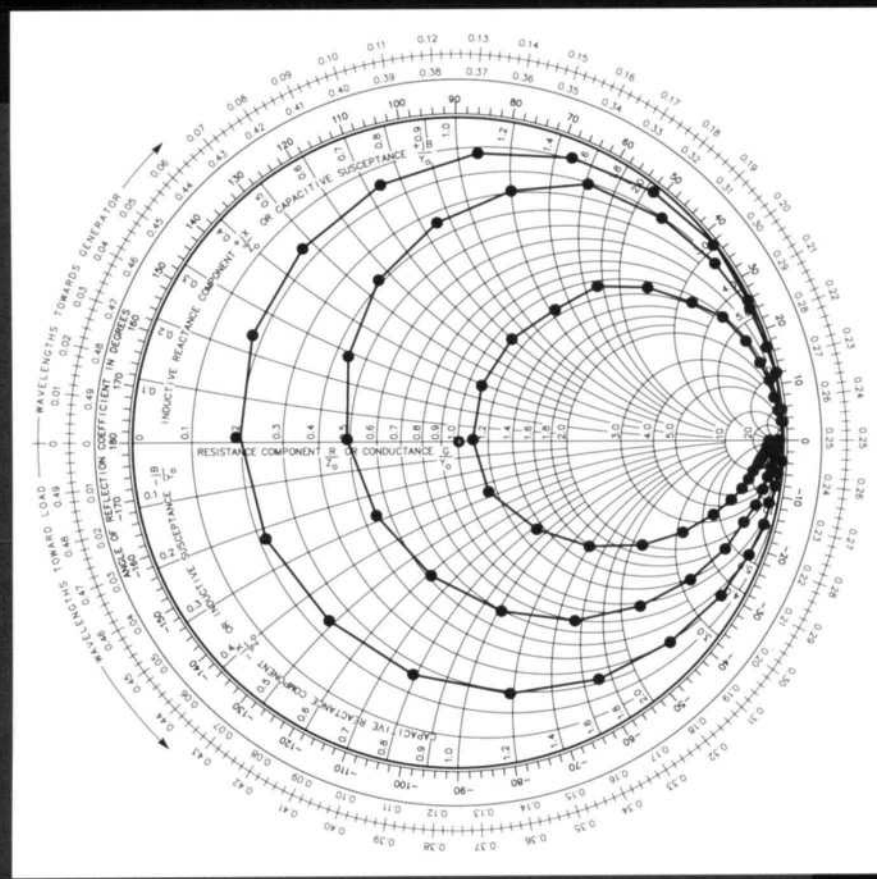
QEX

$1.75

ARRL Experimenter's Exchange

July 1996



Tuner Losses Can Hurt!

QEX: The ARRL
Experimenter's Exchange
American Radio Relay League
225 Main Street
Newington, CT USA 06111

QEX (ISSN: 0886-8093 USPS 011-424) is published monthly by the American Radio Relay League, Newington, CT USA.

Second-class postage paid at Hartford, Connecticut and additional mailing offices.

Offices

225 Main St, Newington, CT 06111-1494 USA
Telephone: 860-594-0200
Telex: 650215-5052 MCI
Fax: 860-594-0259 (24 hour direct line)
Electronic Mail: MCIMAILID: 215-5052
Internet:qex@arrl.org

Subscription rate for 12 issues:

In the US: ARRL Member $15, nonmember $27;

US, Canada and Mexico by First Class Mail: ARRL Member $28, nonmember $40;

Elsewhere by Surface Mail (4-8 week delivery): ARRL Member $20, nonmember $32;

Elsewhere by Airmail: ARRL Member $48, nonmember $60.

QEX subscription orders, changes of address, and reports of missing or damaged copies may be marked: *QEX* Circulation. Postmaster: Form 3579 requested. Send change of address to: American Radio Relay League, 225 Main St, Newington, CT 06111-1494.

Members are asked to include their membership control number or a label from their *QST* wrapper when applying.

In order to insure prompt delivery, we ask that you periodically check the address information on your mailing label. If you find any inaccuracies, please contact the Circulation Department immediately. Thank you for your assistance.

About the Cover

W9CF explains why your T-network tuner may not be delivering as much power to your antenna as you expect.

ISSUE NO. 174

Features

3 A Complete DSP Design Example Using FIR Filters
By John Wiseman, KE3QG

16 Estimating T-Network Losses at 80 and 160 Meters
By Kevin Schmidt, W9CF

21 The Copper Wire Gauge for Electrical Technology
By Antonio L. Eguizabal, VE7FIF

Columns

24 Upcoming Technical Conferences

July 1996 QEX Advertising Index

American Radio Relay League: 15, 26, 27, 28, 29
Communications Specialists Inc: 30
Down East Microwave, Inc: 30
HAL Communications Corp: 31
K6PY's Direction+: 15
PacCom: 32, Cov IV
PC Electronics: 30
Sescom, Inc: 23
Tucson Amateur Packet Radio Corp: 31
Z Domain Technologies, Inc: 30

THE AMERICAN RADIO RELAY LEAGUE

The American Radio Relay League, Inc, is a noncommercial association of radio amateurs, organized for the promotion of interests in Amateur Radio communication and experimentation, for the establishment of networks to provide communications in the event of disasters or other emergencies, for the advancement of radio art and of the public welfare, for the representation of the radio amateur in legislative matters, and for the maintenance of fraternalism and a high standard of conduct.

ARRL is an incorporated association without capital stock chartered under the laws of the state of Connecticut, and is an exempt organization under Section 501(c)(3) of the Internal Revenue Code of 1986. Its affairs are governed by a Board of Directors, whose voting members are elected every two years by the general membership. The officers are elected or appointed by the Directors. The League is noncommercial, and no one who could gain financially from the shaping of its affairs is eligible for membership on its Board.

"Of, by, and for the radio amateur, "ARRL numbers within its ranks the vast majority of active amateurs in the nation and has a proud history of achievement as the standard-bearer in amateur affairs.

A bona fide interest in Amateur Radio is the only essential qualification of membership; an Amateur Radio license is not a prerequisite, although full voting membership is granted only to licensed amateurs in the US.

Membership inquiries and general correspondence should be addressed to the administrative headquarters at 225 Main Street, Newington, CT 06111 USA.

Telephone: 860-594-0200
Telex: 650215-5052 MCI
MCIMAIL (electronic mail system) ID: 215-5052
FAX: 860-594-0259 (24-hour direct line)

Officers

President: RODNEY STAFFORD, KB6ZV
5155 Shadow Estates, San Jose, CA 95135

Executive Vice President: DAVID SUMNER, K1ZZ

Purpose of *QEX*:

1) provide a medium for the exchange of ideas and information between Amateur Radio experimenters
2) document advanced technical work in the Amateur Radio field
3) support efforts to advance the state of the Amateur Radio art

All correspondence concerning *QEX* should be addressed to the American Radio Relay League, 225 Main Street, Newington, CT 06111 USA. Envelopes containing manuscripts and correspondence for publication in *QEX* should be marked: Editor, *QEX*.

Both theoretical and practical technical articles are welcomed. Manuscripts should be typed and doubled spaced. Please use the standard ARRL abbreviations found in recent editions of *The ARRL Handbook*. Photos should be glossy, black and white positive prints of good definition and contrast, and should be the same size or larger than the size that is to appear in *QEX*.

Any opinions expressed in *QEX* are those of the authors, not necessarily those of the editor or the League. While we attempt to ensure that all articles are technically valid, authors are expected to defend their own material. Products mentioned in the text are included for your information; no endorsement is implied. The information is believed to be correct, but readers are cautioned to verify availability of the product before sending money to the vendor.

Empirically Speaking

Math Anxiety

One of the more common remarks we hear from potential—not current—*QEX* subscribers is: "There's too much math in *QEX* for me." We understand that point of view. After all, most amateurs are not professional engineers, scientists or mathematicians, and for those who are not, the math can be daunting.

QEX is a departure from most other ARRL publications, too. Most of the books and periodicals ARRL publishes are targeted toward a general amateur audience. We don't expect the readers of those publications to be able to handle abstruse mathematics—we *know* they can't, for the most part. Of course, you can't entirely eliminate math from electronics. Most of the time we just try to keep the math to the minimum necessary to allow the reader to accomplish his goals. But in *QEX*, we don't restrict the technical level—or the math.

Part of the reason *QEX* exists is because some amateurs *are* professionals in electronics, and they need a place they can come to exchange ideas without the kinds of restrictions that must, of necessity, be applied to broad-interest publications.

Just to confirm that this approach is the right one, we asked some time ago (January 1994) in this space whether the content of *QEX* was "too technical." The overwhelming response was that is was *not* too technical. *QEX* readers want the whole story and are willing to slog through the math if that's what's necessary to get it.

We feel that using *QEX* as a vehicle for this kind of material—instead of *QST*—is appropriate. Printing a few thousand copies of *QEX* and mailing them to the few thousand people who are likely to find the material of interest makes more sense than filling 170,000-plus ARRL members' mailboxes with material that the vast majority will just flip past.

We bring this up to make a point. It is *QEX* that is the chosen ARRL publication for "more technical" articles, by which we mean articles that are cast at a level that will be useful only to those amateurs who can handle more complicated math than is usual in amateur publications. Which doesn't mean that every *QEX* article has to be full of math, only that if math is needed to cover the subject of the article in depth, math is what we'll use. We do try to achieve a mix of articles that will appeal to both amateur experimenters and "amateur professionals."

So, if you have an idea for an article you think is worth publishing, and if that article contains technical material that is of a type you don't see in *QST*, likely it is *QEX* that is the best outlet for your article. And we'd love to have the opportunity to publish it.

This Month in *QEX*

DSP is all the rage, and you can find a fair amount of material that describes DSP algorithms and techniques. But how do you translate that into a working project? You've got to select an algorithm, find a DSP chip that will do the job and code the program. Where to start? John Wiseman, KE3QG, leads you through the process by providing "A Complete DSP Design Example Using FIR Filters."

That T-network low-band tuner is pretty handy; it'll tune just about anything and is easy to adjust. But do you know how much power you're losing in the tuner? You might be surprised. It's worth spending a bit of time "Estimating T-Network Losses at 80 and 160 Meters," as Kevin Schmidt, W9CF, did.

What can you say about wire gauges? Plenty, as it turns out. A look at the AWG table may lead you to think someone made it up randomly, but as Antonio L. Eguizabal, VE7FIF, explains, "The Copper Wire Gauge for Electronic Technology" is actually quite logical.

'Tis the season for conferences, and "Upcoming Technical Conferences" again brings you the latest on the subject.—*KE3Z, email: jbloom@arrl.org.*

A Complete DSP Design Example Using FIR Filters

Following this example through from start to finish will give you a good introduction to the process of designing a DSP application.

By John Wiseman, KE3QG

The purpose of this article is to illustrate a design example for a digital signal processing (DSP) algorithm, from basic conception through real-time implementation. The algorithm chosen for illustration is the finite impulse response (FIR) filter. This class of digital filter is capable of excellent performance in amateur-radio applications, is fairly easy to implement in a DSP chip, and its implementation uses a number of high-performance features unique to DSP architectures. By describing the low-level details, I hope to provide insights into important aspects of DSP design and implementation. These issues are pertinent not only to FIR filters, but also to other more advanced algorithms such as adaptive filtering, frequency domain processing, demodulation and other functions that amateur-radio experimenters will find of interest. I'll discuss simulation of algorithm performance before implementation, real-time performance estimation, floating-point versus fixed-point DSP chip architectures, and how certain unique features of DSP chips can be effectively used.

20 Amaryllis Lane
Newtown, PA 18940
email: johnw@asl.hitachi.com

FIR Filter Design

The implementation equation for a finite impulse response (FIR) filter is:

$$y(n) = \sum_{m=0}^{M-1} c(m)x(n-m) \qquad \text{Eq 1}$$

where M is the number of coefficients, or taps, in the filter, $x(n\text{-}m)$ are the sampled data inputs, $y(n)$ is the filtered output and $c(m)$ are the filter coefficients. The values of the coefficients also represent the filter's impulse response. To implement a specific filter, we must first find appropriate values of c, then calculate Eq 1 in real-time to produce an output y. Compare this equation to that of an infinite impulse response (IIR) filter:

$$y(n) = \sum_{m=0}^{M} b(m)x(n-m) - \sum_{m=1}^{N} a(m)y(n-m) \qquad \text{Eq 2}$$

Eq 2 results in both poles and zeroes in the transfer function, unlike the FIR transfer function of Eq 1, which contains only zeroes. Because of this, the IIR filter is more susceptible to instabilities due to quantization of filter coefficients, an issue that will be discussed later in detail. Another important issue is that FIR filters with

symmetric coefficients are guaranteed to have linear phase response. Proof of this is beyond the scope of this article but is generally given in signal processing textbooks dealing with FIR filter theory.[1,2] Because of their inherent stability and linear phase characteristics, as well as the fact that they are easy to derive coefficients for, FIR filters provide good first-time DSP design experience, even though they will in general require considerably more coefficients to provide the equivalent amount of attenuation compared to IIR filters.

One of the easiest and most straightforward ways of deriving the coefficients for an FIR filter is to use what is known as the Fourier series method. This method takes advantage of the fact that the frequency response of the filter is really a periodic function with the same period as that of the sampling frequency used to sample the analog input signal. (See Fig 1A.) Because of this periodicity of the frequency response "waveform," we can derive the Fourier series coefficients and use them as our FIR filter coefficient values. The equation for deriving these coefficients is:

$$C_n = \int_{-0.5}^{0.5} H(\nu)\cos(2\pi n\nu)d\nu \qquad \text{Eq 3}$$

where n is the coefficient number, ν is the frequency, and $H(\nu)$ is the desired frequency response. In our case, the desired frequency response is of magnitude one in the passband and magnitude zero outside of the passband, so this equation reduces to:

$$C_n = \int_{-f_c}^{f_c} \cos(2\pi n\,\nu)d\,\nu = 2\left|\frac{\sin(2\pi n\,\nu)}{2\pi n}\right|_0^{f_c} = \frac{\sin(2\pi n f_c)}{\pi n} \qquad \text{Eq 4}$$

for an even-function low-pass filter. Note that f_c represents the desired filter cut-off frequency, expressed as a fraction of the sampling frequency f_s. Because of this, its range will be 0 to 0.5, as this represents the range of frequencies under the Nyquist limit (1.0 is the sampling rate itself). A note of caution is warranted here. Some literature uses the fractional notation as it is used in this article, where 0.5 is the Nyquist limit and 1.0 is the sampling frequency, while other sources have been known to define their Nyquist limit at 1.0 and the sampling frequency at 2.0. Yet other sources may define these values in angular frequency terms, as π and 2π, respectively.

As an example, a quick design for a 7-tap FIR filter with f_c=0.125 (for example, 900 Hz with f_s = 7200 Hz) may be done with a hand calculator. Because of coefficient symmetry about zero, we only have to calculate the following:

$c_3 = 0.07503$ $c_2 = 0.15915$ $c_1 = 0.22508$

By using symmetry:

$c_{-3} = 0.07503$ $c_{-2} = 0.15915$ $c_{-1} = 0.22508$

The "zero" coefficient cannot be directly calculated as it results in 0/0, but elementary calculus (L'Hopital's rule) tells us that the result will be $2f_c$, so:

$c_0 = 0.25000$

From this basic result for a low-pass filter, it is interesting to see how to obtain other filters as well. A high-pass filter with the same cut-off frequency can be obtained from the low-pass coefficients by using the following transformation:

$$C_{n(HP)} = (-1)^n C_{n(LP)} \qquad \text{Eq 5}$$

Using the low-pass coefficients from the previous example gives the following high-pass coefficients:

c_{-3}=–0.07503 c_{-2}=0.15915 c_{-1}=–0.22508 c_0=0.25000 c_1=–0.22508 c_2=0.15915 c_3=–0.07503

I tend to use band-pass filters the most, and they are easily designed by changing the limits of integration on the formula for the Fourier series coefficients. An example with the lower passband edge at f_{cl} and the upper passband edge at f_{cu} (Fig 1B) is:

[1]Notes appear on page 13.

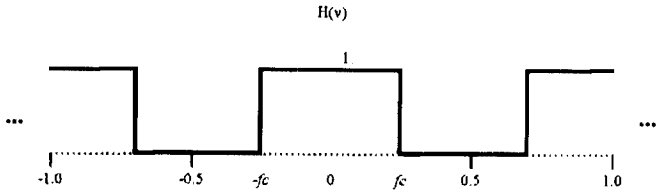


Figure 1A
Desired Lowpass Filter Frequency Response

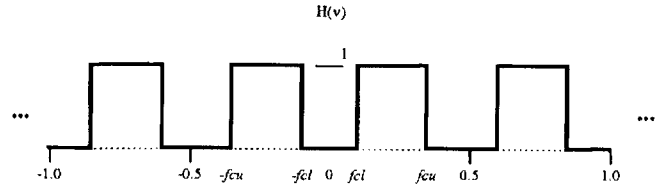


Figure 1B
Desired Bandpass Filter Frequency Response

Fig 1—A desired low-pass filter frequency response is shown at (A), a desired band-pass filter frequency response at (B).

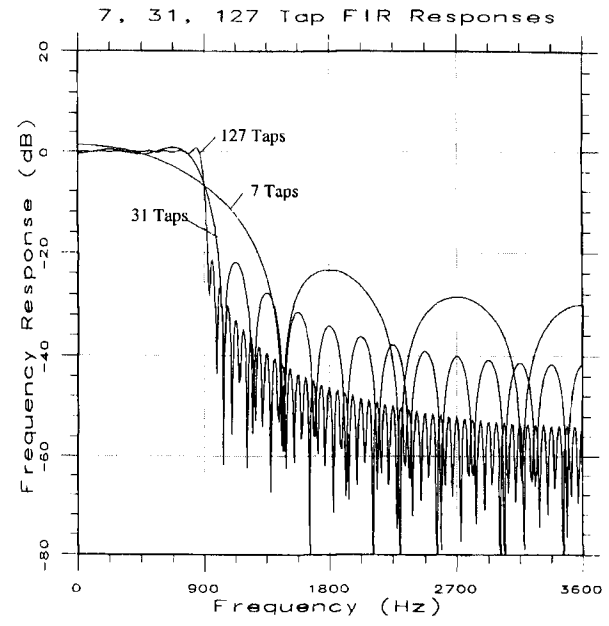


Fig 2—The responses of three different FIR filter designs, each of which is designed for a cut-off at 900 Hz with a sampling frequency of 7200 Hz.

$$C_n = \left.\frac{\sin(2\pi n\,v)}{\pi n}\right|_{f_{cl}}^{f_{cu}} = \frac{\sin(f_{cu}2\pi n)}{\pi n} - \frac{\sin(f_{cl}2\pi n)}{\pi n}$$

Eq 6

Computer-Generated Coefficients

The formulas for generating FIR filter coefficients are fairly simple and easy to use, but they're impractical to use directly for more than a few high-precision values. This job can be made almost trivial by writing a software program that can calculate the coefficients for arbitrary cut-off frequencies and arbitrary numbers of filter coefficients. Listing 1 is a C program I wrote to perform this function. Included in the program is a function that also calculates the frequency response for the newly designed filter, so you can verify its performance before implementation. The response values can be inspected visually or imported into a plot program for more detailed analysis.

As an example, Fig 2 shows the response of three different FIR filter designs. The filters were designed to have a low-pass cut-off frequency of 900 Hz, which corresponds to a factor of 0.125 for a sampling frequency of 7200 Hz. The filters were designed with 7, 31 and 127 coefficients, and you can see that the desired frequency response is more closely approximated by using more coefficients, as would be expected from Fourier theory.

As I said, the Fourier series method is easy and straightforward to implement, but it is somewhat limited in its capabilities. Other mathematical methods that are often used to derive the coefficients include maximally flat approximations, least-squared error designs and minimax error designs. A powerful example of the latter technique is the Remez exchange algorithm, which can be used to design filters for an arbitrary set of specifications with the minimum number of filter coefficients. Variations of these algorithms are provided on supplemental disks to various DSP textbooks for those experimenters who do not desire to write their own coefficient generating software.[3,4,5]

Window Functions

The coefficients for a low-pass FIR filter of 127 taps are plotted in Fig 3. Note that the coefficients do not approach zero, but instead are abruptly truncated at each end of the sequence. This abruptness causes ringing in the passband, known as the Gibbs phenomenon (Note 2), and relatively poor sidelobe attenuation for filters designed with this method. (An example of how the frequency response is affected is given in the next section.) Fig 4 shows the same filter coefficients multiplied by the values for a Hamming window. Notice that the coefficient values now approach zero at the ends of the sequence. *Windowing* is an important concept and is used in frequency domain processing as well, for purposes such as reducing spectral "leakage" in discrete Fourier transforms.

The window coefficients that are used with the examples in this article are from the formula for the generalized Hamming window:

$$w_n = \alpha + (1-\alpha)\cos(n\pi/N) \qquad \text{Eq 7}$$

When the constant $\alpha = 0.5$, the window is known as a *Hanning* (or *Hann*) window. If $\alpha = 0.54$, it is known as a *Hamming* window. As a complement to the pervious 7-coefficient low-pass filter example, a 7-coefficient Hamming window sequence can be generated with a hand calculator, with $N=(7-1)/2=3$. This gives: $w_3=0.08000$, $w_2=0.31000$, $w_1=0.77000$ and $w_0=1.00000$

Because the function is symmetric (try the calculation for $n=-3$), the remaining values are: $w_{-3}=0.08000$, $w_{-2}=0.31000$ and $w_{-1}=0.77000$

A new set of windowed coefficients is now created by multiplying the previously generated FIR coefficients with these windowing values on a pairwise basis like this:

$c_{-3}\times w_{-3}$, $c_{-2}\times w_{-2}$, ..., $c_3\times w_3$

The FIR filter design C program generates four different windows: triangular, Hanning, Hamming and Blackman. You can select which window to use or, if a 0 is entered for this

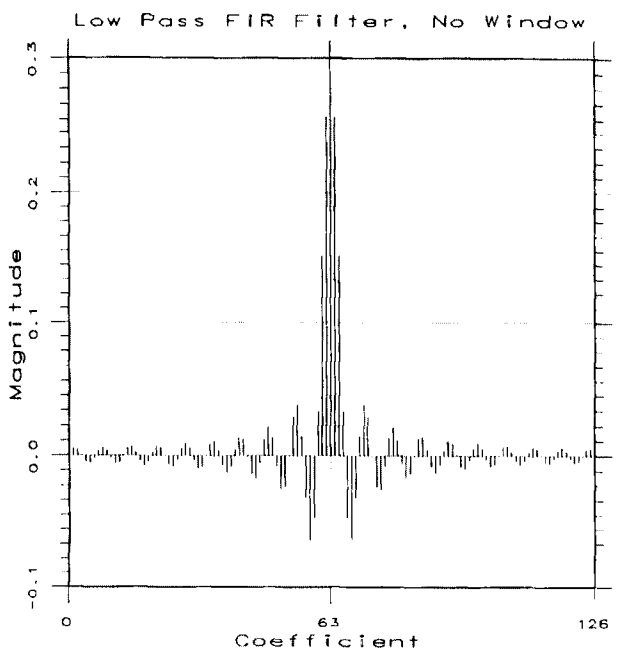


Fig 3—Coefficients for a low-pass FIR filter of 127 taps.

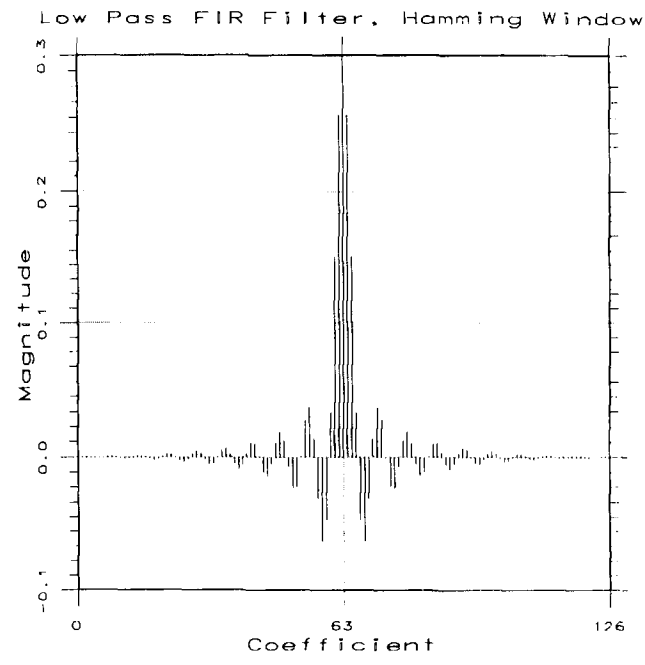


Fig 4—The same filter as Fig 3 but with the coefficients multiplied by the values of a Hamming window.

selection, the default is to use no window, which is actually a rectangular function. This is by no means the complete list of possible window functions. Others, such as the Kaiser window, can be fairly easily added to the program.

Which window function should be used for a particular filtering problem is a trade-off. Experimentation will help show you what parameters work best for a given level of interference, your own personal listening tastes, what particular DSP chip architecture the filter will be implemented on, and so on.

Simulated Filter Performance

Fig 5 is a plot of the output data from the C program for an FIR band-pass filter with 127 coefficients. In this case, the lower cut-off frequency is 0.2 (7200×0.2=1440 Hz) and the upper cut-off frequency is 0.3 (7200× 0.3=2160 Hz). No windowing function is used. Fig 5 shows significant ringing in the passband due to the Gibbs phenomenon. Another problem with this filter design is that the sidelobes are not attenuated much relative to the passband. These problems can be helped significantly by the use of windowing functions.

Fig 6 is a plot of the same filter design, this time using a Hanning window. The passband ringing is virtually gone, and the sidelobe attenuation is considerably better. The trade-off in use of windowing can also be seen—the passband skirts are slightly less steep than for the nonwindowed filter.

Fig 7 is another plot of the original filter design, but a Hamming window is used in this case. Comparing this plot directly with Fig 6 shows that a fairly small change in the windowing function can have significant effects on the resulting filter frequency response. In this case, the first sidelobes outside of the passband are attenuated by about 10 dB more than with the use of a Hanning window. Again, there is a price to pay: the attenuation of the extended sidelobes is not nearly as good, approximately 65 dB compared with 120 dB. Passband roll-off at the skirts is comparable between the two windowing functions. It should be clear that the choice of windowing function is based largely on the allowable passband ripple and sidelobe attenuation as well as the needed filter skirt shape.

One interesting point about these plots is that the data is calculated using double-precision floating-point arithmetic in the C program. In the DSP chips we'll use, calculations are performed with single-precision floating point (TMS320C30) or 16-bit fixed-point (TMS320C50) arithmetic. The move to single-precision floating-point implementation with the C30 will not make a significant difference, but using the C50 with 16-bit (fixed-point) filter coefficients will result in deviation from the simulated frequency response performance. I have added a feature to the C program that calculates the frequency response using 16-bit coefficients, and a plot of this for the Hanning window case is shown in Fig 8. Contrast this plot with that of Fig 6. Since the quantization noise of a 16-bit value is approximately 96 dB, it is not reasonable to expect the 120-dB filter attenuation of Fig 6. If that kind of performance is needed on a 16-bit processor, and if you can tolerate running slower and using more memory, a 32-bit double-precision math scheme could be implemented.

Fig 9 shows the response of a 16-bit version of the filter with a Hamming window. The differences between this plot and the corresponding plot of Fig 7 are much smaller than we saw between Figs 6 and 8, as the filter attenuation is not that large to begin with, fitting better into the available dynamic range of the 16-bit system.

This is part of the process that the DSP designer must perform to optimize the two variables of performance and price. In general, the cost of DSP chips rises as the word width increases, and the cost of analog-to-digital and digital-to-analog converters

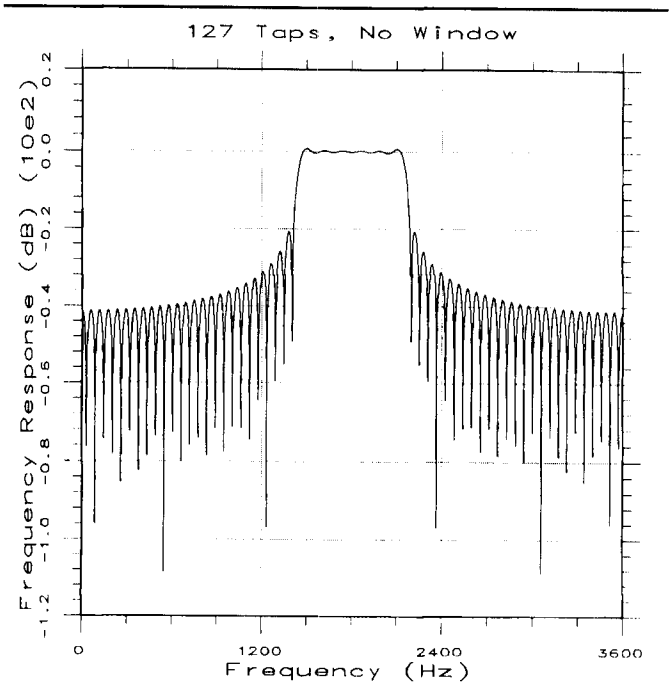


Fig 5—The frequency response calculated by the C program for the 127-tap filter of Fig 3. No windowing is used.

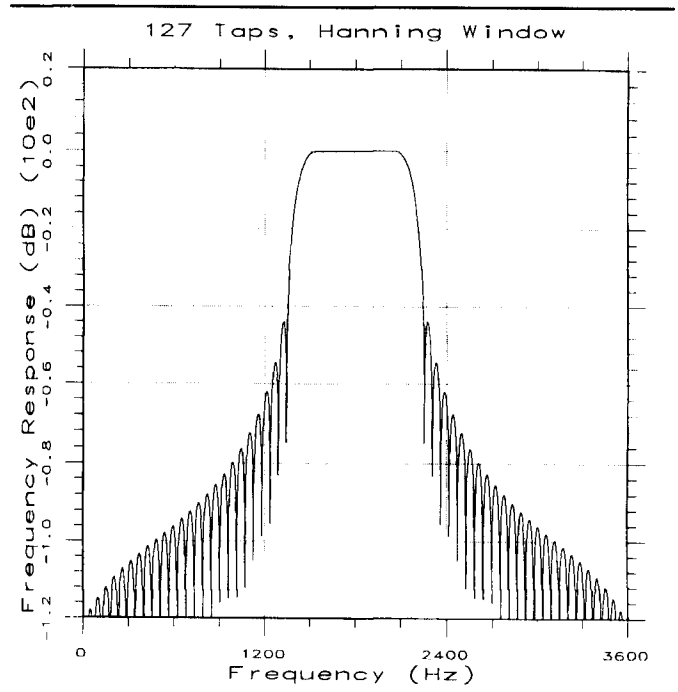


Fig 6—The frequency response for the filter design of Fig 3 but using a Hanning window.

will follow the same trend. In a high-volume product, the cost differences between a 16-bit, 24-bit, or even a floating-point DSP chip may force a compromise in performance in the final product. This need not be the case for the amateur-radio experimenter, however, as the cost of DSP starter kits is relatively low for all of these architectures.

DSP Hardware Platform Selection

Now that we know how to generate coefficients for an FIR filter that satisfy particular requirements, we are ready to implement the design in a DSP chip. The two chips that I'll cover in this article are the Texas Instruments TMS320C30 and the TMS320C50. These two popular DSP chips are representative of floating-point architectures (C30) and 16-bit fixed-point architectures (C50). Both chips are available on internal PC half-cards that contain fast single-cycle static RAM external to the DSP chip, a PC interface and an analog interface circuit (AIC). The AIC is a single chip containing a 14-bit analog-to-digital converter (ADC) for the incoming signal, a 14-bit digital-to-analog converter (DAC) for the processed outgoing signal, antialiasing filters and a programmable sampling frequency. This particular configuration is known as a C30 or C50 Evaluation Module, or EVM for short.

The C50 is also available as part of a starter kit called the DSK. This board has the C50 DSP chip and the AIC but is a stand-alone board that connects to a PC with a serial cable. No external RAM is provided on the DSK, but the internal RAM of the C50 should be more than enough for most amateur-radio experiments. This board is available from Texas Instruments for $99 and includes a code assembler and loader. The software tools are slightly more limited than the full set available with the EVM boards (no linker is provided, so absolute addressing is required), but again they will easily suit the filtering example given here, as well as much more complicated algorithms that you might develop on your own. The DSK is meant to compete with starter kits available from other manufacturers such as Motorola, for their 56002, and Analog Devices, for their

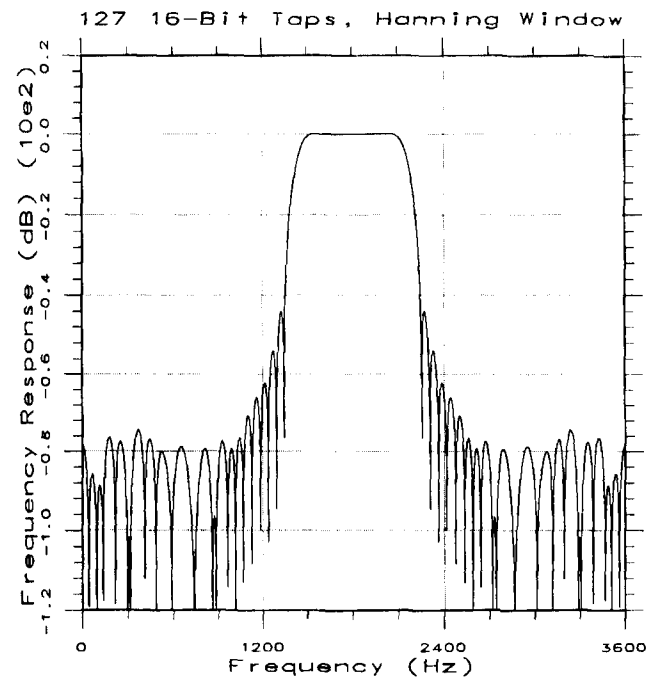


Fig 8—The deviated frequency response of the filter using 16-bit coefficients and a Hanning window. Contrast this plot with that of Fig 6.

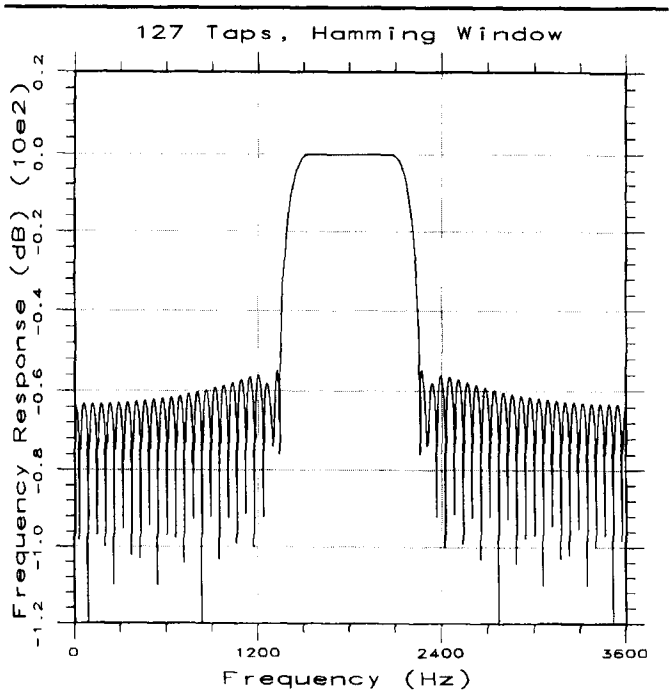


Fig 7— The frequency response for the filter design of Fig 3 but using a Hamming window.

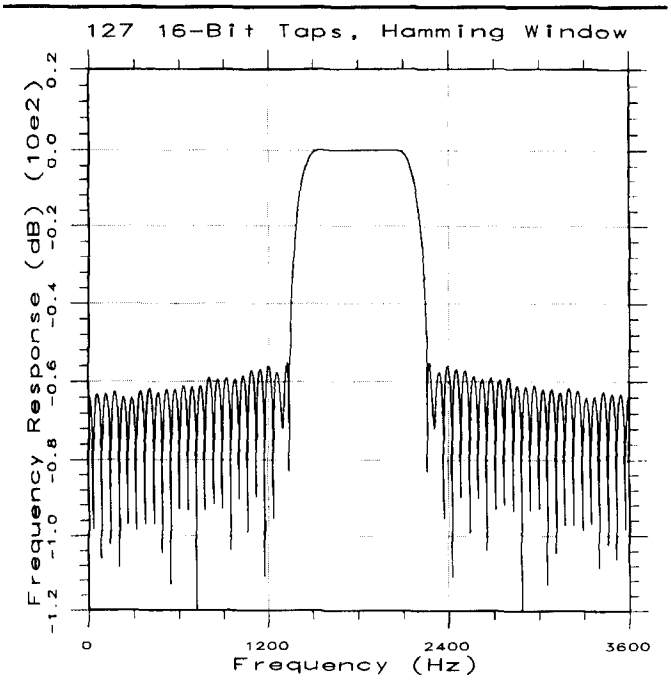


Fig 9—The deviated frequency response of the filter using 16-bit coefficients and a Hamming window. Contrast this plot with that of Fig 7.

2101. As I was writing this article, I came across an announcement from Texas Instruments that a single-bus version of the C30, the C31, would also be available in a DSK starter kit for $99. Although all of these DSP boards can be used to perform the filtering tasks described here, some architectures may provide you with hardware or software features that you might find more appealing to work with. I suggest you check the various manufacturers' data sheets on these boards and the individual DSP chips themselves before settling on a platform to experiment with yourself, as you might find some features, such as the 16-bit ADC/DAC on the Motorola board, to be more beneficial to your algorithms.

Real-Time Performance

Before attempting to implement any algorithm on a DSP chip, a real-time performance analysis should be done. In this case, the analysis is very straightforward and easy to do. The first thing that is needed is to select a reasonable sampling frequency for the analog-to-digital converter within the AIC. A quick look at the AIC data sheet shows that the minimum sampling rate that can be programmed with the standard clock is 7.2 kHz. To prevent aliasing, the analog signal must be sampled at a frequency at least twice that of the highest frequency contained within the signal. Since it's desirable that the same sampling clock be used for both SSB and CW, the maximum frequency of the incoming analog signal could be limited to 2.8 kHz by the AIC analog filter and the Nyquist sampling theorem would be satisfied. If we band-limit our incoming signal to 2.8 kHz, then 7.2 kHz/2.8 kHz=2.5, an acceptable value.

The DSP chip has to perform all of its computations for a given input sample before the next sample arrives. In this case, samples arrive every 1/7200=139 μs. The processor clock speeds for the C30 and C50 EVM boards are 30 and 50 MHz, respectively. Both DSP chips have machine cycles operating at speeds of one-half of the clock, so the C30 runs at a speed of 15 million cycles per second. If the assumption is made that all instructions will execute in one cycle (not always the case) then we can calculate the number of real-time instructions available as 15,000,000/7200. This simple calculation shows that we will have 2083 processor cycles available for our algorithm to successfully complete within the available sampling time before new data is ready. This is quite a lot of "horsepower," and we will have plenty of time left over after implementing a sophisticated time-domain FIR filter.

By extrapolation we can see that if a very high-performance DSP (40 million instructions per second is not unreasonable today) is to be used, a fairly high sampling rate of 40 kHz would leave 1000 instructions per sampling interval for algorithm processing. This is essentially what is done for low-IF digital signal processing today.

Software Implementation

Once you've selected a hardware platform that is capable of executing your algorithm, the next step is to write (and debug!) the software. DSP chips have some unique architectural features that assist in efficient execution of DSP algorithms. Even an experienced programmer of general-purpose microprocessors will have some learning to do before being able to write code for a DSP chip.

Probably the principal difference between general-purpose processors and DSP chips is that the latter make use of a *Harvard* architecture. This scheme uses separate program and data memory areas, which allows simultaneous access of instructions and operands. Refer to the programmer's manual for the processor you're using to learn the specifics of the memory layout of the chip.

Other DSP-specific differences include support for hardware-assisted circular buffering and for parallel execution of common DSP operations, both of which we need to make use of in implementing the FIR filter.

Circular Buffering

The assembly code examples make use of the concept of *circular buffering*. The equation for an FIR filter (Eq 1) shows that M data samples need to be collected so that a sum-of-products operation can be performed with the coefficients, resulting in one output value. For a filter of length 7, 7 input data samples need to be stored. For a filter with 127 coefficients, 127 input data samples need to be stored. As each new sample arrives, it must be placed into storage and the oldest stored sample must be discarded. And the order in which the coefficients are used with the samples is important, too: the first coefficient is multiplied by the oldest sample, the second coefficient by the next-to-oldest sample and so on. In a standard microprocessor, a variable-length buffer would have to be set up in memory and a pointer structure would have to be maintained to keep track of where incoming data was placed to avoid having to shift and move the entire block of previously stored data values. A circular buffer is a DSP chip feature that is designed into the hardware to help with this potentially high-overhead function.

Fig 10 shows how circular buffers are used for both the FIR coefficients and the incoming data values in the 7-tap filter. The c_n values are placed in consecutive memory locations, with lower memory addresses being towards the left in this example. The vertical arrows above the memory locations indicate the position of a memory pointer, with the pointer starting out the calculation from position 1. Likewise, the incoming data values are placed into separate consecutive memory locations, and in the first example x_n is the most recently received data value from the analog-to-digital converter. The filter calculation is now performed by multiplying the c_{-3} value that the coefficient pointer is accessing times the x_{n-6} data value that the data pointer is accessing, placing the product in the accumulator. Now both circular buffers *automatically update their respective pointers* to the next memory locations indicated by arrow number 2. These data values are then multiplied together, and the results are added to the previous results and again stored in the accumulator. This process is repeated until all the coefficients and stored data values have been accessed.

At this time, the circular buffers are each updated, but in slightly different manners. The coefficient buffer values remain constant—no new coefficients are going to be added or changed during the filtering operation. So the pointer to this buffer is simply set up to return to position 1 for the next calculation cycle. The data circular buffer handling is different, however, as a new incoming value (from the ADC) must be stored. When that sample value arrives, the incoming value x_{n+1} must replace the oldest value in the buffer, x_{n-6}. But since x_{n+1} is the most recent data value, it must be the last to be accessed from memory during the calculation cycle, being used in conjunction with the c_3 coefficient. To accommodate this, the starting position of the pointer to the input data buffer is *advanced by one position* after the

new sample value is stored, as seen in the second data buffer line of Fig 10. The next filter calculation now starts with the pointers in position 1 of their respective buffers, so that the first multiply/accumulate starts with c_{-3} and x_{n-5}. When the data buffer pointer reaches the highest position in memory, it then *wraps back* to the lowest position automatically to access the next value consecutively. This wrapping around is why the concept is referred to as a circular buffer. If you were to cut out the memory buffers from Fig 10 with scissors and tape the lowest memory position to the highest, it would reinforce the circular nature of what is actually happening.

Circular buffers greatly simplify the handling of both the filter coefficients and the incoming data values to be filtered. Not only do they help to feed the multiply/accumulate logic with data quickly, but they provide an efficient memory structure so that only $2M$ memory locations need to be dedicated to the storage of coefficients and data values.

C30 Assembly Code Example

Fig 11 gives an assembly language code listing for an FIR filter implementation on the C30 EVM. After processor reset, an initialization routine is run that leads to the routine labeled LOOP. A polling loop at the start of the routine continually checks to see if the analog interface chip (AIC) is ready to accept another data word from the DSP. If not, the device is continually repolled. This function could also be done with an interrupt service routine, but I chose to use polling as it is easier to follow in the coded examples. If the AIC is ready, an integer data value is read by the DSP and converted to floating-point format for further mathematical processing. This floating-point data value is then stored in the circular buffer that has been previously set up in internal memory.

Now that data has been converted into the correct format and placed into the circular buffer, we are ready to compute the filter output by implementing the sum-of-products operation of Eq 1 using the stored filter coefficients. Again, hardware within the DSP chip is used to make this operation extremely fast and efficient. The RPTS instruction forces the C30 to execute the next instruction in a repeating loop with no performance penalty. This is advantageous to the developer in two ways. First, the DSP will be executing only the multiply-add instruction in consecutive cycles, so speed will be maximized. Second, the constant that is called TAPS in the RPTS instruction can be defined at the beginning of the program and easily modified so that the filter length can be changed quickly. As your DSP programs grow in complexity, this becomes a major benefit in helping with code maintenance.

Analyzing the one instruction that is actually calculating the sum-of-products for the filter shows that quite a lot is happening during this single machine cycle. First, the presence of the || symbol indicates that a parallel operation is taking place in the DSP. An MPYF3 (a floating-point multiplication with 3 operands) is taking place at the same time as an ADDF (a float-

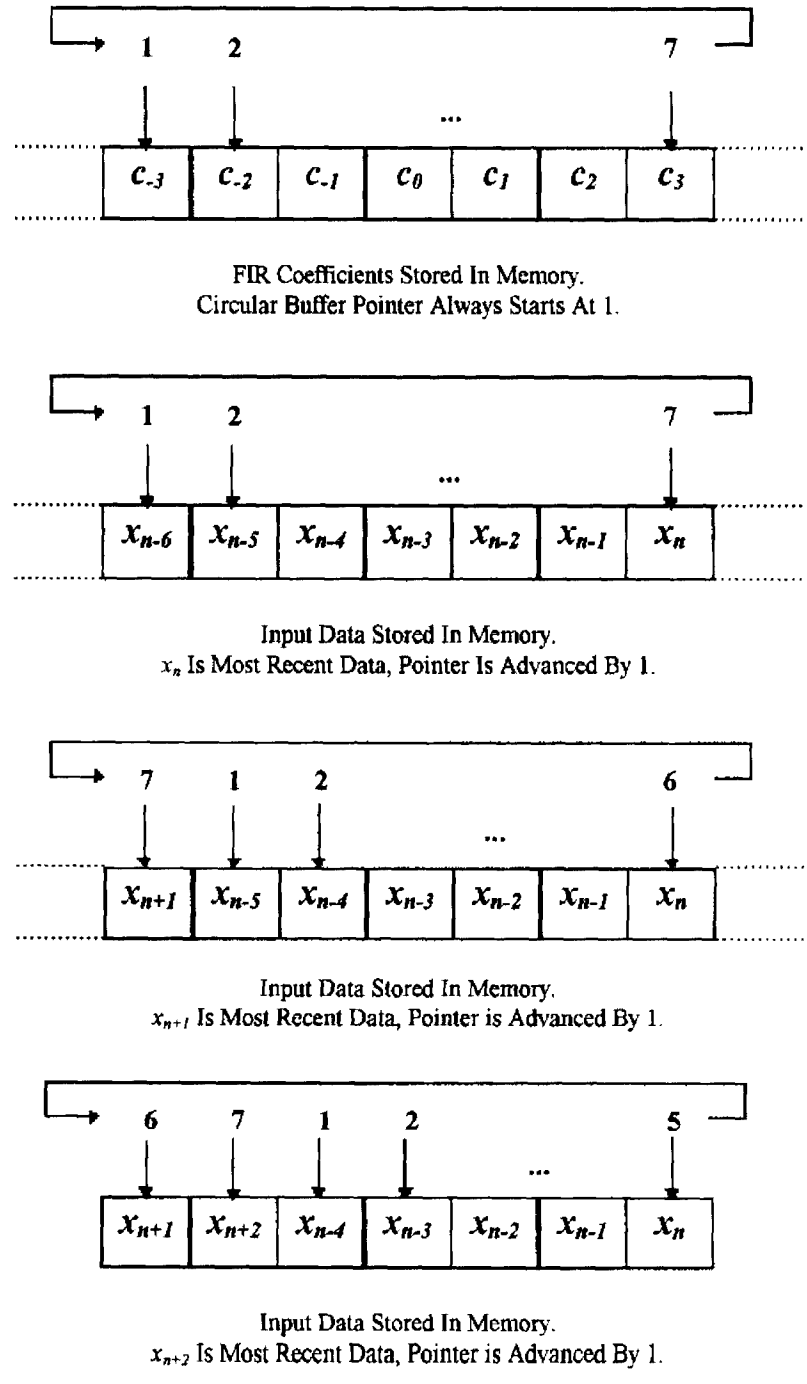


Fig 10—An example of how the coefficient and data circular buffers are managed for a 7-tap filter.

ing-point addition). Not all individual instructions can be combined into parallel instructions. Since the entire instruction word itself is limited to 32 bits, a small subset of these bits is available to be allocated to instruction decoding, register allocation, addressing mode selection, etc. Because of this, a limited number of instructions can be paralleled. The designers of the chip chose to allow parallel execution of those instruction pairs that are most useful in typical signal processing algorithms. The MPYF3 | | ADDF instruction is a good example, as it implements a sum-of-products operation, the heart of an FIR filter calculation.

The first two operands of the MPYF3 portion of the instruction are used to get the data values from the input circular buffer and the coefficient values, both now in internal DSP RAM. The multiplier portion of the circuitry multiplies these two values together and places the floating-point result into the 3rd operand, R6. At the same time (in parallel), an adder in the ALU adds the floating-point values in R6 and R5 with the resulting value placed, or accumulated, into R5. Both input operands for the multiply instruction are retrieved by circular address generators. The input data circular buffer is advanced one extra position after every run through the entire routine to make room for an incoming data word, while the coefficient circular buffer is reset back to the starting point of the coefficient table located in RAM. (See Fig 10.)

It is important to understand that because the instructions are operating in parallel, the ADDF portion of the instruction is operating on data that is one cycle *behind* in time! The architecture of the C30 is such that R6 is updated by the MPYF3 portion *after* the current value in R6 is read by the ADDF portion. Because of this, after the desired number of sum-of-products operations is performed, a final ADDF instruction must be tacked on to the end to accumulate the final product (in R6) into the sum. *After this instruction*, R5 will contain the correct answer for the filter calculation.

After the final addition is completed, the filter output data must be converted from floating-point back to integer form for output to the AIC digital-to-analog converter (DAC). This conversion is accomplished in one cycle with the FIX instruction. Finally, the lower 2 bits of the 16-bit data word must be set to zero, as the AIC accepts as data only the upper 14 bits, using the lower two bits during the initial programming and set-up phase.

Now that the calculation is complete and the filtered data word has been output from the DSP to the outside world, the routine executes a branch back to begin polling for the availability of more input data. The branch execution is enabled by the BD instruc-

```
; The following code implements an FIR filter of length TAPS on the TMS320C30 floating point DSP
; chip.
; Initialization code is not shown. Register assignments are not optimized, and may need to be modified
; in other applications.
;
; AR1 points to the serial port control register.
; AR2 points to the data receive register.
; AR3 points to the data transmit register.
; AR5 points to the start of the filter coefficient table.
; AR6 points to the input circular buffer.
; AR7 points to the output data mask.
;
; R1 contains the transmit bit mask (preloaded).
; R2 contains the input value converted to floating point.
; R5 is used as an accumulator.
; R6 is used as a temporary buffer for the multiplier.

LOOP  TSTB   *AR1,R1              ; check transmit bit - R1 is mask
      BZD    LOOP                 ; loop if zero, continue if one (delayed)
                                  ; next 3 instructions are always executed
      LDI    0.0,R5               ; clear "accumulator" for next filter run
      LDI    0.0,R6               ; clear temp register for next filter run
      FLOAT  *AR2,R2              ; AR2 points to data receive register
                                  ; convert integer to Floating point
                                  ; delayed branch occurs at this point
      STF    R2,*AR6++%           ; store data in circular buffer
      RPTS   TAPS-1               ; repeat next instruction for # of taps in filter
      MPYF3  *AR6++%,*AR5++%,R6   ; input circular buffer x coefficient table
||    ADDF   R6,R5                ; accumulate into R5
      ADDF   R6,R5                ; final accumulate
      BD     LOOP                 ; delayed branch to LOOP
      FIX    R5,R5                ; convert answer to integer format
      AND3   *AR7,R5              ; mask lower 2 bits to zero (AR7 => mask)
      STI    R5,*AR3              ; store output in data TX register
                                  ; delayed branch occurs at this point

; This is the coefficient file for a lowpass FIR filter of length 7, with cutoff frequency at 0.125.
; The coefficients are in floating point format suitable for usage in the C30 assembler.

COEFF .asect  "coeff",0100h       ; filter coefficients start at memory location 0100h
      .word   0.0750263596
      .word   0.1591549431
      .word   0.2250790790
      .word   0.2500000000
      .word   0.2250790790
      .word   0.1591549431
      .word   0.0750263596
```

Fig 11—Assembly-language code listing for an FIR filter implementation on the C30 EVM.

```
; The following code implements an FIR filter of length TAPS on the TMS320C50 16-bit DSP chip.
; Initialization code is not shown. Register assignments are not optimized, and may need to be modified
; in other applications.
;
; AR1 points to the serial port control register.
; AR2 points to the data receive register.
; AR3 points to the data transmit register.
; AR6 points to the input data circular buffer.
;
; The Accumulator Buffer must be loaded with the 32-bit mask FFFC0000h during initialization.

LOOP  BIT    *,4                  ; check XRDY in serial port control register
      BCND   LOOP,NTC             ; loop if zero, continue if one
      MAR    *,AR2                ; change pointer register to AR2
      LACL   *,AR6                ; input data from data receive register and place
                                  ; in lower accumulator, change pointer register to AR6
      SACL   *+                   ; store input data in circular buffer, advance circular
                                  ; buffer one position
      RPTZ   TAPS-1               ; repeat next instruction for # of taps in filter
                                  ; automatically clear accumulator and product register
                                  ; for next filter run
      MAC    COEFF,*+             ; multiply-accumulate input circular buffer and
                                  ; coefficient table
      APAC                        ; final accumulate
      MAR    *,AR3                ; change pointer register to AR3
      BD     LOOP                 ; delayed branch to LOOP
      ANDB                        ; mask lower 2 bits of upper accumulator word to zero
                                  ; by AND of accum. with accum. buffer
      SACH   *,AR1                ; store output in data transmit register, change pointer
                                  ; to AR1 before returning to LOOP
                                  ; delayed branch occurs here

; This is the coefficient file for a lowpass FIR filter of length 7, with cutoff frequency at 0.125
; The coefficients are in hex format suitable for usage in the C50 assembler.

COEFF .asect  "coeff",0100h       ; filter coefficients start at program memory 0100h
      .word   099ah
      .word   145fh
      .word   1ccfh
      .word   2000h
      .word   1ccfh
      .word   145fh
      .word   099ah
```

Fig 12—Assembly\-language code listing for an FIR filter implementation on the C50 EVM.

tion, whose mnemonic is short for branch delayed. Under normal branching operation there is a three-instruction delay (two-instruction delay in the C50) in performing the actual branch after the instruction is read. If this branch were the last instruction in the routine, there would effectively be a waste of three instruction cycles because of internal processor pipeline delays. The delayed branch helps the programmer make use of this time by potentially allowing one to fill these gaps with useful instructions. In the routine shown, the conversion to integer and the mask/store operations are performed just prior to when the branch back to LOOP is actually implemented. The only caveat to using delayed branches is that during a conditional test the programmer must be careful, because the suceeding two instructions will be executed *regardless of whether the branch is actually taken*.

The second portion of the assembly language program is labeled as COEFF, and it contains the actual filter coefficients as generated from the C program. For convenience, the C program prints the floating point coefficients into a separate file along with the .word assembler directive. The .asect directive tells the assembler that the data is to be assembled into the program at absolute address 0100h, which was chosen somewhat arbitrarily. To change filters in this basic set-up, generate a coefficient file for the desired filter parameters, then include the data values into the assembly language file at the COEFF entry point. If the number of coefficients has changed, modify the value of TAPS and then reassemble the code.

C50 Assembly Code Example

Fig 12 shows an assembly code listing for the C50 EVM that performs the same filtering function as the C30 code analyzed above. At a glance, it should be apparent that there are some interesting differences between the two. First, the instruction mnemonics are not the same even though the two DSP chips are designed and produced by the same manufacturer. Second, there are no parallel instructions in the C50 architecture. And third, it takes more assembly-language instructions to do the same thing with the C50, at least for this coding example with my particular style.

Again, a simple polling scheme is implemented as the starting point of the FIR filtering routine. When the check is successful and the routine has been entered, data is input into the DSP and placed in the lower 16 bits of the accumulator. Note the difference in architecture here. The C50 has an accumulator register dedicated to that purpose alone, while the C30 can use any general-purpose register with the addition instructions to effectively implement an accumulator. The SACL instruction then takes this data and stores it into RAM in a predefined circular buffer. Again, an architectural difference between the DSP chips is apparent in that the incoming data in the C50 must be routed through the accumulator before being placed in the RAM circular buffer. The C30, on the other hand, can use a parallel instruction along with any general-purpose register to increase efficiency.

Like the C30, the C50 has a hardware repeat-next-instruction feature that is used to maintain single-cycle execution of the time-critical sum-of-products calculation. In the C50, this is accomplished with the MAC instruction. The MAC mnemonic is short for multiply/accumulate, and the instruction allows the output of the multiplier to be routed directly into the accumulator in the same machine cycle. The two inputs are defined in the instruction to be COEFF, which is my label for the start of the filter coefficient data table in RAM, and the data circular buffer that is pointed to by the value contained in register AR6. This instruction is specially designed for repeating: once the starting points in memory for the two operands are loaded, they are automatically in-cremented to the next value by special addressing hardware. The plus (+) sign in the instruction indicates that the starting value of the circular buffer is advanced one position for the next run through the routine. As was the case in the C30, the multiply/accumulate operation is registered (or pipelined) so that the value in the accumulator is actually one cycle behind what the multiplier is presently calculating. Again, a final accumulator cycle needs to be performed. The C50 has a special instruction to do this, APAC, which automatically uses the correct operands for the update.

In the C30 filter coding example, the fact that floating-point arithmetic is used in the sum-of-products calculation means that the answer appeared in a DSP register needing only a conversion instruction (FIX) before storing the integer result in the AIC DAC. In the C50, things are not quite as simple due to the fact that the architecture is based on 16-bit fixed-point words. Notice that in this example, no data conversion is necessary during data input because we are reading a two's complement integer from the ADC. Likewise, no conversion will be necessary at the output to the AIC DAC once the lower 2 bits are masked to zero. The price we pay is that we must keep track of the radix point in the accumulator. This is fairly simple in this application, but more sophisticated algorithms can result in this becoming very complicated!

The fixed-point multiplier in the C50 multiplies two

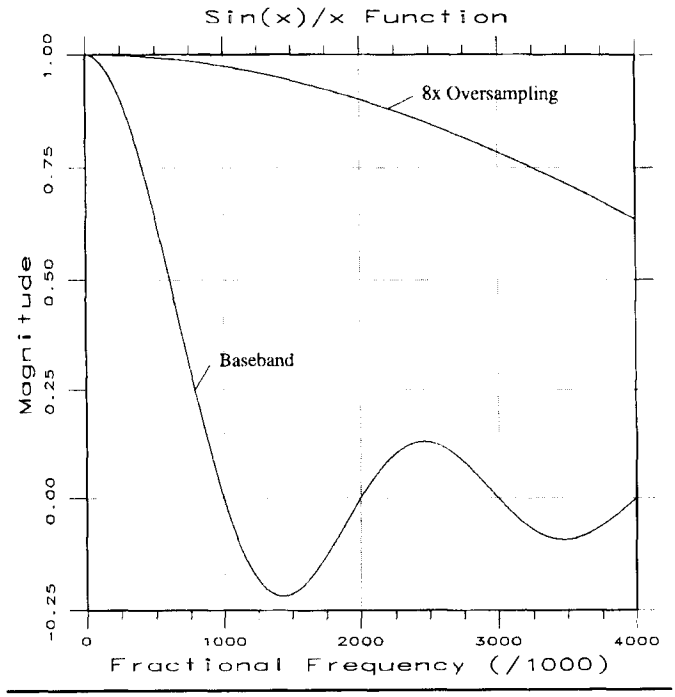


Fig 13—Output signal reconstruction by a DAC causes the signal amplitude to vary with frequency, as shown here. To aid in plotting, frequencies were multiplied by 1000. Thus, the 1000 point on the *x* axis represents the sampling frequency. Note that the baseband curve shows attenuation to about 0.6 of the dc value at the Nyquist frequency, while the 8× oversampling curve shows almost no attenuation at that point.

16-bit words together, producing a 32-bit result. If the two input words are unsigned, the output is also unsigned. In this example, the input data from the AIC ADC is 16-bit two's complement words (14-bit resolution, with the lower two bits set to zero) and the stored FIR filter coefficients are 16-bit two's com-plement words as well. Because of the way binary numbers are multiplied together, the output of a multiplication of two two's complement numbers is another two's complement number, but with an additional sign bit on the left. The accumulator in the C50 can be set up to automatically compensate for this extra sign bit by shifting the word left one bit. Again, hardware comes in handy because without this an extra instruction it would be necessary to use an extra instruction to shift the accumulator value by one bit.

After the sum-of-products calculation is compete, the answer is contained in the accumulator as a two's complement 32-bit number. Since we are only interested in transferring 16 bits to the AIC, the obvious thing to do is to write the upper half of the accumulator out since it is this half that contains the most significant bits of data, as well as the sign bit. The C50 provides instructions to *store accumulator high word* (SACH, used in this code) and *store accumulator low word* (SACL). Since the lower half of the accumulator represents extended-resolution bits in this filtering example, they can effectively be ignored. As with the C30 example, the lower two bits of the output word are masked to zero for compatibility with the AIC data format.

The second part of the assembly routine is similar to the one used for the C30 code, except that the data is not in floating-point format but in 16-bit hex format instead. This conversion is accomplished with a function in the C program, and rounding is provided for greater accuracy in the results. Again, changing these values along with the value of TAPS (if required) will change the filter performance after reassembly.

Comparison of DSP Functionality

Several things are apparent when comparing the two assembly language routines on the C30 and the C50. Special-purpose hardware is designed into both DSP chips to help make typical signal processing operations more efficient—like circular buffering, sum-of-product calculations that can be performed in consecutive cycles by utilizing repeated multiply-addition type operations and delayed branch instructions that can be used to efficiently handle both unconditional and conditional program branches. Because the examples given in this article are somewhat limited in scope, other DSP-specific hardware features were not used. One such major feature is automatic bit-reversal addressing capability, available on both the C30 and the C50. This feature helps reduce the programming overhead associated with the coding of fast Fourier transform (FFT) algorithms, which are used extensively in signal processing.

It is also fairly obvious that these DSP chips, even though they are designed and built by the same manufacturer, handle solving the same problems in different ways. The two coding examples show that floating-point arithmetic alleviates the need for worrying about accumulator overflow or scaling but necessitates a conversion to and from integer form for I/O purposes. Parallel instructions available on the C30 can help with overall efficiency and speed, at the expense of forcing the programmer to think in terms of pipelining, but not all algorithms may be able to take advantage of this fully. Because the C50 is dealing with an instruction word that is only 16 bits wide compared to 32 bits for the C30, it is more limited in its coding of instructions to support features such as indirect addressing. If the programmer tends to use indirect addressing frequently (as I do), the C50 instructions that are used solely to change the current address pointer will appear often. This is done in the C50 example twice, with MAR instructions.

Because the assembly language instructions are different between DSP chips, and because DSP programs can become extremely difficult to design and maintain when they incorporate many functions in a real-time environment, many DSP developers will choose to develop their code in the C language. Most (if not all) commercially available DSP chips now have C compilers available that will generate assembly language code from the higher level language of C. C provides certain advantages over assembly code, such as portability, speed of development and maintainability, but inherently some inefficiencies will result in the compiled code. Because of this, developers will typically code their algorithms in C, then test the compiled code to make sure that real-time performance is met. If it is not, the problem module or modules can be identified and rewritten in optimized assembly language, then called via the main C program. As a matter of fact, C-language code development is so pervasive that its use is influencing the design of DSP chip architectures. The C30 was designed from the start to be efficiently coded with C, and this can be seen in its larger, more general-purpose register file among other things.

Final System Considerations

A potential pitfall of practical implementations of sampled systems such as the one here arises when a digital-to-analog converter is used to convert the DSP algorithm output back to an analog signal. Practical DACs implement a *zero-order sample-and-hold* on the output signal, which essentially clamps the output voltage at a particular analog value for the entire clocking period. An analog smoothing filter is then used to eliminate the resulting "stair-step" from the output waveform. This process results in an amplitude roll-off of the signal as frequency is increased, reaching 0 at the sampling frequency, as shown in Fig 13. Of course, we won't usually be interested in any output frequency above Nyquist (half the sampling frequency)—the analog smoothing filter should remove any such signals anyway. But at the Nyquist frequency, the output is down to about 0.6 of its dc value. This is quite a serious degradation of the system frequency response that we so carefully tried to design our filter for in the first place!

The theoretical explanation of this process is that the output signal from the processor, which is formed using *ideal sampling*, is convolved with a rectangular pulse, equal in duration to the time between clock pulses. The outcome of this convolution process can be visualized as a multiplication of the signal's frequency spectrum with the Fourier transform of a rectangle, a $\sin(x)/x$ function. This is the function plotted in Fig 13.

There are basically three solutions to this problem. The first is to use what is called a $\sin(x)/x$ compensation filter, which essentially is an analog peaking filter at the output, designed such that the overall output is flat across the frequency spectrum. This is the technique that is used in the AIC provided on the two EVM boards that I have used. In this case, the programmer need only worry about initializing the AIC correctly so that the compensation filter is used. The second solution,

which can be used if the available DAC does not have this built-in compensation filter, is to program the compensation into the DSP code itself. Texas Instruments has detailed a method for designing a first-order digital filter that can be programmed in about 7 instructions per sampling interval, for very little overhead.[6] The third solution is to use an oversampling DAC. A typical oversampling rate for commercial DACs is 8 times, and a plot of how the $\sin(x)/x$ function for this case compares to the baseband case is included in Fig 13. In this case, very little distortion of the frequency response is introduced by the DAC.

Summary

This article has shown a complete algorithm-to-assembly code design example for developing real-time FIR filters on DSP chips. The basic principles demonstrated can be extended to many different algorithms and applications. Because of the widespread availability of personal computers and reasonably priced software packages, such as C compilers, it is fairly easy to develop an algorithm simulation environment to test performance prior to implementation. Even if it is not desired to experiment at an algorithmic level, you can go right to the implementation phase by using design software available with textbooks and from other commercial sources. To go along with the personal computer, several DSP chip manufacturers are now offering low-priced DSP starter kits that give the home experimenter the ability to generate and download assembly language programs from the PC for around $100. Because these boards generally have analog I/O interfaces included on them, they are a natural fit for amateur-radio experimentation.

FIR filters implemented on DSP chips can provide dramatic results in amateur-radio applications and these algorithms can be the starting point for the design and implementation of more sophisticated functions, such as IIR filters and adaptive noise reduction. As prices go down and processing power goes up, DSP chips are starting to work their way up in the radio architecture from predominantly audio processing to performing IF processing functions such as demodulation. What better time could there be to start your own DSP experiments?

About the Author

John Wiseman, KE3QG, was first licensed ten years ago as KA5WTO and has been an active shortwave and AM radio listener for more than 25 years. He enjoys working HF SSB and CW, and experimenting with his homebrew transceiver. John has worked with high-performance DSP chips for several years and has published several papers on DSP applications. When he finds the spare time, John enjoys implementing various DSP algorithms in his homebrew radio. He holds a BSEE from the University of Massachusetts and an MSEE from the University of New Mexico. John is currently employed by Hitachi America in Princeton, New Jersey, as a Senior Researcher designing hardware for advanced video systems.

Notes

[1]Parks, T. W. and Burrus, C. S., *Digital Filter Design*, John Wiley & Sons, 1987, ISBN 0-471-82896-3.

[2]Mitra, S. K. and Kaiser, J. F., Editors, *Handbook for Digital Signal Processing*, John Wiley & Sons, 1993, ISBN 0-471-61995-7.

[3]Alkin, O., *Digital Signal Processing—A Laboratory Approach Using PC-DSP*, Prentice Hall, 1994, ISBN 0-13-328139-6.

[4]Thede, L., *Analog and Digital Filter Design Using C*, Prentice Hall, 1996. ISBN 0-13-352627-5.

[5]Embree, P. M., *C Algorithms for Real-Time DSP*, Prentice Hall, 1995, ISBN 0-13-337353-3.

[6]*Linear Circuits Data Book—Volume 2*, "TLC32046 Wide-Band Analog Interface Circuit," Texas Instruments, 1992.

Listing 1

```
/* This C program calculates FIR filter coefficients for lowpass */
/* and bandpass configurations.  Highpass filters may be designed*/
/* by modifying the function Coefficient() as shown.  Floating   */
/* point values may be used directly on the C30 DSP, and a       */
/* conversion routine is given to allow hex output for the C50.  */
/* This program compiles on a Sun workstation, and previous      */
/* versions have been compiled on a PC with a Borland C compiler.*/
/* File names may have to be changed for your machine...         */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX 4096

void Conversion(void);
void Coefficient(void);
void Window(void);
void Symmetric(void);
void FreqResponse(void);
void FreqRespons_16bit(void);
void ArrayMultiply(double WindowWt);

int m, l, n2, i, j, TapNumber, FreqRespPoints, WindowType;
double q, q1, q2, am;
double pi;
double LPCoeffs[MAX], HPCoeffs[MAX], BPCoeffs[MAX], Coeffs_16bit[MAX];

/*****************************************************************/
/*                                                               */
/* The function Coefficient() calculates the tap values for      */
/* specified filter parameters.  The Fourier series method is    */
/* used in this function.                                        */
/*                                                               */
/*****************************************************************/

void Coefficient(void)
{

float LowerFC, UpperFC;

printf("\nLS FIR Bandpass Filter Design With Windows\n");
printf("\nEnter:  # of Taps, LowerFC, UpperFC, Window Type, # of Freq
Response Points\n\n");
scanf("%d%f%f%d%d", &TapNumber, &LowerFC, &UpperFC, &WindowType,
&FreqRespPoints);

n2=TapNumber/2;
m=(TapNumber+1)/2;
l=TapNumber-1;
pi=3.141592654;
am=(TapNumber+1.0)/2.0;

/* Calculate the FIR filter coefficients */

if (m==am)
{
        LPCoeffs[m-1]=2.0*UpperFC;
        HPCoeffs[m-1]=((2.0*LowerFC)+((0.5-(2.0*LowerFC))*2.0));
        BPCoeffs[m-1]=(LPCoeffs[m-1]+HPCoeffs[m-1])-1.0;
}

for (j=0; j<n2; ++j)
{
        q=pi*((j+1)-am);
        LPCoeffs[j]=(sin(UpperFC*2.0*q))/q;
        HPCoeffs[j]=(-(sin(LowerFC*2.0*q)))/q;

/* HPCoeffs[j]=(pow(-1.0,j))*(-(sin(LowerFC*2.0*q))/q);     */
/* Uncommenting this line and commenting out the previous one   */
/* will allow highpass filter coefficient design.               */

        BPCoeffs[j]=LPCoeffs[j]+HPCoeffs[j];

}
return;

}
```

```
/*************************************************************/
/*                                                           */
/* The function Window() calculates the window values for    */
/* triangular, Hanning, Hamming, Blackman, or rectangular    */
/* (no window - default) windows.                            */
/*                                                           */
/*************************************************************/

void Window(void)
{

double WindowWt;

q=pi/am;
q1=pi/(am-1.0);

switch (WindowType)
{

/* Multiply by a triangular window */

case 1:
       for (j=0; j<am; ++j)
       {
       WindowWt=(j+1)/am;
       ArrayMultiply(WindowWt);
       }
break;

/* Multiply by a Hanning window */

case 2:
       for (j=0; j<am; ++j)
       {
       WindowWt=0.5-(0.5*cos((j+1)*q));
       ArrayMultiply(WindowWt);
       }
break;

/* Multiply by a Hamming window */

case 3:
       for (j=0; j<am; ++j)
       {
       WindowWt=0.54-(0.46*cos(j*q1));
       ArrayMultiply(WindowWt);
       }
break;

/* Multiply by a Blackman window */

case 4:
       for (j=0; j<am; ++j)
       {
       WindowWt=0.42-(0.5*cos((j+1)*q))+(0.08*cos(2*(j+1)*q));
       ArrayMultiply(WindowWt);
       }
break;
}

return;
}

/*************************************************************/
/*                                                           */
/* The function Symmetric() calculates the symmetric portion */
/* of the filter coefficients, then writes these floating    */
/* point values to the file "bpfilter.txt".  These same values */
/* are written to the file "bpfilter.ftp", but each value is */
/* preceeded with the assembler directive .word for easy     */
/* inclusion in the C30 assembly language code.              */
/*                                                           */
/*************************************************************/

void Symmetric(void)
{

FILE *f;
FILE *ff;

f=fopen("bpfilter.ftp", "w");
ff=fopen("bpfilter.txt", "w");

/* Equate symmetric coefficients */

for (j=0; j<((TapNumber-1)/2); l-=2, ++j)
{
       BPCoeffs[j+1]=BPCoeffs[j];
}

/* Print coefficients */

printf("\nThe calculated FIR coefficients are:\n\n");
printf("       BP coefficients\n");
printf("           Decimal\n\n");

for (j=0; j<TapNumber; ++j)
{
       printf("%18f\n",BPCoeffs[j]);
       fprintf(f, "         .word  %15.10f\n",BPCoeffs[j]);
       fprintf(ff, "%15.10f\n", BPCoeffs[j]);
}

fclose(f);
fclose(ff);

return;
}
```

```
/*************************************************************/
/*                                                           */
/* The function FreqResponse() calculates the frequency      */
/* response of the calculated FIR filter coefficients for    */
/* verification purposes.  The array a[j] is the frequency   */
/* response normalized to one, while the array c[j] is the   */
/* response in dB.  These values are printed on the screen,  */
/* and c[j] is written to the file "freqresp.txt" for usage  */
/* with plot programs.                                       */
/*                                                           */
/*************************************************************/

void FreqResponse(void)
{

double bt;
double a[MAX], c[MAX];

FILE *f;

f=fopen("freqresp.txt", "w");

/* Calculate the frequency response */

printf("\nThe calculated frequency response is:\n\n");
printf("          Magnitude                Decibels\n\n");

q2=pi/FreqRespPoints;

for (j=0; j<FreqRespPoints; ++j)
{
       if (am==m)
       {
       bt=0.5*BPCoeffs[m-1];
       }

       for (i=0; i<n2; ++i)
       {
       bt=bt+(BPCoeffs[i]*cos(q2*(am-(i+1))*j));
       }

a[j]=2*bt;

/* Calculate frequency response in decibels */

c[j]=20.0*(log10(fabs(a[j])));
printf("%18f%25f\n", a[j],c[j]);
fprintf(f, "%10f\n", c[j]);

}

fclose(f);

return;
}

/*************************************************************/
/*                                                           */
/* The function Conversion() converts the floating point FIR */
/* filter coefficients to 16-bit hex values (2's complement). */
/* These values are rounded to the nearest bit, then stored  */
/* in the file "bpfilter.hex".  Each coefficient is preceeded */
/* by the assembler directive .word for easy usage in C50    */
/* assembly code.                                            */
/*                                                           */
/*************************************************************/

void Conversion(void)
{

long int hexnum;

FILE *ff;

ff= fopen("bpfilter.hex", "w");

for (j=0; j<TapNumber; ++j)
{

hexnum=(long int)(BPCoeffs[j]*(float)524288.0);
if ((hexnum & 15) >= 8)
{
       hexnum += 16;
}

       hexnum = ((hexnum << 12) & 4294901760);
       hexnum = (hexnum >> 16) &65535;
       printf("%04x\n", hexnum);
       fprintf(ff, "         .word  %04xh\n", hexnum);

       Coeffs_16bit[j] = hexnum;

}

fclose(ff);

return;
}

/*************************************************************/
/*                                                           */
/* The function FreqResponse_16bit() calculates the FIR filter */
/* frequency response for 16-bit resolution coefficients, for */
/* comparison with the "ideal" double precision floating point */
/* coefficients calculated elswhere.                         */
/*                                                           */
/*************************************************************/
```

```
void FreqResponse_16bit(void)
{

double bt;
double a[MAX], c[MAX];

FILE *f;

f=fopen("freqresp_16bit.txt", "w");

/* Calculate the frequency response */

printf("\nThe calculated frequency response is:\n\n");
printf("          Magnitude                Decibels\n\n");

q2=pi/FreqRespPoints;

for (j=0; j<FreqRespPoints; ++j)
{
        if(Coeffs_16bit[j] >= 32768)
        {
          Coeffs_16bit[j] = (double)(Coeffs_16bit[j]/(float)32768.0);
          Coeffs_16bit[j] -= 2.0;
        }

        else
        {
          Coeffs_16bit[j] = (double)(Coeffs_16bit[j]/(float)32768.0);
        }
}

for (j=0; j<FreqRespPoints; ++j)
{
        if (am==m)
        {
          bt=0.5*Coeffs_16bit[m-1];
        }

        for (i=0; i<n2; ++i)
        {
          bt=bt+(Coeffs_16bit[i]*cos(q2*(am-(i+1))*j));
        }

a[j]=2*bt;

/* Calculate frequency response in decibels */

c[j]=20.0*(log10(fabs(a[j])));
printf("%18f%25f\n", a[j],c[j]);
fprintf(f, "%10f\n", c[j]);

}

fclose(f);

return;
}

/*************************************************************/
/*                                                           */
/* The function ArrayMultiply() multiplies the FIR filter    */
/* coefficients with the appropriate window values on a pair-*/
/* wise basis.                                               */
/*                                                           */
/*************************************************************/

void ArrayMultiply(double WindowWt)
{

BPCoeffs[j] *= WindowWt;

return;
}
/*************************************************************/

main()
{

Coefficient();
Window();
Symmetric();
FreqResponse();
Conversion();
FreqResponse_16bit();

return(0);

}

/*************************************************************/
```

Estimating T-Network Losses at 80 and 160 Meters

T-network tuners are common, but are they good? You may be surprised.

By Kevin Schmidt, W9CF

T-network tuners are popular for matching antennas for 160 through 10 meters. Recent articles by Frank Witt, AI1H, and Andrew Griffith, W4ULD, addressed how to measure tuner loss for various resistive loads, gave some example calculations and measurements and showed how to adjust the tuner for minimum loss.[1,2] Measuring your tuner's loss is the only sure way to know, but by examining the worst case losses at various standing wave ratios, some useful simplifications and estimates of tuner losses can be made. Simple computer programs can accurately analyze your tuner once you know the component values, and all the numerical results here can be obtained in this way, but often a "back of the envelope" calculation can give additional insight. You can, for example, look at a hamfest tuner, or read the advertised specifications of a tuner, and easily make an educated guess of the sort of power loss that you can expect on 80 and 160 meters.

[1]Notes appear on page 20.

6510 S Roosevelt St
Tempe, AZ 85283

Fig 1A gives the schematic diagram of a typical T-network tuner. Since most of the loss is in the coil, lossless capacitors are assumed, and coil loss is included by the equivalent parallel resistance QX_L. Fig 1B shows an equivalent circuit where the desired impedance R_0=50 Ω in series with the input capacitor C_1 is transformed into its equivalent parallel resistance and reactance. The output resistance R and reactance X, in series with the output capacitor C_2, are also transformed into their parallel equivalents.

The input impedance will be 50 Ω if the tuner elements are selected so that the parallel equivalent output resistance in parallel with the coil loss resistance gives the parallel equivalent input resistance:

$$\frac{R_0}{R_0^2 + X_{C1}^2} = \frac{R}{R^2 + (X_{C2} + X)^2} + \frac{1}{QX_L}$$

Eq 1

and if the parallel reactances are tuned to resonance:

$$0 = \frac{X_{C1}}{R_0^2 + X_{C1}^2} + \frac{X_{C2} + X}{R^2 + (X_{C2} + X)^2} + \frac{1}{X_L}$$

Eq 2

The usual T network has C_1, C_2 and L all variable. Since there are only two matching equations, many combina-

tions will provide a match. The optimum combination is the one that minimizes power loss. The fractional power loss is the ratio of parallel equivalent input resistance to the the parallel coil resistance.

Notice that the parallel equivalent source and load resistances are larger than either of the original source or load resistances. In other words, any T network will transform the load resistance to a higher value that must also be higher than 50 Ω. It then transforms this high value down to 50 Ω to produce a match. Typical examples at 80 m would be a 10-Ω load resistance transformed to a 4-kΩ parallel equivalent, which is transformed back to 50 Ω, while a 100-Ω load resistance might be transformed to 1 kΩ before being transformed back to 50 Ω. The loss mechanism is now easier to see. For these typical cases, the coil reactance will be around a few hundred ohms. The parallel equivalent coil resistance for a coil of Q=100 would be 10 or 20 kΩ. This resistance is enormous compared to 50 Ω, and initially you might be tempted to ignore it, but it is placed across a point in the circuit where the impedance is transformed to a few kΩ. This means that the loss would be on the order of 10%, hardly negligible if a 1500-W transmitter is used—unless your coil is designed to dissipate 150 W.

As discussed by Griffith (Note 2), a typical T network designed for 160 through 10 m has compromises. One of these is that the capacitors typically have a maximum value of 200 to 300 pF. Another is that L is usually a roller inductor. Tom Rauch, W8JI, who has investigated the Q of some roller inductors, tells me that a rough estimate of the Q of high-quality, off-the-shelf, commercial roller inductors would range from a low of around 20 at low values of inductance, up to a maximum Q around 100. Custom roller inductors can have a higher Q. Since 80 and 160 m are the lower frequency limits of these tuners, and the antennas to be tuned there are often relatively short, or far from resonance, losses for these bands are an important concern. Problems also occur at the high frequency limit of these tuners with large stray reactances, minimum component values and low coil Q. Here I will concentrate only on performance at lower frequencies.

Maximum loss occurs with low-impedance loads. In Fig 2 I show a Smith chart where I have plotted points joined by straight lines to approximate contours of constant loss. The inner group of points corresponds to a loss of 0.3 dB, the middle group 0.5 dB and the outer group 1.0 dB of loss. Because the contours are centered toward the right hand side of the

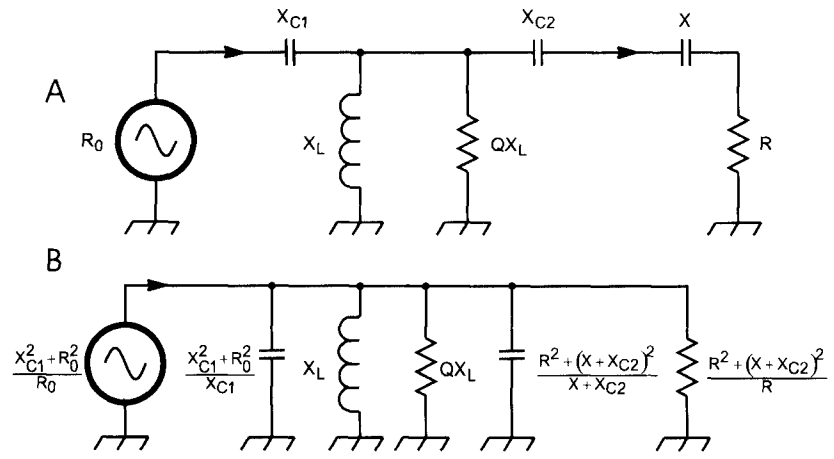


Fig 1—A typical T network connected between a source designed for a termination of R_0 and a load impedance consisting of a resistance R and a reactance X. The coil loss is shown as a parallel equivalent resistance (A). An equivalent circuit for the T network where all elements have been transformed into parallel equivalents (B). Matching requires the load and source parallel equivalent resistances to be equal and all the parallel equivalent reactors to resonate.

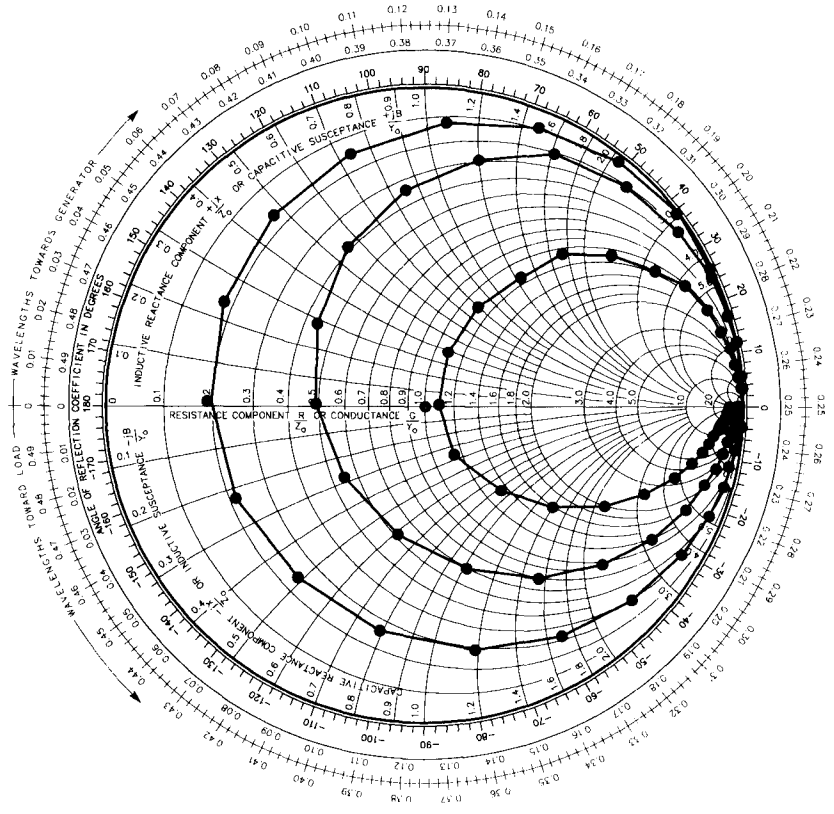


Fig 2—A Smith chart, normalized to 50 Ω, showing points of constant loss for a T network with 250-pF capacitors and a coil Q of 100. The inner set of dots are at points with 0.3 dB of loss. The points have been joined by lines to guide the eye. The outer set of points corresponds to 1 dB loss and the middle set corresponds to 0.5 dB of loss.

chart, which corresponds to higher impedances, lower losses tend to occur at higher impedances and higher losses at lower impedances for a given SWR. The contours are shifted toward the top of the chart, which indicates somewhat lower losses for inductive rather than capacitive loads. Table 1 shows the loss at 3.7 MHz for a tuner with capacitors with a maximum value of 250 pF, coil Qs of 50, 100 and 200, and purely resistive loads. The tuner is adjusted to give the least loss.

Loads with significant reactance also can be matched with a T network. Table 2 shows the worst-case loss and the load that causes the maximum loss in the network for an SWR of s, calculated by a straightforward numerical search on a computer. The load shown is that for a Q of 100. Detailed analytic calculations show that the worst-case loss for SWR greater than about 2 occurs at an impedance that is slightly capacitive, and is given approximately by:

$$R = \frac{R_0}{s}$$

$$X = \frac{R_0^2}{2X_{C2}} \qquad \text{Eq 3}$$

which agrees with the load calculated numerically and shown in Table 2.

A comparison of Tables 1 and 2 shows that while the maximum loss for a given SWR is at a slightly capacitive load, an excellent approximation to the worst case loss at a given SWR is given by calculating with a purely resistive load with:

$$R = \frac{R_0}{s} \qquad \text{Eq 4}$$

This numerical result is verified by analytical calculations that show that the additional loss for the reactive load over the purely resistive is given roughly by an additional factor of $R_0^2/\left(4X_{C2}^2\right)$, which changes the calculated loss by only a few percent.

The usefulness of these results is that the worst-case loss can be approximated simply. For typical capacitor values used and with these low-resistance loads, the magnitude of the capacitive reactances of C_1 and C_2 at 80 and 160 m is significantly larger than either R_0 or the load resistance R; a 250-pF capacitance corresponds to roughly 175 Ω at 80 m and 350 Ω at 160 m. If the loss is assumed small, Eqs 1 and 2 can be approximated by:

$$X_{C1}^2 = \frac{R_0}{R} X_{C2}^2 = sX_{C2}^2 \qquad \text{Eq 5}$$

and

$$\frac{1}{X_L} = -\frac{1}{X_{C1}} - \frac{1}{X_{C2}} \qquad \text{Eq 6}$$

The ratio of the power dissipated in the coil P_{loss} to the power input P is approximately:

$$\frac{P_{loss}}{P} = \frac{X_{C1}^2}{R_0 Q X_L} \qquad \text{Eq 7}$$

Using Eqs 5 and 6, this becomes:

$$\frac{P_{loss}}{P} = \frac{\left(s+\sqrt{s}\right)\left|X_{C2}\right|}{R_0 Q} \qquad \text{Eq 8}$$

This equation shows that the value of $|X_{C2}|$ should be minimized to minimize the loss. Therefore, the capacitors should be adjusted to have the largest value they can while achieving a match. For this low-impedance case, C_2 should be set to its maximum value. The loss in the T network in dB is:

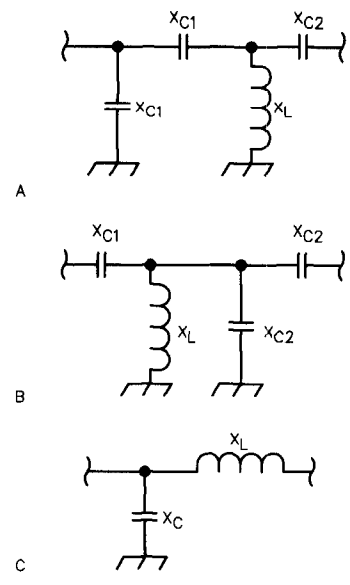


Fig 3—(A) The Ultimate transmatch circuit, (B) the SPC transmatch circuit and (C) an L network for matching low-resistance loads.

Table 1—Calculated loss in dB for a T-network tuner at 3.7 MHz using the full equivalent circuit of Fig 1B with input and output capacitances of 250 pF, with resistive loads, *R*, and coil Q shown.

R	*SWR*	*Loss (dB) Q=50*	*Loss(dB) Q=100*	*Loss(dB) Q=200*
1	50:1	7.47	4.99	3.08
2.5	20:1	4.62	2.79	1.57
5	10:1	3.00	1.69	0.91
10	5:1	1.85	1.00	0.52
25	2:1	0.95	0.49	0.25
50	1:1	0.62	0.31	0.15
100	2:1	0.53	0.26	0.13
250	5:1	0.43	0.21	0.10
500	10:1	0.37	0.18	0.08
1000	20:1	0.39	0.20	0.10
2500	50:1	0.61	0.31	0.15

Table 2—Calculated worst-case loss and corresponding load for a T-network tuner at 3.7 MHz using the full equivalent circuit of Fig 1B with input and output capacitances of 250 pF. The *R* and *X* values shown are for Q=100.

R	*X*	*SWR*	*Loss(dB) Q=50*	*Loss(dB) Q=100*	*Loss(dB) Q=200*
50	0	1:1	0.62	0.31	0.15
26	–7	2:1	0.97	0.50	0.25
10	–8	5:1	1.89	1.02	0.53
5	–8	10:1	3.05	1.72	0.93
2.5	–8	20:1	4.69	2.84	1.60
1	–8	50:1	7.56	5.06	3.13

$$L_{dB}^{T} = -10\log_{10}\left(1-\frac{P_{loss}}{P}\right) \qquad \text{Eq 9}$$

Since the approximate formula is only good at small values of loss, I can expand the logarithm without making the approximation worse, using:

$$\log_{10}(1-x) \approx -\frac{x}{\ln(10)} \qquad \text{Eq 10}$$

If the frequency f is given in MHz and the maximum capacitance of the capacitors is written as C_{max} and given in pF, then using R_0=50 Ω:

$$L_{dB}^{T} \approx 14{,}000\frac{s+\sqrt{s}}{C_{max}fQ} \qquad \text{Eq 11}$$

The results of Eq 11 are shown in Table 3 and can be compared with those of Table 2. For losses less than a dB or two, the agreement is good.

Eq 11 allows us to estimate the worst possible loss that can occur with an output SWR of s. The loss can be a lot smaller; for example if the SWR is 5, but corresponds to a purely resistive load of 250 Ω, Eq 11 greatly overestimates the loss. However, if the SWR of 5 corresponds to a purely resistive load of 10 Ω, Eq 11 should fairly accurately predict the loss.[3]

Only the product of the Q value of the coil and the maximum value of the output capacitor needs to be measured or estimated to use Eq 11. The maximum value of the capacitor is often given in the tuner specifications. If not, it can be easily measured or estimated from handbook formulas from the size, spacing and number of plates. The coil Q can be guessed, or, for a more accurate estimate, measured using an RF bridge or Q meter. Alternatively, measuring the loss for a 50-Ω load and then applying Eq 11 will give a value of QC_{max} at the measurement frequency. This matched loss can be measured by matching a 50-Ω dummy load with your tuner and using a power meter to measure the input and output powers. Once QC_{max} is known, it can be used to calculate the worst case loss at other SWR values.

Another popular tuner uses a differential T network. In this network the capacitors C_1 and C_2 are ganged together so that their values sum to approximately C_{max}. The worst-case loss can be calculated as before and is:

$$L_{dB}^{diffT} \approx 14{,}000\frac{\left(1+\sqrt{s}\right)^2}{C_{max}fQ} \qquad \text{Eq 12}$$

The worst-case loss of the differential T network is a factor of 2 worse at an SWR of 1, but becomes the same as the standard T for large values of SWR. This disadvantage is offset by the convenience of having only two components to adjust, and by the fact that one source of operator error is eliminated since a really bad set of component values cannot be chosen. This is unlike the standard network where the operator can set the components to values that produce a match but greatly increase losses.

Other T-type configurations can be examined. The Ultimate and SPC transmatches are shown in Fig 3A and 3B respectively.[4,5] In these, one of the capacitors in the network is replaced with a two-section variable. For the Ultimate transmatch, at 80 or 160 m, the reactance of the capacitor across the input is significantly larger than 50 Ω, so it has little effect. You can simply ignore it in the loss analysis here; the extra section just increases the cost of the transmatch without improving it. The SPC transmatch has the second section of the output capacitor connected across the coil. The worst-case loss of this circuit is always greater than the standard T network. The analysis above is easily extended by adding this additional capacitance across the coil. For the SPC network, the loss, when matching load resistances smaller than R_0, is given by:

$$L_{dB}^{SPC} \approx 14{,}000\frac{2s+\sqrt{s}}{C_{max}fQ} \qquad \text{Eq 13}$$

where C_{max} is the maximum capacitance of one section of the output capacitor for the SPC circuit. The loss is 50% more than a standard T network for an SWR of 1, increasing to double the loss as the output resistance drops. Unlike the differential T network, there do not appear to me to be any benefits from this circuit that offset this additional loss.

It is amusing to compare the T-network results with those of a simple L network designed to match a resistive low impedance load, as shown in Fig 3C. The result is:

$$L_{dB}^{L} = \frac{10}{\ln(10)}\frac{\sqrt{s-1}}{Q} \qquad \text{Eq 14}$$

For this resistive load, the simple L network is better by an overall factor of $|X_{C2}|/R_0$ which is about a factor of 7 for C_{max} of 250 pF at 160 m. In addition, the loss for a load of 50 Ω is zero (where the L and C values are both zero), and it increases less with SWR than for the T networks. The penalty is the limited matching range. An L-network tuner needs to be reconfigured to match a wide range of loads; this switching of components can get complicated.

The peak voltage across the output capacitor of a T network for these loads can also be calculated within these same approximations. Since the series capacitors' reactance is significantly

Table 3—Worst-case loss for a T-network tuner at 3.7 MHz with input and output capacitances of 250 pF, using Eq 11.

SWR	*Loss(dB) Q=50*	*Loss(dB) Q=100*	*Loss(dB) Q=200*
1:1	0.61	0.30	0.15
2:1	1.03	0.52	0.26
5:1	2.19	1.10	0.55
10:1	3.98	1.99	1.00
20:1	7.41	3.70	1.85
50:1	17.28	8.64	4.32

Table 4—Worst-case loss on 80 m calculated from Eq 11 and compared to the loss measured in Note 1 for the Heathkit SA-2040 tuner.

SWR	*Measured Loss (dB)*	*Calculated Loss (dB)*
1:1	0.6	0.6
2:1	0.8	0.8
4:1	1.2	1.3
8:1	2.6	2.4
16:1	4.1	4.5

larger than 50 Ω, the voltage across the source can be ignored to get an estimate of the peak voltage. The peak input current is $I = \sqrt{2P/R_0}$. The peak voltage across the input capacitor is therefore:

$$V = I|X_{C1}| = \sqrt{\frac{2P}{R_0}}|X_{C1}| \qquad \text{Eq 15}$$

Substituting as above for the value of X_{C1} in terms of C_{max} in pF, the frequency f in MHz and the SWR s gives the approximate peak voltage across the capacitors for the standard T network with R_0=50 Ω:

$$V = \frac{100{,}000}{\pi}\frac{\sqrt{P}}{fC_{max}}\sqrt{s} \qquad \text{Eq 16}$$

This equation also works for both the Ultimate transmatch and SPC tuner circuits. For the differential T network, the relationship between C_1 and C_2 changes the result to:

$$V = \frac{100{,}000}{\pi}\frac{\sqrt{P}}{fC_{max}}\left(1+\sqrt{s}\right) \qquad \text{Eq 17}$$

The power-loss equations can be compared to some measurements given in Frank Witt's article (Note 1). The only T network that he measured was a Heath SA-2040, which has the Ultimate transmatch configuration. This tuner does not cover 160 m. The input capacitor has a maximum value of 125 pF and the output capacitor has a maximum value of 170 pF. Because these values are not equal, a little care is needed. For a 50-Ω input and output impedance, the capacitor values must be equal. This means that the output capacitor can have a maximum value of 125 pF for a match. However, when the output resistance is reduced, Eq 5 can be applied to show that for loads below about $(125/170)^2 \times 50$ Ω, or about 27 Ω, the full 170-pF value of the output capacitor can be used. The measured loss with a 50-Ω load at 80 m was 13%, which corresponds to 0.6 dB. Converting this into a Q value using C_{max} of 125 pF gives a Q of approximately 100, a reasonable value. Using this Q value and a C_{max} value of 170 pF in Eq 11 gives the calculated values at other loads, shown in Table 4 along with the measured values. The values agree within about 10%. This level of agreement is partially fortuitous, but these results show that the simple "back of the envelope" calculations work.

Table 5—The "back of the envelope" formulas for the worst-case loss and maximum peak voltage as a function of the SWR *s* derived in the text. *f* is the frequency in MHz, C_{max} is the maximum value of the output capacitor in pF, Q is the coil Q and *P* is the power. The formulas for the ultimate transmatch are the same as for the standard T network.

Network	*Loss in dB*	*Peak Voltage*
Standard T	$14{,}000\frac{s+\sqrt{s}}{C_{max}fQ}$	$\frac{100{,}000}{\pi}\frac{\sqrt{P}}{fC_{max}}\sqrt{s}$
Differential T	$14{,}000\frac{\left(1+\sqrt{s}\right)^2}{C_{max}fQ}$	$\frac{100{,}000}{\pi}\frac{\sqrt{P}}{fC_{max}}\left(1+\sqrt{s}\right)$
SPC	$14{,}000\frac{2s+\sqrt{s}}{C_{max}fQ}$	$\frac{100{,}000}{\pi}\frac{\sqrt{P}}{fC_{max}}\sqrt{s}$

For convenience, I have gathered the approximate loss and peak voltage formulas in Table 5.

These results point out some fundamental problems in using a 160 through 10-m T-network tuner at 80 and especially 160 m. Even feeding a resistive 50-Ω load with a tuner with 250-pF capacitors and a coil Q of 100 gives a loss of about 0.6 dB. With a 1500-W transmitter, the coil will have to dissipate about 180 W. I doubt if many tuners can stand up to that. Increasing the SWR to 3 increases the worst-case loss to 1.3 dB, and the dissipation could increase to almost 400 W. Clearly, a real 1500-W 160-m T-network tuner needs to have significantly larger capacitors and a high-quality coil in a large cabinet to minimize loss.

I would like to thank Tom Rauch, W8JI, for reading a preliminary version of this work, for many helpful comments and for providing me with reasonable estimates of the Q of the coils in these networks.

About the Author

Kevin Schmidt, W9CF, was first licensed in 1966 as WA9THN. He received an AB degree from Washington University and a PhD from the University of Illinois. Kevin is an associate professor of physics and astronomy at Arizona State University, where he uses and develops methods to simulate the quantum mechanical behavior of many-particle systems using Monte Carlo techniques and high-performance computers.

Notes

[1]Witt, Frank, AI1H, "How to Evaluate Your Antenna Tuner," Part 1, *QST*, April 1995, p 30; Part 2, *QST*, May 1995, p 33.

[2]Griffith, Andrew, W4ULD, "Getting the Most Out of Your T-network Antenna Tuner," *QST*, January 1995, p 44.

[3]Analytic calculations show that the least loss with a purely resistive load will occur approximately when both capacitors are set to the maximum value and the load resistance is $R = X_C^2/R_0$, where X_C is the reactance of one of the capacitors. The minimum loss with a resistive load is approximately half that of the loss for a matched load of R_0=50 Ω if the coil Q remains the same. This result and Eq 11 give reasonable upper and lower bounds to the loss.

[4]McCoy, Lewis G., W1ICP, "The Ultimate Transmatch," *QST*, July 1970, p 24.

[5]Straw, R. D., Editor, *The ARRL Antenna Book*, 17th Edition,"A Transmatch for Balanced or Unbalanced Lines," p 25-8.

The Copper Wire Gauge for Electrical Technology

An explanation of the AWG wire gauge system, with some handy tricks for converting between gauge and size.

By Antonio L. Eguizabal, VE7FIF

Introduction

Insulated copper wire is used extensively in electrical and electronics applications. In amateur radio use, it is predominant. Copper is a metal with excellent thermal, electrical, mechanical and physical characteristics; unequaled in its cost-benefit properties. Hence its widespread use throughout the world of electrotechnology. Most amateur radio circuits use insulated circular-cross-section (round) copper wire. Presently, in Canada, the US and other countries, the round copper wire is based on the American Wire Gauge (AWG) system. In this article, the basis for this gauge is presented, together with some of its interesting properties.

151 West Osborne Road
North Vancouver, BC V7N 2P9
Canada

Sizing of Round Copper Wire

Circular-cross-section, bare copper wire can be sized in several ways. For smaller conductors (up to No. 4/0 or 0000 AWG) it follows the American Wire Gauge.[1] For larger conductors, the tendency is to use the cross-section area expressed in circular mils (the area corresponding to a circle of 0.001 inches, or 1 mil, in diameter). The AWG system uses inches and feet and has no direct equivalent in the metric system.

The metric world usually sizes round copper conductors by their cross-section area in square millimeters (mm^2) and sometimes uses a system based on the diameter in millimeters for small wires.[2] In the UK, round copper wire follows the Standard Wire Gauge (SWG)[3], which is similar to the AWG but is not always exactly equivalent, as shown in the wire tables of Note 2. To make matters more confusing, other systems are used for other materials, such as the Steel Wire Gauge, the Birmingham Wire Gauge, the Old English Wire Gauge, the old Paris Gauge, and so on. For a history of wire gauges, the reader can consult Note 4.

The AWG System

The American Wire Gauge was invented by J. R. Brown in 1857 and is also known as the Brown and Sharp gauge. It is the prevalent wire gauge in North America and other countries for solid, round, bare copper wire of diameter less than 0.46 inches. Its use

[1]Notes appear on page 23.

is a defacto standard for Canada and the US, being specified for electrical wiring using copper wire as regulated by the Canadian Electrical Code and the National Electrical Code in the United States.[5,6]

Together with other gauges such as the ones already mentioned, the AWG system has the property that its sizes represent approximately the successive steps in the process of wire drawing (pulling through a hard steel die of known diameter).

Its numbers are retrogressive—a larger number denoting a smaller wire—corresponding to the successive drawing operations. For example, No. 0 AWG could be the first pass and No. 1 AWG the second pass through a smaller diameter die. Actually, the AWG system starts at No. 0000 (or 4/0 AWG) and stops at No. 50 AWG. The gauge numbers obey a mathematical relation and are not arbitrarily chosen as it may appear at first glance.

The basis of the AWG is a mathematical law. I briefly mentioned the relation in a previous article, which is repeated here for completeness.[7] The gauge is specified by two diameters and the law that a given number of intermediate diameters are formed by a geometrical progression. Thus, the diameter of No. 0000 or 4/0 AWG is defined as 0.4600 inches and the diameter of No. 36 AWG is 0.005 inches (see Notes 1 and 2). There are 38 sizes between these two, hence the ratio of any diameter to the next diameter of a larger size is given by:

$$\sqrt[39]{\frac{0.4600}{0.0050}} = \sqrt[39]{92} = 1.1229322 \qquad \text{Eq 1}$$

This is called the *progression constant* of the AWG system. It can be verified with a calculator by taking the logarithm (any base will do) of 92, dividing by 39 and then taking the antilog (same base) of the quotient. A good calculator will display as many digits as shown here—or more. My HP-41CV shows 1.122 932 197 when FIX is set at the maximum of 9 digits.

This number has some interesting properties:

a) The *square* of the progression constant is $(1.1229322)^2=1.2610$.

b) The *sixth power*, that is the ratio of any diameter to the diameter of the sixth greater number is: $(1.1229322)^6=2.0050$.

c) The fact that b) is so *close to the number 2*, is the basis of numerous useful relations and computation short cuts.

From the above, the following are *approximate rules* which apply to the AWG system and they are easy to remember:

1. An *increase of three gauge* numbers (say from No. 21 AWG to No. 18 AWG) *doubles the area* and weight, which consequently cuts in half the dc resistance.

2. An *increase of six gauge* numbers (say from No. 18 AWG to No. 12 AWG) *doubles the diameter.*

3. An *increase in ten gauge* numbers (say from No. 12 AWG to No. 2 AWG) *increases the area and weight by ten* and this reduces the dc resistance by ten times.

4. A *No. 10 AWG* round copper wire has *a diameter of about 0.10 inches*, an area of about ten thousand circular mils and a dc resistance of approximately 1 Ω per 1000 feet (standard annealed copper at 20° C).

5. The *weight of No. 2 AWG* copper wire is approximately *200 pounds per 1000 feet.*

6. The *diameter of No. 12* AWG is very close to *2 mm* (exactly 2.05 mm).

7. The *diameter of No. 18 AWG* is very close to *1 mm* (exactly 1.02 mm).

8. The *diameter of No. 24 AWG* is very close to *0.5 mm* (exactly 0.511 mm).

9. The *diameter of No. 30 AWG* is very close to *0.25 mm* (exactly 0.254 mm).

These approximate rules are easy to apply when winding coils, transformers or inductors for ham radio use. No consideration is given here to the insulation thickness, as this varies from manufacturer to manufacturer in the case of enamelled wire (ie, Formvar, a good thin insulating material). The insulating layer thickness also varies with the size of the wire. For demanding applications, the supplier's technical data should be carefully checked. For example, see Note 8.

An Additional Thought

For those interested in obtaining the diameter of a round copper wire given its AWG number, without the use of tables or reference handbooks (Notes 2 and 3), I have written a short program for the popular HP-41 CV programmable calculator. Using a computer, I believe, is a bit of an overkill in this case. The actual program is shown in Table 1 and it is not to difficult to rewrite for other programmable

Table 1—HP-41CV Program for Converting AWG into mils and mm.

01 LBL ALPHA WIRE ALPHA	<program name>
02 LBL 01	<begins branch/loop>
03 ALPHA WIRE SIZE? ALPHA	
04 XEQ ALPHA PROMPT ALPHA	<asks for AWG number>
05 XEQ ALPHA INT ALPHA	<makes AWG integer>
06 STO 01	
07 51	
08 STO 02	
09 RCL 01	
10 RCL 02	
11 X ≤ Y	<checks if AWG≤50>
12 GTO 01	<if not start again>
13 92	<calc of prog const>
14 LN	
15 39	
16 ÷	
17 STO 03	<stores $1n\ k_{awg}$>
18 RCL 01	<calc geom progression>
19 CHS	
20 36	
21 +	
22 x	
23 e^x	
24 5	
25 x	<diameter in mils>
26 R/S	
27 0.0254	
28 x	<diameter in mm>
29 GTO ..	<end>

machines. A very small error may appear in the calculated diameter when compared to the wire tables in Note 2, as these numbers are rounded off to reduce the decimal digits.

The following relation, derived from Eq 1, is used to convert an AWG number to diameter:

$$d_N = 5\exp[36 - N]\,k_{awg} \text{ (mils)}$$

$$k_{awg} = \frac{\ln 92}{39} \qquad \text{Eq 2}$$

The above equation is used as the basis of the program shown in Table 1. You must load it first by setting the HP-41CV to "Prgm." Once loaded, it will stay in memory. To run this program, called "Wire," do the following:

a) Do XEQ ALPHA WIRE ALPHA. This invokes the WIRE program.

b) Press R/S: This will display a WIRE SIZE? prompt.

c) Enter the AWG number, then press R/S.

d) The display shows the diameter in mils. Press R/S again for mm.

e) Press RS to continue with another AWG computation.

Please note that for large wire sizes you can enter the numbers as:

No. 1/0 AWG: enter as "0"
No. 2/0 AWG: enter as "–1"
No. 3/0 AWG: enter as "–2"
No. 4/0 AWG: enter as "–3", etc.

Conclusion

The American Wire Gauge system has been briefly presented, with its basic properties and some easy rules to remember when building projects using round copper wire. A small error exists in the exact calculation of the diameter of the wire, as the insulation thickness of the enamel is ignored. This is a small error (about 0.001 inch or 1 mil) that does not greatly affect the practice of winding small coils, inductors or transformers for amateur radio use. The calculator program shown converts AWG numbers to diameters in mils and mm for any gauge of wire larger than or equal to No. 50 AWG.

About the Author

Antonio L. Eguizabal is a graduate of the University of Santiago with a BSc (Hons) and MASc from the University of British Columbia, both in Electrical Engineering. He was first licensed as CE3ACO in 1967. His interestes include analog, digital, RF circuits and telecommunication systems, as well as experimenting with novel antennas for professional and amateur work. Antonio is also a volunteer radio operator with the North and West Vancouver Emergency Program.

Notes

[1]Dwight, H.B., Professor at MIT, *Electrical Coils and Conductors, Their Characteristics and Theory*, Chapter 5, McGraw Hill 1945.

[2]Fink, D.G, and Beaty, H.W., *Standard Handbook for Electrical Engineers*, Chapter 4, Twelfth Edition, McGraw Hill, 1987.

[3]Staff of Siemens A.G., *Electrical Engineering Handbook*, Chapters 1 and 8, John Wiley and Sons, 1990.

[4]*NBS Handbook 100*, National Bureau of Standards, National Technical Information Service, US Department of Commerce, Washington, DC.

[5]*Canadian Electrical Code*, Section 2-116, Canadian Standards Association, 1990, Rexdale, Ontario.

[6]*National Electrical Code Handbook*, Article 110-6, National Fire Protection Association, 1990, Quincy, Massachussetts.

[7]Eguizabal, A.L., "Inductance of Solenoid Coils: A Radio Amateur View, Part II, *The Canadian Amateur*, February 1994, p 53.

[8]*Belden Wire and Cable Master Catalog*, Cooper Industries, 1989, Richmond, Indiana.

Upcoming Technical Conferences

Eastern VHF/UHF Conference

The 22nd Annual Eastern VHF/UHF Conference will be held August 23-25, 1996, at the Quality Inn and Conference Center, Vernon, Connecticut.

Friday will feature informal gatherings in the Hospitality Room. Registration, formal talks, VHF-SHF band sessions, banquet and a VHF-microwave trivia quiz are on Saturday's agenda. A VHF-UHF Swap 'n' Sell and antenna gain measuring will take place on Sunday.

Advanced registration (includes a copy of the *Proceedings*) is $20, $25 at the door. Sunday-only registration is $5. Extra *Proceedings* are $12 each. Banquet tickets are available by mail. To register (make checks payable to Eastern VHF/UHF Society) or for more information, contact Rae Bristol, K1LXD, 328 Mark Drive, Coventry, CT 06238; tel: 860-742-8650. For information on the Swap 'n' Sell, contact Mark Casey, N1LZV, 303 Main Street, Hampden, MA 01036; tel 413-566-2445.

For hotel reservation, contact Lori Torizer at 1-800-235-4667. The Quality Inn and Conference Center is located at 51 Hartford Turnpike (Route 83), Vernon, CT 06066.

1996 ARRL and TAPR Digital Communications Conference

The 15th ARRL and TAPR Digital Communications Conference will be held September 20-22, 1996, at the Quality Inn Seattle Airport in Seattle, Washington (minutes from SeaTac airport).

This year's co-hosts are the Puget Sound Amateur Radio TCP/IP Group and Boeing Employees Amateur Radio Society (BEARS).

The ARRL and TAPR Digital Communications Conference is an international forum for radio amateurs in digital communications, networking, and related technologies to meet, publish their work, and present new ideas and techniques for discussion. Presenters and attendees will have the opportunity to exchange ideas and learn about recent hardware and software advances, theories, experimental results and practical applications. The Digital Communications Conference is not just for the digital expert, but for digitally-orientated amateurs of all levels of experience.

This year's conference will again provide an entire morning with beginning and intermediate presentations on selected topics in digital communications. Some of the topics will include: APRS, Satellite Communications, TCP/IP, Digital Radio, Spread Spectrum and other introductory topics. Come to the conference and hear these topics presented by the experts!

In addition to the presentation of papers on Friday and Saturday, three workshops will be held during the conference. On Friday, Keith Sproul, WU2Z, will hold a workshop on APRS packet-location software. Keith is the Chair of the TAPR APRS Special Interest Group, developer of the Macintosh and more recent co-developer of the Windows95 version of APRS, and a leader in the area of APRS technology. This is a unique opportunity to gain insight into this fast growing new digital aspect of amateur operations that combines computers, packet radio and GPS (Global Positioning Satellites). On Sunday, Dewayne Hendricks, WA8DZP, will conduct a workshop focusing on "How to utilize Part 15 wireless Radios for Ham Applications." Dewayne is an expert in the area of commercial wireless systems; his company WarpSpeed Imagineering, focuses on wireless Internet connectivity. This workshop presents an opportunity to learn how Personal Communications Technology (handheld and small business wireless systems) can be used in the amateur service. A second Sunday workshop will focus on Wireless Networking using the WA4DSY 56K RF modem Technology. This workshop will focus on the technology and accessories of creating and maintaining 56K networks using the WA4DSY modem and equipment compatible with it such as routers, digital driver cards, transverters and repeaters. Use of WA4DSY 56K equipment in the 219-220 band will also be discussed.

Full information on the conference and hotel information can be obtained by contacting Tucson Amateur Packet Radio, 8987-309 E. Tanque Verde Road #337, Tucson, AZ 85749-9399. Phone: 817-383-0000. Fax: 817-566-2544. Internet: tapr@tapr.org. Web: **http://www.tapr.org**.

Microwave Update 1996

Microwave Update 1996 will be held October 4-6, 1996, at the Ramada Camelback Hotel in Phoenix, Arizona.

For those arriving Thursday, October 3, there will be tours of the Phoenix area electronic surplus stores. Tours will begin at 9:30 AM and continue at various times throughout the day. An X-Band EME demonstration is planned for Thursday night at WA7CJO's QTH.

The technical program will be devoted to frequencies above 902 MHz and will include microwave test equipment, low-noise amplifiers, TWT power amplifiers and other pertinent microwave topics. In addition to the predominant "Who's Who in North America Microwave" speakers, we currently have speakers scheduled from Japan and Europe as well.

We will have a microwave flea market on both Friday and Saturday night. Noise figure measurements are also planned. There will be a microwave equipment auction held on Saturday afternoon that promises to have some high-power TWT ampliferes along with other useful "junk."

The Ramada Camelback Hotel offers free airport transportation. Room rates for the conference are $62 per night for singles, $72 for doubles. For reservations call 800-688-2021 or 602-264-9290; fax 602-264-3068.

Conference resistration is $40 prior to September 22 and $45 at the door. To register, or for more information, contact Jim Vogler, WA7CJO, 2540 E. Heatherbrae Drive, Phoenix, AZ 85016. Tel/fax: 602-954-0541.

20th Annual Mid Atlantic States VHF Conference

The 20th Annual Mid Atlantic States VHF Conference will be held October 5, 1996, at the Horsham Days Inn in Horsham, Pennsylvania.

Sponsored by the Mt. Airy VHF Radio Club, the conference continues to present a wide variety of technical papers covering all aspects of 50 MHz through light frequencies. Talks on operating, propagation, construction and theory are among those requested. Speakers are encouraged to contact the conference chairman early.

Rooms may be reserved at the Days Inn by calling 215-674-2500. Mention "Packrats" or "Hamarama" for a discount. A dinner will be held on Saturday evening. On Sunday, Hamarama, our annual hamfest will be held at the Bucks County Drive-In a few miles north of Warrington, PA.

For more information contact: John Sorter, KB3XG, Conference Chairman, at 1214 N. Trooper Road, Norristown, PA 19403. Tel: 610-584-2489. Email: JohnKB3XG@aol.com.

1996 AMSAT-NA Space Symposium and Annual Meeting

The 1996 AMSAT-NA Space Symposium and Annual Meeting will be held November 8-10, 1996, in Tucson, Arizona.

You know those times when you've been too busy or thought "ho-hum" about a ham event, so your buddy went, and you "missed it"!? This event is going to be good.

All the Southwest charm of Tucson—clear, broad, blue skies, at a season when our temperatures are the most comfortable. Friendly Southwest people and experiences.

Meet and share with hams from around the world.

Discuss and learn about the latest ham radio satellites under development.

Participate in Satellite Beginners' forums or immerse yourself in the intricate details of Amateur Space Technology.

Enjoy demonstrations of the latest satellite ground stations or take notes on how to put together the absolutely cheapest satellite ham station.

Bring your family; have them enjoy the tour to the Kitt Peak Radio Telescope with you. Or see them in the evening after they have spent the day enjoying the tourist sites in the area.

Plan to *not* to miss it. For information call Heather Johnson, N7DZU, tel: 520-749-5106, email: n7dzu@ azstarnet.com, or Larry Brown,NW7N, tel: 520-886-1957, email: nw7n@amsat.org. □□

Feedback

Our Apologies to Mr. Wien

Parker Cope's, W2GOM, March 1996 *QEX* article "Designing a Wein-Bridge Oscillator," has stirred up some correspondence.

The inventor's name is actually Wien. The misspelling of his name crept into several major reference books about thirty years ago and the error has become widespread.

The original Wien-bridge article, "Meassuring Inductance with the 'Optical Telephone'," (a somewhat primitive oscilloscope using a light beam reflected off a speaker) appeared in *Annalen der Physik* in 1891. □□