A DSP Version of Coherent-CW (CCW)

Obtain the advantages of coherent detection of Morse signals using DSP.

by Bill de Carle, VE2IQ

Coherent CW (CCW) is an old technique for digging weak CW signals out of the noise. The system synchronizes the receiver to the sender's keying. Thereafter, due to the rules of Morse code (each "dah" equals exactly three "dit-times;" and spaces between marking elements are always an integral multiple of one dit time) the receiver knows precisely when the carrier is allowed to turn on or off. This is useful information that would otherwise have to be transmitted—at the expense of additional power and/or bandwidth. Of course, the sender must have an absolutely rock-steady rhythm for the scheme to work. For this reason, CCW stations employ special keyers or keyboards which can generate perfectly timed code. Traditionally, CCW uses a keying rate of 12 WPM (100 milliseconds per dit) and a received audio tone of 800 Hz.

The receiver divides its time into 100-ms windows, or frames. Once synchronized, there is complete certainty that the incoming tone is either present or absent for the entire 100-ms window—the rules of Morse code prevent it from switching somewhere in the middle. This knowledge makes it possible to use a matched filter (sometimes called an *integrate-and-dump* filter) to reduce the receive bandwidth down to about 9 Hz (main lobe), eliminating most of the noise while letting the CW tone pass through unscathed (see Fig 1). All coherent 800-Hz energy received during the window is integrated (accumulated), and only at the very end of each window does the receiver make its decision—and the question is: "during the previous 100 milliseconds, was the key at the transmitter up or down?"

The question is answered by comparing the measured amplitude of the 800-Hz tone to some threshold value—more received energy means the key probably was down, less energy means it probably was up. If the receiver decides the key was down, it sounds a local sidetone oscillator for the next 100 ms. Apart from the 100-ms delay between the incoming audio and the regenerated tone, the sidetone signal follows the received code faithfully, and of course there is absolutely no electrical noise present from that point on since the signal has been completely rebuilt by the receiver. It is like a repeater on a fiber-optic cable. At each stage, hardware makes a firm decision based on a marginal situation then announces confidently whether the window was marking or spacing. Noise (at least in its common form) is completely removed from the incoming signal, so a human operator can copy it without stress. That is, as long as the hardware makes the right decision at the end of each window! If the signal-to-noise ratio is so awful that the matched filter gets a wrong answer, a different kind of noise enters the picture: the system still generates perfect code at its output, but it's no longer the same message the transmitter originated. Then the human operator can step in and say, "Hey, that didn't make sense!," and try to figure out what the real message must have been based on context, prior knowledge, etc.

It has been argued that a seasoned, skilled CW operator can perform the same filtering operation as CCW performs with his brain, concentrating on the incoming code while blissfully ignoring all the other stuff in the receiver's passband at the time. Maybe so, but it's pretty hard work. An equally narrow (9-Hz) conventional audio filter wouldn't help either: the circuit's response time would be so slow it would spread out the leading (attack) and falling (decay) edges of the keyed waveform in time, blurring the distinction between marks and

29 Sommet Vert
St. Adolphe d'Howard, QC
J0T 2B0 Canada
24-hr BBS: 514 226 7796

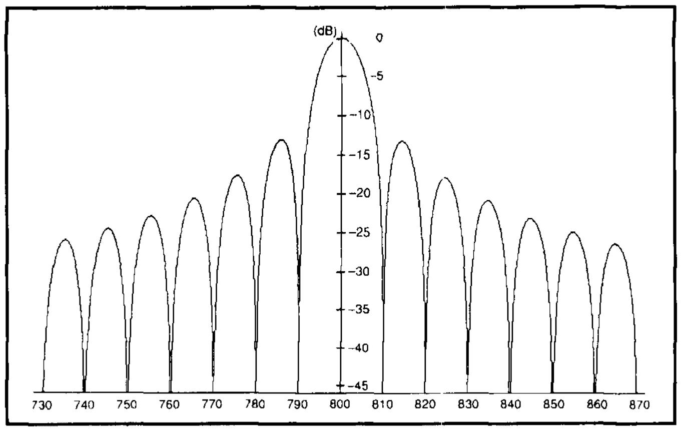


Fig 1—Amplitude response of an integrate-and-dump filter for a small range of audio frequencies near 800 Hz.

This is the characteristic response for both the traditional matched filter built with analog parts (operational amplifiers, resistors, capacitors) and for the DSP version described in this article, built with computer code. Notice the notches every 10 Hz (810, 820, 830 Hz...), and the peaks in the sidelobes—also every 10 Hz, but centered on 815, 825, etc. These sidelobes sometimes allow enough energy from QRM to get through the filter to hurt us; they could be removed (at least drastically attenuated) with further digital processing if we were willing to pay the price: additional computing time. We chose to go with the classic filter response to satisfy the CCW purists and to keep the number crunching requirements reasonable so the program can run on slower computers.

Those infinitely deep notches every 10 Hz can be put to practical use. A Lowfer (160..190 KHz) beacon operator might choose a carrier frequency of say 187.530 kHz, placing it halfway between two power-line harmonics. When the integrate-and-dump filter is centered on this carrier, *all* harmonics of the 60-Hz power line will be notched out simultaneously. Fine business!

spaces. After all the votes are in, it would seem that coherent CW carries an advantage of some 20 dB over regular CW at 12 WPM. So why doesn't everybody do it that way? Answer: up until now it has been quite difficult to get a CCW station on the air. Once synchronization had been established, in order to keep the transmitter and receiver from drifting apart, expensive frequency standards were needed at each end of the link. Transceivers had to be stabilized, usually by phase-locking their master oscillators to some common external standard such as WWVB. And that integrate-and-dump filter circuit was no piece of cake to build and align. These technical challenges have kept all but the most dedicated devotees away from the mode until now.

Can DSP Help?

Some years ago I designed and built a DSP engine dedicated to receiving CCW. It worked, but it was far too complicated—some fifty ICs on the board. I concluded that no one else would ever build one of those things, and I was right. At the time, personal computers were just coming on the scene and I didn't think there was enough processing power in them to do the job. But then there appeared ATs, 386s, 486s, and I thought it might be worth another look. Maybe, with very careful coding, we could do the DSP function on existing ham-shack computers with a minimum amount of external hardware. If so, the cost would be next to nothing and many more people could get in on CCW. After doing an audio spectrum analyzer project (see "A Receiver Spectral Display using DSP," in Jan 1992 *QST*) I realized that a CCW program using the same analog interface was feasible. This interface is a Sigma-Delta analog-to-digital coverter circuit that measures about 4 inches by 1.8 inches and runs off a 9-V battery. It uses a handful of common CMOS chips and can be built in an evening. It also can be purchased assembled and tested, ready to hook up. This circuit samples the received audio 7,200 times each second, converts the measured voltages to numbers, and passes these to the computer through one of its serial ports (eg, COM1) running at 115 kbaud. No modifications to the radio or the computer are required. It is exactly the same board described in the *QST* article and also in the '93 *Handbook*. Some of you will already have one.

What Does the Software Look Like?

The primary thing to keep in mind is that we are going to be hard pressed for time. At 7,200 samples per second, a new sample point's numerical value is fed into the computer every 139 microseconds. The software has to service the UART (serial port) interrupt, read the measured voltage, and perform the DSP filtering operation 7200 times per second. With real tight coding it can be done, and in the time left over we can do some other pretty neat things as well. The challenge is to reduce the time spent handling interrupts to the absolute minimum. The computer has to be able to process the incoming numerical samples at least as fast as they are acquired (ie, in less than 139 μs per sample, on average).

For this algorithm to work, we cannot afford to turn off the interrupts while we do some heavy computing; the interrupts have to be always enabled to guarantee that we don't miss a single sample. DOS systems usually have several "background" tasks that become active periodically, sometimes shutting off the system interrupts for up to a millisecond at a time. So the program's first action is to take over the DOS timer interrupt to make sure no other program gets control of the machine—even for a millisecond—while the DSP algorithm is running. In writing the program, I concentrated on shaving cycles from the serial-port interrupt service routine. In real-time programming, what's crucially important is to reduce the amount of time the computer spends on things which are done often (eg, 7,200 times every second). Things which happen less frequently can be coded a little more sloppily. How to minimize the interrupt service time? Well, we would like very much to avoid any particularly long machine instructions, such as multiply or divide.

The classic integrate-and-dump (I&D) filter works by first shifting the frequency of the incoming 800-Hz tone

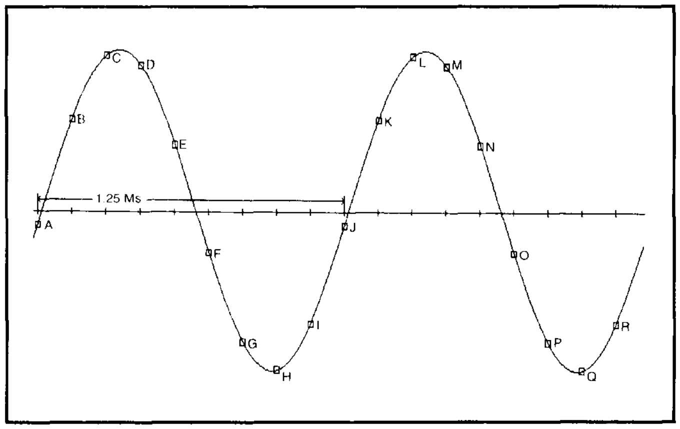


Fig 2—Each cycle of an 800-Hz sinewave takes 1.25 milliseconds. Digitizing the audio at the rate of 7,200 samples per second, we measure the instantaneous voltage nine (9) times during this same period. The analog waveform advances through forty (40) degrees of phase between samples.

To measure the amplitude of this signal (which may be buried in the noise), we use the principle of least squares to "fit" an 800-Hz sinewave on to the sampled data points. A least-squares fit gives us the best estimate of the signal's amplitude and phase. The formula requires us to multiply each voltage sample by the sine of angle X, and then to sum the resulting product into an accumulator. We must also multiply by the cosine of X, summing the result into a different accumulator. The angle X is advanced by 40 degrees after each sample. This normally takes two multiplications and two additions per sample, plus the overhead required to service the interrupt, read the voltage, advance the phase angle, etc.

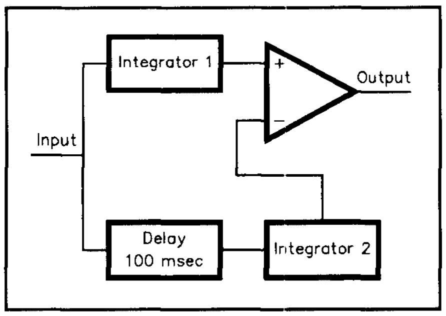
Notice that after the first nine samples (A through I), the tenth sample (J) will occur at exactly the same relative position on the second cycle as our (A) sample did on the first cycle. In other words, the angle X for sample J should be the same as the angle we used for sample A. Similarly, sample K will have the same phase angle as sample B and so on. This is a stroke of luck for us, and happens only because we have chosen a sampling rate which is an exact integral multiple of the frequency of the sinusoid whose amplitude we want to measure.

down to "baseband" (dc). This is done by mixing the audio with an 800-Hz reference tone. Once at baseband, the energy in the signal is split into two separate channels called *in-phase* (I) and *quadrature* (Q). These channels are then independently integrated over the 100-ms window. This is essentially an averaging-over-time operation and is only feasible because at 0 Hz (dc) the I and Q channels don't change their values with time. However, if the incoming tone is not *exactly* at 800 Hz, a beat note will be generated and will cause problems for us, at least to some degree. Let's take an example. At 800 Hz, a 100-ms frame consists of exactly 80 cycles of that sinusoidal waveform. We mix it with our 800-Hz reference tone and we get exactly 0 Hz, or dc, which can be averaged. Now let's say the incoming frequency was not 800 Hz, but rather 790 Hz. The beat note will be 10 Hz. But if you look at it over 100 ms, or a tenth of a second, there will be only one complete cycle. If we take the average value of its voltage over that 100-ms period we come up with zero! It's positive for half the time, equally negative for the other half, and the average value is zero volts. It matters not one iota what the starting amplitude of that 790-Hz tone was; the output of our I&D filter will be zero. And likewise for any frequency that is spaced away from 800 Hz by an any exact multiple of 10 Hz. Between 800 Hz and 790 Hz, the filter's response takes on intermediate values, ranging all the way from 0 dB at 800-Hz down to minus infinity dB (total attenuation) at 790 Hz.

At the end of each 100-ms integration period, the classic hardware filter freezes the instantaneous voltages on its integrating capacitors and, based on the values of the I and Q channels averaged over the interval, computes the average amplitude of the 800-Hz tone during that period. (It could also compute the phase of the tone relative to that of the 800-Hz local reference oscillator, but that generally is not done.) The hardware version of an I&D filter then has to dump all the charge from those capacitors instantly to start measuring the signal in the next window. This is physically impossible with real hardware integrators, so in practice we always waste a little of each frame right at the beginning while the capacitors are discharged of the voltages built up during the previous frame. (Well, not absolutely impossible: one could set up two complete filters and switch between them on alternate frames, I guess.)

So, how can we get the same answer with a DSP algorithm, get around the old problems, and hopefully avoid having to build the hardware filter at all? Our basic challenge is to estimate the amplitude of an 800-Hz sinusoid which is assumed to be unvarying throughout the entire 100-ms window, and which is likely to be much weaker than any number of interfering carriers present in the receiver's passband at the same time. The classical DSP solution to this would be to emulate the hardware I&D filter in software. We multiply the incoming samples with a unit vector rotating at 800 Hz to get instantaneous values of the I and Q components of the baseband signal, then we integrate (add up) all these values over 100 ms, eventually dividing by the total number of samples taken to obtain the averaged I and Q values. Then we compute the square root of the sum of the squares of the two mutually-orthogonal components, and that's the answer. (We could also calculate the phase of the received sinewave by computing the arctangent of the ratio of the Q and I components.) In this particular case, at 7,200 samples per second, each 100- ms window would consist of 720 samples. Which would mean 720 × 2 multiplications, 720 × 2 additions, plus a whole bunch of other time-consuming stuff at the end of each window (such as calculating the square root, clearing the accumulators, etc). All of this is mathematically equivalent to "fitting" an 800-Hz sinusoid to the sampled data points by least-squares, then solving for its amplitude and phase. That's a lot of number crunching to get done in a tenth of a second, even for today's faster home computers. And there isn't a whole tenth of a second available either—a good portion of the time

Fig 3—This shows how the "running average" version of an integrate-and-dump filter is configured. We integrate the input signal continuously without ever clearing (resetting) the integrator circuit. But we also integrate a delayed version of the same signal (it's delayed by 100 milliseconds), then we subtract the outputs of the two integrators. The result is the integral of the signal over the last 100 milliseconds, and it can be sampled at any time, not just at the end of some particular window. The *Coherent* program uses this scheme to sample the DSP filter's output three times near the end of each window: a little early, at the nominal time, and a little late. These numbers are then compared to see if we need to adjust the timing. In CCW it is important to keep the tracking window on top of the sender's window with the smallest possible amount of stagger. For example, if the window is lagging during a mark-space sequence, not only is there less mark energy available to detect, there is also more (erroneous) space energy due to the part of the mark pulse which is received in the following window. Since the recovered intelligence depends on our ability to distinguish between mark and space, this is a double-whammy that can rapidly degrade copy if we allow the transmitter and receiver to start drifting out of sync.



goes for overhead (pushes, pops, etc) in servicing those 720 interrupts—the DSP algorithm has to operate in whatever time is left over after all that. So it's tough, eh? Well, here is where we start getting lucky. On most computers a multiply instruction takes much longer to execute than a simple addition, so we would be far ahead if we could eliminate those two multiplies per sample—and it just so happens we can. We know that at 7,200 samples per second, each individual cycle of an 800-Hz tone takes exactly 9 samples to cover. In other words, during the time between each sample and the following one, an 800-Hz sinusoid will have advanced through 40 degrees of phase. And 40 × 9 = 360 degrees, which is one complete revolution exactly. Eureka!

After 9 samples have been processed, the coefficients we have to multiply the samples with will start to repeat, taking on the same sequence of values for the next 9 samples, and so on. So of our 720 samples in total, 80 of them will be multiplied by two particular numbers, another 80 will be multiplied by a new two-number set, etc. This good fortune is entirely attributable to the fact that our sampling rate just happens to be an exact integral multiple of the frequency of the sinusoid whose amplitude we want to measure. We can take advantage of it by using something called the distributive law (in algebra): A×B + A×C + A×D equals A×(B+C+D). You get exactly the same answer in the end, but one way needs three multiplies and two adds, the other way needs only one multiply and two adds. We will use this to solve for the average I and Q component values with only 18 multiplies per 720 samples instead of 1440!

But there is another trick available. It turns out that the cosine of 40 degrees (one of our sample phases) has the same value as the cosine of 320 degrees (another one of our points). Likewise, of the remaining seven coefficients, six of them can be paired up in this way, the seventh is unity, and it's real easy to multiply by one! On the sine side, 8 of the 9 coefficients can be paired (allowing for sign changes), the other coefficient is zero (it's even easier to multiply by that, hi!). The bottom line is that by combining terms we can get by with just 8 multiplications instead of the 18 that would otherwise be needed. This is starting to look doable!

What we have to obtain during each 100-ms window are the end values of nine (9) accumulators, and on each of the 720 interrupts we only have to add the current sample into one of those nine accumulators. For each interrupt we sum into another accumulator, and after the ninth sample has been processed we start over with the first accumulator. Thus, aside from the house-keeping associated with servicing the interrupt, checking for overruns and/or clipping, figuring out which accumulator to address and such mundane things, we are left with a simple addition to perform. For sure, at the end of the 100-ms window we must do some further calculations, but we have all the time in the world (relatively speaking) to get them done in the 100 ms during which the data for the next-following window are being acquired.

Now, there is one little complication: I would like to run not one, but three (3) I&D filters concurrently. These three filters should overlap in time so that by comparing the three outputs I can decide if the window phase is drifting and it would be advantageous to make a slight adjustment to the phasing cycle that determines when each window starts and ends. The most economical way to run three such I&D filters concurrently is never to clear the various accumulators. Instead, I keep a "delay line" in memory, consisting of the last 720 samples taken. For each new sample, I add it into the appropriate accumulator, then subtract off the value of the sample taken 720 time slots earlier. In this way the accumulators are guaranteed not to overflow, because in the long run we subtract out as many counts as we add in. Furthermore, at the end of every 9 samples (one cycle of the 800-Hz audio signal)—the nine accumulators always hold exactly the same numbers you'd get if you started 100 ms ago with cleared accumulators and integrated through one complete window's worth of samples. Try doing *that* with analog integrators! It works on the computer because digital fixed point arithmetic is absolute—there are no errors such as would arise in an analog integrator due to charge leaking off a capacitor, component values changing slightly with temperature, etc. Any analog integrator would eventually saturate at one rail or the other due to such errors. The digital integrator can run for hours (or days) with absolutely no accumulated long-term error.

We still must "sample and hold" the values in those nine accumulators whenever we have to compute the amplitude of the measured component over that particular window. This involves moving nine 16-bit numbers to secondary storage positions. It is done with a single block move instruction and has no impact whatsoever on the processing of subsequent samples, so the computer version can start to process each new window immediately (not throwing away any of the incoming energy) as opposed to the analog version, which has to wait for the capacitors to discharge fully before starting a new integration cycle.

At the end of each window, we must then compute the amplitude of the measured 800-Hz component. This involves those eight multiplications mentioned above, then taking the square root of the sum of the squares. This is done in tightly coded assembly language in order to execute in the shortest possible time.

As a fine-tuning aid for the operator, the program also measures the frequency of that 800-Hz component (to the nearest tenth of a Hz) and displays it on the computer screen, updated at the end of each 100-ms window. There is a trick to this: the frequency has to be measured *after* the DSP filtering operation. Otherwise, any nearby strong carrier could disrupt the measurement and give an erroneous reading. In fact, during each *marking* window (once we have decided that the 800-Hz tone was indeed present during that window) we also measure its phase (averaged over the entire 100-ms window) and save this for later reference. On the next *marking* window, as long as it is not too far in time away from the last measured one, we measure the phase again. If there is any slight discrepancy between the frequency of the incoming 800-Hz tone (from the receiver) and our precise 800-Hz reference tone (which is actually determined by the 1.8432-MHz crystal oscillator on the Sigma-Delta board), there will be some phase shift in the detected signal (in the same way the amplitude of the "beat note" varies regularly whenever there is a slight difference between two compared frequencies). If we know the amount of phase shift as well as the time interval over which this phase shift accumulated, we can figure out how much the received frequency differs from the nominal 800-Hz value, and there you have it. The operator can set a software switch to enable this phase comparison to also occur across an intervening *space* frame (or not). If the transmitter is phase-coherent from one key-down to the next, then it is feasible to use those two keydown periods to measure his frequency. If not, then the software only uses phases measured in consecutive marking frames (eg, during a "dah"), when the key is presumably held down for 300 ms continuously and the transmit carrier could not change its phase during that period.

That is the essence of a DSP version of the integrate-and-dump filter. The algorithm runs on just about any IBM-compatible computer. The program incorporating this algorithm is called *Coherent*. There is enough time left over after the DSP calculations to allow for lots of other CCW goodies. For instance, *Coherent* has an auto-tune feature. It is important to keep the incoming audio tone centered in the filter's rather narrow passband. *Coherent* tracks the incoming signal's frequency. If it deviates more than half a hertz from the nominal 800-Hz value, *Coherent* issues a pulse on one of two RS232 control lines to make the receiver tune up or down by 1 Hz. Many modern rigs can tune in precise 1-Hz steps by pressing the MIC up/down buttons, so *Coherent* makes the signals needed to do this automatically. Once the signal has been tuned in initially, the operator can sit back, put his feet up, and leave the driving to the computer. Even if his receiver drifts in frequency (or if the transmitter drifts) it's no problem because the software will retune the radio as necessary to maintain the CW tone at 800 Hz.

Coherent also has a frame-phasing tracking loop. After each 100-ms frame is processed, the program looks at whether the SNR would have been better had the window ended 1 cycle (1.25 ms) earlier or 1 cycle later. If there is consistent evidence that going to a slightly earlier window would improve the SNR, then the program does this automatically. What this means is that once synchronization has been achieved, the operator can let the program track the incoming signal and adjust the phasing as necessary to maximize the SNR advantage the mode is capable of. With this system, special frequency standards and rig stabilization are no longer needed. The only equipment you need to operate CCW is a *reasonably* stable transceiver, the little Sigma-Delta interface board, and a computer.

And, of course, the *Coherent* program also lets you send CCW just by typing on the keyboard. That should go without saying. As well, the program has a "beacon" mode, where a pre-stored CCW message can be scheduled to go out at precise time intervals based on the computer's clock.

And Now, Something to Think About...

We have seen that by synchronizing our receiver to the keying at the remote transmitter it is possible to shrink the passband of our receiving filter down to a rather astonishing nine hertz or so, in the process eliminating much of the noise that would otherwise render the signal unreadable. That is fine for coherent CW stations, but what about ordinary CW—where the guy at the other end is sending by hand and his carrier can turn on or off at any arbitrary time? After all, the overwhelming majority of amateur stations around the world don't use coherent CW. Is there anything we can do with DSP to help dig these weak signals out of the mud?

Consider a train of RF pulses, where a carrier is switched on for, say 100 milliseconds, then switched off for the next 100 milliseconds. Let's assume this signal is received by a normal amateur sideband rig with its local oscillator tuned 800 Hz away from the incoming carrier. Looking at the audio coming out of the speaker, what frequency components are present? Well, that depends on your point of view! On the one hand, if we take the position that for a frequency component to exist it must be present always with unvarying amplitude and phase, then we must presume many frequencies, all adding up to make the on/off pulsed waveform. On the other hand, if we examine the waveform on an oscilloscope, we see 80 cycles of a pure (800 Hz) sinewave inside each pulse with no other frequencies present during either the pulses or the silent periods. Common sense tells us there is but one frequency (800 Hz), and that it is only there some of the time. Since there is only one frequency in the signal, it would make a lot of sense to use an arbitrarily narrow filter (ie, 0-Hz wide) centered on that 800-Hz tone. Such a filter would eliminate *all* the noise (QRM, QRN) except that which happened to be on *exactly* the same frequency. The way we usually design highly selective (narrow) filters is to take many samples spaced over a long interval of time and combine them mathematically to isolate the contributions of all the individual frequencies. Analog filters use the same technique, storing energy in reactive components. The narrower the filter (the higher the "Q") the longer the energy from any given cycle stays around inside the tank circuit. The drawback with all these filters is that it takes a long time for them to attain their final output value after a step change in the input signal (as is the case when a CW carrier is keyed). The sharper a filter is in frequency, the longer it takes (in time) for it to respond. This is why experienced CW operators will tell you it doesn't pay to use IF filters much narrower than about 250 Hz when trying to copy code by ear. Narrower filters actually make it *harder* to copy because they obliterate (smear) the sharp leading edges of the keyed tones which the ear needs to recognize code patterns. But the characteristic time spreading of such filters is not a result of some insurmountable law of physics! It follows entirely from the particular way they were designed: they observe a signal over a long timespan to make fine distinctions in frequency in order to realize the narrow response.

The ideal filter for copying ordinary CW would be 0-Hz wide and have an instantaneous response time. When the key went down at the transmitter, the output of the "sliver" filter at the receiver would reflect the amplitude change immediately.

What approach can we take in designing such a filter? A good question is: for any given signal, how much of it do we need—how long do we have to observe it before we can break it down into its component frequencies and state what the amplitude at any specific frequency must be? Could we take just a momentary snippet out of a waveform, analyze it extensively on a fast computer and figure out its complete spectral content just from that tiny portion we looked at? The answer will surprise you. The answer is yes! In fact, there is no minimum amount of time for which we need to observe a waveform in order to completely characterize it. In theory, we could sample a complex signal for just one instant and immediately know the amplitude of an 800-Hz sinusoid in it—regardless of whatever other frequencies might be present. The calculation gets a whole lot more complicated when many other frequencies are present, but it still can be done. The most straightforward case is when we know there is only one sinusoid, say at 800 Hz, and we want to ascertain its amplitude with just one instantaneous glance at the waveform. Hmmm—if we knew the phase it would be easy. With only a single voltage measurement taken at a known point along a sine curve (phase), we can determine its amplitude. Not knowing the phase in advance, we have to solve for it. Which means we need at least two (2) independent measurements, both taken at the same instant in time. The actual sampled voltage will do for one of them. For the other we can use the first derivative—the rate the voltage is changing at that particular moment. This derivative can be obtained without taking any time: convert the voltage to a current, run it through an inductor, and measure the instantaneous voltage across the inductor. Here we have an implementation of our "ideal" CW filter for the simplest case where there is only one frequency component to resolve. When there are many frequency components to separate, we will obviously need more information, but it is all available in that same instant; the higher order derivatives are mutually orthogonal, hence independent, and they're there for the measuring. So it is possible (at least in principle) to design a CW receiving filter with arbitrarily narrow bandwidth (approaching 0 Hz) and a virtually instantaneous response time. What's needed is hardware to differentiate a signal repeatedly and a very fast computing machine to crunch the numbers.

Obtaining the Software

The following can be ordered from the author:

Coherent CCW software package, $20

Bare circuit board for constructing Sigma-Delta interface, $24

Assembled and tested Sigma-Delta board, ready to hook up, $95

All prices in US dollars, and please include $5 for airmail shipment to anywhere on the planet.

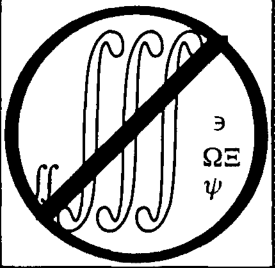
For more information on CCW, contact:

CCW Interest Group
Peter Lumb, G3IRM
2 Briarwood Ave
Bury St Edmunds
Suffolk IP33 3QF
England

DSP

Without Tears™

Learn DSP and Put your Knowledge to work Immediately!



Coming to a Place Near You
Ft. Lauderdale
Albuquerque
Washington D.C.
Chicago, Dallas
San Jose, Long Beach

Call Z Domain Technologies, Inc.
(800)-967-5034
(404)-664-6738
Call 9-5 EST. Ask for brochure.
Our 2-Day Advanced Course is ready. Call for info

By Taking This 3-Day Course You will really learn DSP. Guaranteed!