Measuring 9600-Baud Radio BER Performance

DSP techniques make testing a G3RUH-compatible radio easy.

By Jon Bloom, KE3Z

One of the jobs of the ARRL Lab is to test the performance of equipment sold to amateurs. With the new crop of 9600-baud radios coming out, we had to develop a technique for testing their performance. The best way to test the performance of a radio used for digital communication is, by far, to test the bit-error rate (BER) that the radio provides under various conditions. BER is a conceptually simple metric that answers the question: How many of the bits get through correctly when a data stream is passed through the system?

The system we developed uses a Texas Instruments DSP Starter Kit (DSK) board that includes a TMS320C26 processor. Figs 1 and 2 show the circuitry of the BER test box built in the ARRL Lab. The DSP generates a test signal that is passed through the system under test. For receiver testing, the signal modulates a low-noise FM signal generator that feeds the radio being tested. The demodulated output of the radio goes to the DSP input so that the demodulated signal can be compared to the transmitted signal. To test transmitters, the DSP output test signal is applied to the modulation input of the transmitter under test. The transmitter RF output is attenuated to a low level, then applied to the Lab-built test box where it is mixed with an unmodulated signal from a signal generator. The resulting IF signal is demodulated by a low-distortion demodulator, and the demodulated signal is routed to the DSP input for comparison with the generated signal.

225 Main Street
Newington, CT 06111
email: jbloom@arrl.org

G3RUH Signals

The 9600-baud system used for amateur packet radio, both terrestrially and via the UoSat packet satellites, uses signals handled by the G3RUH modem design. In this system, the binary data stream coming out of a TNC is first "scrambled" to remove any dc component of the signal. Scrambling, which has nothing to do with encryption or data hiding, is simply encoding that ensures that, on average, there are as many 1-bits in the data stream as there are 0-bits. The average voltage of the data signal is thus constant. The scrambled data signal is then used to generate shaped pulses. The shaped-pulse signal is what is applied to the modulation input of the FM transmitter.

On reception, the shaped-pulse

signal is filtered and limited, to recover the scrambled data stream. A clock-recovery circuit generates a clock signal that is synchronous with the incoming data. Using this clock, the modem descrambles the data stream, with the end result being a binary data signal identical to the transmitted data signal.

Since our test system has to generate and receive signals like those of the G3RUH modem, some discussion of the pulse shaping used is necessary. The general problem to be solved when sending digital data using FSK is to limit the bandwidth of the baseband (data) signal before applying it to the modulator. But you have to be careful how you do that.

When we limit the bandwidth of a pulsed signal, we necessarily stretch the pulses in time. That is, a signal that is bandlimited cannot also be time limited. That means that in limiting the bandwidth, we cause each pulse to overlap adjacent pulses. In theory, each pulse overlaps *all* of the other pulses, but as you go further away in time from a particular pulse, its amplitude gets smaller and smaller.

So, we need to find some way of allowing the pulses to overlap their neighbors without interference. One approach to doing so is to shape the

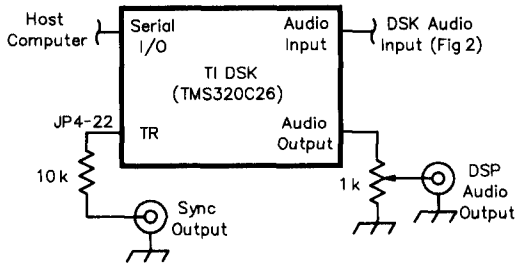


Fig 1—Diagram of the BER test connections to the TI DSK.

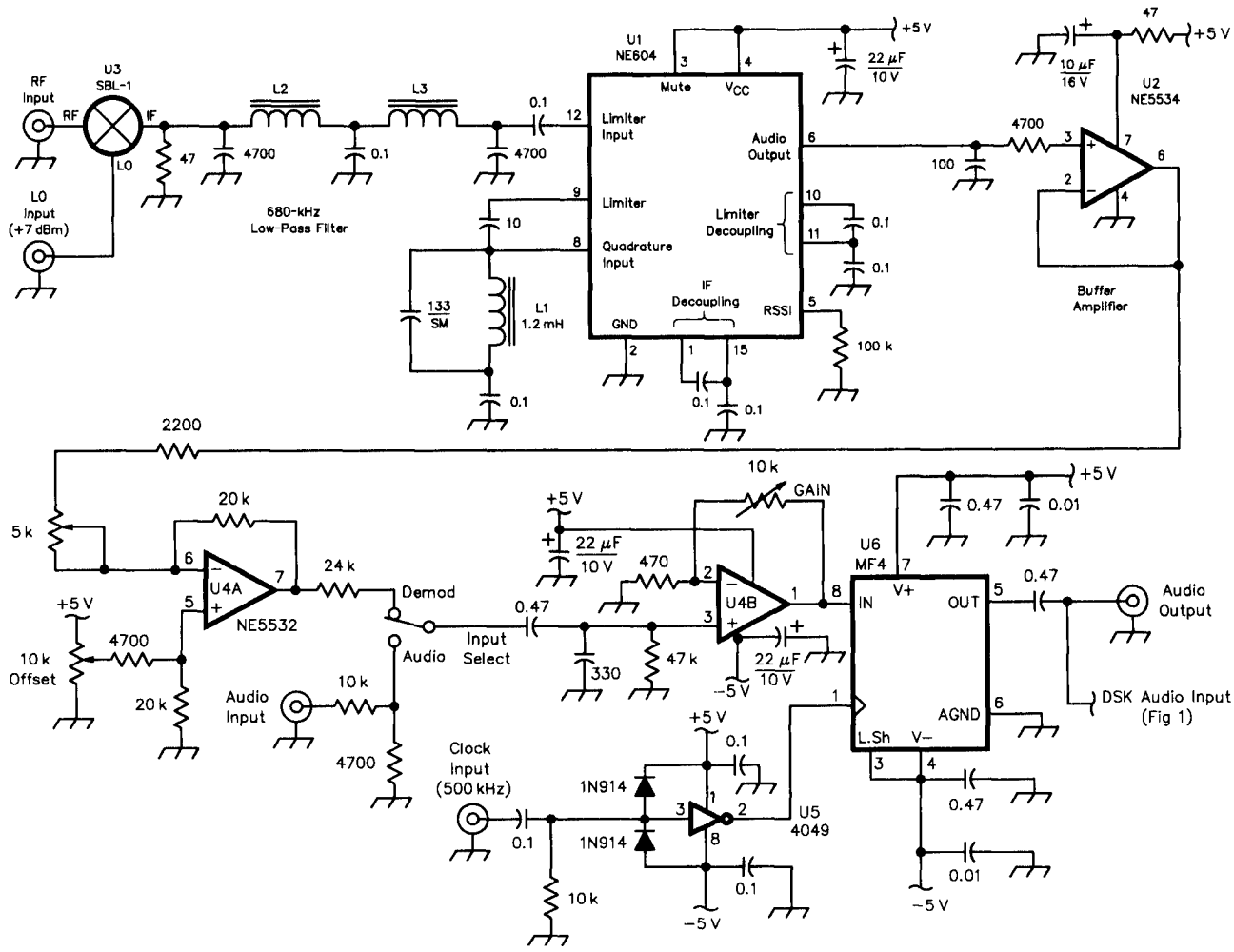


Fig 2—Schematic diagram of the BER test box circuit.
L1—1.2-mH Toko 10RB fixed inductor. (Digi-Key part TK4401-ND.)
L2, L3—41 turns #28 enam wire on a T-50-1 toroid core.

pulses so that, while each pulse does overlap its neighbors, the amplitude of that pulse is zero at the center of each of the other pulses. That way, we can sample the signal at the center of each pulse period and see only the signal from the current pulse; none of the other pulses contributes any amplitude to the signal at that time.

The spectrum of one pulse that has these characteristics is straightforward. It is flat from 0 Hz out to some chosen frequency, then rolls off with a cosine-shaped curve, reaching –6 dB at one-half the baud rate.[1] It continues rolling off with this cosine curve until it reaches zero. In the G3RUH system, the spectrum begins rolling off at 2400 Hz, reaches the –6-dB point at 4800 Hz and reaches zero at 7200 Hz.

In the time domain, the pulse shape that results from such a spectrum has a maximum at the center of the bit period, then decreases in amplitude as we move away from the center of the bit. The pulse signal goes negative, passing through zero at the center of the preceding and following bits. As we go farther away in time from the bit center, the signal alternates between positive and negative values, always passing through zero at the center of each bit. There are other spectra that have pulse shapes that reach zero at the center of all the other pulses, but the benefit of the raised-cosine spectrum is that the pulse amplitude falls off rapidly with time. This is important because any amplitude or phase distortion present in the system is likely to cause the zero-crossing points of the pulse to shift in time, causing ISI. Since the amplitude of the pulse is small near the center of other pulses, the potential for harmful ISI is also small.

Since each bit of the shaped-pulse signal now extends across multiple bit periods—both preceding and following bits periods—we must take this into account in generating our signal. And, since this is DSP, what we are generating is a *sampled* version of the signal. What we must end up with at any sample is a signal that comprises a component from the current bit, preceding bits and following bits. Theoretically, we need components from *all* of the bits in the data stream, but the amplitude of each pulse falls off so rapidly that only a few successive bits need be used to generate any given sample. In this system, we chose to include components of the current bit and the four preceding and four following bits.

Note that only 1-bits contribute to the signal; 0-bits generate no pulse. If we were to send a continuous stream of 0-bits, we'd get no pulses at all. (Of course, the scrambler circuit ensures that will never occur in a real transmitted data stream.) So, for each sample we need to add up the contribution of the current bit and the contributions of eight other bits, some of which may be 0-bits that make no con-

[1]Notes appear on page 23.

User-Defined Parameters:

Sampling rate: $f_s := 38400$ Baud rate: $R := 9600$ # bits: $b := 9$

System bandwidth: $B := 7200$

Calculated parameters:

$f_0 := \frac{R}{2}$ $f_1 := 2 \cdot f_0 - B$ $n := \frac{f_s}{R} \cdot b$ $n = 36$

$f_\Delta := B - f_0$

$h_e(t) := \left(\frac{\sin(2 \cdot \pi \cdot f_0 \cdot t)}{2 \cdot \pi \cdot f_0 \cdot t}\right) \cdot \left[\frac{\cos(2 \cdot \pi \cdot f_\Delta \cdot t)}{1 - (4 \cdot f_\Delta \cdot t)^2}\right]$ Impulse response function

$t_d := \frac{1}{f_s}$ Sampling period

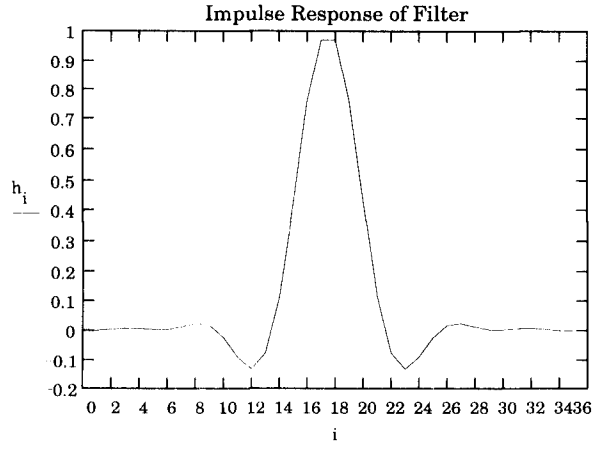
$i := 0, 1 .. n - 1$ $t_i := \left(i - \frac{n-1}{2}\right) \cdot t_d$ Time values

$h_i := h_e(t_i)$ Impulse response of filter

$scale := \frac{2}{(\max(h) - \min(h))}$ $scale = 1.81101$

$a_i := |h_i|$ $s_i := \frac{h_i \cdot scale}{\sum a}$ Normalization of filter coefficients $j := 0, 1 .. floor\left(\frac{n-1}{2}\right)$

WRITEPRN(IMPULSE) := s_i



s_j
-0.001
$-6.1271 \cdot 10^{-4}$
$7.4531 \cdot 10^{-4}$
0.0019
0.0016
$2.9393 \cdot 10^{-4}$
$3.8545 \cdot 10^{-4}$
0.0036
0.0075
0.0054
-0.0086
-0.0311
-0.0452
-0.0269
0.0393
0.1453
0.2571
0.329

Fig 3—This *Mathcad* worksheet calculates the impulse response of the FIR filter used to generate the shaped-pulse signal.

tribution. The contribution of a particular preceding or following 1-bit is the amplitude of its pulse at the present time. To keep things simple, we use a sampling rate that is a multiple of the baud rate—and thus a multiple of the pulse rate.

The method used to implement this pulse forming is an FIR filter. The impulse response of the filter is simply the samples that comprise a single shaped pulse, extending over nine bit periods. Our sampling rate is four times the bit rate, so the impulse response is 4×9=36 samples long. A *Mathcad 5.0* worksheet, shown in Fig 3, calculates the impulse response.

If we feed a single sample of amplitude 1 into the filter, preceded and followed by 0-amplitude samples, the resulting output will be a single copy of our shaped pulse, as shown in the *Mathcad* graph in Fig 3. To generate our data stream, we input the present data value (1 or 0) at one sample, follow it with three samples of 0, then input the next data value. At any time, the FIR filter contains zeroes in all except (possibly) 9 locations, representing the 9 successive data bits. The output of the filter at any sample time comprises components of 9 data pulses, which is what we want for our shaped-pulse signal.

Now that we know how to generate the shaped-pulse signal from a data stream, we need to think about how to generate the data stream itself. We want to generate a data stream that mimics the scrambled signal of the G3RUH modem. The scrambler in the modem is a tapped shift register with feedback. The logic equation for this circuit is:

$$y = x_0 \oplus x_{12} \oplus x_{17}$$

where y is the output of the scrambler, x_0 is the current input bit, x_{12} is the 12th previous input bit and x_{17} is the 17th previous input bit. Generating the test data stream is done by making x_0 always a 1, implementing a shift register in software, and calculating y for each new data bit—once every four samples. Note that this is essentially the same signal generated by the G3RUH modem in its BER test mode.

Counting Bit Errors

Having generated a test signal, we now need to consider how to compare the signal coming back from the system under test to the signal we transmitted. In passing through the tested system, the signal will be delayed by some amount, and the amount of delay will vary from one system to another. What we need to do is determine what the system delay is, then remember what our transmitted signal was that far back in the past in order to compare it to the samples of the received signal. But we are only interested in the value of the received signal at one time during each bit period: the center of the bit.

The DSP software handles this need using a two-step delay process. First, the operator will tell the DSP system how many samples of delay there are between the transmitted and received signals. Of course, the system delay is not likely to be kind enough to let the center of the received bits fall right onto one of our samples—there are only four samples per bit, after all. So, the second step is to adjust the phase of the DSP's transmit and receive sample clocks. The combination of these two techniques lets us adjust the DSP's delay with fine resolution.

This calibration procedure is performed using a dual-channel oscilloscope. One channel is connected to the *sync* signal from the DSK, the other channel is used to display the demodulated signal that is being fed into the DSK input. The operator first commands the DSK to generate a calibration signal. The DSK does so by passing a single 1-bit through the FIR filter described previously, followed by 0-bits. The 1-bit is repeated every 72 samples, resulting in a single shaped pulse every 2 ms. The oscilloscope is triggered on the sync signal and set for 0.2 μs/div. The operator then sends commands to the DSK to alter the number of samples of delay between the transmitted and received signals. The sync signal, a short pulse, is output during the sample the DSK believes to be the center of the received bit. Thus the operator simply adjusts the delay value, which causes the

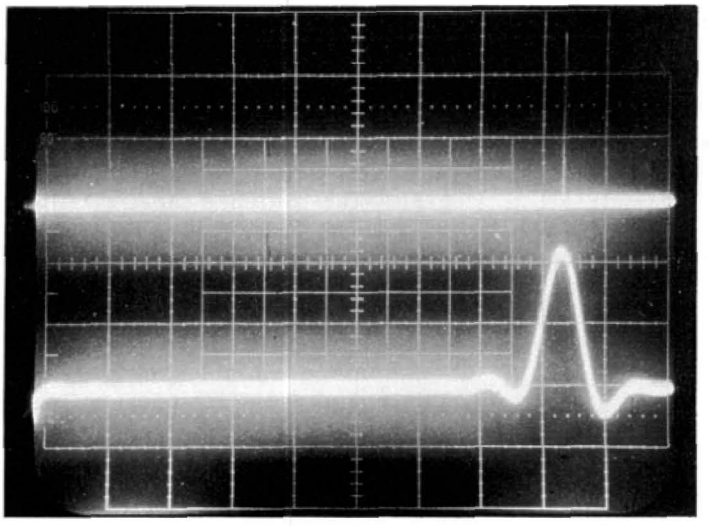


Fig 4—The first-phase calibration signal. When properly calibrated, the calibration signal (lower trace) is aligned with the sync pulse (upper trace) as shown. The oscilloscope is set for 0.2 μs/div. The sync pulse is very narrow and hard to see in this photograph but shows up well on the oscilloscope screen.

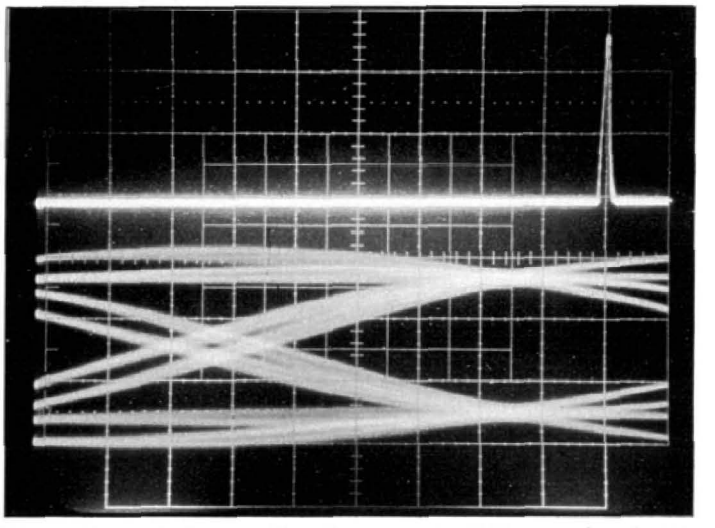


Fig 5—Second-phase calibration uses the BER test signal and sync pulse to produce an eye pattern. The oscilloscope is set for 10 μs/div. The center of the eye is adjusted to lead the sync pulse by 14 μs as shown.

received signal trace on the oscilloscope to move relative to the sync pulse. When the sync pulse is aligned with the shaped pulse of the received signal (Fig 4), the DSK is at the proper sample-delay value.

To achieve final calibration, the DSK clock phases must be adjusted to take into account the part of the system delay that is less than one sample period long. This is done by commanding the DSK to output its test signal and switching the oscilloscope to 10 μs/div. Now the oscilloscope displays a sync pulse at the right side of the screen, along with the *eye pattern* of the received data (Fig 5). The proper sampling point of the bits is the point where the eye is most "open." However, there is a 14-μs delay in the DSK system between the actual sampling point and the sync pulse. Thus, the operator commands the DSK to step its clock phase until the center of the eye leads the sync pulse by 14 μs. This is not a hugely critical setting—a few-microsecond error isn't normally detectable in the measured BER. The delay does have to be in the ballpark, however.

There is one more detail to consider during calibration. Depending on the system being tested, the received signal may be inverted from the transmitted signal; the positive-going pulses we sent may now be negative-going. If this is the case, the DSK will count all the correctly received bits as bad and all the bad bits as correct. So, the DSK supports a user command to invert the sense of the received data. The polarity of the received signal is most easily seen during the first calibration phase (Fig 4), and if the pulse is seen to be inverted then, the DSK's "invert" command should be issued. That will cause no difference in the displayed signal, but the DSK will know "which way is up."

DSP Program Operation

The DSP program used to perform BER measurements is called DBERT. The object file, DBERT.DSK, is downloaded into the DSK from the host computer. Once DBERT is running, it communicates with the host computer via the DSK's serial port. The serial I/O communication mechanism was described in a previous article.[2] When serial I/O is occurring, the DSP interrupts must be disabled. This keeps the DSP from executing its signal-processing operations during serial I/O. For that reason, DSK operation is controlled by the host computer using a command-response sequence: The host sends a command to the DSK, the DSK executes that command—including any needed signal processing—and reports completion of the command back to the host, if needed. Each command to the DSK from the host is a single ASCII character. Some commands result in the DSK needing to send data back to the host. In that case, once the command is completed by the DSK, it sends the data back to the host via the serial interface. Table 1 lists the commands supported by the DBERT program.

Because the DSK has to stop its signal I/O while communicating with the host, there is a potential problem in taking a measurement. When the DSK begins generating the test signal, it is possible that the system being tested will exhibit a response to the start-up transient from the DSK that invalidates the result. So, the BER test signal is started and the DSK waits 20,000 samples (about a half-second) before starting to sample the received signal.

We wanted to be able to test at consistent signal-to-noise ratios in order to establish reference levels for comparing different radios to one another. Since each unit will exhibit a unique sensitivity, we needed some way of adjusting the input power level to get a fixed output signal-to-noise ratio. The solution was to include a SINAD measurement function in the DBERT program. When the DSK receives the SINAD command from the host, it generates a 1-kHz sine wave. About a half-second after it begins generating the sine wave, the SINAD measurement software begins sampling the input signal. It then takes 8192 samples for measurement. Each measured sample is squared and added to a running sum (the *signal-plus-noise* value). The samples are also run through a narrow 1-kHz IIR notch filter that removes the 1-kHz test signal, leaving only the noise. The output samples from this filter are also squared and added to a (separate) running sum (the *noise* value). After 8192 samples have been processed, the two 32-bit sum values are returned to the host computer. The host can then calculate the SINAD by expressing the ratio of the signal-plus-noise value to the noise value in dB. A more detailed description of this technique was published in an earlier *QEX* article.[3]

The analog I/O of the DSK is performed by a TLC32040 integrated analog subsystem. This chip includes a 14-bit D/A, 14-bit A/D, sample-clock generator and input and output programmable switched-capacitor filters (SCF). The TLC32040 is driven by a 10-MHz clock produced by the processor chip. The frequency of this clock, in combination with the programmable dividers in the analog chip, determines the available sampling rates. The design of the BER test software calls for a 38,400 sample-per-second (sps) rate. Unfortunately, this exact rate isn't possible with the 10-MHz clock. The nearest available rate is approximately 38,461.5 sps. This translates to a 0.16% error in the speed of the test signals, which is negligible in the context of BER measurements. However, this error is suf-

Table 1—DBERT Commands

Command	*Return value*	*Description*
I	None	Generates test BER signal
C	None	Generates calibration signal
4	Error count (2 words)	Performs 10,000-bit BER test
5	Error count (2 words)	Performs 100,000-bit BER test
+	None	Increment sample delay
-	None	Decrement sample delay
A	None	Advance clock phase
0	None	Normal data polarity
1	None	Inverted data polarity
N	None	Generate 1-kHz sine wave
Q	None	Quiet (DSK output = 0 V)
S	SINAD value (4 words)	Take SINAD measurement
V	Input voltage (1 word)	Report input voltage value
T	None	Generate test signal (4800-Hz sine wave)

ficient to make the test signal unreadable by a G3RUH modem since the clock-recovery loop in that design has a very narrow lock range. I discovered this early on in the development of this system. To check that the problem was in fact the speed difference and not some problem with the signal I was generating, I temporarily removed the 40-MHz master oscillator on the DSK board—from which the 10-MHz clock is derived—and connected a signal generator to the clock input through a Schmitt trigger inverter. By setting the signal generator to generate exactly the clock frequency needed to get a 9600-baud signal, I found that the G3RUH modem was quite happy to consider my BER test signal a proper one.

Another problem with the TLC32040 is that the input SCF is a band-pass filter that normally cuts off at about 300 Hz. While the filter cutoff frequency is programmable, it can't be set anywhere near the very low frequency needed for this application. So, the DBERT program configures the TLC32040 to operate without its input filter. This requires that an external antialiasing filter be added. As shown in Fig 2, and described in more detail later, an SCF that cuts off at 10 kHz was added to the BER test box to fill this need.

The final characteristic of the TLC32040 to note is that it includes no internal sin*x*/*x* correction—not surprising in a chip with a programmable sample rate. Fortunately, with a sampling rate of 38400 Hz and signals of only up to 7200 Hz, the sin*x*/*x* roll-off at the upper end of the signal spectrum is only about ½ dB. Still, this seems worth correcting. The TLC32040 data sheet describes a first-order IIR filter that can be used to perform this correction. The *Mathcad* worksheet of Fig 6 shows the calculation of the coefficients of this sin*x*/*x* correction filter, which is implemented in the DBERT program. Fig 6 also shows the predicted output response both before and after the correction, as well as the estimated group delay of the correction filter.

As noted, the calibration procedure requires a sync pulse. The DSK has no uncommitted output signal lines usable for outputting a pulse, so DBERT outputs the sync pulse on the serial data output line. Since the sync pulse is less than a microsecond long, this won't usually bother the serial I/O chip of the host computer. (We did find one computer that was occasionally confused by the presence of the sync pulse.) The RS-232 signal is available on one of the DSK's expansion connectors. A 10-kΩ resistor connected to this point brings the sync signal out to the front panel of the BER test box. The presence of the resistor protects the RS-232 line from accidental shorting or application of another signal.

Host Application Software

We used two different programs. One, which we won't cover in detail here, manages the Lab's computer-controlled signal generator for

$f_s := 38400$ Sampling frequency $f := 4800$ Upper frequency of passband

$f_n := \frac{f}{f_s}$ $f_n = 0.125$ Normalized upper frequency

$A := \left(\frac{\pi \cdot f_n}{\sin(\pi \cdot f_n)}\right)^2$ $A = 1.0530293$ Amplitude correction at f_n

$f_L := \frac{1}{f_s}$ $f_L = 2.6041667 \cdot 10^{-5}$ Normalized lower passband frequency

$A_L := \left(\frac{\pi \cdot f_L}{\sin(\pi \cdot f_L)}\right)^2$ $A_L = 1$ Amplitude correction at f_L

Compute needed first-order IIR coefficients

$p_1 := 1$ $p_2 := 2$

Given

$A \cdot (1 - 2 \cdot p_1 \cdot \cos(2 \cdot \pi \cdot f_n) + p_1^2) - p_2^2 \cdot (1 - p_1)^2 = 0$

$A_L \cdot (1 - 2 \cdot p_1 \cdot \cos(2 \cdot \pi \cdot f_L) + p_1^2) - p_2^2 \cdot (1 - p_1)^2 = 0$

$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} := \text{Find}(p_1, p_2)$

$v_1 = 0.1049616$ $v_2 = 1$

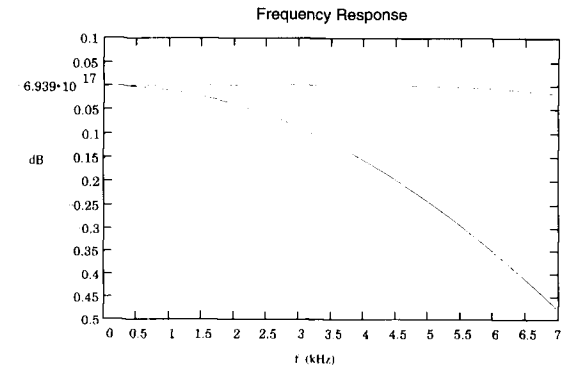
Resulting first-order IIR filter: $y[n]=v_2(1-v_1)x[n]+v_1y[n-1]$

Computation of Corrected System Response

$S(f) := \frac{\sin\left(\pi \cdot \frac{f}{f_s}\right)}{\pi \cdot \frac{f}{f_s}} \cdot M(f)$ $M(f) := \frac{v_2 \cdot (1 - v_1) \cdot \exp\left(j \cdot 2 \cdot \pi \cdot \frac{f}{f_s}\right)}{\exp\left(j \cdot 2 \cdot \pi \cdot \frac{f}{f_s}\right) - v_1}$

$P(f) := \arg(M(f))$

$f_x := 1, 70 .. 7000$ $\omega_x := 2 \cdot \pi, 70 \cdot 2 \cdot \pi .. 7000 \cdot 2 \cdot \pi$



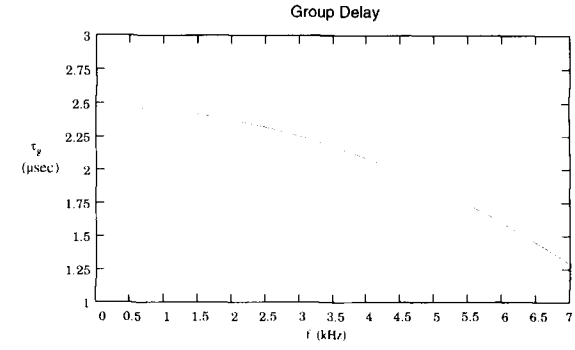


Fig 6—Correction of sin*x*/*x* roll-off is performed with a first-order IIR filter having the coefficients calculated by this *Mathcad* worksheet.

stepped BER measurements at various signal levels and frequencies. The other program, BERT.EXE, is a C program that communicates with the DSK under operator control. Communication with the DSK is normally performed at 19,200 baud, although slower speeds can be used. To ensure that no serial overruns occur, BERT uses interrupt-driven serial I/O.

The BERT program is quite straightforward. It accepts keyboard commands from the operator and sends them to the DSK. The command set is the same as that of DBERT—BERT just passes these commands through to the DSK—with a couple of additions. Also, BERT "knows" about those DSK commands that generate a response from DBERT. When such a command is given, BERT waits for the response from the DSK, then converts the incoming serial bytes to binary values. In the case of BER test values (commands *4* and *5*), it prints out the reported number of errors and the BER, calculating the latter based on the number of bits the DSK was commanded to use in its test. For the SINAD command (*S*), BERT computes and prints the SINAD and the distortion percentage using the values returned by DBERT.

The DBERT program can perform BER tests using 10,000 samples or 100,000 samples. The BERT program also provides a 1-million sample BER test (the *6* command), which it performs by commanding the DSK to perform a 100,000-sample BER test 10 times. BERT sums the values from each of these tests to get the final result. As each BER test result is returned from the DSK, BERT prints the running total of samples, errors and BER. The *L* command performs the same function, except that it stops if 100 or more total errors have been reported.

If the operator selects the calibrate (*C*), idle (*I*), sine-wave (*N*) or quiet (*Q*) commands, BERT remembers the selected command. Whenever a BER or SINAD test command is performed, BERT waits for that command to complete, then sends the appropriate command to place DBERT in the most recently selected mode from among those listed.

BER Test Box Hardware

The BER test box contains several op-amp amplifiers used to keep the input signals within the range of the TLC32040 analog input. The box also contains a mixer and a demodulator, shown in Fig 2, designed by ARRL Lab Engineer Zack Lau, KH6CP. To test transmitters, the transmitter output is attenuated with a high-power attenuator down to a level that the SBL-1 mixer, U3, can handle. The LO input of the SBL-1 is driven by a +7-dBm signal from a signal generator set to a frequency 373-kHz below or above the transmitter frequency. The output from the SBL-1 passes through a low-pass filter that removes the sum frequency, leaving only the 373-kHz difference frequency. This signal is applied to an NE604 FM IF chip that contains a limiter, IF amplifier and quadrature FM demodulator. The frequency at which the demodulator operates is set by L1 and its associated silver-mica capacitor. When Zack built this circuit, the components he used just happened to fall at 373 kHz. If you reproduce the circuit, it's likely that your copy will work at a slightly different frequency. This shouldn't present a problem; the exact operating frequency isn't critical.

Since the NE604 operates from a single 5-V supply, its output is a positive voltage. This signal is fed into U4A, which amplifies the signal and also removes the positive offset, so that the result is zero volts when the input signal is at 373 kHz. U4B amplifies this demodulated signal or an external input signal (usually the output of a receiver being tested) with adjustable gain. The signal is finally filtered in an MF4 4th-order Butterworth low-pass SCF, U6, that acts as the antialiasing filter for the DSP input. The cut-off frequency of U6 is determined by its input clock, with the clock being 50 times the cutoff frequency. In our test set, this clock is supplied by an external function generator, but it could as easily be provided by a crystal oscillator and divider chain.

Interpreting BER Measurements

Bit errors are (or should be) due mainly to corruption of the signal by noise. Thus, they should be random. That being the case, if you make the same measurement several times, you are likely to get different results each time. But the more bits you send through the system, the more the effect of noise is averaged and the more consistent the measurements will be. That raises the question: How many bits do you need to use to get a valid result? The answer to that question depends on what you mean by "a valid result." The more bits you use, the closer your measurement will be to the "true" BER you would get if you sent an *infinite* number of bits through the system. Since sending an infinite number of bits through the system is a little, well, impractical, we need to come up with some guidelines for selecting a finite number of bits to use.

Unfortunately, in order to do that we have to make some assumptions about the character of the noise in the system. Fortunately, the assumptions we make hold up pretty well for real systems. (There's nothing new about this; we make assumptions about the character of system noise all the time. For example, when we relate noise to system bandwidth we often assume that the noise is uniformly distributed across the spectrum of interest.)

What we find is that with a given number of bits in the sample set, and a given number of errors within those bits, we can establish a *confidence interval*. The confidence interval tells us how likely it is that our measurement is within some specified factor of the "true" BER. For example, we might find that our measurement gives us an 80% confidence that the value is within a factor of 3 of the true BER. The end result is that we can get as good a measurement as we want if we are willing to wait for enough bits to go through the system. Of course, at a low BER we don't get many errors, so we need to send a *lot* of bits!

What's interesting about the confidence interval is that it depends on the number of errors detected. Suppose you made two BER measurements. If for the second measurement you double the number of bits sent and get double the number of errors, you end up with the same BER but a higher confidence. On the other hand, if you double the number of bits but measure the same number of errors (you're measuring at a lower noise level, for instance), you get a lower BER but the same confidence interval. What that means is that all we need to do is to ensure that we have at least the number of errors needed to establish the desired confidence interval.

There are two useful sets of numbers we've used here in the ARRL Lab for our BER testing. If you measure 10 bit errors, you are 95% sure that you are within a factor of about 2 of the true BER. And if you get 100 bit errors, you are 99% sure that you are within a factor of about 1.3 of the true BER.[4] If you look at a curve of BER versus sig-

nal-to-noise ratio, you'll find that a factor-of-2 difference in BER occurs with a fraction of a dB change in signal-to-noise ratio. That suggests that a measurement that results in 10 bit errors is a pretty good one, and a measurement that results in 100 bit errors is a *very* good measurement. That's why the *L* command exists in the BERT program. It pops 100,000 bits at a time through the system, stopping when 100 or more errors have been reported—because continuing to measure more bit errors is a waste of time—or when 1 million bits have been sent. Using a million bits means that a BER of 1×10^{-4} (100 bit errors) or worse can be measured with great accuracy, and a BER of 1×10^{-5} (10 bit errors) can be measured with decent accuracy. Of course, if you have time on your hands you can measure a BER of 1×10^{-6} by sending 10 million bits through the system. (That takes over 17 minutes at 9600 baud!)

Conclusion

The system described here has been used to measure a number of 9600-baud radios. Some of the test results will be presented in an upcoming *QST* article, and future *QST* reviews of 9600-baud radios will include measurements made using this system. The system has proven to be effective and easy to use. I should add that we spot-checked the results obtained with this system by measuring BER using a G3RUH modem. It gave BER results that were consistently slightly better because its input filter cuts off at a lower frequency than the SCF in the BER test box, improving the signal-to-noise ratio.

The software for this system, including the source code, is available for downloading from the ARRL BBS (203-666-0578) and via the Internet using anonymous FTP from ftp.cs.buffalo.edu. The file name is QEXBERT.ZIP.

Notes

[1]Couch, L. W., *Digital and Analog Communication Systems* (New York: Macmillan, 1993), p 179.

[2]Bloom, J., KE3Z, "Measuring System Response with DSP," *QEX*, February 1995, pp 11-23.

[3]Bloom, J., KE3Z, "Measuring SINAD Using DSP", *QEX*, June 1993, pp 9-13.

[4]Jeruchim, M. C., Balaban, P. and Shanmugan, K. S., *Simulation of Communication Systems* (New York: Plenum Press, 1992), pp 492-501. □□

Upcoming Technical Conferences

40th Annaul West Coast VHF/UHF Conference

May 5-7, 1995, Sheraton Cerritos Hotel, Town Center, 12725 Center Court Drive, Cerritos, California.

For more information please call 714-990-9203 or fax 714-990-1340.

1995 ARRL Digital Communications Conference

September 8-10, 1995, La Quinta Conference Center, Arlington, Texas—just minutes from Dallas/Fort Worth Airport. Co-hosted by Tucson Amateur Packet Radio (TAPR) and the Texas Packet Radio Society.

More information will be released soon, or contact the TAPR office at 8987-309 E. Tanque Verde Road #337, Tucson, AZ 85749-9399, tel: 817-383-0000; fax: 817 566-2544; Internet: tapr@tapr.org

Call for papers: Deadline for receipt of camera-ready papers is **July 21, 1995**. Contact Maty Weinberg at ARRL HQ (tel: 203-666-1541; fax: 203-665-7531; Internet: lweinberg@arrl.org) for infomation on submitting papers.

Microwave Update 95

October 26-28, La Quinta Inn, Arlington, Texas.

For more information contact: Al Ward, WB5LUA, 2306 Forest Grove Estates Road, Allen, TX 75002 or Kent Britain, WA5VJB, 1626 Vineyard, Grand Prairie, TX 75052-1405.