A DSP-Based Audio Signal Processor

This project uses a low-cost DSP board and serves both as an introduction to DSP techniques and as a useful station accessory.

By Johan Forrer, KC7WW, 26553 Priceview Drive, Monroe, OR 97456

This article presents the theory of operation and implementation details of a digital signal processor-based (DSP) audio signal processor (ASP). Such an ASP may be used with a communication receiver or incorporated as an integral part of a home-brew receiver project.

The ASP consists of several components: a digital beat-frequency oscillator (BFO), selectable band-pass filters for CW, SSB and other digital modes, a denoiser based on the least-mean-squares (LMS) technique and a Wiener-filter autonotcher (removing carriers or heterodynes). Several advanced concepts are applied in this project, such as multirate processing, adaptive filtering and frequency shifting. These all are of fundamental importance to anyone wishing to learn more about the finer points of DSP. These principles may be considered the “tools of the trade” for working with DSP. This not only applies to audio signal processing, but also is becoming evident in contemporary digital radios.

The DSP platform used for this project is a low-cost evaluation module by Analog Devices called the *EZ-KIT Lite*. [1] However, any DSP platform with modest memory and processor speed may be used. The *EZ-KIT Lite* was considered ideal for this project because of its 16-bit audio interface, 33 million instruction per second (MIPS) ADSP-2181 DSP, 32k words of on-chip memory, included software development tools and low cost.

This article describes a number of components that make up an ASP, their functionality and how they are engineered and implemented. The objective is to expose the reader to the background that is essential for future involvement in DSP as there is no substitute or reward greater than trying it yourself.

Background

The W9GR DSP project that appeared in *QST* nearly four years ago encouraged many to build kits or to try their hand at DSP development. [2] Since then, several offerings of low-cost DSP evaluation modules (EVM) have brought powerful, yet affordable, DSP to the amateur experimenter. The *EZ-KIT Lite* used in this project is an example of contemporary EVMs that offer substantial amounts of on-chip memory combined with high clock rates—these modules are capable of doing serious DSP work.

A major hurdle for newcomers to DSP is the steep learning curve associated with DSP theory. In addition, implementation details for a typical DSP platform often seem a formidable prospect. The situation has improved a lot over the four years since the W9GR article was first written. Those interested in DSP now have access to reasonably good DSP filter design tools, simulation packages, and a wealth of literature and software examples, much of which is available on the Internet.

ASP Architecture

Module Descriptions

Fig 1 shows a top-down overview of the different modules of the ASP and their interconnectivity. It shows that there are several ways to interconnect modules depending on the type of processing required.

Input always passes through the digital BFO and may subsequently be routed “straight through” to the output combiner when no signal processing is desired. Otherwise, a specific filter may be placed in line, either with or without further processing. If filtering only is desired, the filter output is routed directly to the output combiner; the denoiser or autonotcher modules—or both—may be selected and placed in the signal path for further signal processing.

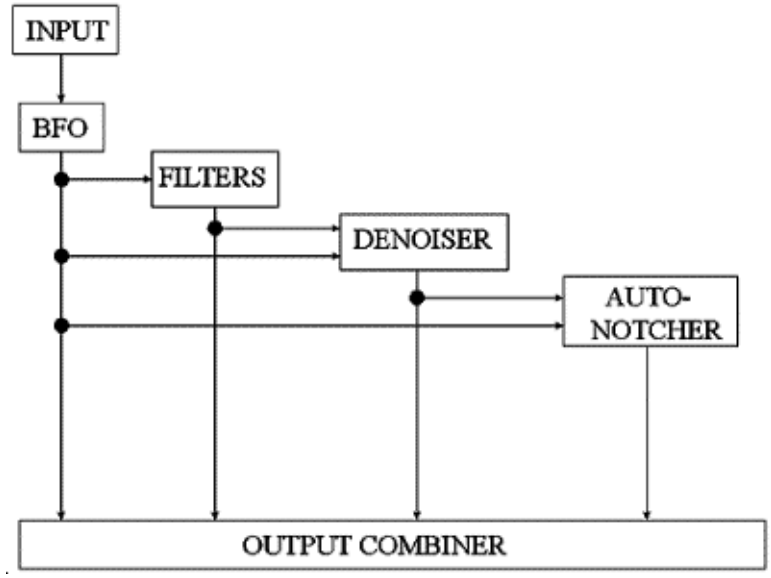

Fig 1—Outline of the audio signal processor (ASP) architecture. Each module has one output node and at least one input node. Module interconnections are user selectable and configured by software.

Digital BFO

Fine frequency resolution is a desirable feature for any receiving system. For example, changing received CW pitch to fall within a narrow-filter passband or adjusting an SSB signal for better clarity may be desired. It may also be an advantage to provide an alternative means to provide finer frequency resolution when working with a receiver that does not provide small frequency stepping. Such fine frequency resolution is essential for tuning RTTY, WEFAX or SSTV signals.

The digital BFO uses a Hilbert transformer to implement frequency shifting of signals received in the audio passband. The technique is an old one and is also used for other purposes such as the phasing method of SSB modulation/demodulation. Before the advent of DSP, such frequency shifting hardware was very difficult to build and tune. DSP achieves this nearly impossible feat with the Hilbert transform, which has the ability to do a perfect shift of a band of frequencies such that a constant group delay is exhibited thoughout the whole band.

Refer to references 10 to 12 for an in-depth treatment of the Hilbert transform, which is an essential part of the BFO. The operation of the BFO may further be formalized as follows.

Let the audio signal be represented by the time-domain signal *x(t)*. This is a real signal, ie, it has identical positive and negative frequency components. For the frequency-domain representation, let *Fp(X)* represent the positive frequencies and *Fn(X)* represent the negative frequencies. Thus:

$$F(x) = F_n(x) + F_p(x) \qquad \text{Eq 1}$$

The signal is subsequently band-limited by the codec's antialias filter and sampled at 18.9 ksps by the codec. The digitized signal is then passed through the Hilbert transformer as the first step in producing a special signal, called an analytic signal. The Hilbert transformer has the unique property that it delays all positive frequencies by +90° and all negative frequencies by –90°. Eq 2 shows this phase shifting (note the use *–j* and *+j* to indicate phase shifting) in the frequency domain.

$$\hat{F}(x) = -jF_p(x) + jF_n(x) \qquad \text{Eq 2}$$

To form the time-domain representation, *z(t)*, a complex signal, we combine *x(t)*, the real part, with the output of the Hilbert transformer, $\hat{x}(t) = H(x)$, the complex part, as shown in Eq 3. This is known as an analytic signal.

$$z(t) = x(t) + j\hat{x}(t)$$ Eq 3

Eq 3 has interesting properties—it contains only positive frequencies. This is shown in Eqs 4, 5 and 6.

$$Z(t) = F(t) + j\hat{F}(t)$$ Eq 4

$$Z(x) = F_p(x) + F_n(x) + j\left[-jF_p(x) + jF_n(x)\right]$$ Eq 5

$$Z(x) = 2F_p(x)$$ Eq 6

This property of Eq 6, that it only contains positive frequencies, is of great significance when working with phasing-type modulation schemes in communication systems. For example, selective cancellation of certain frequencies is possible without the need for additional filtering. Use of this property is made in the following section.

The next step is to do the actual frequency-shifting operation. What we are after is a real signal, *y(t)*, that has both positive and negative frequencies, with the positive frequencies shifted up in frequency—and the negative frequencies shifted down in frequency—by an amount determined by the BFO. This frequency shift is obtained by mixing (multiplying) a signal generated using a numerically controlled oscillator (NCO), which is our BFO, with the analytic input signal, *z*(*t*), as shown in Eq 7.

$$y(t) = R_e\left[z(t)e^{j\omega t}\right]$$ Eq 7

The NCO frequency is represented in complex notation as *e jwt*. Note that we need to take the real part of the result, as after the frequency shifting is done we want to process only real numbers. Substituting and expanding Eq 7 results in Eq 8.

$$y(t) = R_e\left[(I + jQ)(\cos(\omega t) + j\sin(\omega t))\right]$$ Eq 8

where *I* and *Q* refer to the in-phase and quadrature component signals of the analytic input signal. Eq 8 then reduces to our wanted signal as shown in Eq 9.

$$y(t) = I\cos(\omega t) - Q\sin(\omega t)$$ Eq 9

A diagram of the processing is shown in **Fig 2**. Note that the negative sign produces the upper sideband (USB), or frequency up-shift. If a down-shift in frequency is desired, the sign should be a positive, in which case the lower sideband (LSB) results. The frequency response of the Hilbert transformer design used in the project is shown in **Fig 6**. See Notes 11 and 12 for details on determining the Hilbert transform coefficients.

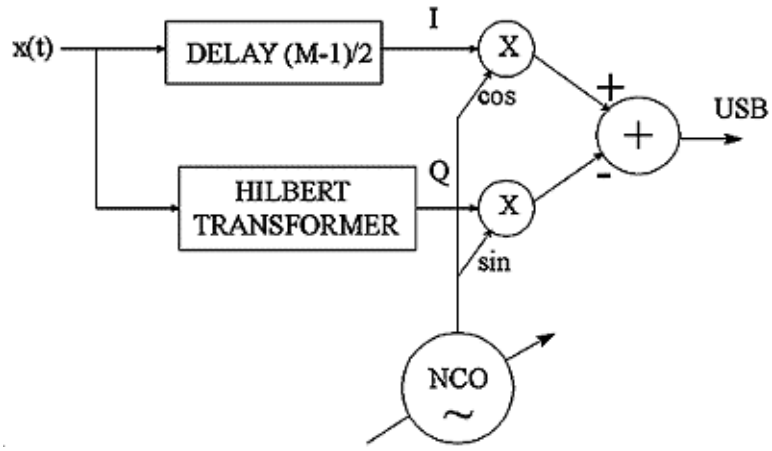

Fig 2—Hilbert transformer-based BFO. An analytic input signal is formed and mixed with a numerically controlled quadrature output oscillator (NCO). The real-valued output of the difference between the upper (in-phase or "I" branch) and lower (out-of-phase or quadrature "Q" branch) produce the upper sideband (USB). This corresponds to shifting the band of input frequencies up in frequency.

Filters

The ASP includes the passband-shaping filters listed in **Table 1**. Digital filters can be tailored to meet specific criteria, such as the nature of their transition zones (filter skirts), stop-band rejection (adjacent-channel suppression) or pass- or stop-band ripple. Several interacting factors are involved: the sampling rate, steepness of the transition zones and the desired bandwidth. The designer can experiment with filter order as a means to achieve the desired effects, however, more often than not, some compromises must be made either in the final bandwith, transition zones and/or amount of stop-band rejection.

Filter requirements also vary. CW filters, for example, need steep skirts and good adjacent-channel rejection, typically in the order of –60 dB or better. Too narrow a filter is hard to use in practice because, if the signal is just slightly off frequency, the operator may have a hard time locating and placing it in the filter's passband. Two CW filters are provided, one wide and the other very narrow. Filters for SSB need a wider bandwidth, with somewhat relaxed adjacent-channel rejection requirements than for CW or RTTY. Stop-band rejection on the order of –40 to –50 dB is usually adequate. RTTY filters, like CW, also require good adjacent-channel rejection, the optimal bandwidth being determined by the baud rate and FSK shift. It's always a good idea to allow a little extra bandwidth for operator convenience—there usually is little degradation is performance by allowing a small amount of frequency tolerance.

Having some idea of filter requirements, practical implications need to be considered next. Finite impulse response filters (FIR) are used in this application. These were designed using the Parks-McClellan algorithm (Remez exchange, equiripple) using *Matlab*. [8] An excellent public-domain program is available as an alternative (Note 9), however, the reader may use one of any number of filter-design software packages. The magnitude responses for the six filters are shown in **Figs 3** to **5** and comply with the specifications for bandwidth and stop-band rejection described in **Table 1**.

Table 1—Passband-shaping filters used in the audio signal processor. These are all finite-impulse response (FIR) filters. The associated frequency responses, passband bandwidth (BWp), stop-band bandwidth (BWs), and filter order are shown. The sampling frequency is 18.9 ksps. CW filters are centered at 800 Hz, SSB filters assume a voice-grade channel, and RTTY is for high-tone pair standard (2125/2295 Hz). The various stop-band bandwidth values were obtained by trial-and-error for the given sample rate and filter order.

Filter	*Freq.Response (Hz)*	*Bandwidth (Hz)*	*Sample Rate (kSPS)*	*Filter Order*
CW 100 Hz	750-850	BWp 100 BWs 400	2.3625 2.3625	60
CW 500 Hz	550-1050	BWp 500 BWs 700	2.3625 2.3625	60

SSB narrow	300-2700	BWp 1200 BWs 1400	9.4500 9.4500	120
SSB narrow	300-1500	BWp 2400 BWs 2600	9.4500 9.4500	120
RTTY 50 baud	2075-2345	BWp 270 BWs 465	9.4500 9.4500	120
RTTY 200 baud	2030-2390	BWp 360 BWs 560	9.4500 9.4500	120

The codec is programmed to run at 18.9 ksps. At this rate, filter order becomes an important concern as it determines whether there will be sufficient clock ticks between samples to execute all of the DSP code. For example, a CW filter with stop-band rejection of –60 dB would require an extraordinarily large filter order to match the filter shown in **Fig 3** and may be starving for clock cycles. However, the use of multirate processing reduces this computational load substantially. In the case of the CW filter, decimation by eight reduces the sample rate to 2.3625 ksps and only requires a filter order of 60 to meet the –60 dB stop-band rejection requirement.

Multirate processing requires additional overhead due to a low-pass filter at the decimating front end and a similar low-pass filter at the interpolating output stage. Note that CW operation normally does not require the noise processing functions because of the narrow bandwidth. Autonotching should also be bypassed for CW as the autonotcher would attempt to null out every CW signal in the passband!

For SSB and RTTY, a decimation factor of two is applied. These filters may optionally be used with noise reduction or autonotching. However, digital data signals may suffer waveform distortion effects and possibly other timing-related problems, so it usually is not a good idea to use such processing with data signals.

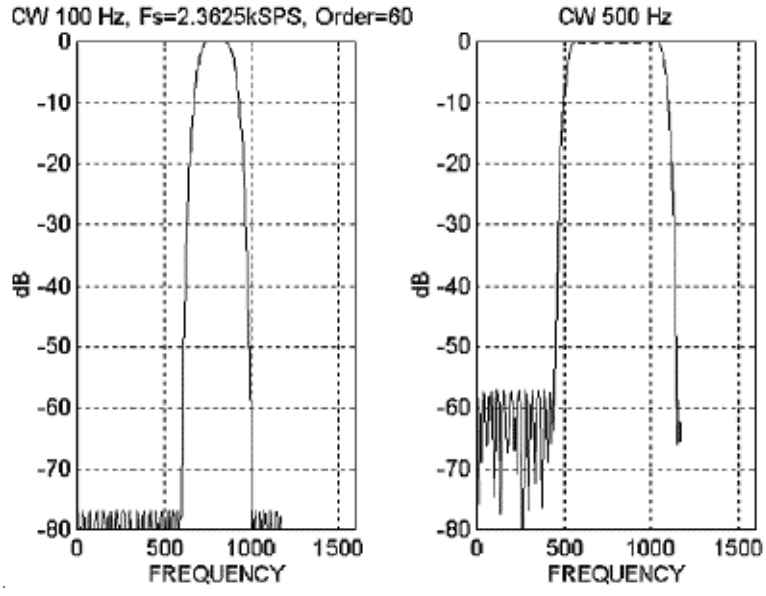

Fig 3—CW filter centered at 800 Hz. The left-hand figure has a 100-Hz bandwidth and the right-hand figure has a 500-Hz bandwidth. The sample rate is 2.3625 ksps and the filter order is 60.

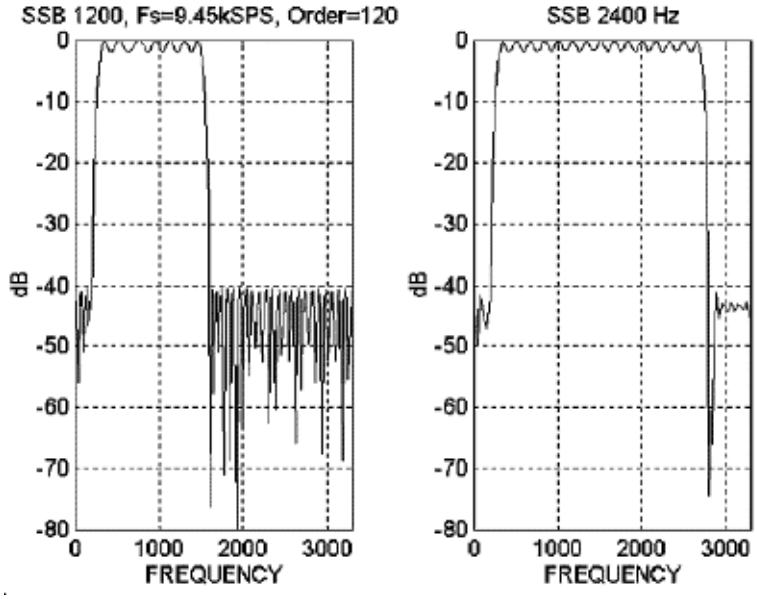

Fig 4—SSB filters. SSB filters assume a voice grade channel (300-3000 Hz). The left-hand figure is 1200 Hz wide and the right-hand figure is 2400 Hz wide. The sample rate is 9.45 ksps and the filter order is 120.

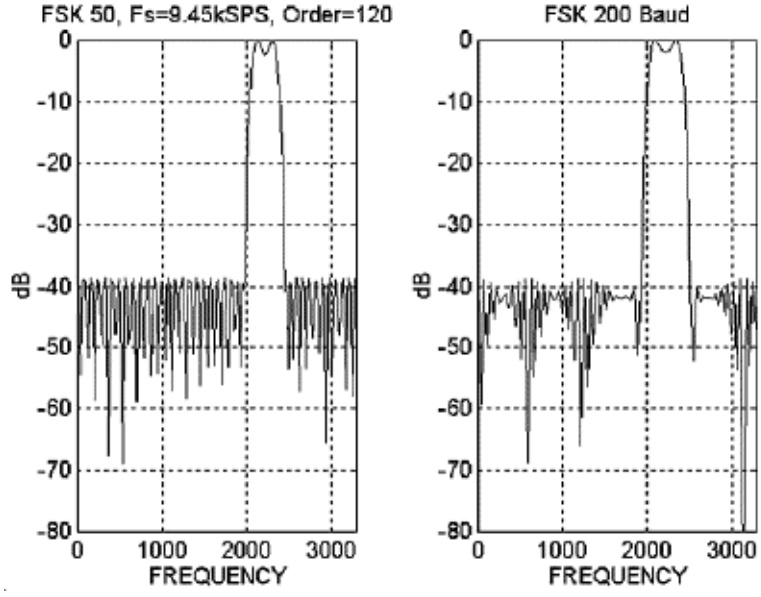

Fig 5—FSK filters. The FSK filters are the standard high-tone pair centered at 2210 Hz. The filter in the left-hand figure is designed for 50-baud operation and the filter in the right-hand figure is for 200-baud operation. The sample rate is 2.3625 ksps and the filter order is 60.

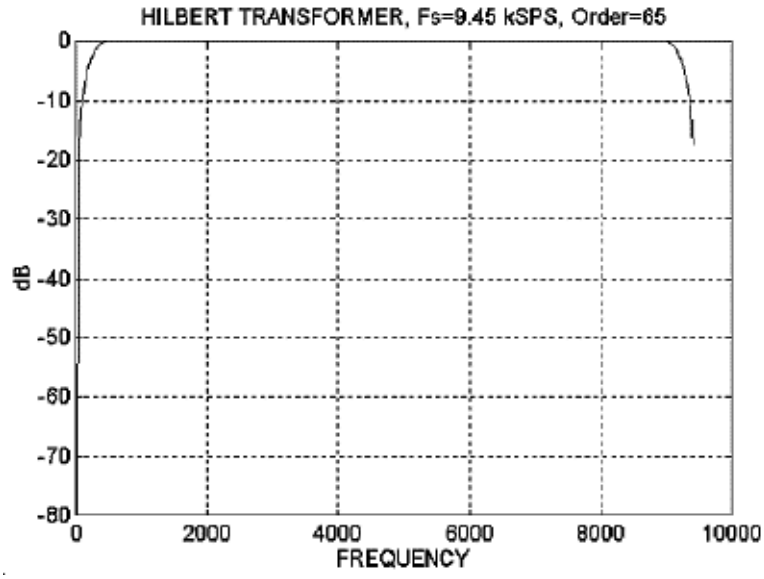

Fig 6—Magnitude response of the Hilbert transform of order 65. When used with a causal filter, it will have approximately unity gain, a group delay of (65-1)/2 samples, and approximately 90° phase shift.

Denoiser

Denoising using DSP may be achieved by several means: least mean squares (LMS), autocorrelation and spectral subtraction. [3, 5] The effectiveness of each of these depends on the nature of the noise and, to some extent, the nature of the signal. There also appear to be some psychological effects—operators often prefer one technique over another as a matter of taste. In certain instances it also appears that using a combination of these techniques may have advantages. This subject is beyond the scope of this article, and the reader is encouraged to study the references listed in Notes 2, 3, 4, 7 and 11 for the LMS technique and Notes 3, 5 and 6 regarding the spectral subtraction technique.

This project implements the LMS method as described by Hershberger. [2, 4] The architecture for the LMS algorithm is shown in **Fig 7**. A delayed version of the input signal is passed through a tunable filter and then compared to the unprocessed input signal. The difference signal is then used to tune the variable filter in order to drive the difference signal to zero. The premise is that speech signals exhibit a substantial amount of coherence. That is, the delayed signal will have high correlation with the raw input. Noise, on the other hand, tends to have a random nature and will not show the same degree of correlation.

The variable filter thus becomes a time-varying filter. An FIR structure is used as given in Eq 10.

$$y_k = \sum_{n=0}^{L} b_n(k) x_{k-n}$$

Eq 10

The algorithm effectively tunes the set of coefficients, *bn(k)*, in order to drive the difference signal, or error signal, *e*, as small as possible. The LMS algorithm uses the method of steepest descent. In this case the set of coefficients may be considered a vector, *Bk*, at instant *k*, that needs to be updated for a minimum mean squared error (MMSE). The usual procedure for minimizing a function, in this case the error squared, ε^2, is followed. An estimate of the amount that the coefficient vector *Bk* needs to change is determined from the gradient, given in Eq 11.

$$\nabla_k = \frac{\partial E\left[\varepsilon^2\right]}{\partial B_k}$$

Eq 11

Here, *E* refers to the “expected value,” or mean.

Since we are dealing with a system where there potentially may be many local minima on the error surface, and subsequently little chance for achieving an absolute MMSE, the algorithm only makes small adjustments at a time to the coefficient vector, *Bk*, in order to steer it towards MMSE. This correction is shown in Eq 12.

$$B_{k+1} = B_k - \mu\nabla_k \qquad \text{Eq 12}$$

Here, the factor, μ, determines the rate of change. The gradient given in Eq 11 is difficult to compute for a dynamic system. However, it may be estimated from the instantaneous error as shown in Eq 13.

$$\hat{\nabla} = \frac{\partial\varepsilon_k^2}{\partial B_k} = 2\varepsilon_k \frac{\partial(x_k - y_k)}{\partial B_k} \qquad \text{Eq 13}$$

Where *xk* is the reference (or input signal in our case), and *yk* represents the output of the time-varying filter. Since *xk*, the input signal, is independent of the output of the time-varying filter, it may be considered a constant and we may drop it from the partial derivative. If we then substitute Eq 10 for *yk*, Eq 13 simplifies to:

$$\hat{\nabla}_k = -2\varepsilon_k X_k \qquad \text{Eq 14}$$

where *Xk*, is the L-element input vector (input delay line buffer). The estimated coefficient vector, *Bk+1*, in Eq 12 then becomes:

$$B_{k+1} = B_k + 2\mu\varepsilon_k X_k \qquad \text{Eq 15}$$

Eq 15 is the classic LMS formula. It is evident that the dynamics of the tuned filter depend on several factors; the rate at which the coefficients can be adapted, the sampling rate and the feedback loop factor, 2μ.

Reyer and Hershberger further suggest the use of a “decay” factor to allow the filter to return to a quiet setting for situations where there is no input signal.[4] The final form for adapting the variable filter’s coefficients is shown in Eq 16.

$$B_{k+1} = (1-d)B_k + 2\mu\varepsilon_k X_k \qquad \text{Eq 16}$$

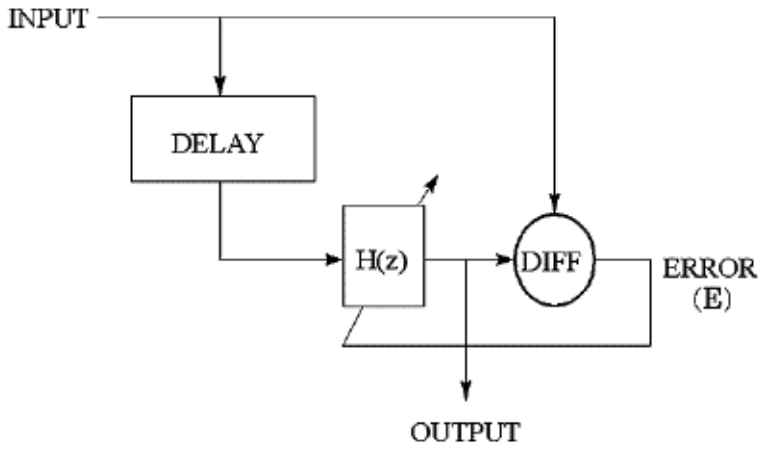

Fig 7—Generalized adaptive filter architecture for implementing the least-mean-square (LMS) denoiser.

Autonotcher

The autonotcher is based on a Wiener filter, which is basically identical to the LMS denoiser except that coherent signals are subtracted from the input. [2, 4] This is achieved by simply using the same algorithm except for where the output is taken. This would correspond to the point in **Fig 7** labeled ERROR. This makes code implementation really simple.

The working parameters such as the amount of delay in the signal path and the decay and convergence factors, and *d* and μ, respectively, are different for denoising than for auto-notching functions. These factors are maintained and adjusted separately.

A useful feature in this implementation allows for both the LMS denoiser and Wiener autonotcher to be placed in series. This is useful when monitoring SSB transmissions as it helps in removing heterodynes and reduces listening stress by removing background noise. Mileage, however, varies and much remains subject to personal preferences.

Software

Software for the ASP is available for downloading. [13] The software package consists of several modules—a control program that resides on the PC and DSP code for downloading to the *EZ-KIT*. Source code for both the PC control program, written in C, and the DSP, written in assembly language, is provided and may be used as a basis for further experimentation. I encourage you to explore and modify the code—remember that there is no greater reward than trying and succeeding in doing it yourself.

Acknowledgment

I wish to acknowledge the contributions that Adrian Nash, G4ZHZ, has made in providing ideas and suggestions for this project. The project started out on a DSP sound card and initially only provided the denoising function. Adrian's version provided the ideas for the BFO and filters. The version presented in this article combined all these functions and was ported to the *EZ-KIT*.

Notes

[1]*EZ-KIT Lite* is manufactured by Analog Devices, One Technology Way, PO Box 9106, Norwood, MA 02062-9196.

[2]Hershberger, D. "Low-Cost Digital Signal Processing for the Radio Amateur," *QST*, September 1992, pp 43-51.

[3]Hershberger, D. "**DSP—An Intuitive Approach**," *QST*, February 1996, pp 39-42.

[4]Reyer, S. E., and Hershberger, D., "Using the LMS Algorithms for QRM and QRN Reduction," *QEX*, September 1992, pp 3-8.

[5]Hall, D., "Spectral Subtraction Eliminates Noise from Speech in Real Time," *Personal Engineering*, May 1995, pp 51-54.

[6]Boll, S. F., "Suppression of Acoustic Noise in Speech Using Spectral Subtraction," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, April 1979, Vol ASSP-27(2), pp 113-120.

[7]Widrow, B., Glover, J. R., McCool, J. M., Kaunitz, J., Williams, C. S., Hearn, R. B., Zeidler, J. R., Dong, E. and Goodlin, R. C., "Adaptive Noise Canceling: Priciples and Applications," *Proceedings of the IEEE*, December 1975, Vol 63 (12), pp 1692-1716.

[8]*Matlab*, The MathWorks, Inc, Cochituate Place, 24 Prime Park Way, Natick, MA 01760. **(http://www.mathworks.com)**

[9]Egil Kvaleberg, Hysebybakken 14A, N-0379 Oslo, Norway. (**http://www.oslonet.no/home/egilk**)

[10]Frerking, M. E., *Digital Signal Processing in Communication Systems*. 1993, Van Nostrand Reinhold. New York. (ISBN: 0442016166). (See various chapters on the Hilbert transform).

[11]Stearns, S. D. and David, R. A., *Signal Processing Algorithms in* Matlab. 1996, Prentice-Hall, Inc, NJ (ISBN: 0130451541), pp 339-343.

[12]Oppenheim, A. V. and Shafer, R. W., *Discrete-Time Signal Processing*, Chapter 10. 1989, Prentice-Hall, Inc, NJ (ISBN: 013216292).

[13]The software, in file qexasp.zip, is available from **http://www.arrl.org/qexfiles/**.