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 นพทีปกังวาล. 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

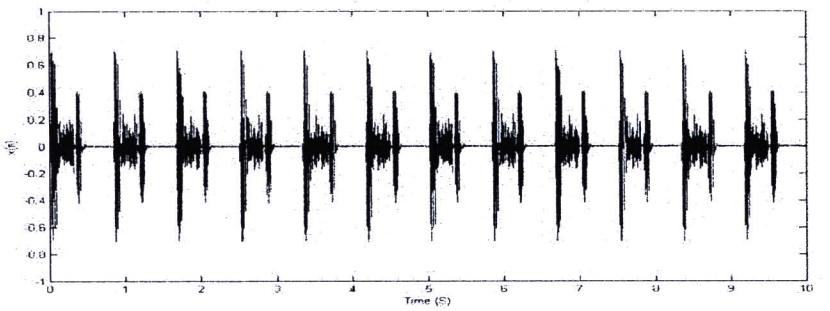
สัญญาณเสียงที่ใช้อยู่ในรูปแบบไฟล์นามสกุล Wave ความยาว 10 วินาที มีอัตราสุ่ม 8000 Hz ขนาด 16 Bits ข้อมูลเสียงที่ใช้มี 2 ชนิด คือ

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

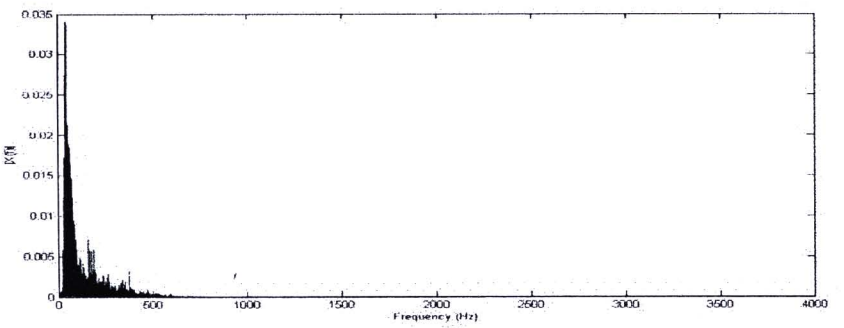
โดยสัญญาณเสียงที่ใช้ในงานวิจัยนี้ ประกอบไปด้วย สัญญาณเสียงหัวใจ และสัญญาณเสียงรบกวน ดังนี้

1. สัญญาณเสียงหัวใจ [20]

เลือกใช้สัญญาณเสียงหัวใจของคนที่เป็นโรค Pan Systolic Murmur ซึ่งมีเสียง Murmur ระหว่างเสียง S1 และ S2



สัญญาณเสียง Pan Systolic Murmur ในโดเมนเวลา

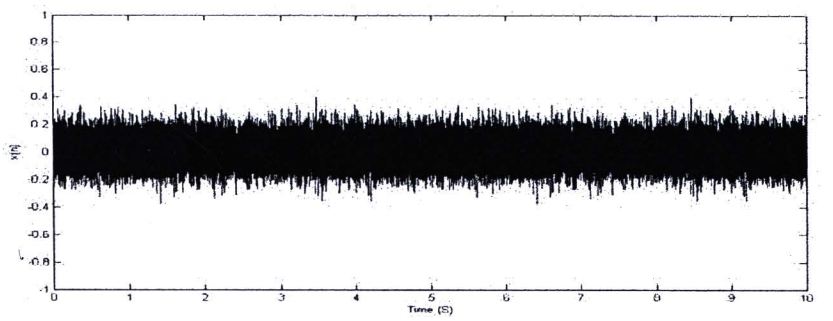


สัญญาณเสียง Pan Systolic Murmur ในโดเมนความถี่

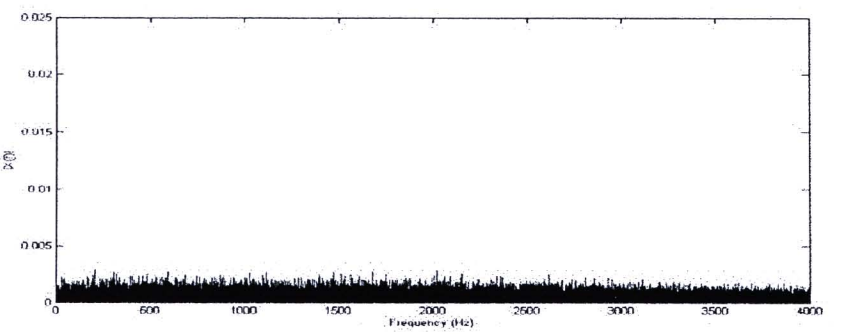
2. สัญญาณเสียงรบกวนภายนอก [21]

เลือกใช้สัญญาณเสียงรบกวนภายนอกที่มีความดังสม่ำเสมอ (Steady State Noise) ที่มีลักษณะแตกต่างกัน 4 ชนิด ดังนี้

2.1. White Noise เป็นสัญญาณเสียงรบกวนที่มีกำลังของสัญญาณในช่วงความถี่ต่างๆ เท่าๆ กัน

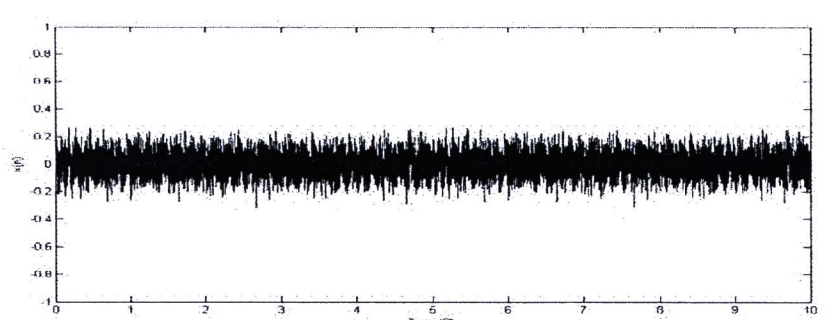


สัญญาณเสียง White Noise ในโดเมนเวลา

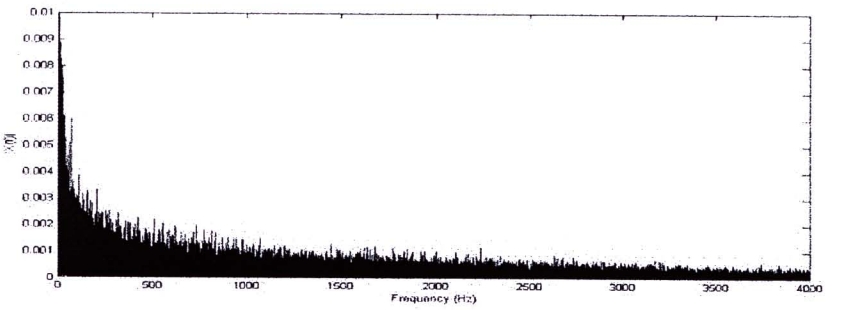


สัญญาณเสียง White Noise ในโดเมนความถี่

2.2. Pink Noise เป็นสัญญาณเสียงรบกวนที่กำลังของสัญญาณจะแปรผกผันกับความถี่ คือ ในช่วงความถี่ที่สูงขึ้นกำลังของสัญญาณจะลดลง

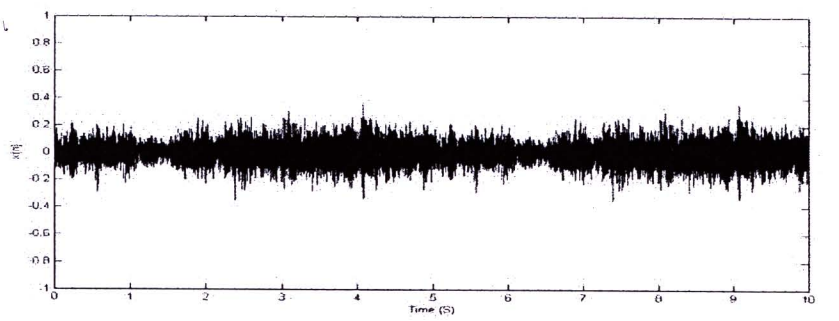


สัญญาณเสียง Pink Noise ในโดเมนเวลา

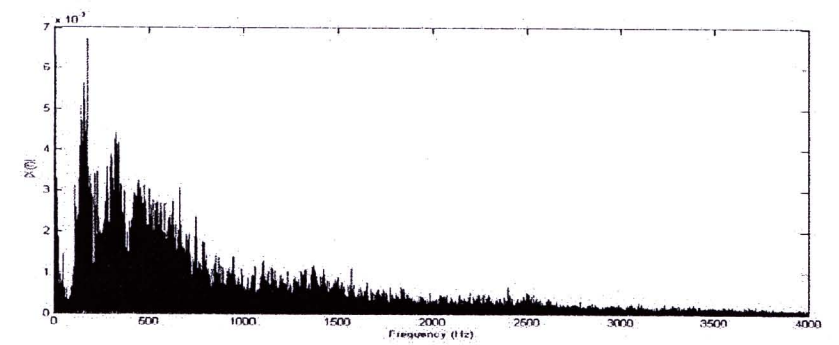


สัญญาณเสียง Pink Noise ในโดเมนความถี่

2.3. Babble Noise เป็นสัญญาณเสียงรบกวนที่บันทึกจากสภาพแวดล้อมที่มีคน จำนวนมากพูดคุยกัน

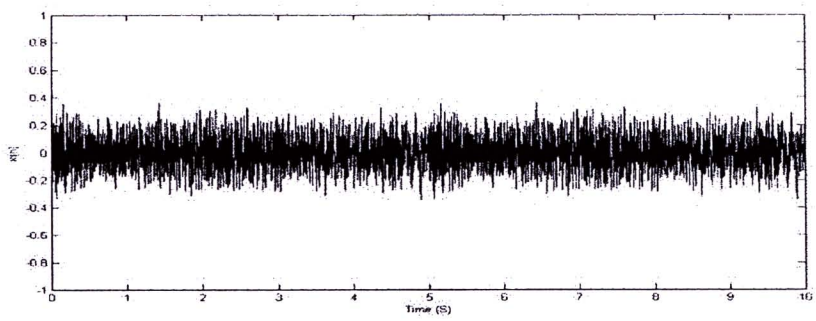


สัญญาณเสียง Babble Noise ในโดเมนเวลา

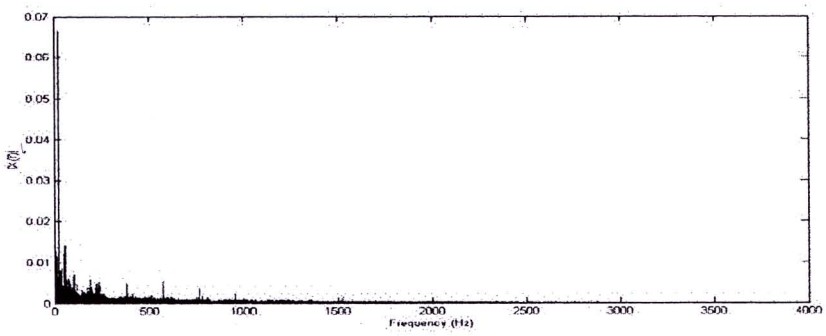


สัญญาณเสียง Babble Noise ในโดเมนความถี่

2.4. Factory Noise เป็นสัญญาณเสียงรบกวนที่บันทึกจากสภาพแวดล้อมที่มีเสียง การทำงานของเครื่องจักรในโรงงาน



สัญญาณเสียง Factory Noise ในโดเมนเวลา



สัญญาณเสียง Factory Noise ในโดเมนความถี่

ภาคผนวก ข.
โค้ดที่ใช้ในการคำนวณค่าต่าง ๆ 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

สมการที่ 25 สามารถเขียนเป็นโค้ดโปรแกรมได้ดังนี้

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

สมการที่ 26 สามารถเขียนเป็นโค้ดโปรแกรมได้ดังนี้

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

สมการที่ 27 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 - โปรแกรมส่วนที่ใช้ในการแปลงสัญญาณแอนะลอกเป็นดิจิตอล และแปลงสัญญาณดิจิตอลกลับมาเป็นสัญญาณแอนะลอก
   - 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