำนวน 2 ตัว เบอร์ A16 ของบริษัท XYCON ทำงานที่แรงดัน 3.0–18.0 โวลต์ วงจรแสดงในรูปที่ 23 ทำหน้าที่ขยายเสียงจากไมโครโฟนแต่ละตัวให้ดังเพียงพอก่อนนำไปเข้าตัวประมวลผล



รูปที่ 23 วงจร Amplifierเบอร์ A16

3. Anti Aliasing Filter

งานวิจัยนี้เลือกใช้วงจร 5th-Order Low Pass Filter จำนวน 2 ตัว เบอร์ MAX7427 ของบริษัท MAXIM ทำงานที่แรงดัน 3.0 โวลต์ วงจรแสดงในรูปที่ 24 ทำหน้าที่กรองเสียงความถี่สูงจากไมโครโฟนแต่ละตัวออกไป เพื่อป้องกันการเกิดเอเลียดซิ่ง (Aliasing) ที่เกิดจากการใช้ค่าอัตราสุ่ม (Sampling Rate) น้อยเกินไป ในการแปลงสัญญาณแอนะลอกให้เป็นสัญญาณดิจิตอล ซึ่งทำให้สัญญาณดิจิตอลที่ได้มีความผิดเพี้ยน เนื่องจากงานวิจัยนี้เลือกใช้ค่าอัตราสุ่มที่ 8000 Hz จากทฤษฎีการสุ่มสัญญาณ จะได้ว่าสัญญาณแอนะลอกควรมีความถี่สูงสุดน้อยกว่า 4000 Hz จึงจำเป็นต้องกรองความถี่ที่สูงกว่านี้ออกไป โดยค่าความถี่ตัด (f_C) สามารถปรับเปลี่ยนได้โดยการเปลี่ยนขนาดของตัวเก็บประจุ (C) ระหว่างขา CLK กับ GND ซึ่งคำนวณได้จากสมการที่ 21 และ 22 งานวิจัยนี้เลือกใช้ตัวเก็บประจุขนาด 47 pF จะได้ว่าความถี่ตัดอยู่ที่ประมาณ 3723 Hz



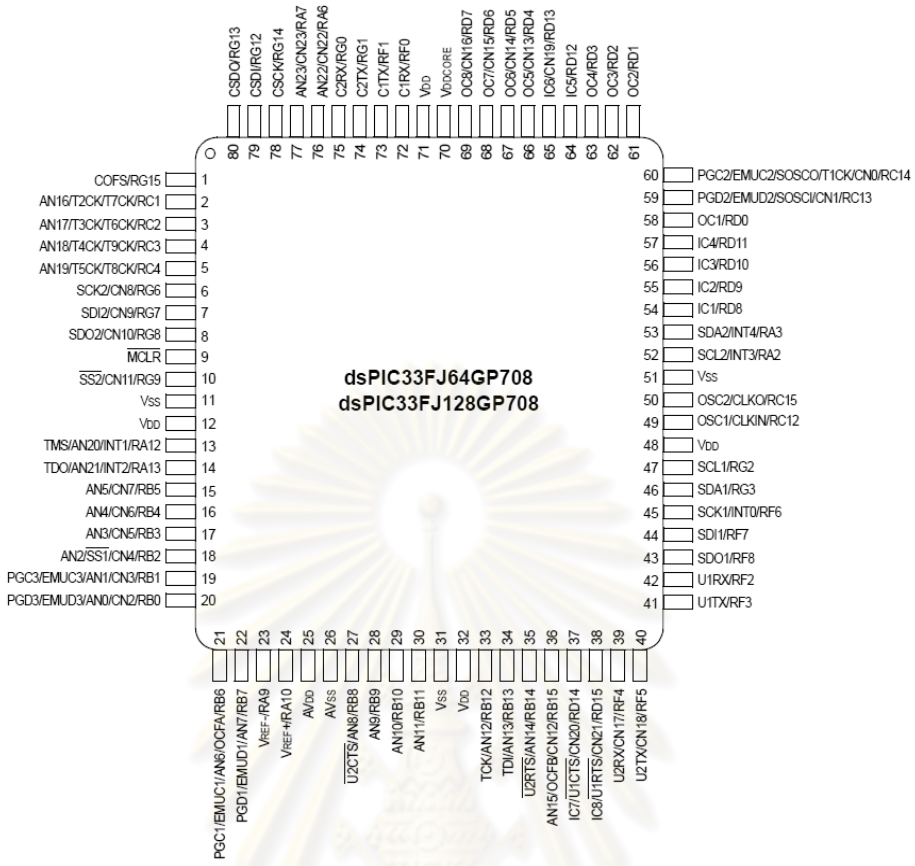
รูปที่ 24 วงจร Anti Aliasing Filter เบอร์ MAX7427 [16]

$$f_C = \frac{f_{CLK}}{100} \tag{21}$$

$$f_{CLK}(kHz) = \frac{17.5 \times 10^3}{C(pF)} \tag{22}$$

4. DSP Board

งานวิจัยนี้เลือกใช้บอร์ดประมวลผล ET-dsPIC33WEB V1.0 ของบริษัท ETT ที่ใช้ตัวประมวลผลขนาด 16 Bits เบอร์ dsPIC33FJ128GP708 ของบริษัท MICROCHIP ทำงานที่แรงดัน 3.0–3.6 โวลต์ ซึ่งมี Pin Diagrams ดังแสดงในรูปที่ 25 คุณสมบัติของ dsPIC ที่แตกต่างจาก PIC ทั่วไป คือ ความสามารถทางด้านการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing, DSP) ทำให้มีสมรรถนะในการคำนวณสูงขึ้น แต่ยังคงความใช้งานง่าย และราคาต่ำของ PIC ไว้



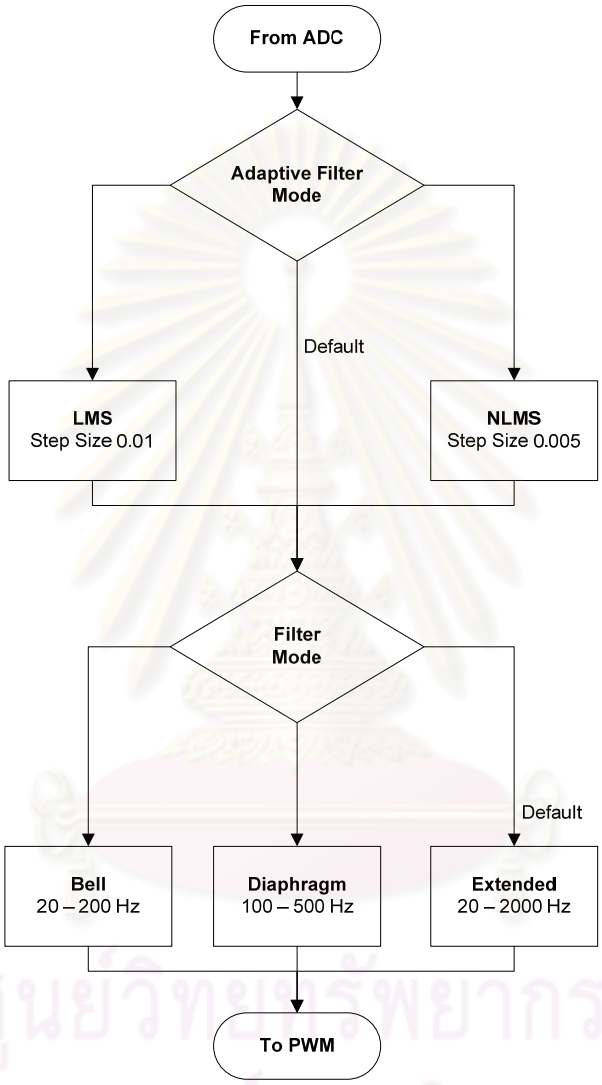
รูปที่ 25 Pin Diagrams ของตัวประมวลผล เบอร์ dsPIC33FJ128GP708 [17]

งานวิจัยนี้ใช้ภาษา C ในการพัฒนาโปรแกรมควบคุมการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ โดยให้ตัวประมวลผลใช้ความเร็วในการประมวลผล (F_{CY}) อยู่ที่ประมาณ 40 ล้านคำสั่งต่อวินาที (Million Instuctions Per Second, MIPS) จะได้ว่าแต่ละคำสั่งใช้เวลาประมาณ 0.025 μs โดยตัวประมวลผลทำหน้าที่ต่างๆ ดังนี้

4.1. แปลงเสียงจากไมโครโฟนแต่ละตัว ซึ่งเป็นสัญญาณแอนะลอกให้เป็นสัญญาณดิจิตอล โดยใช้โมดูล ADC ใช้การสุ่มสัญญาณพร้อมกัน (Simultaneous Sampling) และใช้เทคนิคการเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access, DMA) เพื่อลดการทำงานของตัวประมวลผล โดยใช้ค่าอัตราการสุ่ม 8000 Hz ขนาด 10 Bits จะได้ว่า มีการสุ่มสัญญาณทุกๆ 125 μs และมีระดับของสัญญาณ 1024 ขั้น

4.2. ประมวลผลสัญญาณดิจิตอล เพื่อปรับปรุงสัญญาณเสียงให้มีคุณภาพเหมาะสมต่อการนำไปตรวจฟัง โดยใช้ ตัวกรองแบบธรรมดา (Filter) และ ตัวกรองแบบปรับตัวได้ (Adaptive Filter) โดยมีขั้นตอนในการประมวลผลดังแสดงในรูปที่ 26 เนื่องจากทำการประมวลผลสัญญาณเป็นเฟรม เฟรมละ 32 จุด จะได้ว่าแต่ละ

เฟรมใช้เวลาประมาณ 4 ms ดังนั้นเวลาที่ใช้การประมวลผลแต่ละเฟรมต้องไม่เกิน 4 ms เมื่อพิจารณาถึงความล่าช้าของสัญญาณ จะได้ว่า ต้องทำการเก็บสัญญาณใช้เวลา 1 เฟรม และ ประมวลผลสัญญาณใช้เวลาอีก 1 เฟรม ดังนั้นสัญญาณจะล่าช้าไป 2 เฟรม ซึ่งเมื่อคิดเป็นเวลาจะได้ประมาณ 8 ms

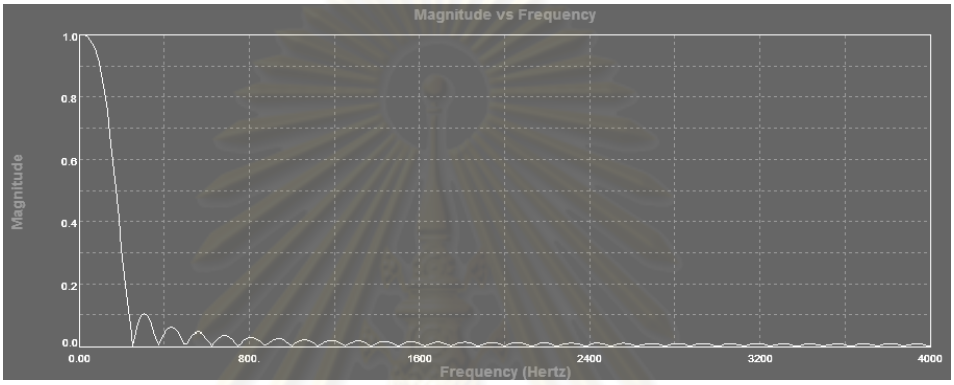


รูปที่ 26 ขั้นตอนการการประมวลผลสัญญาณดิจิตอลของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

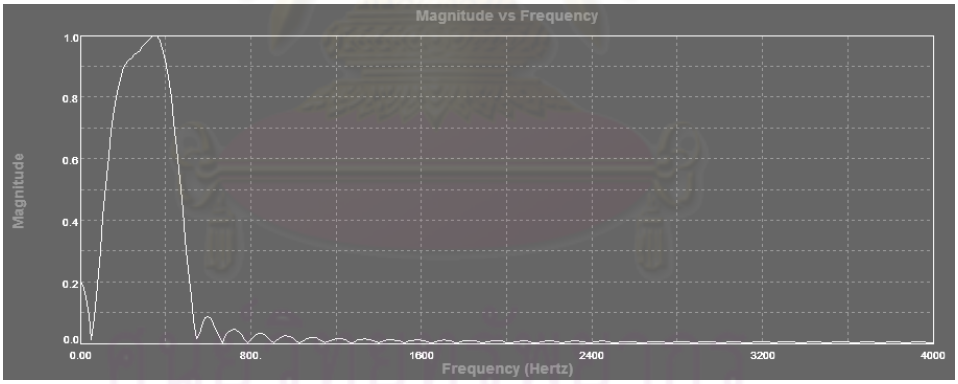
4.2.1. ตัวกรองธรรมดา (Filter) ทำการกรองความถี่เสียงจากไมโครโฟนตัวที่ 1 ให้อยู่ในช่วงที่ใช้ตรวจฟัง โดยแบ่งช่วงความถี่ที่ใช้ในการตรวจฟังออกเป็น 3 ช่วง ได้แก่ ช่วง Bell จะกรองเสียงให้อยู่ในช่วงความถี่ 0-200 Hz ช่วง Diaphragm จะกรองเสียงให้อยู่ในช่วงความถี่ 100-500 Hz และช่วง Extended จะกรองเสียงให้อยู่ในช่วงความถี่ 0-2000 Hz ซึ่งมี

ผลตอบสนองทางความถี่ ดังแสดงในรูป 27 ถึง 29 โดยใช้ตัวกรองแบบ FIR หน้าต่างที่ใช้ คือ แบบ Kaiser และ ค่าอันดับที่ใช้ คือ 64

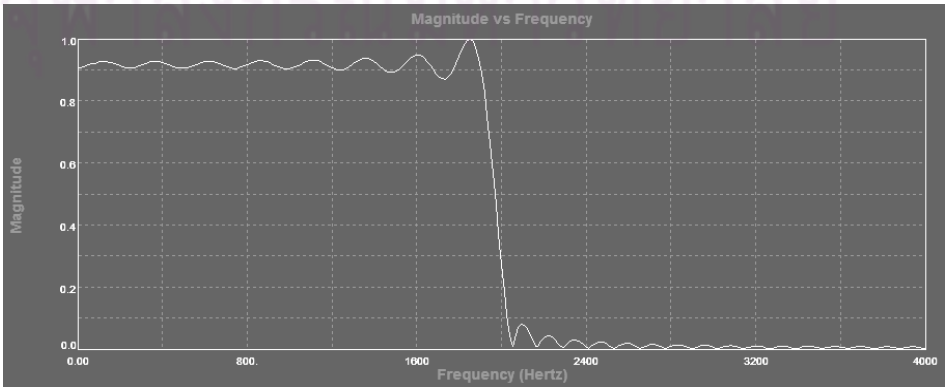
เมื่อพิจารณาจำนวนคำสั่งต่อการเรียกใช้ตัวกรองแต่ละครั้ง คือ 53+N(4+L) โดยที่ N คือ จำนวนข้อมูลในแต่ละเฟรม และ L คือ ค่าอันดับ ซึ่งในที่นี้ใช้ N = 32 และ L = 64 จะได้ว่า จำนวนคำสั่งต่อการเรียกใช้ตัวกรองแต่ละครั้ง คือ 2229 คำสั่ง ดังนั้น การเรียกใช้ตัวกรองแต่ละครั้งใช้เวลาประมวลผลประมาณ 55.725 μ s



รูปที่ 27 ผลการตอบสนองความถี่ในช่วง Bell



รูปที่ 28 ผลการตอบสนองความถี่ในช่วง Diaphragm



รูปที่ 29 ผลการตอบสนองความถี่ในช่วง Extended

4.2.2.ตัวกรองแบบปรับตัวได้ (Adaptive Filter) ทำการลดเสียงรบกวนภายนอก โดยใช้อัลกอริม LMS และ NLMS ใช้ค่าอัตราการปรับตัว (Step Size) และค่าอันดับ (Order) ที่ได้จากโปรแกรมจำลองการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ โดยค่า Step size ที่ใช้ คือ 0.01 และ 0.005 ตามลำดับ และ ค่า Order ที่ใช้ คือ 32

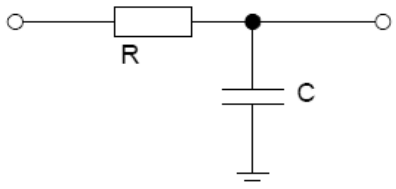
เนื่องจากจำนวนคำสั่งต่อการเรียกใช้ตัวกรองแบบปรับตัวได้แต่ละครั้ง สำหรับอัลกอริทึม LMS และ NLMS คือ 61 + N(13 + 5L) และ 66 + N(49 + 5L) ตามลำดับ ในที่นี้ใช้ N = 32 และ L = 32 จะได้ว่า จำนวนคำสั่งต่อการเรียกใช้ตัวกรองแบบปรับตัวได้แต่ละครั้ง คือ 5597 และ 6754 คำสั่ง ดังนั้น การเรียกใช้ตัวกรองแบบปรับตัวได้แต่ละครั้ง สำหรับอัลกอริทึม LMS และ NLMS ใช้เวลาประมวลผลประมาณ 139.925 และ 168.850 μs ตามลำดับ

4.3. แปลงเสียงซึ่งเป็นสัญญาณดิจิตอลให้เป็นสัญญาณแอนะลอก โดยใช้เทคนิค Pulse Width Modulation (PWM) เนื่องจากตัวประมวลผลไม่มีโมดูล DAC โดยต้องการแทนข้อมูลขนาด 10 Bits สามารถคำนวณความถี่ของ PWM (F_{PWM}) ได้จากสมการที่ 23 จะได้ว่า Bits = 10 และ F_{CY} = 40 MHz ดังนั้น F_{PWM} มีค่าประมาณ 39 kHz

$$Bits = \frac{\log(F_{CY} / F_{PWM})}{\log 2} \qquad (23)$$

5. RC Filter

งานวิจัยนี้เลือกใช้วงจร 1th-Order RC Filter วงจรแสดงในรูปที่ 30 ทำหน้าที่กรองความถี่สูงออกไป เพื่อช่วยลดเสียงรบกวนที่เกิดจากวงจรไฟฟ้า และการใช้เทคนิค PWM [18] โดยสามารถคำนวณความถี่ตัด (f_C) ได้จากสมการที่ 24 งานวิจัยนี้เลือกใช้ตัวต้านทานขนาด 7.5 KΩ และ ตัวเก็บประจุขนาด 10 nF จะได้ว่าความถี่ตัดอยู่ที่ประมาณ 2122 Hz



รูปที่ 30 วงจร RC Filter

$$f_C = \frac{1}{2\pi RC} \qquad (24)$$

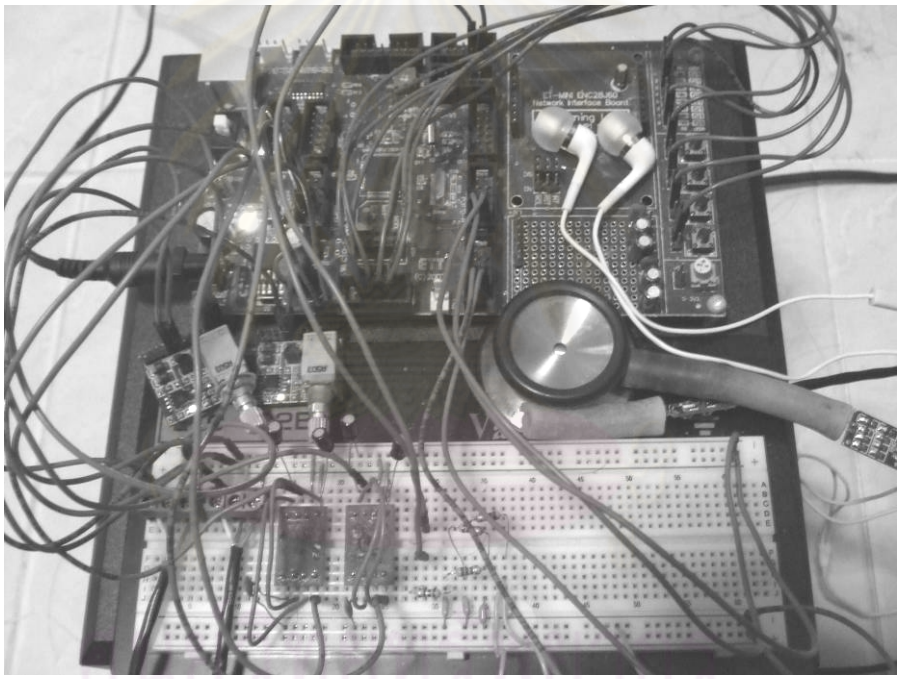
6. Audio Amplifier

งานวิจัยนี้เลือกใช้ชุดวงจร Amplifier เบอร์ A16 ของบริษัท XYCON ทำงานที่แรงดัน 3.0–18.0 โวลต์ ทำหน้าที่ขยายเสียงก่อนออกหูฟัง โดยสามารถปรับระดับความดังของเสียงได้

7. Headphone

งานวิจัยนี้เลือกใช้หูฟังเบอร์ MDR-E11LP ของบริษัท SONY ทำหน้าที่แสดงผลของเสียงที่สนใจจะตรวจฟังออกทางหูฟัง

ในขั้นต้นอุปกรณ์หลายส่วน ถูกต่อแยกออกมาจากบอร์ดประมวลผล โดยนำมาต่อบนบอร์ดทดลอง ดังแสดงในรูปที่ 31 เพื่อสะดวกในการปรับปรุงและแก้ไขวงจร



รูปที่ 31 บอร์ดประมวลผล บอร์ดทดลอง และอุปกรณ์อื่นๆ

บทที่ 4
การทดลองและการวิเคราะห์ผล

4.1 การทดลองบนโปรแกรมจำลองการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

ในการทดลองนี้จะทำการวิเคราะห์ผลของการเปลี่ยนแปลงพารามิเตอร์ของอัลกอริทึมแบบปรับตัวได้ วิเคราะห์คุณภาพของสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกโดยอัลกอริทึมแบบปรับตัวได้ และทำการเปรียบการใช้ข้อมูลเสียงจำลอง และข้อมูลเสียงจริง ทั้งนี้เพื่อศึกษาและหาพารามิเตอร์ที่เหมาะสมของอัลกอริธึมแบบปรับตัวได้ ที่จะนำมาใช้ในการลดเสียงรบกวนภายนอกของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

4.1.1 การวิเคราะห์ผลของการเปลี่ยนแปลงพารามิเตอร์ในอัลกอริทึม

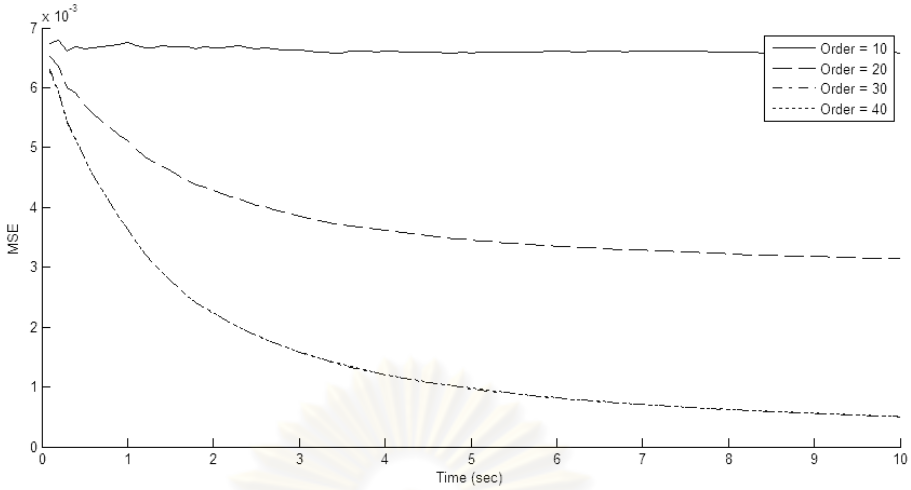
ในการทดลองนี้จะทำการวิเคราะห์ผลของการเปลี่ยนแปลง ค่าอันดับ (Order) และ ค่าอัตราการปรับตัว (Step Size) ของอัลกอริทึม LMS และ NLMS โดยประเมินจากกราฟแสดงค่า Mean Square Error (MSE) ณ เวลาต่างๆ ที่วัดจากสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอก ($S_{denoise}$) เปรียบเทียบกับ สัญญาณเสียงหัวใจที่ไม่มีเสียงรบกวนภายนอก ($S_{original}$) ซึ่งสามารถคำนวณได้จากสมการที่ 25

$$MSE = E[(S_{denoise} - S_{original})^2] \qquad (25)$$

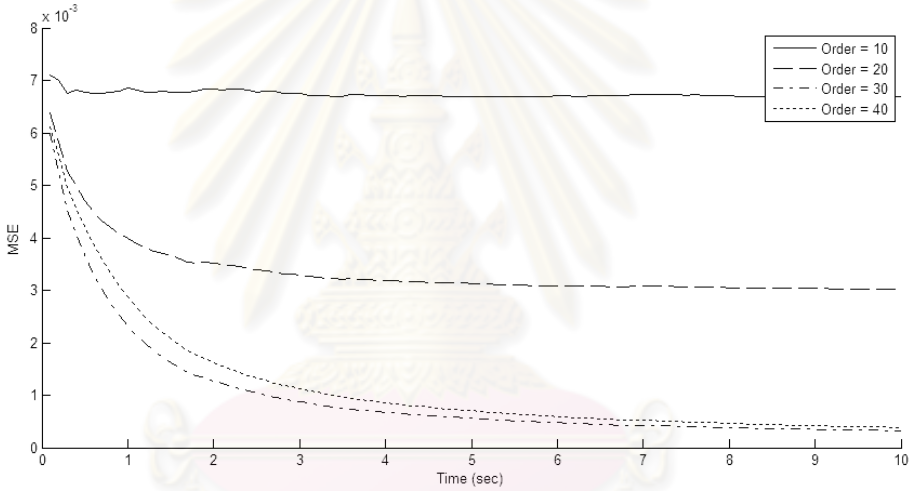
ในการทดลองนี้ เสียงหัวใจที่ใช้ คือ Pan Systolic Murmur เสียงรบกวนภายนอกที่ใช้ คือ White Noise ข้อมูลเสียงที่ใช้ คือ ข้อมูลเสียงจำลอง (Simulated Data)

1. <u>การวิเคราะห์ผลของการเปลี่ยนแปลงค่าอันดับ (Order)</u>

ในการทดลองนี้จะทำการเปลี่ยนค่าอันดับ (Order) เป็น 10, 20, 30 และ 40 และกำหนดค่าอัตราการปรับตัว (Step Size) เป็นค่าเดียวกันตลอด เพื่อดูแนวโน้มผลของการเปลี่ยนแปลงค่า Order



รูปที่ 32 กราฟแสดงค่า MSE ที่ค่าอันดับต่างๆ ของอัลกอริทึม LMS

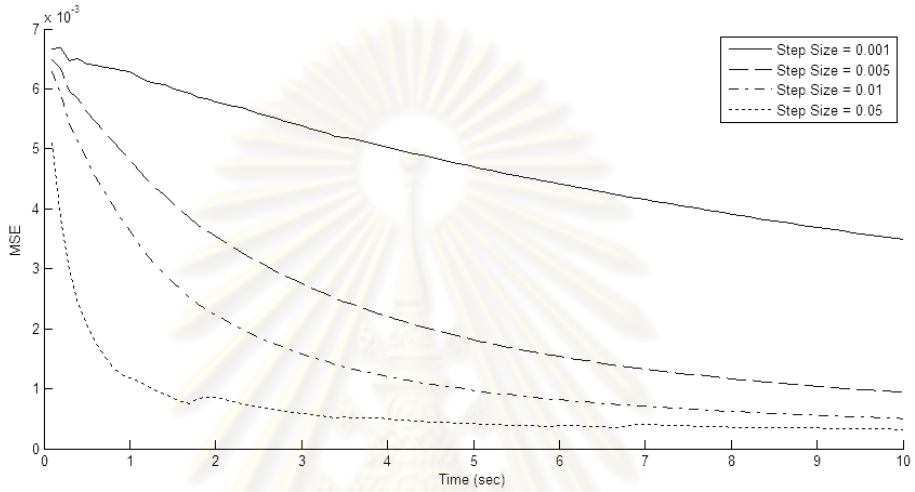


รูปที่ 33 กราฟแสดงค่า MSE ที่ค่าอันดับต่างๆ ของอัลกอริทึม NLMS

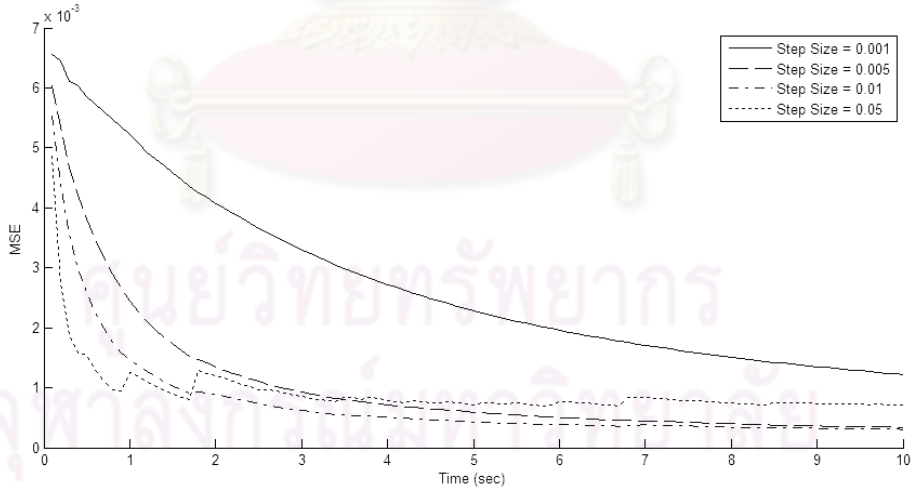
จากรูปที่ 32 และ 33 แสดงให้เห็นแนวโน้มของผลที่เกิดจากการเปลี่ยนแปลงค่าอันดับ (Order) คือ ค่า Order ที่มากขึ้น จะส่งผลให้ค่า MSE ที่ลู่เข้าค่าที่เริ่มคงที่ นั้นจะมีค่าน้อยลง ซึ่งแสดงให้เห็นว่า สัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกนั้นจะมีความผิดพลาดน้อยลง แต่การใช้ค่า Order ที่มากขึ้นจะส่งผลให้อัลกอริทึมต้องทำการประมวลผลมากขึ้นด้วย ค่า Order ที่เลือกใช้สำหรับอัลกอริทึม LMS และ NLMS คือ 32 แต่สำหรับอัลกอริทึม S-LMS และ S-NLMS ค่า Order จะเฉลี่ยไปตามจำนวนย่านความถี่ย่อย ซึ่งในการทดลองนี้แยกย่านความถี่ย่อยออกเป็น 4 ย่าน ดังนั้น ค่า Order ที่เลือกใช้สำหรับอัลกอริทึม S-LMS และ S-NLMS คือ 8

2. <u>การวิเคราะห์ผลของการเปลี่ยนแปลงค่าอัตราการปรับตัว (Step Size)</u>

ในการทดลองนี้จะทำการเปลี่ยนค่าอัตราการปรับตัว (Step Size) เป็น 0.001, 0.005, 0.01 และ 0.05 และ กำหนดค่าอันดับ (Order) เป็นค่าดียวกันตลอด เพื่อดูแนวโน้มผลของการเปลี่ยนแปลงค่า Step Size



รูปที่ 34 กราฟแสดงค่า MSE ที่ค่าอัตราการปรับตัวต่างๆ ของอัลกอริทึม LMS



รูปที่ 35 กราฟแสดงค่า MSE ที่ค่าอัตราการปรับตัวต่างๆ ของอัลกอริทึม NLMS

จากรูปที่ 34 และ 35 แสดงให้เห็นแนวโน้มของผลที่เกิดจากการเปลี่ยนแปลงค่าอัตราการปรับตัว (Step Size) คือ ถ้าใช้ค่า Step Size ที่มีค่ามาก จะใช้เวลาน้อยในการที่ค่า MSE จะลู่เข้าสู่ค่าที่เริ่มคงที่ ซึ่งแสดงให้เห็นว่า จะใช้เวลาน้อยในการที่สัญญาณเสียงหัวใจจะได้รับการลดเสียงรบกวนภายนอกจนถึงระดับที่เริ่มคงที่ แต่ค่า MSE ที่ลู่เข้าค่าที่เริ่มคงที่ นั้นจะมีค่ามาก ซึ่งแสดงให้

เห็นว่า สัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกนั้นจะมีความผิดพลาดมาก ในทางกลับกัน ถ้าใช้ค่า Step Size ที่มีค่าน้อย จะใช้เวลามากในการที่ค่า MSE จะลู่เข้าสู่ค่าที่เริ่มคงที่ ซึ่งแสดงให้เห็นว่า จะใช้เวลามากในการที่สัญญาณเสียงหัวใจจะได้รับการลดเสียงรบกวนภายนอกจนถึงระดับที่เริ่มคงที่ แต่ค่า MSE ที่ลู่เข้าค่าที่เริ่มคงที่ นั้นจะมีค่าน้อย ซึ่งแสดงให้เห็นว่าสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกนั้นจะมีความผิดพลาดน้อย ดังนั้นจึงไม่ควรใช้ค่า Step Size ที่มากหรือน้อยเกินไป

4.1.2 การวิเคราะห์คุณภาพของสัญญาณเสียงหัวใจ

ในการทดลองนี้จะทำการวิเคราะห์คุณภาพของสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกโดยอัลกอริทึม LMS, NLMS, S-LMS และ S-NLMS โดยพิจารณาจากค่า Signal to Noise Ratio (SNR) ของสัญญาณเสียงหัวใจที่มีเสียงรบกวนภายนอก (S_{noisy}) และสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอก ($S_{denoise}$) ซึ่งสามารถคำนวณได้จากสมการที่ 26 และ 27

$$SNR_{noisy} = 10\log(\frac{E[S^2{}_{original}]}{E[(S_{noisy} - S_{original})^2]}) \qquad (26)$$

$$SNR_{denoise} = 10\log(\frac{E[S^2{}_{original}]}{E[(S_{denoise} - S_{original})^2]}) \qquad (27)$$

ในการทดลองนี้จะใช้ค่าอันดับ (Order) ที่ได้จากการทดลองที่ผ่านมา และใช้ค่าอัตราการปรับตัว (Step Size) ต่างๆ ตั้งแต่ 0.001 ถึง 1 เสียงหัวใจที่ใช้ คือ Pan Systolic Murmur เสียงรบกวนภายนอกที่ใช้ คือ White Noise, Pink Noise, Babble Noise และ Factory Noise ข้อมูลเสียงที่ใช้ คือ ข้อมูลเสียงจำลอง (Simulated Data)

ตารางที่ 1 แสดงค่า SNR ของอัลกอริทึม LMS

Step Size	SNR (dB)				
	White	*Pink*	*Babble*	*Factory*	*Average*
-	5.5114	6.6085	6.9677	3.6642	5.6880
0.001	9.3513	11.6106	11.8861	12.7096	11.3894
0.005	15.0062	15.4041	16.9378	13.8609	11.8370
0.01	17.6132	16.6118	19.1757	12.3716	16.4431
0.05	18.7710	13.3157	22.5449	7.1821	15.4534

ตารางที่ 2 แสดงค่า SNR ของอัลกอริทึม NLMS

Step Size	SNR (dB)				
	White	*Pink*	*Babble*	*Factory*	*Average*
-	5.5114	6.6085	6.9677	3.6642	5.6880
0.001	13.8895	16.1176	18.2643	15.0897	15.8402
0.005	19.0900	15.6824	20.3950	11.4956	16.6658
0.01	19.1301	12.8628	19.1295	8.8125	14.9837
0.05	15.0700	5.9560	12.8382	3.4013	9.3164

ตารางที่ 3 แสดงค่า SNR ของอัลกอริทึม S-LMS

Step Size	SNR (dB)				
	White	*Pink*	*Babble*	*Factory*	*Average*
-	5.5114	6.6085	6.9677	3.6642	5.6880
0.001	6.7897	7.6355	7.5741	6.7158	7.1788
0.005	7.5399	9.4524	9.1187	10.3929	9.1260
0.01	8.4083	10.6619	10.5921	11.8051	10.3669
0.05	13.0806	14.2686	15.4002	14.1152	14.2162
0.1	15.7938	15.8992	17.6314	13.5375	15.7155
0.5	18.9395	15.8613	22.0308	8.5585	16.3476
1	17.8995	12.2589	22.0306	6.3651	14.6386

ตารางที่ 4 แสดงค่า SNR ของอัลกอริทึม S-NLMS

Step Size	SNR (dB)				
	White	*Pink*	*Babble*	*Factory*	*Average*
-	5.5114	6.6085	6.9677	3.6642	5.6880
0.001	13.5355	14.2359	15.3720	12.9578	14.0253
0.005	18.4737	17.5154	20.6942	15.4780	18.0403
0.01	18.5111	16.3786	21.0580	13.9103	17.4645
0.05	14.3797	9.6279	17.1350	7.5898	12.1830

จากตารางที่ 1-4 เมื่อพิจารณา ค่า Step Size ที่ให้ค่า SNR เฉลี่ยที่ดีที่สุดของแต่ละอัลกอริทึม จะได้ว่า ค่า Step Size ที่ควรเลือกใช้สำหรับอัลกอริทึม LMS และ S-LMS คือ 0.01 และ 0.5 ตามลำดับ ส่วน ค่า Step Size ที่ควรเลือกใช้สำหรับอัลกอริทึม NLMS และ S-NLMS คือ 0.005 สังเกตได้ว่า ค่า Step Size ของอัลกอริทึม S-LMS จะมากกว่าของอัลกอริทึม LMS เนื่องจากเมื่อแยกย่านความถี่จะส่งผลให้ค่ากำลังของสัญญาณแต่ละย่านมีค่าลดลง ส่งผลให้จำเป็นต้องปรับค่า Step Size ให้มากขึ้น ตามหลักการเลือกใช้ค่า Step Size ที่เหมาะสม ซึ่งจะแปรผกผันกับค่ากำลังของสัญญาณขาเข้า แต่สำหรับอัลกอริทึม S-NLMS ค่า Step Size ที่เหมาะสมจะเป็นค่าที่ใกล้เคียงกับของอัลกอริทึม NLMS เนื่องจาก การเปลี่ยนแปลงค่ากำลังของสัญญาณ แทบจะไม่ส่งผลต่อการทำงานของอัลกอริทึม NLMS เพราะอัลกอริทึม NLMS มีส่วนของการ Normalized สัญญาณขาเข้า

ตารางที่ 5 แสดงค่า SNR เฉลี่ยที่ดีที่สุดของแต่ละอัลกอริทึม

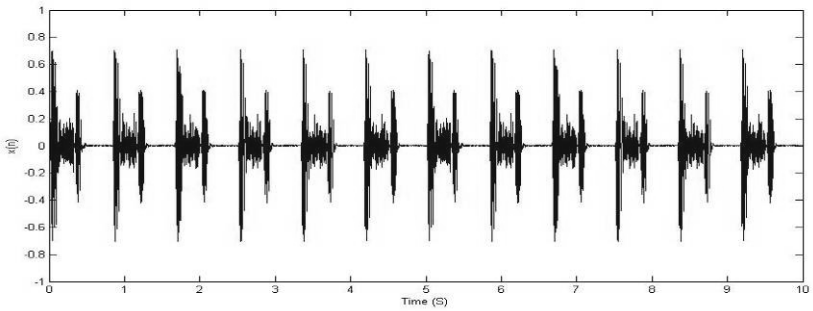
Algorithm	SNR (dB)				
	White	*Pink*	*Babble*	*Factory*	*Average*
-	5.5114	6.6085	6.9677	3.6642	5.6880
LMS	17.6132	16.6118	19.1757	12.3716	16.4431
NLMS	19.0900	15.6824	20.3950	11.4956	16.6658
S-LMS	18.9395	15.8613	22.0308	8.5585	16.3476
S-NLMS	18.4737	17.5154	20.6942	15.4780	18.0403

จากตารางที่ 5 แสดงให้เห็นว่า ค่า SNR ของสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอกมีค่าสูงขึ้นมากพอสมควร โดยพบว่าอัลกอริทึม S-NLMS ให้ค่า SNR เฉลี่ยสูงที่สุด ส่วนอัลกอริทึมอื่นๆ จะให้ค่า SNR เฉลี่ยใกล้เคียงกัน

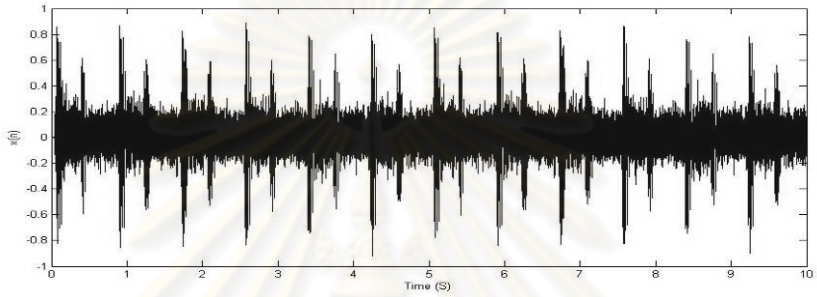
4.1.3 การเปรียบเทียบการใช้ข้อมูลเสียงจำลองและข้อมูลเสียงจริง

ในการทดลองนี้จะทำการเปรียบเทียบการใช้ข้อมูลเสียงจำลอง (Simulated Data) และข้อมูลเสียงจริง (Actual Data) ที่ผ่านการลดเสียงรบกวนภายนอก โดยใช้อัลกอริทึม LMS และ NLMS โดยพิจารณาจากค่า Signal to Noise Ratio (SNR)

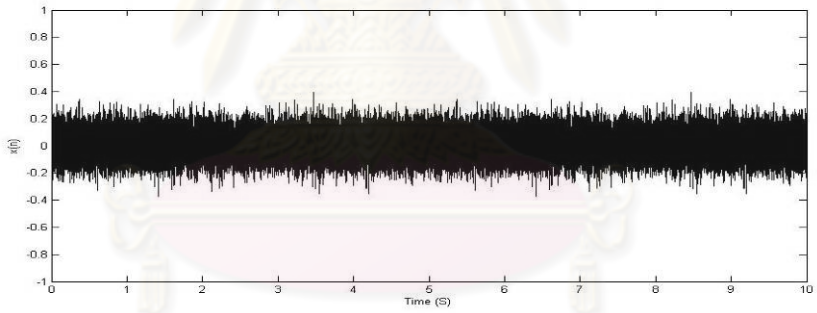
ในการทดลองนี้จะใช้ค่าอันดับ (Order) และค่าอัตราการปรับตัว (Step Size) จากการทดลองที่ผ่านมา เสียงหัวใจที่ใช้ คือ Pan Systolic Murmur เสียงรบกวนภายนอกที่ใช้ คือ White Noise ข้อมูลเสียงที่ใช้ คือ ข้อมูลเสียงจำลอง (Simulated Data) และข้อมูลเสียงจริง (Actual Data)



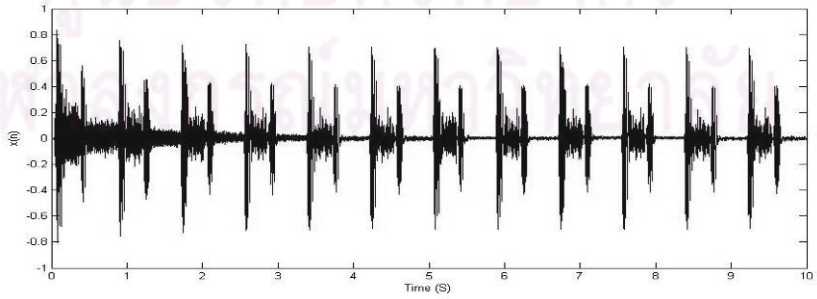
รูปที่ 36 ข้อมูลเสียงจำลอง : สัญญาณเสียงหัวใจที่ไม่มีเสียงรบกวนภายนอก



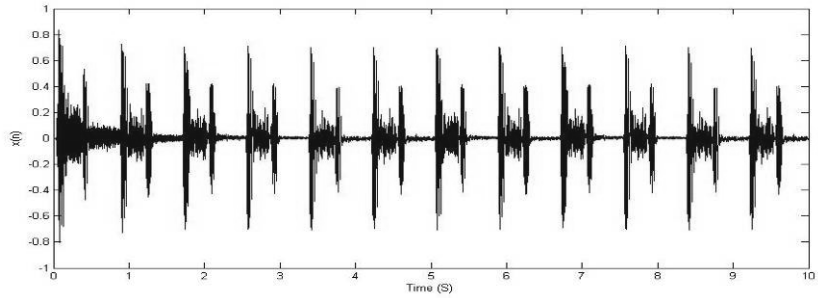
รูปที่ 37 ข้อมูลเสียงจำลอง : สัญญาณเสียงหัวใจที่มีเสียงรบกวนภายนอก (ไมโครโฟนตัวที่ 1)



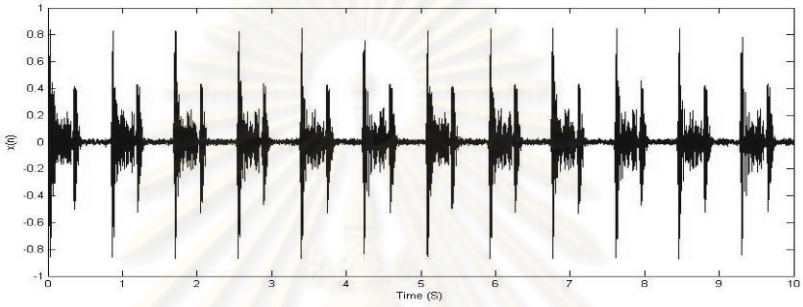
รูปที่ 38 ข้อมูลเสียงจำลอง : สัญญาณเสียงรบกวนภายนอก (ไมโครโฟนตัวที่ 2)



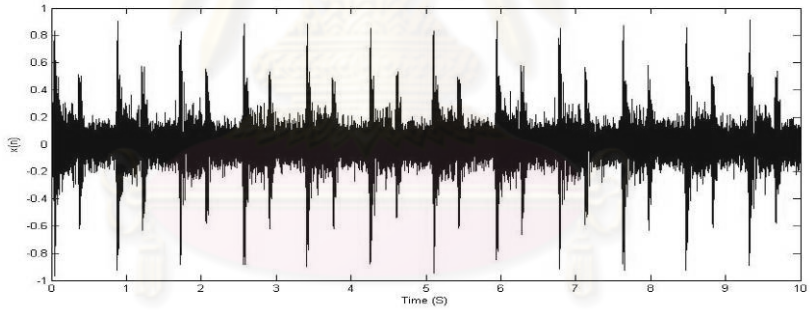
รูปที่ 39 ข้อมูลเสียงจำลอง : สัญญาณเสียงหัวใจที่ผ่านอัลกอริทึม LMS



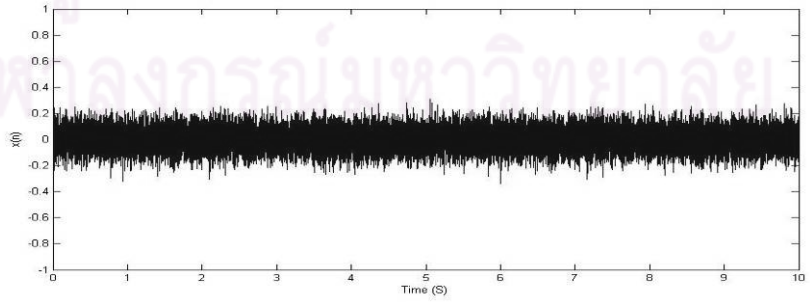
รูปที่ 40 ข้อมูลเสียงจำลอง : สัญญาณเสียงหัวใจที่ผ่านอัลกอริทึม NLMS



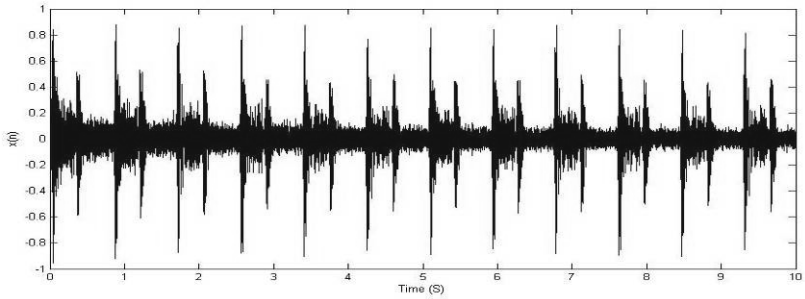
รูปที่ 41 ข้อมูลเสียงจริง : สัญญาณเสียงหัวใจที่ไม่มีเสียงรบกวนภายนอก



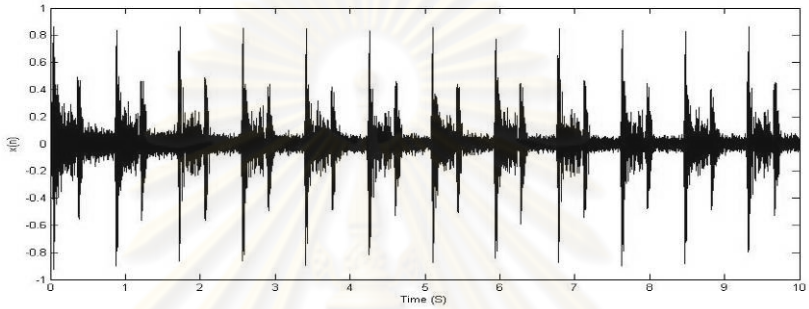
รูปที่ 42 ข้อมูลเสียงจริง : สัญญาณเสียงหัวใจที่มีเสียงรบกวนภายนอก (ไมโครโฟนตัวที่ 1)



รูปที่ 43 ข้อมูลเสียงจริง : สัญญาณเสียงรบกวนภายนอก (ไมโครโฟนตัวที่ 2)



รูปที่ 44 ข้อมูลเสียงจริง : สัญญาณเสียงหัวใจที่ผ่านอัลกอริทึม LMS



รูปที่ 45 ข้อมูลเสียงจริง : สัญญาณเสียงหัวใจทีผ่านอัลกอริทึม NLMS

ตารางที่ 6 แสดงค่า SNR ของข้อมูลแต่ละชนิด

Data	SNR (dB)		
	-	*LMS*	*NLMS*
Simulated	5.5187	18.1182	20.4901
Actual	6.8807	12.4933	18.0148

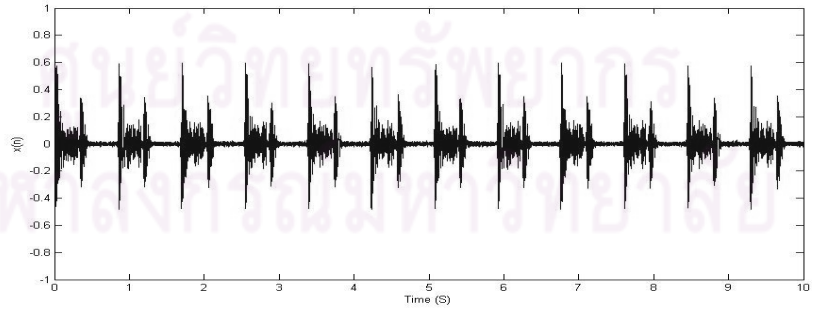
จากรูปที่ 36 ถึง 45 และตารางที่ 6 แสดงให้เห็นว่า การลดเสียงรบกวนภายนอก โดยอัลกอริทึม LMS และ NLMS เมื่อใช้ข้อมูลเสียงจริง ลดเสียงรบกวนภายนอกได้น้อยกว่า เมื่อใช้ข้อมูลเสียงจำลอง ทั้งนี้เนื่องจากประสิทธิภาพของการลดเสียงรบกวนภายนอกขึ้นอยู่กับความสัมพันธ์ของเสียงรบกวนภายนอกในไมโครโฟนตัวที่ 1 และ 2 สำหรับข้อมูลเสียงจำลอง เสียงรบกวนภายนอกของไมโครโฟนตัวที่ 1 เกิดจากการนำเสียงรบกวนภายนอกของไมโครโฟนตัวที่ 2 มาผ่านวงจรกรองความถี่ต่ำผ่านที่ 3000 Hz ส่งผลให้ความสัมพันธ์ของเสียงรบกวนภายนอกมีความซับซ้อนน้อย ในขณะที่ข้อมูลเสียงจริง ความสัมพันธ์ของเสียงรบกวนภายนอกของไมโครโฟนตัวที่ 1 และ 2 ขึ้นอยู่กับตำแหน่งและระยะห่างระหว่างไมโครโฟน ส่งผลให้ความสัมพันธ์ของเสียงรบกวนภายนอกมีความซับซ้อนมาก ดังนั้น การลดเสียงรบกวนภายนอกโดยใช้ข้อมูลเสียงจำลองจึงสามารถลดเสียงรบกวนภายนอกได้มากกว่าการใช้ข้อมูลเสียงจริง

เมื่อพิจารณาถึงประสิทธิภาพของอัลกอริทึม LMS และ NLMS จะเห็นว่า เมื่อใช้ข้อมูลเสียงจริง ค่า SNR ของอัลกอริทึม LMS จะต่ำลงมาก ในขณะที่ค่า SNR อัลกอริทึม NLMS จะต่ำลงเล็กน้อย เนื่องจากการใช้ข้อมูลเสียงจริง ค่ากำลังของสัญญาณขาเข้าอาจจะเปลี่ยนแปลงไปจากการใช้ข้อมูลเสียงจำลอง ซึ่งได้ทำการทดลองและหาค่าอัตราการปรับตัว (Step Size) ที่เหมาะสมแล้ว สำหรับอัลกอริทึม LMS เมื่อมีการเปลี่ยนแปลงค่ากำลังของสัญญาณขาเข้า จะทำให้ค่า Step Size ที่เหมาะสมเปลี่ยนแปลงไปด้วย ตามหลักการการเลือกใช้ค่า Step Size ที่เหมาะสม ซึ่งจะแปรผกผันกับค่ากำลังของสัญญาณขาเข้า แต่สำหรับอัลกอริทึม NLMS จะมีส่วนของการ Normalized สัญญาณขาเข้า ทำให้การเปลี่ยนแปลงค่ากำลังของสัญญาณขาเข้าแทบจะไม่ส่งผลต่อการปรับตัวของอัลกอริทึม ซึ่งเป็นข้อดีของอัลกอริทึม NLMS

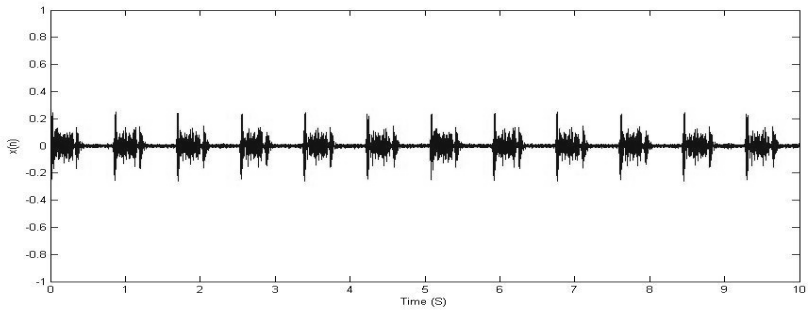
4.2 การทดลองบนต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

ในการทดลองนี้จะทำการวิเคราะห์คุณภาพของสัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอก โดยเลือกใช้รูปแบบของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ 2 รูปแบบ คือ ทั่วไป (Default) และ ลดเสียงรบกวนภายนอก (Denoise) ซึ่งรูปแบบ Default จะกรองเสียงให้อยู่ในช่วง Extended ส่วนรูปแบบ Denoise จะกรองเสียงให้อยู่ในช่วง Extended และ ลดเสียงรบกวนภายนอกโดยใช้อัลกอริทึม LMS (เนื่องจากอัลกอริทึม NLMS มีปัญญาเรื่องการ Saturation ของข้อมูลเมื่อประมวลผลบนบอร์ดประมวลผล จึงเลือกใช้อัลกอริทึม LMS แทน) โดยพิจารณาจากค่ากำลังของสัญญาณเสียงรบกวนภายนอกที่ลดลง

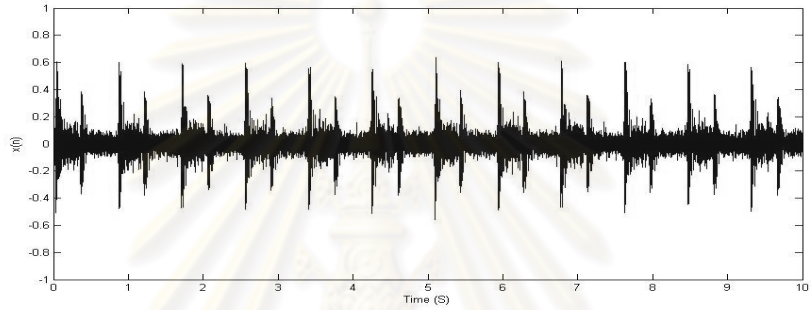
ในการทดลองนี้ เสียงหัวใจที่ใช้ คือ Pan Systolic Murmur เสียงรบกวนภายนอกที่ใช้ คือ White Noise ข้อมูลเสียงที่ใช้ คือ ข้อมูลเสียงจริง (Actual Data)



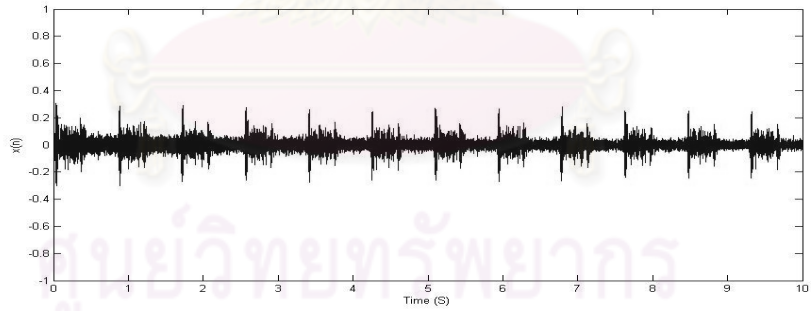
รูปที่ 46 ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ : สัญญาณเสียงหัวใจที่ไม่มีเสียงรบกวนภายนอกที่ผ่านรูปแบบ Default



รูปที่ 47 ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ : สัญญาณเสียงหัวใจที่ไม่มีเสียงรบกวนภายนอกที่ผ่านรูปแบบ Denoise



รูปที่ 48 ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ : สัญญาณเสียงหัวใจที่มีเสียงรบกวนภายนอกที่ผ่านรูปแบบ Default



รูปที่ 49 ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ : สัญญาณเสียงหัวใจที่มีเสียงรบกวนภายนอกที่ผ่านรูปแบบ Denoise

จากรูปที่ 46 ถึง 49 แสดงให้เห็นว่า สัญญาณเสียงหัวใจที่มีเสียงรบกวนภายนอกที่ผ่านรูปแบบ Denoise จะมีระดับของเสียงรบกวนภายนอกค่อยๆ ต่ำลง โดยพบว่ากำลังของสัญญาณเสียงรบกวนภายนอก ที่วัดจากช่วงที่ไม่มีเสียงหัวใจ ตอนเริ่มต้นเปรียบเทียบกับตอนสุดท้าย ลดลงประมาณ 5 dB เมื่อให้อาสาสมัครแพทย์ฟัง พบว่า สามารถได้ยินเสียง Murmur ได้ชัดเจนขึ้น แม้ว่าสัญญาณเสียงหัวใจที่ผ่านรูปแบบ Denoise จะมีลักษณะของเสียงหัวใจที่ลดลง

ไปบ้าง โดยเฉพาะเสียง S1 และ S2 ซึ่งเป็นเสียงความถี่ต่ำ เนื่องจากผลของการทำงานของตัวกรองแบบปรับตัวได้ แต่ปัญหานี้สามารถแก้ไขได้โดยการขยายเสียงในช่วงความถี่ต่ำให้ดังขึ้น

บทที่ 5
บทสรุป

5.1 สรุปผลการวิจัย

จากการทดลองบนโปรแกรมจำลองการทำงานของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์และการทดลองบนต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ สามารถสรุปเป็นผลวิจัย ดังนี้

- ค่าอันดับ (Order) มีผลต่อการทำงานของอัลกอริทึมแบบปรับตัวได้ โดยการใช้ค่า Order ที่น้อยเกินไป จะทำให้ค่าที่ลู่เข้ามีความผิดพลาดมาก การใช้ค่า Order ที่มากขึ้นจะส่งผลให้ค่าที่ลู่เข้ามีความผิดพลาดน้อยลง แต่การใช้ค่า Order ที่มากขึ้นจะส่งผลให้อัลกอริทึมต้องทำการประมวลผลมากขึ้นและทำให้การลู่เข้าช้าลงด้วย ดังนั้นจึงไม่ควรใช้ค่า Order ที่น้อยหรือมากเกินไป
- ค่าอัตราการปรับตัว (Step Size) มีผลต่อการทำงานของอัลกอริทึมแบบปรับตัวได้ โดยการใช้ค่า Step Size ที่น้อยเกินไป จะทำให้ระบบลู่เข้าช้า แต่ค่าที่ลู่เข้าจะมีความผิดพลาดน้อย ในทางกลับกัน การใช้ค่า Step Size ที่มากเกินไป จะทำให้ระบบลู่เข้าเร็ว แต่ค่าที่ลู่เข้าจะมีความผิดพลาดมาก ดังนั้น จึงไม่ควรใช้ค่า Step Size ที่น้อยหรือมากเกินไป
- แม้ว่าความซับซ้อนของการประมวลผลในส่วนของอัลกอริทึมแบบปรับตัวได้ แบบแยกย่านจะน้อยกว่าแบบเต็มย่าน แต่เมื่อพิจารณาการประมวลผลในส่วนแผงตัวกรอง (Filterbank) ที่ต้องใช้ในการแยกย่าน และรวมย่านความถี่จะพบว่ามีค่าสูงมาก ดังนั้นถ้าจะนำมาใช้ในระบบฝังตัว จึงควรใช้ตัวประมวลผลร่วมที่มีแผงตัวกรอง (Filterbank Co-processor) เพื่อเป็นการลดภาระการประมวลผลของตัวประมวลผลหลัก
- ประสิทธิภาพของการลดเสียงรบกวนภายนอกโดยตัวกรองแบบปรับตัว ขึ้นอยู่กับความสัมพันธ์ของเสียงรบกวนภายนอกในไมโครโฟนตัวที่ 1 และ 2 ถ้าความสัมพันธ์ระหว่างเสียงรบกวนภายนอกทั้งสองมีค่ามากจะส่งผลให้ประสิทธิภาพของการลดเสียงรบกวนภายนอกมีมากขึ้น ดังนั้น การลดเสียงรบกวนภายนอกโดยใช้ข้อมูลเสียงจำลองจึงสามารถลดเสียงรบกวนภายนอกได้มากกว่าการใช้ข้อมูลเสียงจริง
- การลดเสียงรบกวนภายนอกโดยตัวกรองแบบปรับตัวได้ เมื่อใช้ข้อมูลเสียงจริงซึ่งค่ากำลังของสัญญาณขาเข้าอาจจะเปลี่ยนแปลงไปจากข้อมูลเสียงจำลอง การใช้อัลกอริทึม LMS จะไม่สามารถทนต่อการเปลี่ยนแปลงค่ากำลังของสัญญาณขาเข้าได้

แต่สำหรับอัลกอริทึม NLMS จะมีส่วนของการ Normalized สัญญาณขาเข้า ทำให้การเปลี่ยนแปลงค่ากำลังของสัญญาณขาเข้าแทบจะไม่ส่งผลต่อการปรับตัวของอัลกอริทึม ดังนั้น เมื่อใช้ข้อมูลเสียงจริง อัลกอริทึม NLMS จึงให้ผลที่ดีกว่า LMS

- การใช้ตัวกรองธรรมดา (Filter) ที่มีค่าน้ำหนักคงที่ กรองเสียงให้อยู่ในช่วงความถี่ที่ต้องการ สามารถลดเสียงรบกวนที่ไม่อยู่ในช่วงความถี่ช่วงนั้นได้ แต่ไม่สามารถลดเสียงรบกวนที่มีความถี่อยู่ในช่วงนั้นได้ ดังนั้น การใช้ตัวกรองธรรมดาไม่สามารถลดเสียงรบกวนภายนอกที่มีความถี่อยู่ในช่วงที่ใช้ในการตรวจฟังได้
- ตัวกรองแบบปรับตัวได้ (Adaptive Filter) ที่มีค่าน้ำหนักปรับเปลี่ยนตลอดเวลา โดยอาศัยเงื่อนไขทางสถิติของสัญญาณและของโมเดลของสิ่งแวดล้อมที่สร้างขึ้นในการคำนวณ สามารถลดเสียงรบกวนที่มีความถี่อยู่ในย่านเดียวกับเสียงที่ได้ต้องการได้ ดังนั้น การใช้ตัวกรองแบบปรับตัวได้จึงสามารถลดเสียงรบกวนภายนอกที่มีความถี่อยู่ในช่วงที่ใช้ในการตรวจฟังได้
- การใช้ตัวกรองแบบธรรมดา และตัวกรองแบบปรับตัวได้ร่วมกัน จะทำให้สามารถลดเสียงรบกวนได้มากขึ้น ดังนั้น จึงควรใช้ตัวกรองแบบธรรมดาและตัวกรองแบบปรับตัวได้ร่วมกัน ในการลดเสียงรบกวนภายนอกของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์

5.2 **ข้อเสนอแนะ**

จากการพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ ผู้ทำวิจัยมีข้อเสนอแนะ ดังนี้

- ควรใช้โมดูล ADC (Analog to Digital Converter) ที่มีความละเอียดมากกว่านี้ เพื่อคุณภาพของเสียงที่ดีขึ้น เนื่องจากการประมวลผลข้อมูลบนตัวประมวลผล dsPIC เป็นแบบ 16 Bits แต่โมดูล ADC ที่ติดมากับตัวประมวลผล มีความละเอียดเพียง 10-12 Bits
- ควรใช้โมดูล DAC (Digital to Analog Converter) โดยตรง เนื่องจากตัวประมวลผลไม่มีโมดูล DAC ทำให้ต้องใช้โมดูล PWM (Pulse Width Modulation) แทน ทำให้เมื่อใช้แทนข้อมูลที่มีความละเอียดสูงกว่า 10 Bits จะทำให้เกิดเสียงรบกวนในช่วงความถี่ที่มนุษย์ได้ยิน
- ควรใช้บอร์ดประมวลผลมีสัญญาณรบกวนน้อยกว่านี้ เนื่องจากเมื่อเสียงผ่านบอร์ดประมวลผลแล้วเสียงที่ได้มีเสียงรบกวนพอสมควร ผู้ทำวิจัยจึงคาดว่าน่าจะเป็นสัญญาณรบกวนจากบอร์ดประมวลผล ทำให้ต้องใช้วงจร RC Filter กรองความถี่ต่ำผ่าน หลังจากเสียงผ่านบอร์ดประมวลผล แต่ก็ยังไม่สามารถลดเสียงรบกวนได้หมด

เนื่องจากไม่สามารถตัดที่ความถี่ที่ต่ำเกินไป เพราะจะเป็นการลดทอนเสียงที่ใช้ในการตรวจฟังไปด้วย

- ควรออกแบบการวางตำแหน่งของไมโครโฟนแต่ละตัว เพื่อให้เสียงรบกวนภายนอกที่ไมโครโฟนแต่ละตัวได้รับมีความสัมพันธ์กันมากที่สุด โดยควรสอดไมโครโฟนตัวที่ 1 ไว้ในส่วนของ Chestpiece แทนที่จะสอดไว้ในท่อนำเสียง และวางไมโครโฟนตัวที่ 2 ไว้ที่ตำแหน่งใกล้ๆ กันนอก Chestpiece เพื่อประสิทธิภาพในการลดเสียงรบกวนภายนอกโดยตัวกรองแบบปรับตัวได้

5.3 การดำเนินงานในอนาคต

จากการพัฒนาต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ ผู้ทำวิจัยมีแนวทางในการดำเนินงานในอนาคต ดังนี้

- ปรับปรุงประสิทธิภาพ การลดเสียงรบกวนภายนอก ของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ให้ดียิ่งขึ้น
- เพิ่มความสามารถของต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์ ในส่วนของการบันทึกข้อมูลเสียง เล่นข้อมูลเสียงที่บันทึก และ ส่งข้อมูลเสียงไร้สายไปแสดงผลบนเครื่องคอมพิวเตอร์
- ประกอบวงจรต่างๆ ลงบนแผ่นวงจรพิมพ์ และจัดใส่กล่อง เพื่อให้ต้นแบบหูฟังแพทย์แบบอิเล็กทรอนิกส์สามารถพกพาได้สะดวกขึ้น

รายการอ้างอิง

[1] Marie-Claude Grenier, Katerie GagnonD, Jacques Genest, Jr., Jocelyn Durand, and Louis-Gilles Durand. Clinical comparison of acoustic and electronic stethoscopes and design of a new electronic stethoscope. American Journal of Cardiology (March 1998) : pp. 653-656.

[2] Jocelyn Durand, Louis-Gilles Durand and Marie-Claude Grenien, Electronic stethescope. U.S. Patent 5602924, February 11, 1997.

[3] P. Várady. Wavelet-Based Adaptive Denoising of Phonocardiographic Records. Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE, pp. 1846 – 1849, 2001.

[4] N. Jatupaiboon, S. Pan-ngum and P. Israsena. Development of an Electronic Stethoscope Prototype. 2nd ECTI-Conference on Application Research and Development, pp. 42-47, 2010.

[5] Wikipedia. Sound [Online]. Available from : http://en.wikipedia.org/wiki/Sound [2010, August 16]

[6] จินดา สามัคคี. การพัฒนาเครื่องต้นแบบเพื่อเก็บบันทึกและวิเคราะห์เสียงเต้นของหัวใจจากหลายตำแหน่งบริเวณหน้าอก. วิทยานิพนธ์ปริญญามหาบัณฑิต, สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น, 2546.

[7] พิพัฒน์ นพที่ปกังวาล. หมวดที่ 5 ความปลอดภัยสภาพแวดล้อมในการทำงาน [Online]. Available from : http://kmcenter.rid.go.th/kmc01/pdf/7/4/safty/g5.pdf [2010, August 16]

[8] พรชัย ภววงษ์ศักดิ์. การประมวลผลสัญญาณดิจิตอลเบื้องต้น [Online]. Available from: http://www.kmitl.ac.th/~kskasems/dsp/DSP_R10.pdf [2010, August 16]

[9] Haykin, Simon. Adaptive Filter Theory. Prentice-Hall, 2001, pp. 203-212, 231-278.

[10] F. Belloni, D. Della Giustina, S. Riboldi, M. Riva and E. Spoletini. Towards a Computer-Aided Diagnosis by means of Phonocardiogram Signals. Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium, pp. 2770 – 2775, 2007.

[11] Yi Luo. Portable Bluetooth Visual Electrical Stethoscope Research. Communication Technology, 2008. ICCT 2008. 11th IEEE International Conference, 2008.

[12] K. Hung and Y.T. Zhang. Usage of Bluetooth in Wireless Sensors for Tele-Healthcare. EMBS/BMES Conference, 2002. Proceedings of the Second Joint, pp. 1881 – 1882, 2002.

[13] Ying-Wen Bai and Chao-Lin Lu. Digital Stethoscope Uses the Adaptive Noise Cancellation Filter and the Chebyshev IIR Bandpass Filter to Reduce the Noise of the Heart Sound. Enterprise networking and Computing in Healthcare Industry, 2005. HEALTHCOM 2005. Proceedings of 7th International Workshop, pp. 278 – 281, 2005.

[14] Julie Johnson, David Hermann, Melody Witter, Etienne Cornu, Robert Brennan and Alain Dufaux. An Ultra-Low Power Subband-Based Electronic Stethoscope. Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference, 2006.

[15] B. Farhang-Boroujeny. Adaptive Filters : Theory and Applications. John Wiley and Sons Ltd., Chichester, 1998, pp. 293-320.

[16] Maxim Datasheet. MAX7426/MAX7427 [Online]. Available from : http://datasheets.maxim-ic.com/en/ds/MAX7426-MAX7427.pdf [2010, August 16]

[17] Microchip Datasheet. dsPIC33FJXXXGPX06/X08/X10 [Online]. Available from : http://www1.microchip.com/downloads/en/DeviceDoc/70286C.pdf [2010, August 16]

[18] Microchip Datasheet. 16-Bit Language Tools Libraries [Online]. Available from : http://ww1.microchip.com/downloads/en/DeviceDoc/51456F.pdf [2010, August 16]

[19] Microchip Application Notes. Using PWM to Generate Analog Output [Online]. Available from : http://ww1.microchip.com/downloads/en/AppNotes/00538c.pdf [2010, August 16]

[20] Heart Sound Data. Pan Systolic Murmur [Online]. Available from : http://solutions.3m.com/wps/portal/3M/en_US/Littmann/stethoscope/education/heart-lung-sounds [2010, August 16]

[21] Noise Data. White Noise, Pink Noise, Babble Noise and Factory Noise [Online]. Available from : http://spib.rice.edu/spib/select_noise.html [2010, August 16]

[22] โศรฎา แข็งการ และ กนต์ธร ชำนิประศาสน์. การใช้ MATLAB สำหรับงานทางวิศวกรรม [Online]. Available from : www.kmutt.ac.th/science/book/intromatlab_th.pdf [2010, August 16]

ภาคผนวก

ภาคผนวก ก.
สัญญาณเสียงที่ใช้ในการทดลอง

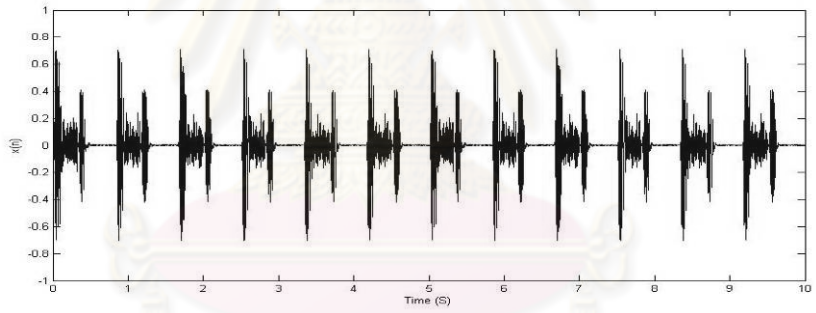
สัญญาณเสียงที่ใช้อยู่ในรูปแบบไฟล์นามสกุล Wave ความยาว 10 วินาที มีอัตราสุ่ม 8000 Hz ขนาด 16 Bits ข้อมูลเสียงที่ใช้มี 2 ชนิด คือ

- ข้อมูลเสียงจำลอง (Simulated Data) ได้มาจาก โปรแกรมสร้างข้อมูลเสียงจำลอง
- ข้อมูลเสียงจริง (Actual Data) ได้มาจาก การอัดเสียงหัวใจในสภาวะที่มีเสียงรบกวนภายนอก

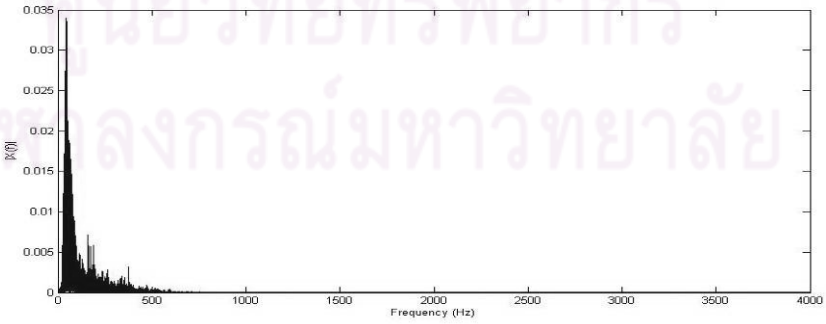
โดยสัญญาณเสียงที่ใช้ในงานวิจัยนี้ ประกอบไปด้วย สัญญาณเสียงหัวใจ และสัญญาณเสียงรบกวน ดังนี้

1. <u>สัญญาณเสียงหัวใจ</u> [20]

เลือกใช้สัญญาณเสียงหัวใจของคนที่เป็นโรค Pan Systolic Murmur ซึ่งมีเสียง Murmur ระหว่างเสียง S1 และ S2



สัญญาณเสียง Pan Systolic Murmur ในโดเมนเวลา

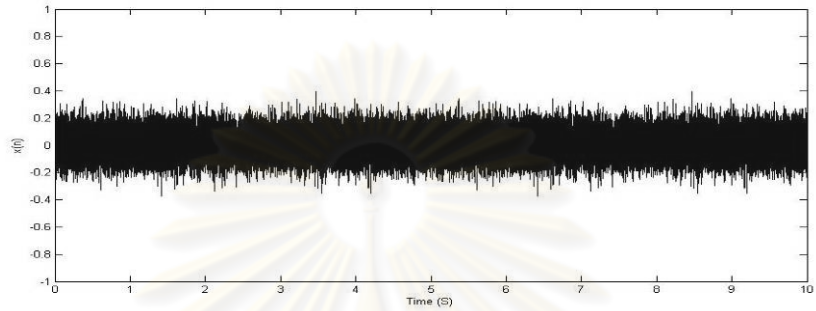


สัญญาณเสียง Pan Systolic Murmur ในโดเมนความถี่

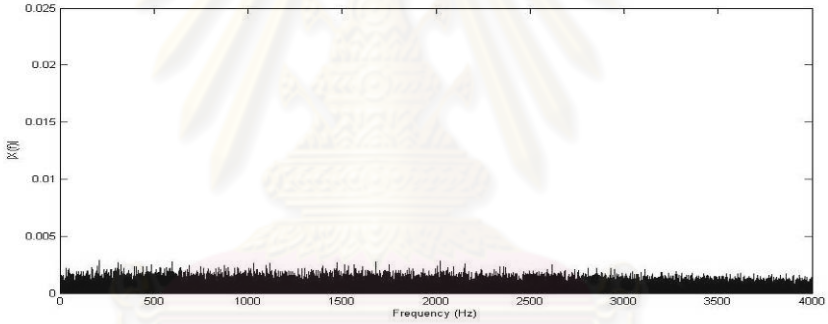
2. <u>สัญญาณเสียงรบกวนภายนอก</u> [21]

เลือกใช้สัญญาณเสียงรบกวนภายนอกที่มีความดังสม่ำเสมอ (Steady State Noise) ที่มีลักษณะแตกต่างกัน 4 ชนิด ดังนี้

2.1. White Noise เป็นสัญญาณเสียงรบกวนที่มีกำลังของสัญญาณในช่วงความถี่ต่างๆ เท่าๆ กัน

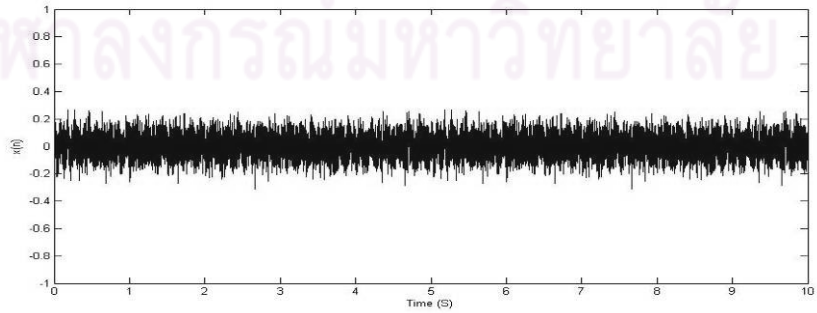


สัญญาณเสียง White Noise ในโดเมนเวลา

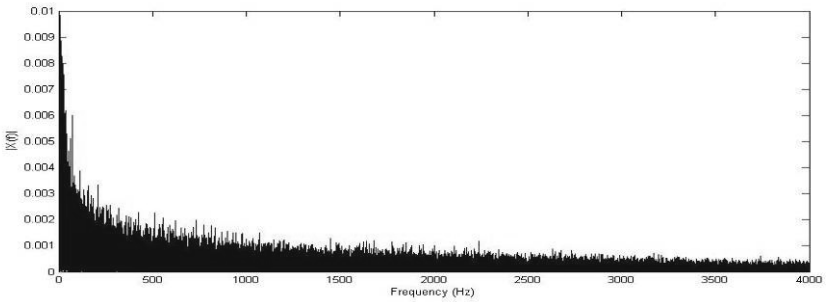


สัญญาณเสียง White Noise ในโดเมนความถี่

2.2. Pink Noise เป็นสัญญาณเสียงรบกวนที่กำลังของสัญญาณจะแปรผกผันกับความถี่ คือ ในช่วงความถี่ที่สูงขึ้นกำลังของสัญญาณจะลดลง

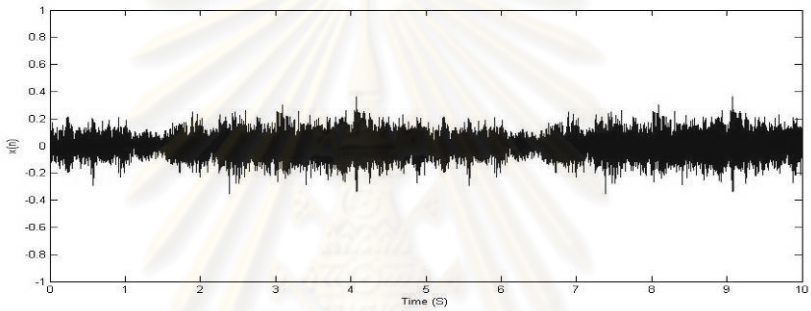


สัญญาณเสียง Pink Noise ในโดเมนเวลา

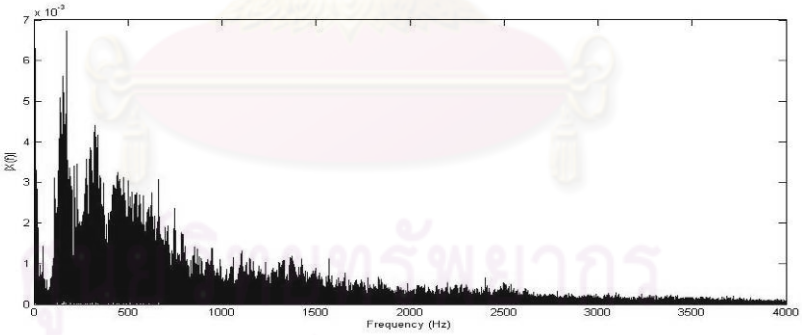


สัญญาณเสียง Pink Noise ในโดเมนความถี่

2.3. Babble Noise เป็นสัญญาณเสียงรบกวนที่บันทึกจากสภาพแวดล้อมที่มีคนจำนวนมากพูดคุยกัน

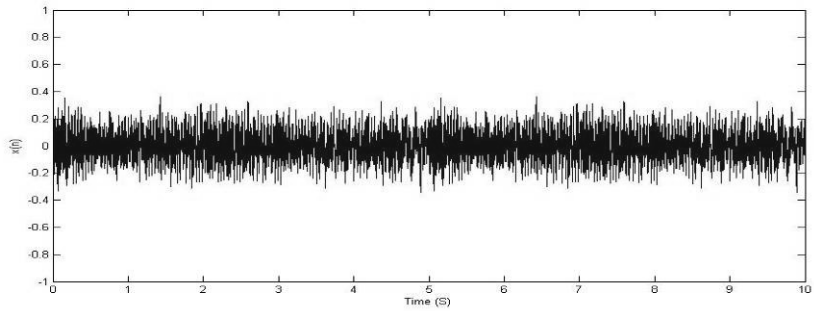


สัญญาณเสียง Babble Noise ในโดเมนเวลา

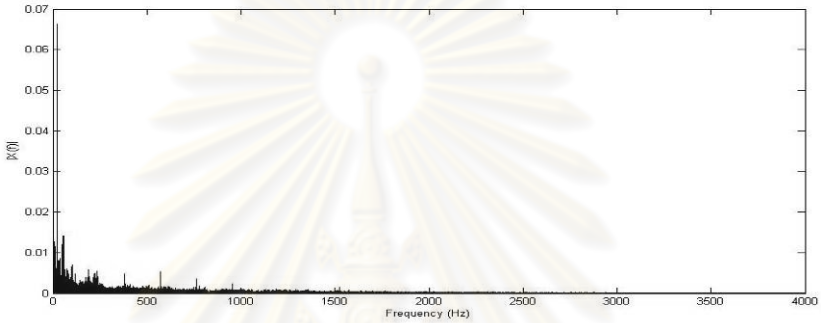


สัญญาณเสียง Babble Noise ในโดเมนความถี่

2.4. Factory Noise เป็นสัญญาณเสียงรบกวนที่บันทึกจากสภาพแวดล้อมที่มีเสียงการทำงานของเครื่องจักรในโรงงาน



สัญญาณเสียง Factory Noise ในโดเมนเวลา



สัญญาณเสียง Factory Noise ในโดเมนความถี่

ภาคผนวก ข.
โค้ดที่ใช้ในการคำนวณค่าต่างๆ บนโปรแกรม MATLAB

MATLAB เป็นโปรแกรมคอมพิวเตอร์สมรรถนะสูงเพื่อใช้ในการคำนวณทางเทคนิค MATLAB ได้รวมการคำนวณ การเขียนโปรแกรมและการแสดงผลรวมกันอยู่ในตัวโปรแกรมเดียว ได้อย่างมีประสิทธิภาพ และอยู่ในลักษณะที่ง่ายต่อการใช้งาน นอกจากนี้ลักษณะของการเขียนสมการในโปรแกรมก็จะเหมือนการเขียนสมการคณิตศาสตร์ที่เราคุ้นเคยดีอยู่แล้ว งานที่ทั่วไปที่ใช้ MATLAB ก็เช่น การคำการคำนวณทั่วไป การสร้างแบบจำลองและการทดสอบแบบจำลอง การวิเคราะห์ข้อมูล การแสดงผลในรูปกราฟทั้งโดยทั่วไปและกราฟทางด้านทางวิทยาศาสตร์และวิศวกรรม สามารถสร้างโปรแกรมในลักษณะที่ติดต่อกับผู้ใช้ทางกราฟฟิกส์ คำสั่งหลักๆ ของ MATLAB ที่ใช้งานในงานวิจัยนี้ คือ คำสั่งในส่วนของ ตัวดำเนินการ ฟังก์ชันทั่วไป และ ฟังก์ชันทางสถิติ [22]

ตัวดำเนินการ	การดำเนินการ
+	บวก
-	ลบ
*	คูณ
.*	คูณเชิงสมาชิก
/	หาร
./	หารเชิงสมาชิก
^	ยกกำลัง
.^	ยกกำลังเชิงสมาชิก

ฟังก์ชันทั่วไป	การดำเนินการ
abs(x)	หาค่า absolute ของ x
sqrt(x)	หาค่า รากที่ 2 ของ x
round(x)	ทำให้ x เป็นจำนวนเต็ม โดยปัดค่าให้เป็นจำนวนเต็มที่ใกล้ x ที่สุด
fix(x)	ทำให้ x เป็นจำนวนเต็ม โดยปัดค่า x ให้เป็นจำนวนเต็มที่ใกล้ศูนย์ที่สุด
floor(x)	ทำให้ x เป็นจำนวนเต็ม โดยปัดค่า x ให้เป็นจำนวนเต็มที่ใกล้ x ไปทาง -infinity

ceil(x)	ทำให้ x เป็นจำนวนเต็ม โดยปัดค่า x ให้เป็นจำนวนเต็มที่ใกล้ x ไปทาง + infinity
sign(x)	บอกเครื่องหมายของ x โดยจะเป็น -1 ถ้า x < 0, เป็น 1 ถ้า x > 0 และเป็น 0 ถ้า x = 0
rem(x,y)	หาเศษที่ได้จากการหาร x ด้วย y หรือ เศษของ x/y
exp(x)	หาค่า exponential ของ x
log(x)	หาค่า ln(x) หรือ natural logarithm ของ x
log10(x)	หาค่า log10 x หรือ logarithm ฐาน 10 ของ x

ฟังก์ชันทางสถิติ	**การดำเนินการ**
max(x)	จะให้ค่าที่มากที่สุดของ x ถ้า x เป็น matrix จะได้ผลเป็น row vector ที่บรรจุค่าสูงสุดในแต่ละ column ของ x ถ้า x เป็น complex number จะได้ผลเป็นค่าที่มีขนาดสูงสุด
[y,ind] = max(x)	จะให้ผลเป็น row vector y ที่บรรจุค่าที่สูงสุดในแต่ละ columnของ matrix x และ ind จะบรรจุตำแหน่งของแต่ละ element ในแต่ละcolumn ของ x
max(A,B)	จะให้ผลเป็น matrix มีขนาดเท่ากับ A และ B โดยแต่ละ element ที่ตำแหน่ง (i, j) จะเป็นค่าที่สูงสุดเมื่อเทียบระหว่าง aij และ bij สำหรับการคำนวณเพื่อหาค่าน้อยที่สุด จะใช้ function **min** ซึ่งมีการใช้เช่นเดียวกับ function max
sum(x)	ถ้า x เป็น vector จะได้ผลเป็นการรวมค่าทั้งหมดของ element ใน x แต่ถ้า x เป็น matrix จะได้ผลเป็น row vector ที่บรรจุผลบวกแต่ละ column ของ x ไว้
cumsum(x)	ถ้า x เป็น vector จะได้ผลเป็นการรวมสะสม ของ x นั่นคือ element ที่ 2 เป็นผลรวมของ element 1 กับ 2 ของ x , element ที่ 3 จะเป็นการรวมของ element ที่ 1, 2 และ 3 ของ x ไปเรื่อย ๆ ถ้า x เป็น matrix MATLAB จะพิจารณาเช่นเดียวกันในแต่ละ column ของ x
prod(x)	จะได้ผลคล้ายกับ **sum** แต่เปลี่ยนเป็นผลคูณของ element
cumprod(x)	จะได้ผลคล้ายกับ **cumsum** แต่เปลี่ยนเป็นผลคูณของ element
mean(x)	ถ้า x เป็น vector จะได้ผลเป็นค่ากลางของ element ของ x ถ้า x เป็น matrix จะได้ row vector ที่มีแต่ละ element เป็นค่ากลางของแต่ละ

	column ของ x
median(x)	ถ้า x เป็น vector จะได้ผลเป็น median ของ element ของ x ถ้า x เป็น matrix จะได้ row vector ที่มีแต่ละ element เป็นค่า medien ของแต่ละ column ของ x
std(x)	ถ้า x เป็น vector จะได้ผลเป็นส่วนเบี่ยงเบนมาตรฐาน (standard deviation) ของ element ทั้งหมดของ x ถ้า x เป็น matrix จะได้ผลเป็น row vector ที่แต่ละ element เป็นค่าเบี่ยงเบนมาตรฐานของแต่ละ column ของ x
cov(x)	ถ้า x เป็น vector จะได้ผลเป็น Varian ของ x ถ้า x เป็น matrix จะได้ผลเป็น diagonal matrix ซึ่งแต่ละค่าจะเป็น Varian ของแต่ละ column ของ x **corrcoef(A)** จะได้ผลเป็น correlation matrix ของ matrix A **sort(x)** ถ้า x เป็น vector จะได้ผลเป็น vector ที่มีการเรียงลำดับของ element จากค่าน้อยไปมาก ถ้า x เป็น matrix จะได้ผลเป็น matrix ขนาดเท่ากันโดยแต่ละ column จะเป็นการเรียงลำดับจากน้อยไปหามากของแต่ละ column ของ x
corrcoef(A)	จะได้ผลเป็น correlation matrix ของ matrix A
sort(x)	ถ้า x เป็น vector จะได้ผลเป็น vector ที่มีการเรียงลำดับของ element จากค่าน้อยไปมาก ถ้า x เป็น matrix จะได้ผลเป็น matrix ขนาดเท่ากันโดยแต่ละ column จะเป็นการเรียงลำดับจากน้อยไปหามากของแต่ละ column ของ x
[y,ind] = sort(x)	จะได้ y เป็นผลของคำสั่ง sort (x) ส่วน ind คือผลที่ได้ในการเทียบ index ของ element ก่อนมีการเรียงลำดับของ x

จากตาราง ตัวดำเนินการ ฟังก์ชันทั่วไป และ ฟังก์ชันทางสถิติ ทำให้เราสามารถเขียนโค้ดของโปรแกรมที่ใช้ในการคำนวณค่าจากสมการต่างๆ ที่ใช้ในงานวิจัย ได้ดังนี้

<u>สมการที่ 25</u> สามารถเขียนเป็นโค้ดโปรแกรมได้ดังนี้

```
MSE = (sum((denoise(1:sample)-original(1:sample)).^2))/sample
```

<u>สมการที่ 26</u> สามารถเขียนเป็นโค้ดโปรแกรมได้ดังนี้

```
SNR_noisy = 10*log10(mean(original.^2) / mean((noisy- original).^2))
```

<u>สมการที่ 27</u> สามารถเขียนเป็นโค้ดโปรแกรมได้ดังนี้

```
SNR_denoise = 10*log10(mean(original.^2) / mean((denoise- original).^2))
```

โดยที่ original คือ สัญญาณเสียงหัวใจที่ไม่มีเสียงรบกวนภายนอก ($S_{original}$)

denoise คือ สัญญาณเสียงหัวใจที่ผ่านการลดเสียงรบกวนภายนอก ($S_{denoise}$)

noisy คือ สัญญาณเสียงหัวใจที่มีเสียงรบกวนภายนอก (S_{noisy})

sample คือ จำนวนจุดของสัญญาณที่ต้องการนำมาคำนวณ

ภาคผนวก ค.
โค้ดที่ใช้ควบคุมการทำงานของตัวประมวลผล dsPIC

ตัวประมวลผล dsPIC สนับสนุนการเขียนโปรแกรมควบคุมได้ทั้งภาษา Assembly และภาษา C โดยเฉพาะเครื่องมือพัฒนาโปรแกรมด้วย MPLAB C30 ได้เตรียมไลบรารีเชื่อมต่อภายนอก (dsPIC Peripheral Libraries) ให้พร้อมใช้เพื่อควมคุมการใช้งานโมดูลต่างๆ ใน dsPIC รวมถึงไลบรารีที่เกี่ยวกับการประมวลผลสัญญาณดิจิตอล (DSP Library) และไลบรารีมาตรฐานภาษา C พร้อมกับฟังก์ชันคณิตศาสตร์ (Standart C Libraries with Math Functions)

โค้ดของโปรแกรมที่ใช้ควมคุมการทำงานของตัวประมวลผล dsPIC จะประกอบไปด้วย 2 ส่วนหลัก คือ Sources Files และ Headrer Filtes

1. Sources Files
   - 1.1. main.c - โปรแกรมส่วนที่ใช้ควบคุมการทำงานของตัวประมวลผล dsPIC
   - 1.2. adcdacDrv.c - โปรแกรมส่วนที่ใช้ในการแปลงสัญญาณแอนะลอกเป็นดิจิตอลและแปลงสัญญาณดิจิตอลกลับมาเป็นสัญญาณแอนะลอก
   - 1.3. switch.c - โปรแกรมส่วนที่ใช้ควมคุมการทำงานของสวิตซ์
   - 1.4. Bell.s - โปรแกรมที่เก็บค่าน้ำหนักที่ใช้กรองเสียงให้อยู่ในช่วงความถี่ Bell
   - 1.5. Diaphragm.s - โปรแกรมที่เก็บค่าน้ำหนักที่ใช้กรองเสียงให้อยู่ในช่วงความถี่ Diaphragm
   - 1.6. Extended.s - โปรแกรมที่เก็บค่าน้ำหนักที่ใช้กรองเสียงให้อยู่ในช่วงความถี่ Extened
   - 1.7. fir.s - โปรแกรมการทำงานของตัวกรองธรรมดา
   - 1.8. firlms.s - โปรแกรมการทำงานของตัวกรองแบบปรับตัวได้โดยใช้อัลกอริทึม LMS
   - 1.9. firlmsn.s - โปรแกรมการทำงานของตัวกรองแบบปรับตัวได้โดยใช้อัลกอริทึม NLMS
2. Header Files
   - 2.1. dsp.h - ไลบรารีการประมวลผลสัญญาณดิจิตอล
   - 2.2. adcdacDrv.h – กำหนดรายละเอียดของฟังก์ชันและค่าต่างๆ ให้กับ adcdacDrv.c
   - 2.3. switch.h – กำหนดรายละเอียดของฟังก์ชัน และค่าต่างๆ ให้กับ switch.c

main.c

```
#include     "p33fxxxx.h"
#include     "stdio.h"
#include     "..\h\dsp.h"
#include     "..\h\adcdacDrv.h"
#include     "..\h\switch.h"
_FOSCSEL  ( FNOSC_FRC );
_FOSC( FCKSM_CSECMD & OSCIOFNC_ON & POSCMD_NONE );
_FWDT( FWDTEN_OFF );
        int         debounce = 0;
extern  int         i;
extern  int         j;
extern  char        m;
extern  char        f;
extern  char        a;
extern  fractional signalIn1A[ PROC_BLOCK_SIZE ];
extern  fractional signalIn1B[ PROC_BLOCK_SIZE ];
extern  fractional signalOut1A[ PROC_BLOCK_SIZE ];
extern  fractional signalOut1B[ PROC_BLOCK_SIZE ];
extern  fractional signalIn2A[ PROC_BLOCK_SIZE ];
extern  fractional signalIn2B[ PROC_BLOCK_SIZE ];
extern  fractional signalOut2A[ PROC_BLOCK_SIZE ];
extern  fractional signalOut2B[ PROC_BLOCK_SIZE ];
extern  fractional signalY2A[ PROC_BLOCK_SIZE ];
extern  fractional signalY2B[ PROC_BLOCK_SIZE ];
extern  fractional LMSmu;
extern  fractional energy[ PROC_BLOCK_SIZE ];
extern        fractional        LMSCoef[LMS_SIZE]        __attribute__
((space(xmemory),far,aligned(PROC_BLOCK_SIZE)));
extern       fractional       DelayBufI[LMS_SIZE]       __attribute__
((space(ymemory),far,aligned(PROC_BLOCK_SIZE)));
extern  FIRStruct  FIRFilterI;
extern  FIRStruct  BellFilter;
extern  FIRStruct  DiaphragmFilter;
extern  FIRStruct  ExtendedFilter;
int main( void ) {
    PLLFBD            = 41;
    CLKDIVbits.PLLPOST = 0;
    CLKDIVbits.PLLPRE  = 0;
    OSCTUN            = 0;
    RCONbits.SWDTEN    = 0;
    __builtin_write_OSCCONH( 0x01 );
    __builtin_write_OSCCONL( 0x01 );
    while( OSCCONbits.COSC != 0b001 )
        ;
    while( OSCCONbits.LOCK != 1 )
    {
    }
    ;
    initSwitch( &debounce );
    initAdc( );
    initTmr3( );
    initOC2( );
    initOC3( );
    initTmr2( );
    initDma0( );
    InitProcDSP( );
    while( 1 ) {
        if( debounce > 0 )
```

```
            debounce--;
        if( CheckSwitchS1( ) == 1 )
        {
            if( LED1 == 1 ) {
                LED1 = 0;
                for( i = 0;
                     i < LMS_SIZE;
                     i++ )
                    LMSCoef[ i ] = 0;
                LMSmu = Q15( 0.01 );
                a     = 'L';
            }
            else
                if( LED1 == 0 )
                {
                    LED1 = 1;
                    a     = ' ';

                }

        }
        if( CheckSwitchS2( ) == 1 )
        {
            if( LED2 == 1 ) {
                LED2 = 0;
                for( i = 0;
                     i < LMS_SIZE;
                     i++ )
                    LMSCoef[ i ] = 0;
                LMSmu = Q15( 0.005 );
                a     = 'N';
            }
            else
                if( LED2 == 0 )
                {
                    LED2 = 1;
                    a     = ' ';

                }
        }

        if( CheckSwitchS3( ) == 1 )
        {
            if( LED3 == 1 ) {
                LED3 = 0;
                f    = 'B';
            }
            else
                if( LED3 == 0 )
                {
                    LED3 = 1;
                    f    = 'D';
                }
        }
        if( i == PROC_BLOCK_SIZE )
        {
            i = 0;
            if( m == 'A' ) {
                m = 'B';
                if( a == 'L' )
```

```
                    FIRLMS( PROC_BLOCK_SIZE, signalY2A,
                            signalIn2A, &FIRFilterI,
                            signalIn1A, LMSmu );
                else
                    if( a == 'N' )
                        FIRLMSNorm( PROC_BLOCK_SIZE,
                           signalY2A, signalIn2A,
                           &FIRFilterI, signalIn1A,
                           LMSmu, energy );
                VectorSubtract( PROC_BLOCK_SIZE,
                                signalOut1A, signalIn1A,
                                signalY2A );
                VectorCopy( PROC_BLOCK_SIZE,
                            signalOut2A, signalIn2A );
                if( f == 'B' )
                    FIR( PROC_BLOCK_SIZE, signalOut1A,
                         signalOut1A, &BellFilter );
                else
                    if( f == 'D' )
                        FIR( PROC_BLOCK_SIZE,
                             signalOut1A, signalOut1A,
                             &DiaphragmFilter );
                    else
                        if( f == 'E' )
                        FIR( PROC_BLOCK_SIZE,
                             signalOut1A, signalOut1A,
                             &ExtendedFilter );
            }
            else
                if( m == 'B' ) {
                    m = 'A';
                    if( a == 'L' )
                        FIRLMS( PROC_BLOCK_SIZE,
                                signalY2B, signalIn2B,
                                &FIRFilterI, signalIn1B,
                                LMSmu );
                    else
                        if( a == 'N' )
                        FIRLMSNorm( PROC_BLOCK_SIZE,
                           signalY2B, signalIn2B,
                           &FIRFilterI, signalIn1B,
                           LMSmu, energy );
                    VectorSubtract( PROC_BLOCK_SIZE,
                       signalOut1B, signalIn1B,
                       signalY2B );
                    VectorCopy( PROC_BLOCK_SIZE,
                                signalOut2B,
                                signalIn2B );
                    if( f == 'B' )
                        FIR( PROC_BLOCK_SIZE,
                             signalOut1B, signalOut1B,
                             &BellFilter );
                    else
                        if( f == 'D' )
                        FIR( PROC_BLOCK_SIZE,
                             signalOut1B, signalOut1B,
                             &DiaphragmFilter );
                        else
                        if( f == 'E' )
                        FIR( PROC_BLOCK_SIZE,
```

```
                                signalOut1B, signalOut1B,
                                &ExtendedFilter );
                }
        }
    }
}
```

adcdacDrv.c

```
#include     "p33fxxxx.h"
#include     "stdio.h"
#include     "..\h\dsp.h"
#include     "..\h\adcdacDrv.h"
        int         i = 0;
        int         j = 0;
        char        m = 'A';
        char        f = 'E';
        char        a = ' ';
        fractional signalIn1A[ PROC_BLOCK_SIZE ];
        fractional signalIn1B[ PROC_BLOCK_SIZE ];
        fractional signalOut1A[ PROC_BLOCK_SIZE ];
        fractional signalOut1B[ PROC_BLOCK_SIZE ];
        fractional signalIn2A[ PROC_BLOCK_SIZE ];
        fractional signalIn2B[ PROC_BLOCK_SIZE ];
        fractional signalOut2A[ PROC_BLOCK_SIZE ];
        fractional signalOut2B[ PROC_BLOCK_SIZE ];
        fractional signalY2A[ PROC_BLOCK_SIZE ];
        fractional signalY2B[ PROC_BLOCK_SIZE ];
        fractional LMSmu;
        fractional energy[ PROC_BLOCK_SIZE ];
        fractional  LMSCoef[  LMS_SIZE  ]  __attribute__(  (  space  (
xmemory ), far, aligned ( PROC_BLOCK_SIZE ) ) ) ;
        fractional  DelayBufI[  LMS_SIZE  ]  __attribute__(  (  space  (
ymemory ), far, aligned ( PROC_BLOCK_SIZE ) ) ) ;
        FIRStruct  FIRFilterI;
extern  FIRStruct  BellFilter;
extern  FIRStruct  DiaphragmFilter;
extern  FIRStruct  ExtendedFilter;
void initAdc( void )
{
    AD1CON1bits.AD12B    = 0;
    AD1CON1bits.FORM     = 1;
    AD1CON1bits.SSRC     = 2;
    AD1CON1bits.SIMSAM   = 1;
    AD1CON1bits.ASAM     = 1;
    AD1CON2bits.SMPI     = 1;
    AD1CON2bits.BUFM     = 0;
    AD1CON2bits.CHPS     = 1;
    AD1CON3bits.ADRC     = 0;
    AD1CON3bits.ADCS     = 3;
    AD1CHS0bits.CH0SA    = 2;
    AD1CHS0bits.CH0NA    = 0;
    AD1CHS123bits.CH123SA = 1;
    AD1CHS123bits.CH123NA = 0;
    AD1PCFGL             = 0xFFFF;
    AD1PCFGLbits.PCFG1   = 0;
    AD1PCFGLbits.PCFG2   = 0;
    IFS0bits.AD1IF       = 0;
    IEC0bits.AD1IE       = 0;
```

```
    AD1CON1bits.ADON      = 1;
}
struct
{
    int Ch1;
    int Ch2;

} BufferA __attribute__( ( space ( dma ) ) ) ;
struct
{
    int Ch1;
    int Ch2;
} BufferB __attribute__( ( space ( dma ) ) ) ;
;
void initDma0( void )
{
    DMA0CONbits.AMODE = 0;
    DMA0CONbits.MODE  = 2;
    DMA0PAD           = ( int )&ADC1BUF0;
    DMA0CNT           = 1;
    DMA0REQ           = 13;
    DMA0STA           = __builtin_dmaoffset( &BufferA );
    DMA0STB           = __builtin_dmaoffset( &BufferB );
    IFS0bits.DMA0IF   = 0;
    IEC0bits.DMA0IE   = 1;
    DMA0CONbits.CHEN  = 1;
}
unsigned  int DmaBuffer = 0;
void   __attribute__( ( interrupt, no_auto_psv ) ) _DMA0Interrupt(
void )
{
    IFS0bits.DMA0IF = 0;
    if( DmaBuffer == 0 )
    {
        if( m == 'A' )
        {
            signalIn1A[ i ] = SCALE * BufferA.Ch1;
            signalIn2A[ i ] = SCALE * BufferA.Ch2;
        }
        else
            if( m == 'B' )
            {
                signalIn1B[ i ] = SCALE * BufferA.Ch1;
                signalIn2B[ i ] = SCALE * BufferA.Ch2;
            }
    }
    else
    {
        if( m == 'A' )
        {
            signalIn1A[ i ] = SCALE * BufferB.Ch1;
            signalIn2A[ i ] = SCALE * BufferB.Ch2;
        }
        else
            if( m == 'B' )
            {
                signalIn1B[ i ] = SCALE * BufferB.Ch1;
                signalIn2B[ i ] = SCALE * BufferB.Ch2;
            }
    }
```

```
    if( m == 'A' )
    {

        OC2RS = 512  + signalOut1A[ i ] / SCALE;
        OC3RS = 512  + signalOut2A[ i ] / SCALE;

    }
    else
        if( m == 'B' )
        {

            OC2RS = 512  + signalOut1B[ i ] / SCALE;
            OC3RS = 512  + signalOut2B[ i ] / SCALE;

        }
    DmaBuffer ^= 1;
    i++;
}
void InitProcDSP( void )
{
    FIRStructInit( &FIRFilterI, LMS_SIZE, LMSCoef, COEFFS_IN_DATA,
                   DelayBufI );
    for( i = 0; i < LMS_SIZE; i++ )
        LMSCoef[ i ] = 0;
    FIRDelayInit( &FIRFilterI );
    LMSmu = Q15( 0.005 );
    FIRDelayInit( &BellFilter );
    FIRDelayInit( &DiaphragmFilter );
    FIRDelayInit( &ExtendedFilter );
}
void initTmr3( void )
{
    T3CONbits.TON   = 0;
    T3CONbits.TCS   = 0;
    T3CONbits.TGATE = 0;
    T3CONbits.TSIDL = 1;
    TMR3            = 0;
    PR3             = ( Fcy / 8000 ) - 1;
    IFS0bits.T3IF   = 0;
    IEC0bits.T3IE   = 0;
    T3CONbits.TON   = 1;
}
void initOC2( void )
{
    OC2RS             = 512;
    OC2R              = 0;
    OC2CONbits.OCSIDL = 1;
    OC2CONbits.OCTSEL = 0;
    OC2CONbits.OCM    = 0x06;
}
void initOC3( void )
{
    OC3RS             = 512;
    OC3R              = 0;
    OC3CONbits.OCSIDL = 1;
    OC3CONbits.OCTSEL = 0;
    OC3CONbits.OCM    = 0x06;
}
void initTmr2( void )
{
```

```
    T2CONbits.TON   = 0;
    T2CONbits.TCS   = 0;
    T2CONbits.TGATE = 0;
    T2CONbits.TSIDL = 1;
    TMR2            = 0;
    PR2             = 1024;
    T2CONbits.TON   = 1;
}
```

switch.c

```
#include     <p33Fxxxx.h>
#include     "..\h\switch.h"
static            int * debounceCounter;
static  volatile  int  switchS1;
static  volatile  int  switchS2;
static  volatile  int  switchS3;
static  volatile  int  switchS4;
void initSwitch( int  *debounce )
{
    ADPCFG          = 0xFFFF;
    SW1_TRIS        = 1;
    SW2_TRIS        = 1;
    SW3_TRIS        = 1;
    SW4_TRIS        = 1;
    LED1_TRIS       = 0;
    LED2_TRIS       = 0;
    LED3_TRIS       = 0;
    LED4_TRIS       = 0;
    LED1            = 1;
    LED2            = 1;
    LED3            = 1;
    LED4            = 1;
    debounceCounter = debounce;
    switchS1        = 0;
    switchS2        = 0;
    switchS3        = 0;
    switchS4        = 0;
    INTCON2         = 0x001E;
    IFS1bits.INT1IF = 0;
    IEC1bits.INT1IE = 1;
    IFS1bits.INT2IF = 0;
    IEC1bits.INT2IE = 1;
    IFS3bits.INT3IF = 0;
    IEC3bits.INT3IE = 1;
    IFS3bits.INT4IF = 0;
    IEC3bits.INT4IE = 1;
}
int CheckSwitchS1( void )
{
    int wasPressed = 0;
    if( switchS1 == 1 )
    {
        wasPressed = 1;
        switchS1   = 0;
    }
    return ( wasPressed );
}
int CheckSwitchS2( void )
```

```
{
    int wasPressed = 0;
    if( switchS2 == 1 )
    {
        wasPressed = 1;
        switchS2   = 0;
    }
    return ( wasPressed );
}
int CheckSwitchS3( void )
{
    int wasPressed = 0;
    if( switchS3 == 1 )
    {
        wasPressed = 1;
        switchS3   = 0;
    }
    return ( wasPressed );
}

int CheckSwitchS4( void )
{
    int wasPressed = 0;
    if( switchS4 == 1 )
    {
        wasPressed = 1;
        switchS4   = 0;
    }
    return ( wasPressed );
}
void   __attribute__( ( interrupt, no_auto_psv ) ) _INT1Interrupt(
void )
{
    _INT1IF = 0;
    if( *debounceCounter == 0 )
    {
        *debounceCounter = DEBOUNCE_INTERVAL;
        switchS1         = 1;
    }
}

void   __attribute__( ( interrupt, no_auto_psv ) ) _INT2Interrupt(
void )
{
    _INT2IF = 0;
    if( *debounceCounter == 0 )
    {
        *debounceCounter = DEBOUNCE_INTERVAL;
        switchS2         = 1;
    }
}
void   __attribute__( ( interrupt, no_auto_psv ) ) _INT3Interrupt(
void )
{
    _INT3IF = 0;
    if( *debounceCounter == 0 )
    {
        *debounceCounter = DEBOUNCE_INTERVAL;
        switchS3         = 1;
    }
```

```
}
void   __attribute__(  (  interrupt,  no_auto_psv  )  ) _INT4Interrupt(
void )
{
    _INT4IF = 0;
    if( *debounceCounter == 0 )
    {
        *debounceCounter = DEBOUNCE_INTERVAL;
        switchS4         = 1;
    }
}
```

Bell.s

```
; ....................................................................
;    File   Bell.s
; ....................................................................

               .equ BellNumTaps, 64

; ....................................................................
; Allocate and initialize filter taps

               .section .xdata,data,xmemory
               .align 128

BellTaps:
.hword  0xFEE1, 0xFEEB, 0xFEFB, 0xFF11, 0xFF2D, 0xFF50, 0xFF79, 0xFFA8, 0xFFDC
.hword  0x0015, 0x0054, 0x0098, 0x00DF, 0x012A, 0x0177, 0x01C6, 0x0216, 0x0267
.hword  0x02B7, 0x0305, 0x0352, 0x039B, 0x03E1, 0x0422, 0x045E, 0x0494, 0x04C3
.hword  0x04EB, 0x050C, 0x0525, 0x0536, 0x053E, 0x053E, 0x0536, 0x0525, 0x050C
.hword  0x04EB, 0x04C3, 0x0494, 0x045E, 0x0422, 0x03E1, 0x039B, 0x0352, 0x0305
.hword  0x02B7, 0x0267, 0x0216, 0x01C6, 0x0177, 0x012A, 0x00DF, 0x0098, 0x0054
.hword  0x0015, 0xFFDC, 0xFFA8, 0xFF79, 0xFF50, 0xFF2D, 0xFF11, 0xFEFB, 0xFEEB
.hword  0xFEE1

; ....................................................................
; Allocate delay line in (uninitialized) Y data space

               .section .ydata, data, ymemory
               .align 128

BellDelay:
               .space BellNumTaps*2

; ....................................................................
; Allocate and intialize filter structure

               .section .data
               .global _BellFilter

_BellFilter:
.hword BellNumTaps
.hword BellTaps
.hword BellTaps+BellNumTaps*2-1
.hword 0xff00
.hword BellDelay
.hword BellDelay+BellNumTaps*2-1
.hword BellDelay

; ....................................................................
; ....................................................................
; Sample assembly language calling program
;  The following declarations can be cut and pasted as needed into a program
;              .extern _FIRFilterInit
;              .extern _BlockFIRFilter
;              .extern _BellFilter
;
;              .section       .bss
;
```

```
;       The input and output buffers can be made any desired size
;         the value 40 is just an example - however, one must ensure
;         that the output buffer is at least as long as the number of samples
;         to be filtered (parameter 4)
;input:        .space  40
;output:       .space  40
;             .text
;
;
;  This code can be copied and pasted as needed into a program
;
;
; Set up pointers to access input samples, filter taps, delay line and
; output samples.
;             mov     #_BellFilter, W0      ; Initalize W0 to filter structure
;             call    _FIRFilterInit ; call this function once
;
; The next 4 instructions are required prior to each subroutine call
; to _BlockFIRFilter
;             mov     #_BellFilter, W0      ; Initalize W0 to filter structure
;             mov     #input, W1     ; Initalize W1 to input buffer
;             mov     #output, W2    ; Initalize W2 to output buffer
;             mov     #20, W3 ; Initialize W3 with number of required output samples
;             call    _BlockFIRFilter       ; call as many times as needed
```

Diaphragml.s

```
; ......................................................................
;    File   Diaphragm.s
; ......................................................................

              .equ DiaphragmNumTaps, 64

; ......................................................................
; Allocate and initialize filter taps

              .section .xdata,data,xmemory
              .align 128

DiaphragmTaps:
.hword  0xFF12, 0xFEAB, 0xFE68, 0xFE50, 0xFE65, 0xFEA2, 0xFEFD, 0xFF66, 0xFFCC
.hword  0x0018, 0x003A, 0x0023, 0xFFCD, 0xFF34, 0xFE63, 0xFD6B, 0xFC64, 0xFB6B
.hword  0xFA9F, 0xFA20, 0xFA05, 0xFA62, 0xFB3D, 0xFC92, 0xFE50, 0x005B, 0x0291
.hword  0x04C5, 0x06CD, 0x087D, 0x09B4, 0x0A55, 0x0A55, 0x09B4, 0x087D, 0x06CD
.hword  0x04C5, 0x0291, 0x005B, 0xFE50, 0xFC92, 0xFB3D, 0xFA62, 0xFA05, 0xFA20
.hword  0xFA9F, 0xFB6B, 0xFC64, 0xFD6B, 0xFE63, 0xFF34, 0xFFCD, 0x0023, 0x003A
.hword  0x0018, 0xFFCC, 0xFF66, 0xFEFD, 0xFEA2, 0xFE65, 0xFE50, 0xFE68, 0xFEAB
.hword  0xFF12

; ......................................................................
; Allocate delay line in (uninitialized) Y data space

              .section .ydata, data, ymemory
              .align 128

DiaphragmDelay:
              .space DiaphragmNumTaps*2

; ......................................................................
; Allocate and intialize filter structure

              .section .data
              .global _DiaphragmFilter

_DiaphragmFilter:
.hword DiaphragmNumTaps
.hword DiaphragmTaps
.hword DiaphragmTaps+DiaphragmNumTaps*2-1
.hword 0xff00
.hword DiaphragmDelay
.hword DiaphragmDelay+DiaphragmNumTaps*2-1
.hword DiaphragmDelay
```

```
; ......................................................................
; ......................................................................
; Sample assembly language calling program
;  The following declarations can be cut and pasted as needed into a program
;              .extern _FIRFilterInit
;              .extern _BlockFIRFilter
;              .extern _DiaphragmFilter
;
;              .section       .bss
;
;       The input and output buffers can be made any desired size
;         the value 40 is just an example - however, one must ensure
;         that the output buffer is at least as long as the number of samples
;         to be filtered (parameter 4)
;input:        .space  40
;output:       .space  40
;              .text
;
;
;  This code can be copied and pasted as needed into a program
;
;
; Set up pointers to access input samples, filter taps, delay line and
; output samples.
;              mov     #_DiaphragmFilter, W0  ; Initalize W0 to filter structure
;              call    _FIRFilterInit ; call this function once
;
; The next 4 instructions are required prior to each subroutine call
; to _BlockFIRFilter
;              mov     #_DiaphragmFilter, W0  ; Initalize W0 to filter structure
;              mov     #input, W1     ; Initalize W1 to input buffer
;              mov     #output, W2    ; Initalize W2 to output buffer
;              mov     #20, W3 ; Initialize W3 with number of required output samples
;              call    _BlockFIRFilter        ; call as many times as needed
```

Extended.s

```
; ......................................................................
;    File   Extended.s
; ......................................................................

               .equ ExtendedNumTaps, 64

; ......................................................................
; Allocate and initialize filter taps

               .section .xdata,data,xmemory
               .align 128

ExtendedTaps:
.hword  0xFED6, 0xFFC7, 0x013D, 0x004B, 0xFEB0, 0xFFA2, 0x0168, 0x0075, 0xFE7F
.hword  0xFF72, 0x01A0, 0x00AE, 0xFE3E, 0xFF2D, 0x01EE, 0x0102, 0xFDDE, 0xFEC5
.hword  0x0266, 0x0189, 0xFD41, 0xFE0F, 0x033F, 0x028C, 0xFC01, 0xFC7E, 0x054A
.hword  0x0554, 0xF7F6, 0xF5F9, 0x121D, 0x344A, 0x344A, 0x121D, 0xF5F9, 0xF7F6
.hword  0x0554, 0x054A, 0xFC7E, 0xFC01, 0x028C, 0x033F, 0xFE0F, 0xFD41, 0x0189
.hword  0x0266, 0xFEC5, 0xFDDE, 0x0102, 0x01EE, 0xFF2D, 0xFE3E, 0x00AE, 0x01A0
.hword  0xFF72, 0xFE7F, 0x0075, 0x0168, 0xFFA2, 0xFEB0, 0x004B, 0x013D, 0xFFC7
.hword  0xFED6

; ......................................................................
; Allocate delay line in (uninitialized) Y data space

               .section .ydata, data, ymemory
               .align 128

ExtendedDelay:
               .space ExtendedNumTaps*2

; ......................................................................
; Allocate and intialize filter structure

               .section .data
               .global _ExtendedFilter
```

```
_ExtendedFilter:
.hword ExtendedNumTaps
.hword ExtendedTaps
.hword ExtendedTaps+ExtendedNumTaps*2-1
.hword 0xff00
.hword ExtendedDelay
.hword ExtendedDelay+ExtendedNumTaps*2-1
.hword ExtendedDelay

; ..........................................................................
; ..........................................................................
; Sample assembly language calling program
;  The following declarations can be cut and pasted as needed into a program
;              .extern _FIRFilterInit
;              .extern _BlockFIRFilter
;              .extern _ExtendedFilter
;
;              .section       .bss
;
;       The input and output buffers can be made any desired size
;         the value 40 is just an example - however, one must ensure
;         that the output buffer is at least as long as the number of samples
;         to be filtered (parameter 4)
;input:        .space  40
;output:       .space  40
;              .text
;
;
;  This code can be copied and pasted as needed into a program
;
;
; Set up pointers to access input samples, filter taps, delay line and
; output samples.
;              mov     #_ExtendedFilter, W0   ; Initalize W0 to filter structure
;              call    _FIRFilterInit ; call this function once
;
; The next 4 instructions are required prior to each subroutine call
; to _BlockFIRFilter
;              mov     #_ExtendedFilter, W0   ; Initalize W0 to filter structure
;              mov     #input, W1     ; Initalize W1 to input buffer
;              mov     #output, W2    ; Initalize W2 to output buffer
;              mov     #20, W3 ; Initialize W3 with number of required output samples
;              call    _BlockFIRFilter        ; call as many times as needed
```

fir.s

```
;*****************************************************************
;                                                                *
;                    Software License Agreement                  *
;                                                                *
;   The software supplied herewith by Microchip Technology       *
;   Incorporated (the "Company") for its dsPIC controller        *
;   is intended and supplied to you, the Company's customer,     *
;   for use solely and exclusively on Microchip dsPIC            *
;   products. The software is owned by the Company and/or its    *
;   supplier, and is protected under applicable copyright laws. All  *
;   rights are reserved. Any use in violation of the foregoing   *
;   restrictions may subject the user to criminal sanctions under    *
;   applicable laws, as well as to civil liability for the breach of *
;   the terms and conditions of this license.                    *
;                                                                *
;   THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION.  NO       *
;   WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING,    *
;   BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND    *
;   FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE     *
;   COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,  *
;   INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.  *
;                                                                *
;   (c) Copyright 2003 Microchip Technology, All rights reserved.    *
;*****************************************************************

        ; Local inclusions.
```

```
        .nolist
        .include       "dspcommon.inc"              ; MODCON, XMODSRT, XMODEND,
                                                    ; YMODSRT, YMODEND, CORCON,
                                                    ; PSVPAG, COEFFS_IN_DATA,
                                                    ; FIRStruct
        .list

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .section .libdsp, code

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; _FIR: FIR block filtering.
;
; Operation:
;       y[n] = sum_(m=0:M-1){h[m]*x[n-m]}, 0 <= n < N.
;
; x[n] defined for 0 <= n < N,
; y[n] defined for 0 <= n < N,
; h[m] defined for 0 <= m < M as an increasing circular buffer,
; NOTE: delay defined for 0 <= m < M as an increasing circular buffer.
;
; Input:
;       w0 = number of samples to generate (numSamps, N)
;       w1 = ptr to output samples (dstSamps, y)
;       w2 = ptr to input samples (srcSamps, x)
;       w3 = filter structure (FIRStruct, h)
;
; Return:
;       w0 = ptr to output samples (dstSamps, y)
;
; System resources usage:
;       {w0..w6}        used, not restored
;       {w8,w10}        saved, used, restored
;        AccuA          used, not restored
;        CORCON         saved, used, restored
;        PSVPAG         saved, used, restored (if coeffs in P memory)
;        MODCON         saved, used, restored
;        XMODSRT        saved, used, restored
;        XMODEND        saved, used, restored
;        YMODSRT        saved, used, restored
;        YMODEND        saved, used, restored
;
; DO and REPEAT instruction usage.
;       1 DO instructions
;       1 REPEAT intructions
;
; Program words (24-bit instructions):
;       56
;
; Cycles (including C-function call and return overheads):
;       53 + N*(5+M), or
;       56 + N*(9+M) if coefficients in P memory.
;.....................................................................

        .global _FIR    ; export
_FIR:

;.....................................................................

        ; Save working registers.
        push    w8                          ; w8 to TOS
        push    w10                         ; w10 to TOS

;.....................................................................

        ; Prepare CORCON for fractional computation.
        push    CORCON
        fractsetup      w8

;.....................................................................

        ; Prepare CORCON and PSVPAG for possible access of data
        ; located in program memory, using the PSV.
```

```
        push    PSVPAG

        mov     [w3+oCoeffsPage],w10          ; w10= coefficients page
        mov     #COEFFS_IN_DATA,w8            ; w8 = COEFFS_IN_DATA
        cp      w8,w10                        ; w8 - w10
        bra     z,_noPSV                      ; if w10 = COEFFS_IN_DATA
                                              ; no PSV management
                                              ; else
        psvaccess      w8                     ; enable PSV bit in CORCON
        mov     w10,PSVPAG                    ; load PSVPAG with program
                                              ; space page offset
_noPSV:

;.....................................................................

        ; Prepare core registers for modulo addressing.
        push    MODCON
        push    XMODSRT
        push    XMODEND
        push    YMODSRT
        push    YMODEND

;.....................................................................

        ; Setup registers for modulo addressing.
        mov     #0xC0A8,w10                   ; XWM = w8, YWM = w10
                                              ; set XMODEND and YMODEND bits
        mov     w10,MODCON                    ; enable X,Y modulo addressing

        mov     [w3+oCoeffsEnd],w8            ; w8 -> last byte of h[M-1]
        mov     w8,XMODEND                    ; init'ed to coeffs end address
        mov     [w3+oCoeffsBase],w8           ; w8 -> h[0]
        mov     w8,XMODSRT                    ; init'ed to coeffs base address
                                              ; (increasing buffer,
                                              ;  2^n aligned)
        mov     [w3+oDelayEnd],w10            ; w10-> last byte of d[M-1]
        mov     w10,YMODEND                   ; init'ed to delay end address
        mov     [w3+oDelayBase],w10           ; w10 -> d[0]
        mov     w10,YMODSRT                   ; init'ed to delay base address
                                              ; (increasing buffer,
                                              ;  2^n aligned)

;.....................................................................

        push    w1                            ; save return value (y)

;.....................................................................

        ; Perpare to all filter.
        mov     [w3+oDelay],w10                     ; w10 points at current delay
                                              ; sample d[m], 0 <= m < M
        mov     [w3+oNumCoeffs],w4            ; w4 = M
        sub     w4,#3,w4                      ; W4 = M-3
        dec     w0,w0                         ; w0 = N-1

;.....................................................................

        ; Perform filtering of all samples.
        do      w0,_endFilter         ; {     ; do (N-1)+1 times

        ; Prepare to filter sample.
        mov     [w2++],[w10]                  ; store new sample into delay

.ifdef YMEM_ERRATA
    nop
.endif

        clr     a,[w8]+=2,w5,[w10]+=2,w6      ; a  = 0
                                              ; w5 = h[0]
                                              ; w8-> h[1]
                                              ; w6 = d[current]
                                              ; w10->d[next]

        ; Filter each sample.
        ; (Perform all but two last MACs.)
```

```
        repeat  w4                      ; {     ; do (M-3)+1 times
        mac     w5*w6,a,[w8]+=2,w5,[w10]+=2,w6       ; a += h[m]*d[current]
                                        ; w5 = h[m+1]
                                        ; w8-> h[m+2]
                                        ; w6 = d[next]
                                        ; w10->d[next+1]
; }
        ; (Perform second last MAC.)
        mac     w5*w6,a,[w8]+=2,w5,[w10],w6   ; a += h[M-2]*d[current]
                                        ; w5 = h[M-1]
                                        ; w8-> h[0]
                                        ; w6 = d[next]
                                        ; w10->d[next]
        ; (Perform last MAC.)
        mac     w5*w6,a                 ; a += h[M-1]*d[current]

_endFilter:
        ; Save filtered result.
        sac.r   a,[w1++]                ; y[n] =
                                        ;   sum_{m=0:M-1}(h[m]*x[n-m])
                                        ; w1-> y[n+1]
; }

;.....................................................................

        ; Update delay pointer.
        mov     w10,[w3+oDelay]                 ; note that the delay pointer
                                        ; may wrap several times around
                                        ; d[m], 0 <= m < M, depending
                                        ; on the value of N

;.....................................................................

        pop     w0                      ; restore return value

;.....................................................................

        ; Restore core registers for modulo addressing.
        pop     YMODEND
        pop     YMODSRT
        pop     XMODEND
        pop     XMODSRT
        pop     MODCON

;.....................................................................

        ; Restore PSVPAG and CORCON.
        pop     PSVPAG
        pop     CORCON

;.....................................................................

        ; Restore working registers.
        pop     w10                     ; w10 from TOS
        pop     w8                      ; w8 from TOS

;.....................................................................

        return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; OEF
```

firlms.s

```
;*****************************************************************
;                                                                 *
;                       Software License Agreement                *
;                                                                 *
;   The software supplied herewith by Microchip Technology        *
;   Incorporated (the "Company") for its dsPIC controller         *
;   is intended and supplied to you, the Company's customer,      *
;   for use solely and exclusively on Microchip dsPIC             *
;   products. The software is owned by the Company and/or its     *
;   supplier, and is protected under applicable copyright laws. All  *
;   rights are reserved. Any use in violation of the foregoing    *
;   restrictions may subject the user to criminal sanctions under *
;   applicable laws, as well as to civil liability for the breach of *
;   the terms and conditions of this license.                     *
;                                                                 *
;   THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION.  NO        *
;   WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, *
;   BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND *
;   FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE  *
;   COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,  *
;   INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.  *
;                                                                 *
;   (c) Copyright 2003 Microchip Technology, All rights reserved. *
;*****************************************************************

        ; Local inclusions.
        .nolist
        .include       "dspcommon.inc"              ; fractsetup,FIRStruct,
                                                    ; MODCON, XMODSRT, XMODEND,
                                                    ; YMODSRT, YMODEND, CORCON,
                                                    ; COEFFS_IN_DATA
        .list

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .section .libdsp, code

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; _FIRLMS: FIR filtering with LMS coefficient adaptation.
;
; Operation:
;       y[n] = sum(m=0:M-1){h[m]*x[n-m]},
;
;       h(n+1)[m] = h(n)[m] + mu*(r[n]-y[n])*x[n-m], 0<= m < M.
;
; x[n] defined for 0 <= n < N,
; r[n] defined for 0 <= n < N,
; y[n] defined for 0 <= n < N,
; h[m] defined for 0 <= m < M as an increasing circular buffer,
; mu in [-1, 1).
; NOTE: delay defined for 0 <= m < M as an increasing circular buffer.
; NOTE: filter coefficients should not be allocated in program memory,
;       since in this case they cannot be adapted at run time.
;
; Input:
;       w0 = N, number of input samples (N)
;       w1 = y, ptr output samples (0 <= n < N)
;       w2 = x, ptr input samples (0 <= n < N)
;       w3 = filter structure (FIRStruct, h)
;       w4 = r, ptr reference samples (0 <= n < N)
;       w5 = mu.
; Return:
;       w0 = y, ptr output samples (0 <= n < N), or
;       NULL if it was detected that coefficients had been allocated in
;       program memory.
;
; System resources usage:
;       {w0..w7}        used, not restored
;       {w8..w12}       saved, used, restored
;        AccuA          used, not restored
;        AccuB          used, not restored
```

```
;       CORCON        saved, used, restored
;       MODCON        saved, used, restored
;       XMODSRT       saved, used, restored
;       XMODEND       saved, used, restored
;       YMODSRT       saved, used, restored
;       YMODEND       saved, used, restored
;
; DO and REPEAT instruction usage.
;      2 level DO intruction
;      1 level REPEAT intruction
;
; Program words (24-bit instructions):
;      74
;
; Cycles (including C-function call and return overheads):
;      61 + N*(14 + 5*M), or
;      32 if coefficients were allocated in program memory (which implies
;      returning a NULL pointer, as coefficients cannot be adapted at run
;      time if in program memory).
;.....................................................................

       .global _FIRLMS ; export
_FIRLMS:

;.....................................................................

       ; Save working registers.
       push.d  w8                          ; {w8,w9} to TOS
       push.d  w10                         ; {w10,w11} to TOS
       push    w12                         ; w12 to TOS

;.....................................................................

       ; Prepare CORCON for fractional computation.
       push    CORCON
       fractsetup     w7

;.....................................................................

       ; Make sure that coefficients are not located in program memory.
       ; (If they were, they could not be adapted at run time...)
       mov     [w3+oCoeffsPage],w10        ; w10= coefficients page
       mov     #COEFFS_IN_DATA,w8          ; w8 = COEFFS_IN_DATA
       cp      w8,w10                      ; w8 - w10
       bra     z,_noPSV                    ; if w10 = COEFFS_IN_DATA
                                           ; no PSV management
                                           ; else
       mov     #0,w0                       ; w0 = 0 (NULL)
       bra     _restore                    ; restore saved registers
                                           ; and return NULL
_noPSV:

;.....................................................................

       ; Prepare core registers for modulo addressing.
       push    MODCON
       push    XMODSRT
       push    XMODEND
       push    YMODSRT
       push    YMODEND

;.....................................................................

       ; Setup registers for modulo addressing.
       mov     #0xC0A8,w10                 ; XWM = w8, YWM = w10
                                           ; set XMODEND and YMODEND bits
       mov     w10,MODCON                  ; enable X,Y modulo addressing

       mov     [w3+oCoeffsEnd],w8          ; w8 -> last byte of h[M-1]
       mov     w8,XMODEND                  ; init'ed to coeffs end address
       mov     [w3+oCoeffsBase],w8         ; w8 -> h[0]
       mov     w8,XMODSRT                  ; init'ed to coeffs base address
                                           ; (increasing buffer,
                                           ;  2^n aligned)
       mov     [w3+oDelayEnd],w10          ; w10-> last byte of d[M-1]
```

```
        mov     w10,YMODEND                     ; init'ed to delay end address
        mov     [w3+oDelayBase],w10             ; w10 -> d[0]
        mov     w10,YMODSRT                     ; init'ed to delay base address
                                                ; (increasing buffer,
                                                ;  2^n aligned)

;.....................................................................

        push    w1                              ; save return value (y)

;.....................................................................

        ; Perpare to filter all samples.
        mov     [w3+oDelay],w10                       ; w10 points at current delay
                                                ; sample d[m], 0 <= m < M
                                                ; referred to as delay[0]
                                                ; for each iteration...
        mov     w4,w12                          ; w12->r[0]
        mov     [w3+oNumCoeffs],w4              ; w4 = M
        sub     w4,#2,w4                        ; W4 = M-2
        dec     w0,w0                           ; w0 = N-1
        mov     w5,w7                           ; w7 = mu

;.....................................................................

        ; Perform filtering of all samples.
        do      w0,_endFilter           ; {     ; do (N-1)+1 times

        ; Prepare to filter sample.
        mov     [w2++],[w10]                    ; store new sample into delay

.ifdef YMEM_ERRATA
    nop
.endif

        clr     a,[w8]+=2,w5,[w10]+=2,w6        ; a  = 0
                                                ; w5 = h[0]
                                                ; w8-> h[1]
                                                ; w6 = delay[0]
                                                ; w10->delay[1]

        ; Filter each sample.
        ; (Perform all but last MACs.)
        repeat  w4                      ; {     ; do (M-2)+1 times
        mac     w5*w6,a,[w8]+=2,w5,[w10]+=2,w6        ; a += h[m]*delay[m]
                                                ; w5 = h[m+1]
                                                ; w8-> h[m+2]
                                                ; w6 = delay[m+1]
                                                ; w10->delay[m+2]
; }
        ; (Perform last MAC.)
        mac     w5*w6,a                         ; a += h[M-1]*delay[M-1]
                                                ; now:
                                                ; w8-> h[0]
                                                ; w10->delay[0]

        ; Save filtered result.
        sac.r   a,[w1++]                        ; y[n] =
                                                ;   sum_{m=0:M-1}(h[m]*x[n-m])
                                                ; w1-> y[n+1]

        ; With the new output, and the corresponding reference sample,
        ; update the filter coefficients.
        lac     [w12++],b                       ; b  = r[n]
                                                ; w12-> r[n+1]
        sub     b                               ; b  = r[n] - y[n]
        sac.r   b,w5                            ; w5:  current error
        mpy     w5*w7,a                         ; a  = mu*(r[n]-y[n])
        sac.r   a,w5                            ; w5:  attenuated error

        ; Adaptation: h[m] = h[m] + attError*x[n-m].
        ; Here the h[m] cannot be addressed as a circular buffer,
        ; because their values are accessed via a 'LAC' instruction...
        ; Thus, use w9 instead.
```

```
        ; Prepare adaptation.
        dec     w4,w11                          ; w11= M-3
        mov     w8,w9                           ; w9-> h[0]
        clr     a,[w10]+=2,w6                   ; w6 = delay[0]
                                                ; w10->delay[1]

        ; Perform adaptation (all but last two coefficients).
        do      w11,_endAdapt           ; {     ; do (M-3)+1 times
        lac     [w9],a                          ; a  = h[m]
        mac     w5*w6,a,[w10]+=2,w6             ; a += attError*delay[m]
                                                ; w6 = delay[m+1]
                                                ; w9-> delay[m+2]
_endAdapt:
        sac.r   a,[w9++]                        ; store adapted h[m]
                                                ; w9-> h[m+1]
; }
        ; Perform adaptation for second to last coefficient.
        lac     [w9],a                          ; a  = h[M-2]
        mac     w5*w6,a,[w10],w6                ; a += attError*h[M-2]
                                                ; w6 = delay[M-1]
                                                ; w10->delay[M-1]
        sac.r   a,[w9++]                        ; store adapted h[M-2]
        ; Perform adaptation for last coefficient.
        lac     [w9],a                          ; a  = h[M-1]
        mac     w5*w6,a                         ; a += attError*h[M-1]
_endFilter:
        sac.r   a,[w9++]                        ; store adapted h[M-1]
; }

;.....................................................................

        ; Update delay pointer.
        mov     w10,[w3+oDelay]                         ; note that the delay pointer
                                                ; may wrap several times around
                                                ; d[m], 0 <= m < M, depending
                                                ; on the value of N
                                                ; (it is the same as delay[0])

;.....................................................................

        pop     w0                              ; restore return value

;.....................................................................

        ; Restore core registers for modulo addressing.
        pop     YMODEND
        pop     YMODSRT
        pop     XMODEND
        pop     XMODSRT
        pop     MODCON

;.....................................................................

_restore:

        ; Restore CORCON.
        pop     CORCON

;.....................................................................

        ; Restore working registers.
        pop     w12                             ; w12 from TOS
        pop.d   w10                             ; {w10,w11} from TOS
        pop.d   w8                              ; {w8,w9} from TOS

;.....................................................................

        return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; OEF
```

firlmsn.s

```
;*****************************************************************
;                                                                 *
;                       Software License Agreement                *
;                                                                 *
;   The software supplied herewith by Microchip Technology        *
;   Incorporated (the "Company") for its dsPIC controller         *
;   is intended and supplied to you, the Company's customer,      *
;   for use solely and exclusively on Microchip dsPIC             *
;   products. The software is owned by the Company and/or its     *
;   supplier, and is protected under applicable copyright laws. All  *
;   rights are reserved. Any use in violation of the foregoing    *
;   restrictions may subject the user to criminal sanctions under *
;   applicable laws, as well as to civil liability for the breach of *
;   the terms and conditions of this license.                     *
;                                                                 *
;   THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION.  NO        *
;   WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, *
;   BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND *
;   FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE  *
;   COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,  *
;   INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.  *
;                                                                 *
;   (c) Copyright 2003 Microchip Technology, All rights reserved.    *
;*****************************************************************

        ; Local inclusions.
        .nolist
        .include       "dspcommon.inc"              ; fractsetup,FIRStruct,
                                                    ; MODCON, XMODSRT, XMODEND,
                                                    ; YMODSRT, YMODEND, CORCON,
                                                    ; PSVPAG, COEFFS_IN_DATA
        .list

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .section .libdsp, code

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; _FIRLMSNorm: FIR filtering with Normalized LMS coefficient adaptation.
;
; Operation:
;       y[n] = sum_{m=0:M-1}(h[m]*x[n-m]), 0 <= n < N.
;
;       h(n+1)[m] = h(n)[m] + nu*(r[n]-y[n])*x[n-m], 0 <= m < M.
; with
;       nu[n] = mu/(mu*E[n]),
; where
;       E[n] = E[n-1] + (x[n])^2 - (x[n-M+1])^2
; an estimate of the input signal energy at each sample.
;
; x[n] defined for 0 <= n < N,
; r[n] defined for 0 <= n < N,
; y[n] defined for 0 <= n < N,
; h[m] defined for 0 <= m < M as an increasing circular buffer,
; mu in [-1, 1).
; NOTE: delay defined for 0 <= m < M as an increasing circular buffer.
; NOTE: filter coefficients should not be allocated in program memory,
;       since in this case they cannot be adapted at run time.
;
; NOTE that the energy estimate may be also expressed as:
;
;       E[n] = x[n]^2 + x[n-1]^2 + ... + x[n-M+2]^2,
;
; then, to avoid saturation while computing the estimated energy,
; the input signal values should be bound so that
;
;       sum_{m=0:-M+2}((x[n+m])^2) < 1, 0 <= n < N.
;
; Input:
;       w0 = N, number of input samples (N)
;       w1 = y, ptr output samples (0 <= n < N)
```

```
;       w2 = x, ptr input samples (0 <= n < N)
;       w3 = filter structure (FIRStruct, h)
;       w4 = r, ptr reference samples (0 <= n < N)
;       w5 = mu.
;       w6-> E[-1] on start up, and E[N-1] upon return.
; Return:
;       w0 = y, ptr output samples (0 <= n < N)
;       NULL if it was detected that coefficients had been allocated in
;       program memory.
;
; System resources usage:
;       {w0..w7}        used, not restored
;       {w8..w13}       saved, used, restored
;        AccuA          used, not restored
;        AccuB          used, not restored
;        CORCON         saved, used, restored
;        MODCON         saved, used, restored
;        XMODSRT        saved, used, restored
;        XMODEND        saved, used, restored
;        YMODSRT        saved, used, restored
;        YMODEND        saved, used, restored
;
; DO and REPEAT instruction usage.
;       2 level DO intruction
;       1 level REPEAT intruction
;
; Program words (24-bit instructions):
;       92
;
; Cycles (including C-function call and return overheads):
;       66 + N*(50 + 5*M), or
;       36 if coefficients were allocated in program memory (which implies
;       returning a NULL pointer, as coefficients cannot be adapted at run
;.....................................................................

        .global _FIRLMSNorm    ; export
_FIRLMSNorm:

;.....................................................................

        ; Save working registers.
        push.d  w8                              ; {w8,w9} to TOS
        push.d  w10                             ; {w10,w11} to TOS
        push.d  w12                             ; {w12,w13} to TOS

;.....................................................................

        ; Prepare CORCON for fractional computation.
        push    CORCON
        fractsetup     w7

;.....................................................................

        ; Make sure that coefficients are not located in program memory.
        ; (If they were, they could not be adapted at run time...)
        mov     [w3+oCoeffsPage],w10            ; w10= coefficients page
        mov     #COEFFS_IN_DATA,w8              ; w8 = COEFFS_IN_DATA
        cp      w8,w10                          ; w8 - w10
        bra     z,_noPSV                        ; if w10 = COEFFS_IN_DATA
                                                ; no PSV management
                                                ; else
        mov     #0,w0                           ; w0 = 0 (NULL)
        bra     _restore                        ; restore saved registers
                                                ; and return NULL
_noPSV:

;.....................................................................

        ; Prepare core registers for modulo addressing.
        push    MODCON
        push    XMODSRT
        push    XMODEND
        push    YMODSRT
        push    YMODEND
```

```
;.....................................................................

        ; Setup registers for modulo addressing.
        mov     #0xC0A8,w10                 ; XWM = w8, YWM = w10
                                            ; set XMODEND and YMODEND bits
        mov     w10,MODCON                  ; enable X,Y modulo addressing

        mov     [w3+oCoeffsEnd],w8          ; w8 -> last byte of h[M-1]
        mov     w8,XMODEND                  ; init'ed to coeffs end address
        mov     [w3+oCoeffsBase],w8         ; w8 -> h[0]
        mov     w8,XMODSRT                  ; init'ed to coeffs base address
                                            ; (increasing buffer,
                                            ;  2^n aligned)
        mov     [w3+oDelayEnd],w10          ; w10-> last byte of d[M-1]
        mov     w10,YMODEND                 ; init'ed to delay end address
        mov     [w3+oDelayBase],w10         ; w10 -> d[0]
        mov     w10,YMODSRT                 ; init'ed to delay base address
                                            ; (increasing buffer,
                                            ;  2^n aligned)

;.....................................................................

        push    w1                          ; save return value (y)

;.....................................................................

        ; Perpare to filter all samples.
        mov     [w3+oDelay],w10                     ; w10 points at current delay
                                            ; sample d[m], 0 <= m < M
                                            ; referred to as delay[0]
                                            ; for each iteration...
        mov     w4,w7                       ; w7-> r[0] (temporarily)
        mov     [w3+oNumCoeffs],w4          ; w4 = M
        sub     w4,#2,w4                    ; W4 = M-2
        dec     w0,w0                       ; w0 = N-1
        mov     w7,w13                      ; w13->r[0] (permanently)
        mov     w5,w7                       ; w7 = mu (used as numerator)
        mov     w6,w11                      ; w11->E[-1]

;.....................................................................

        ; Perform filtering of all samples.
        do      w0,_endFilter         ; {     ; do (N-1)+1 times

        ; Prepare to normalize.
        mov     [w2],w5                     ; w5 = x[n]
        lac     [w11],a                     ; a = E[n-1]
        mac     w5*w5,a                     ; a += (x[n])^2
        sac.r   a,[w11]                     ;*w11= E[n-1] + (x[n])^2

        ; Prepare to filter sample.
        mov     [w2++],[w10]                ; store new sample into delay

.ifdef YMEM_ERRATA
    nop
.endif

        clr     a,[w8]+=2,w5,[w10]+=2,w6    ; a  = 0
                                            ; w5 = h[0]
                                            ; w8-> h[1]
                                            ; w6 = delay[0]
                                            ; w10->delay[1]

        ; Filter each sample.
        ; (Perform all but last MACs.)
        repeat  w4                  ; {     ; do (M-2)+1 times
        mac     w5*w6,a,[w8]+=2,w5,[w10]+=2,w6      ; a += h[m]*delay[m]
                                            ; w5 = h[m+1]
                                            ; w8-> h[m+2]
                                            ; w6 = delay[m+1]
                                            ; w10->delay[m+2]
; }
        ; (Perform last MAC.)
        mac     w5*w6,a                     ; a += h[M-1]*delay[M-1]
                                            ; now:
```

```
                                        ; w6 = delay[M-1] = x[n-M+1]
                                        ; w8-> h[0]
                                        ; w10->delay[0]

        ; Save filtered result.
        sac.r   a,[w1]                  ; y[n] =
                                        ;   sum_{m=0:M-1}(h[m]*x[n-m])

        ; Continue normalizing.
        lac     [w11],a                 ; a  = E[n-1] + (x[n])^2
        mpy     w6*w6,b                 ; b  = (x[n-M+1])^2
        sub     a                       ; a -= (x[n-M+1])^2
        sac.r   a,[w11]                 ;*w11= E[n] (estimate)
        add     w7,a                    ; a += mu
        sac.r   a,w6                    ; w6 = mu + E[n] (denominator)
        ; Divide.
        push.d  w0                      ; {w0,w1} to TOS
        repeat  #17
        divf    w7,w6                   ; w0 = mu/(mu+E[n])
        mov     w0,w6                   ; w6 = nu[n]
        pop.d   w0                      ; {w0,w1} from TOS

        ; With the new output, and the corresponding reference sample,
        ; compute normalize factor.
        lac     [w1++],a                ; a  = y[n]
                                        ; w1-> y[n+1]
        lac     [w13++],b               ; b  = r[n]
                                        ; w13->r[n+1]
        sub     b                       ; b  = r[n] - y[n]
        sac.r   b,w5                    ; w5:  current error
        mpy     w5*w6,a                 ; a  = nu[n]*(r[n]-y[n])
        sac.r   a,w5                    ; w5:  attenuated error

        ; Adaptation: h[m] = h[m] + attError*x[n-m].
        ; Here the h[m] cannot be addressed as a circular buffer,
        ; because their values are accessed via a 'LAC' instruction...
        ; Thus, use w9 instead.

        ; Prepare adaptation.
        mov     w8,w9                   ; w9-> h[0]
        clr     a,[w10]+=2,w6           ; w6 = delay[0]
                                        ; w10->delay[1]


        ; Perform adaptation (all but last two coefficients).
        dec     w4,w12                  ; w12= M-3
        do      w12,_endAdapt         ; {     ; do (M-3)+1 times
        lac     [w9],a                  ; a  = h[m]
        mac     w5*w6,a,[w10]+=2,w6     ; a += attError*delay[m]
                                        ; w6 = delay[m+1]
                                        ; w9-> delay[m+2]
_endAdapt:
        sac.r   a,[w9++]                ; store adapted h[m]
                                        ; w9-> h[m+1]
; }
        ; Perform adaptation for second to last coefficient.
        lac     [w9],a                  ; a  = h[M-2]
        mac     w5*w6,a,[w10],w6        ; a += attError*h[M-2]
                                        ; w6 = delay[M-1]
                                        ; w10->delay[M-1]
        sac.r   a,[w9++]                ; store adapted h[M-2]
        ; Perform adaptation for last coefficient.
        lac     [w9],a                  ; a  = h[M-1]
        mac     w5*w6,a                 ; a += attError*h[M-1]
_endFilter:
        sac.r   a,[w9]                  ; store adapted h[M-1]
; }

;.....................................................................

        ; Update delay pointer.
        mov     w10,[w3+oDelay]                 ; note that the delay pointer
                                        ; may wrap several times around
                                        ; d[m], 0 <= m < M, depending
                                        ; on the value of N
                                        ; (it is the same as delay[0])
```

```
;.....................................................................

        pop     w0                              ; restore return value

;.....................................................................

        ; Restore core registers for modulo addressing.
        pop     YMODEND
        pop     YMODSRT
        pop     XMODEND
        pop     XMODSRT
        pop     MODCON

;.....................................................................

_restore:

        ; Restore CORCON.
        pop     CORCON

;.....................................................................

        ; Restore working registers.
        pop.d   w12                             ; {w12,w13} from TOS
        pop.d   w10                             ; {w10,w11} from TOS
        pop.d   w8                              ; {w8,w9} from TOS

;.....................................................................

        return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; OEF
```

dsp.h

```
#include        <stdlib.h>
#include        <math.h>
#define Q15(X) \
   ((X < 0.0) ? (int)(32768*(X) - 0.5) : (int)(32767*(X) + 0.5))
typedef struct {
   int numCoeffs;
   fractional* coeffsBase;
   fractional* coeffsEnd;
   int coeffsPage;
   fractional* delayBase;
   fractional* delayEnd;
   fractional* delay;
} FIRStruct;
extern void FIRStructInit (
   FIRStruct* FIRFilter,
   int numCoeffs,
   fractional* coeffsBase,
   int coeffsPage,
   fractional* delayBase
);
extern void FIRInterpDelayInit (
   FIRStruct* filter,
   int rate
);
extern fractional* FIR (
```

```
   int numSamps,
   fractional* dstSamps,
   fractional* srcSamps,
   FIRStruct* filter
);
extern fractional* FIRLMS (
   int numSamps,
   fractional* dstSamps,
   fractional* srcSamps,
   FIRStruct* filter,
   fractional* refSamps,
   fractional muVal
);
extern fractional* FIRLMSNorm (
   int numSamps,
   fractional* dstSamps,
   fractional* srcSamps,
   FIRStruct* filter,
   fractional* refSamps,
   fractional muVal,
   fractional* energyEstimate
);
extern fractional* VectorCopy (
   int numElems,
   fractional* dstV,
   fractional* srcV
);
extern fractional* VectorSubtract (
   int numElems,
   fractional* dstV,
   fractional* srcV1,
   fractional* srcV2
);
```

adcdacDrv.h

```
#define     Fosc              80000000
#define     Fcy               ( Fosc / 2 )
#define     NUMSAMP           2
#define     LMS_SIZE          32
#define     PROC_BLOCK_SIZE   32
#define     SCALE             64
void initAdc( void );
void initTmr3( void );
void initOC1( void );
void initOC2( void );
void initOC3( void );
void initTmr2( void );
void initDma0( void );
void InitProcDSP( void );
void   __attribute__( ( interrupt, no_auto_psv ) ) _DMA0Interrupt(
void );
void  __attribute__( ( interrupt, no_auto_psv ) ) _T3Interrupt( void
);
```

switch.h

```
#define     DEBOUNCE_INTERVAL     25
#define     SW1_TRIS              TRISAbits.TRISA12
#define     SW2_TRIS              TRISAbits.TRISA13
#define     SW3_TRIS              TRISAbits.TRISA2
#define     SW4_TRIS              TRISAbits.TRISA3
#define     LED1_TRIS             TRISAbits.TRISA6
#define     LED2_TRIS             TRISAbits.TRISA7
#define     LED3_TRIS             TRISAbits.TRISA9
#define     LED4_TRIS             TRISAbits.TRISA10
#define     SW1                   PORTAbits.RA12
#define     SW2                   PORTAbits.RA13
#define     SW3                   PORTAbits.RA2
#define     SW4                   PORTAbits.RA3
#define     LED1                  LATAbits.LATA6
#define     LED2                  LATAbits.LATA7
#define     LED3                  LATAbits.LATA9
#define     LED4                  LATAbits.LATA10
void initSwitch( int  *debounce );
int  CheckSwitchS1( void );
int  CheckSwitchS2( void );
int  CheckSwitchS3( void );
int  CheckSwitchS4( void );
```

ประวัติผู้เขียนวิทยานิพนธ์

นายนพดล จตุไพบูลย์ เกิดเมื่อวันที่ 21 ตุลาคม พ.ศ. 2528 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2550 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2551