PSK31: A New Radio-Teletype Mode

Many error-correcting data modes are well suited to file transfers, yet most hams still prefer error-prone Baudot for everyday chats. PSK31 should fix that. It requires very little spectrum and borrows some characteristics from Morse code. Equipment? Free software, an HF transceiver and a PC with Windows *and a sound card will get you on the air.*

By Peter Martinez, G3PLX

[Thanks to the Radio Society of Great Britain for permission to reprint this article. It originally appeared in the December '98 and January '99 issues of their journal, RadCom. *This article includes February 1999 updates from Peter Martinez.*—Ed.]

I've been active on RTTY since the 1960s, and was instrumental in introducing AMTOR to Amateur Radio at the end of the '70s. This improved the reliability of the HF radio link and paved the way to further developments that have taken this side of the hobby more into data transfer, message handling and computer linking, but further away from the rest of Amateur Radio, which is based on two-way contacts between operators.

There is now a gap opening between the data-transfer enthusiasts using the latest techniques and the two-way contact fans who are still using the traditional RTTY mode of the '60s, although of course using keyboard and screen rather than teleprinter. There is scope for applying the new techniques now available to bring RTTY into the 21st century.

This article discusses the specific needs of "live QSO" operating—as opposed to just transferring chunks of error-free data—and describes the PSK31 mode I have developed specifically for live contacts. PSK31 is now becoming popular using low-cost DSP kits. The mode could become even cheaper as the art of using PC sound cards is developed by Amateur Radio enthusiasts.

What is Needed?

I believe that it is the error-correcting process used in modern data modes make them unsuitable for live contacts. I have identified several factors; the first revolves around the fact that all error-correcting systems introduce a time-delay into the link. In the case of an ARQ link like AMTOR or PACTOR, there is a fixed transmission cycle of 450 ms or 1.25 s or more. This delays any key press by as much as one cycle period, and by more if there are errors. With forward-error-correction systems, there is also an inevitable delay, because the information is spread over time. In a live two-way contact, the delay is doubled at the point where the transmission is handed over. I believe that these delays make such systems unpleasant to use in a two-way conversation.

This is not so much a technical problem as a human one. Another factor in this category concerns the way that quality of information content varies as the quality of the radio link varies. In an analogue transmission system such as SSB or CW, there is a linear relationship between the two. The operators are aware of this all the time and take account of it subconsciously: They change the speed and tone of voice instinctively and even choose the conversation topic to suit the conditions. In a digital mode, the relationship between the signal-to-noise ratio (S/N) on the air and the error-rate on

High Blakeband Farm
Underbarrow, Kendal,
Cumbria, LA8 8HP
England

Table 1

The Varicode alphabet. The codes are transmitted left bit first, with “0” representing a phase reversal on BPSK and “1” representing a steady carrier. A minimum of two zeros is inserted between characters. Some implementations may not handle all the codes below 32.

*ASCII**	*Varicode*
0 (NUL)	1010101011
1 (SOH)	1011011011
2 (STX)	1011101101
3 (ETX)	1101110111
4 (EOT)	1011101011
5 (ENQ)	1101011111
6 (ACK)	1011101111
7 (BEL)	1011111101
8 (BS)	1011111111
9 (HT)	11101111
10 (LF)	11101
11 (VT)	1101101111
12 (FF)	1011011101
13 (CR)	11111
14 (SO)	1101110101
15 (SI)	1110101011
16 (DLE)	1011110111
17 (DC1)	1011110101
18 (DC2)	1110101101
19 (DC3)	1110101111
20 (DC4)	1101011011
21 (NAK)	1101101011
22 (SYN)	1101101101
23 (ETB)	1101010111
24 (CAN)	1101111011
25 (EM)	1101111101
26 (SUB)	1110110111
27 (ESC)	1101010101
28 (FS)	1101011101
29 (GS)	1110111011
30 (RS)	1011111011
31 (US)	1101111111
32 (SP)	1
!	111111111
“	101011111
#	111110101
$	111011011
%	1011010101
&	1010111011
‘	101111111
(	11111011
)	11110111
*	101101111
+	111011111
,	1110101
-	110101
.	1010111
/	110101111
0	10110111
1	10111101
2	11101101
3	11111111
4	101110111
5	101011011
6	101101011
7	110101101
8	110101011
9	110110111
:	11110101
;	110111101
<	111101101
=	1010101
>	111010111
?	1010101111
@	1010111101
A	1111101
B	11101011
C	10101101
D	10110101
E	1110111
F	11011011
G	11111101
H	101010101
I	1111111
J	111111101
K	101111101
L	11010111
M	10111011
N	11011101
O	10101011
P	11010101
Q	111011101
R	10101111
S	1101111
T	1101101
U	101010111
V	110110101
W	101011101
X	101110101
Y	101111011
Z	1010101101
[	111110111
\	111101111
]	111111011
^	1010111111
_	101101101
‘	1011011111
a	1011
b	1011111
c	101111
d	101101
e	11
f	111101
g	1011011
h	101011
i	1101
j	111101011
k	10111111
l	11011
m	111011
n	1111
o	111
p	111111
q	110111111
r	10101
s	10111
t	101
u	110111
v	1111011
w	1101011
x	11011111
y	1011101
z	111010101
{	1010110111
|	110111011
}	1010110101
~	1011010111
127	1110110101

*ASCII characters 0 through 31 are control codes. Their abbreviations are shown here in parentheses. For the meanings of the abbreviations, refer to any recent *ARRL Handbook.*

the screen is not so smooth. The modern error-correcting digital modes are particularly bad at this, with copy being almost perfect while the SNR is above a certain level and stopping completely when the SNR drops below this level. The effect is of no consequence in an automatic mailbox-forwarding link, but can badly inhibit the flow of a conversation.

A third factor is a social one: with error-correcting modes, you only get good copy when you are linked to one other station. The copy is decidedly worse when stations are not linked, such as when calling CQ or listening to others. This makes it difficult to meet other people on the air, and there is a tendency to limit contacts to a few close friends or just mailboxes.

These factors lead me to suggest that there is a case for a transmission system that is *not* based on the use of error-correcting codes, when the specific application is that of live contacts. The continued popularity of traditional RTTY using the start-stop system is proof of this hypothesis: There is minimal delay (150 mS), the flow of conversation is continuous, the error-rate is tolerable, and it is easy to listen-in and join-in.

Improving on RTTY

How, then, do we go about using modern techniques that were not available in the '60s, to improve on traditional RTTY? First, since we are talking about live contacts, there is no need to discuss any system that transmits text any faster than can be typed by hand. Second, modern transceivers are far more frequency stable than those of the '60s. We should be able to use much narrower bandwidths than in those days. Third, digital processors are much more powerful than the rotating cams and levers of mechanical teleprinters, so we could use better coding. The drift-tolerant technique of frequency-shift keying, and the fixed-length five-unit start-stop code still used today for RTTY are a legacy of 30-year-old technology limits. We can do better now.

PSK31 Alphabet

The method I have devised for using modern digital processing to improve on the start-stop code, without introducing extra delays due to the error-correcting or synchronization processes, is based firmly on another tradition, namely that of Morse code. Because Morse uses short codes for the more common letters, it is actually very efficient in terms of the average duration of a character. In addition, if we think of it in terms that we normally use for digital modes, Morse code is self-synchronizing: We don't need to use a separate process to tell us where one character ends and the next begins. This means that Morse code doesn't suffer from the "error-cascade" problem that results in the start-stop method getting badly out of step if a start or stop-bit is corrupt. This is because the pattern used to code a gap between two characters *never* occurs inside a character.

The code I have devised is therefore a logical extension of Morse code, using not just one-bit or three-bit code-elements (dots and dashes), but any length. The letter-gap can also be shortened to two bits. If we represent key-up by 0 and key-down by 1, then the shortest code is a single one by itself. The next is 11, then 101 and 111, then 1011, 1101, 1111, but not 1001 since we must not have two or more consecutive zeros inside a code. A few minutes with pencil and paper will generate more. We can do the 128-character ASCII set with 10 bits.

I analyzed lots of English-language text to find out how common was each of the ASCII characters, then allocated shorter codes to the more-common characters. The result is shown in Table 1, and I call it the *Varicode* alphabet. With English text, Varicode has an average code length—including the "00" letter gap—of 6.5 bits per character. By simulating random bit errors and counting the number of corrupted characters, I find that Varicode is 50% better than start-stop code, thus verifying that its self-synchronizing properties work well.

The shortest code in Morse is the most-common letter: "e", but in Varicode the shortest code is allocated to the word space. When idle, the transmitter sends a continuous string of zeros. Fig 1 compares the coding of the same word in ASCII, RTTY, Morse and Varicode.

PSK31 Modulation and Demodulation

To transmit Varicode at a reasonable typing speed of about 50 words per minute needs a bit-rate of about 32 per sec. I have chosen 31.25, because it can be easily derived from the 8-kHz sample-rate used in many DSP systems. In theory, we only need a bandwidth of 31.25 Hz to send this as binary data, and the frequency stability that this implies can be achieved with modern radio equipment on HF.

The method chosen was first used on the amateur bands, to my knowledge, by SP9VRC. Instead of frequency-shifting the carrier, which is wasteful of spectrum, or turning the carrier on and off, which is wasteful of transmitter power capability, the "dots" of the code are signaled by reversing the polarity of the carrier. You can think of this as equivalent to transposing the wires to your antenna feeder. This uses the transmitted signal more efficiently since we are comparing a positive signal before the reversal to a negative signal after it, rather than comparing the signal present in the dot to no-signal in the gap. But if we keyed the transmitter in this way at 31.25 baud, it would generate terrible key clicks, so we need to filter it.

If we take a string of dots in Morse code, and low-pass filter it to the theoretical minimum bandwidth, it will look the same as a carrier that is 100% amplitude-modulated by a sine wave at the dot rate. The spectrum is a central carrier and two sidebands at 6dB down on either side. A signal that is sending continuous reversals, filtered to the minimum bandwidth, is equivalent to a double-sideband suppressed-carrier emission, that is, to two tones either side of a suppressed carrier. The improvement in the performance of this polarity-reversal keying over on-off keying is thus equivalent to the textbook improvement in changing from amplitude-modulation telephony with full carrier to double-sideband with suppressed carrier. I have called

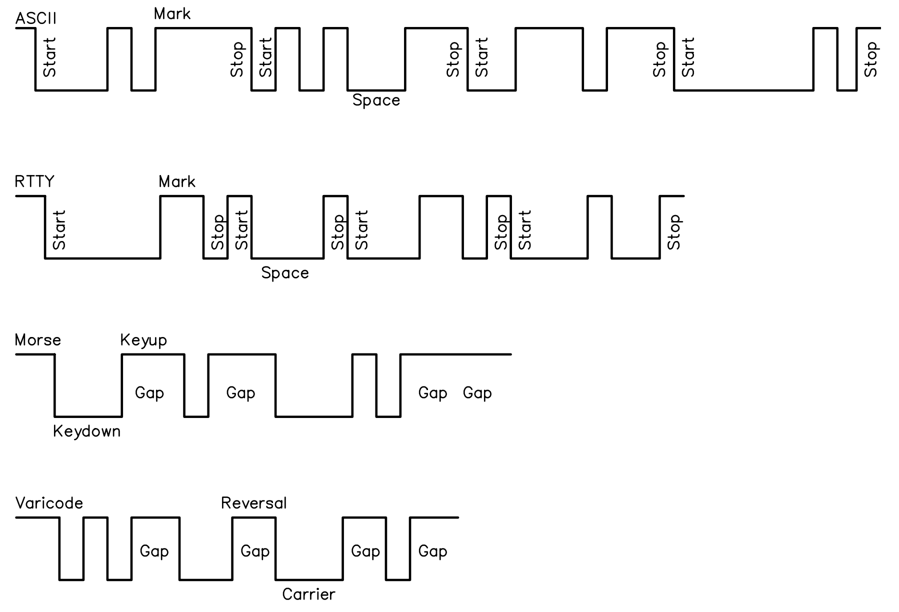


Fig 1—The word "ten" in ASCII, RTTY, Morse and Varicode.

this technique "polarity-reversal keying" so far, but everybody else calls it "binary phase-shift keying," or BPSK. Fig 2 shows the envelope of BPSK modulation and the detail of the polarity reversal.

To generate BPSK in its simplest form, we could convert our data stream to levels of ±1 V, for example, take it through a low-pass filter and feed it into a balanced modulator.

The other input to the balanced modulator is the desired carrier frequency. When sending continuous reversals, this looks like a 1 V (P-P) sine wave going into a DSB modulator, so the output is a pure two-tone signal. In practice we use a standard SSB transceiver and perform the modulation at audio frequencies or carry out the equivalent process in a DSP chip. We could signal logic zero by continuous carrier and signal logic one by a reversal, but I do it the other way round for reasons that will become clear shortly.

There are many ways to demodulate BPSK, but they all start with a bandpass filter. For the speed chosen for PSK31, this filter can be as narrow as 31.25 Hz in theory. A brick-wall filter of precisely this width would be costly, however, not only in monetary terms but also in the delay time through the filter, and we want to avoid delays. A practical filter might be twice that width (62.5 Hz) at the 60-dB-down points with a delay-time of two bits (64 ms).

For the demodulation itself, since BPSK is equivalent to double sideband, the textbook method for demodulating DSB can be used. However, it can also be demodulated by delaying the signal by one bit period and comparing it to the signal with no delay in a phase comparator. The output is negative when the signal reverses polarity and positive when it doesn't.

We could extract the information from the demodulated signal by measuring the lengths of the "dots" and "dashes," as we do by ear with Morse code. It helps to pick the data out of the noise, however, if we know when to expect signal changes. We can easily transmit the data at an accurately timed rate, so it should be possible to predict when to sample the demodulator output. This process is known as synchronous reception, although the term "coherent" is sometimes wrongly used.

To synchronize the receiver to the transmitter, we can use the fact that a BPSK signal has an amplitude-modulation component. Although the modulation varies with the data pattern, it always contains a pure-tone component at the baud rate. This can be extracted using a narrow filter, a PLL or the DSP equivalent, and fed to the decoder to sample the demodulated data. Fig 3 shows block diagrams of a typical BPSK modulator and demodulator.

For the synchronization to work we need to make sure that there are no long gaps in the pattern of reversals. A completely steady carrier has no modulation, so we could never predict when the next reversal was due. Fortunately, Varicode is just what we need, provided we choose the logic levels so that zero corresponds to a reversal and one to a steady carrier. The idle signal of continuous zeros thus generates continuous reversals, giving us a strong 31.25-Hz modulation. Even with continuous keying, there will always be two reversals in the gaps between characters. The average number of reversals will therefore be more than two in every 6.5 bits, and there will never be more than 12 bits with no reversal at all. If we make sure that the transmission always starts with an idle period, then the timing will pull into sync quickly. By making the transmitter end a transmission with a "tail" of unmodulated carrier, it is then possible to use the presence or absence of reversals to squelch the decoder. Hence, the screen doesn't fill with noise when there is no signal.

Getting Going

So much for the philosophy and the theory, but how do you get on the air with this mode? In the first experiments on this mode in early 1996, the route to getting on PSK31 was to obtain one of several DSP starter kits. These are printed-circuit cards, usually with a serial interface to a PC, marketed by DSP processor manufacturers at low cost to help engineers and students become familiar with DSP programming. Some radio amateurs have started to write software for these, not just for RTTY but also for SSTV, packet, satellite and digital-voice experiments. They have audio input and output and some general-purpose digital input/output. The construction work needed is limited to wiring up cables, building a power supply and putting the card into a screened box. The DSP software is freely available, as is the software that runs in the PC to interface to the keyboard and screen, and can be obtained most easily via the Internet. It would certainly be possible to construct a PSK31 modem in hardware, although I know of no one who has done this yet.

However, it became clear late in 1998 that soundcards now common in personal computers are capable of performing the audio input/output function needed for PSK31, with the DSP software running in the PC. At Christmas 1998, I completed a basic *Windows*-based PSK31 program to use the soundcard. The availability of this program has dramatically increased the level of PSK31 activity worldwide. (It's available on the Web: **http://aintel.bi.ehu.es/psk31.html**—*Ed.*)

PSK31 Operating

Since PSK31 performance is the same when calling, listening or in contact, it's easy to progress from listen-

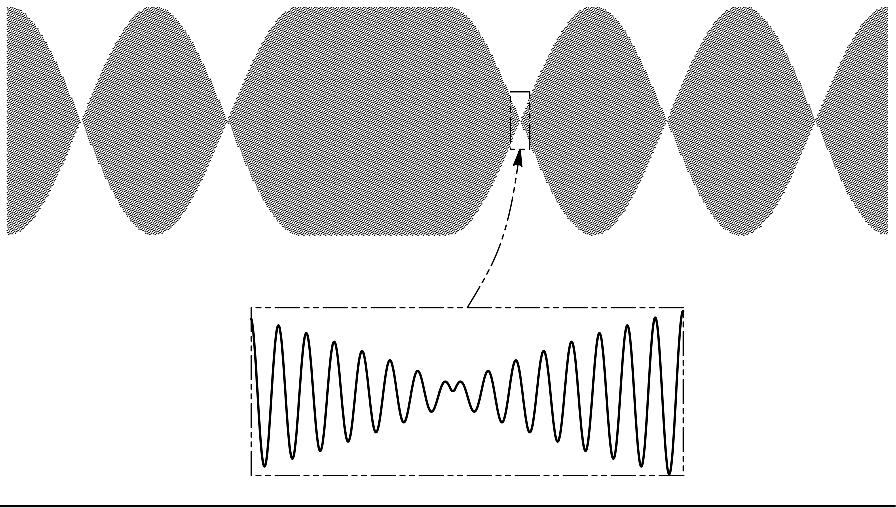


Fig 2—The waveform of BPSK sending the Varicode space symbol., with a close-up of the detail during a phase reversal.

ing to others, to calling CQ, two-way contacts and multi-station nets. The narrow bandwidth and good weak-signal performance do mean learning a few new tricks: First, set the radio dial on one spot. Then fine-tune the audio frequency, while listening through the narrow audio filter rather than the transceiver's loudspeaker, while using an on-screen phase-shift display to center the incoming signal within a few hertz. On transmit, since the envelope of the PSK31 signal is not constant (as is the case for FSK), it is important to keep the transmitter linear throughout. However, since the PSK31 idle is identical to a standard two-tone test signal, it is easy to set up. The worst distortion products will be at ±45 Hz at (typically) 36 dB below PEP.

So far, we've looked at requirements for a live-contact, keyboard/screen communication system, and proposed the narrow-band PSK31 mode as a candidate for a modern equivalent to traditional RTTY. This mode has now been in use on the HF bands by a small but growing band of enthusiasts for about two years. Now, let's look at two recent additions to PSK31.

A Second Look at Error Correction

After getting PSK31 going with BPSK modulation and the Varicode alphabet, several people urged me to add error correction to it in the belief that it would improve it still further. I resisted for the reasons that I gave earlier, namely that the delays in transmission, the discontinuous traffic flow and the inability to listen-in, all make error correction unattractive for live contacts. There is another reason. All error-correcting systems work by adding redundant data bits. Suppose I devise an error-correcting system that doubles the number of transmitted bits. If I wanted to maintain traffic throughput, I would need to double the bit rate. With BPSK that means doubling the bandwidth, so I lose 3dB of S/N and get more errors. The error correction system will have to work twice as hard just to break even! It is no longer obvious that error correction wins. It is interesting to note that with FSK, where the bandwidth is already much wider than the information content, you *can* double the bit-rate without doubling the bandwidth, and error correction *does* work. Computer simulation with BPSK in white noise shows that when the S/N is good, the error-correction system does win, reducing the low error rate to very low levels. At the S/N levels that are acceptable in live amateur contacts, it's better to transmit the raw data slowly in the narrowest bandwidth. It also takes up less spectrum space!

However, there was the suggestion that error correction could give useful results for bursts of noise, which cannot be simulated on the bench, so I decided to try it and do some comparison tests. The automatic repeat (ARQ) method of correcting errors was ruled out. Forward error correction (FEC) seemed to deserve a second look, provided the transmission delay was not too long.

I realized that comparing two systems with different bandwidths and speeds on the air would be difficult. Adjacent-channel interference would be different, as would the effects of multipath. There is, however, another way to double the information capacity of a BPSK channel without doubling its bandwidth and speed. By adding a second, 90° phase-shifted BPSK carrier at the transmitter and a second demodulator in the receiver, we can do the same trick that is used to transmit two color-difference signals in PAL and NTSC television. I call this quadrature polarity-reversal keying, but everybody else calls it quaternary phase-shift keying, QPSK.

There is a 3-dB S/N penalty with QPSK, because we must split the transmitter power equally between the two channels. This is the same penalty as doubling the bandwidth, so we are no worse off. QPSK is therefore ideal for my planned comparison experiment: The adjacent-channel interference, the S/N and the multipath performance would be the same for both.

In the next section, I will think of QPSK not as two channels of binary data, but as a single-channel that can be switched to any of four 90° phase-shift values. By the way, the clock-recovery idea used for BPSK works just as well for QPSK, because the envelope still has a modulation component at the bit-rate.

QPSK and the Convolutional Code

There is a vast amount of available knowledge about correcting errors in data that are organized in blocks of constant length (such as ASCII codes) by transmitting longer blocks. I know of nothing that covers error correction of variable-length blocks like Varicode. There are ways of reducing errors in continuous streams of data with no block structure. (This seems a natural choice for a radio link, since its errors don't have any block structure either.) These are called convolutional codes. One of the simplest forms does actually double the number of data bits; it is therefore a natural choice for a QPSK channel carrying a variable-length code.

The convolutional encoder generates one of the four phase shifts, not from each data bit to be sent, but from a sequence of them. This means that each bit is effectively spread out in time, intertwined with earlier and later bits in a precise way. The more we spread it out, the better will be the ability of the code to correct bursts of noise, but we must not go too far or we will introduce too much transmission delay. I chose a time spread of five bits. The table that determines the phase shift for each pattern of five successive bits is given in the sidebar "The Convolutional Code." The logic behind this table is beyond the scope of this article.

In the receiver, a device called a

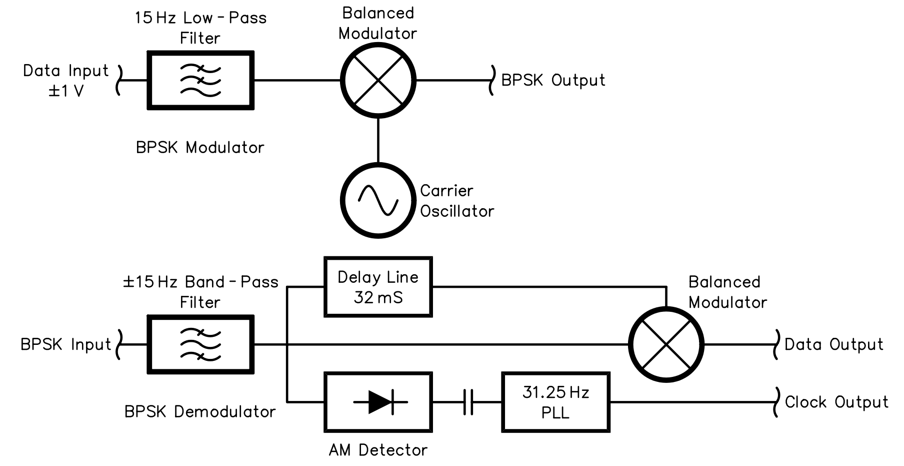


Fig 3—Block diagram of analog BPSK modem.

Viterbi decoder is used. This is not so much a decoder as a whole family of encoders playing a guessing game. Each one makes a different "guess" at what the last five transmitted data bits might have been. There are 32 different patterns of five bits and thus 32 encoders. At each step the phase-shift value predicted by the bit-pattern-guess from each encoder is compared with the actual received phase-shift value, and the 32 encoders are given "marks out of ten" for accuracy. Just as in a knockout competition, the worst 16 are eliminated and the best 16 go on to the next round, taking their previous scores with them. Each surviving encoder then gives birth to two "children," one guessing that the next transmitted bit will be a zero and the other guessing that it will be a one. They all do their encoding to guess what the next phase shift will be and receive scores again, which are added on to their earlier scores. The worst 16 encoders are killed-off again and the cycle repeats.

It's a bit like Darwin's theory of evolution, and eventually all the descendants of the encoders that made the right guesses earlier will be among the survivors and will all carry the same "ancestral genes." If we record the family tree (the bit-guess sequence) of each survivor, we can trace it back to find the transmitted bit-stream. We must wait at least five generations (bit periods), however, before all survivors have the same great great grandmother (who guessed right five bits ago). The whole point is that the scoring system based on the running total ensures that the decoder always gives the most-accurate guess, *even if the received pattern is corrupted.* Although we may need to wait a bit longer than five bit periods for the best answer to become clear. In other words, the Viterbi decoder corrects errors.

The longer we wait, the more accurate it is. I chose a decoder delay of four times the time spread, or 20 bits. We now have a 25-bit delay from one end to the other (800 ms), giving a round-trip delay to a two-way contact of 1.6 seconds. I think this is about the limit before it becomes a nuisance. In any case, the decoder could change to trade performance for delay without incompatibility.

QPSK on the Air

PSK31 operators find QPSK can be very good, but it is sometimes disappointing. In bench tests with white noise, it is actually *worse* than BPSK, confirming the simulation work mentioned earlier, but in conditions of burst noise, improvements of up to five times the character error-rate have been recorded. This performance doesn't come free, however. Apart from the transmission delay, which can be a bit annoying, QPSK is twice as critical in tuning as BPSK. A QPSK signal will start to decode wrong when the phase shift is greater than 45°, and that will be the case when the tuning error is only 3.9 Hz. This could be a problem with some older radios. What tends to happen is that contacts start on BPSK and change to QPSK if it is worth doing and there is no drift. There is one aspect of QPSK that must be kept in mind—it is important for both stations to use the correct sideband—on BPSK it doesn't matter.

Extending the Alphabet

In English-speaking countries, virtually all the characters and symbols that are needed for day-to-day written communications are present in the 128-character ASCII set. However, many other languages have accents, umlauts, tildes and other signs and symbols that are not in the ASCII set,

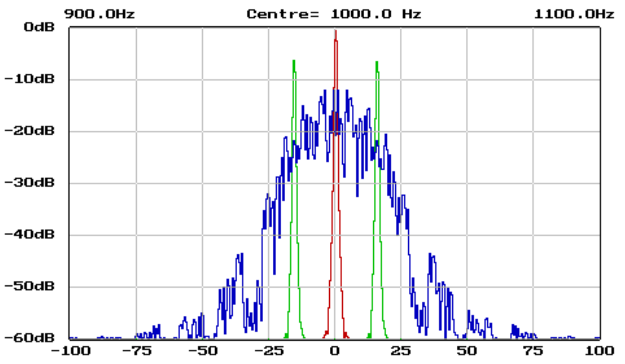


Fig 4—The spectrum of a BPSK signal, idling and sending data, compared with an unmodulated carrier at the same signal level. The carrier is the center pip; the smaller pips are the PSK31 reversals, and the large, ragged hump is noise shaped by the filter.

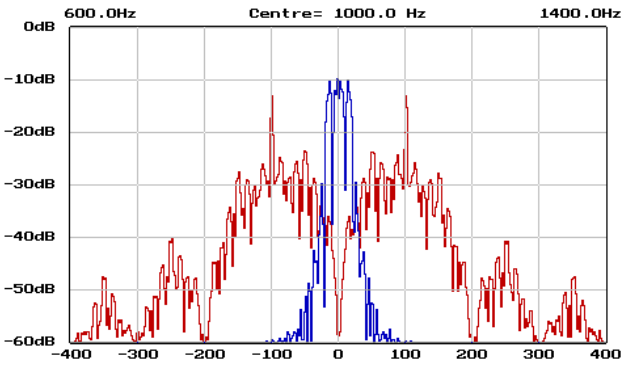
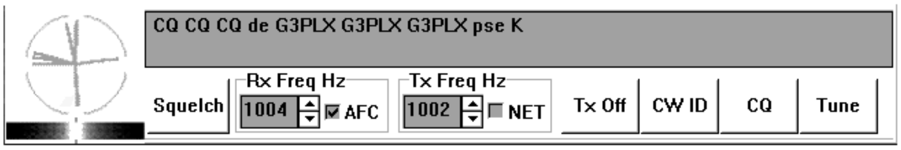


Fig 5—Comparison of the PSK31 spectrum with 100-baud, 200-Hz-shift FSK (AMTOR/PACTOR). The taller, three-hump signal at center is PSK31. The smaller, double-peak (±100 Hz) signal is FSK.



CQ CQ CQ de G3PLX G3PLX G3PLX pse K

Squelch | Rx Freq Hz 1004 ☑ AFC | Tx Freq Hz 1002 ☐ NET | Tx Off | CW ID | CQ | Tune

Fig 6—A screenshot of the *PSK31 Windows* program control panel, receiving a slightly noisy QPSK signal (notice the scope display at left). Fine-tuning controls for receive and transmit audio tones are near the bottom-center of the panel.

Is PSK31 Legal?

Some armchair lawyers have questioned the legality of PSK31 since its Varicode is not specifically mentioned as "legal" digital code in Part 97. Some confusion is understandable, give the wording of 97.309(a). However, the FCC clarified the meaning of the rules in an Order released October 11, 1995 (December 1995 *QST*, p 84). The Order (DA95-2106) reads in part: "This Order amends Section 97.304(a) of the Commission's Rules …to clarify that amateur stations may use any digital code that has its technical characteristics publicly documented. This action was initiated by a letter from the American Radio Relay League, Inc. (ARRL)."

The Order goes on to note that "The technical characteristics of CLOVER, G-TOR and PACTOR have been documented publicly for use by amateur operators, and commercial products are readily available that facilitate the transmission and reception of communications incorporating these codes. Including CLOVER, G-TOR and PACTOR in the rules will not conflict with our objective of preventing the use of codes or ciphers intended to obscure the meaning of the communication. We agree, therefore, that it would be helpful to the amateur service community for the rules to specifically authorize amateur stations to transmit messages and data using these and similar digital codes"

Given that PSK31 is in the public domain for amateur use, that software is readily and freely available and that its emission characteristics clearly meet the standards of Section 97.307 for RTTY/data, there is little doubt that its use by FCC-licensed amateur stations is legal.

However, just to complete the documentation, in a letter to the FCC dated January 27, 1999, ARRL General Counsel Christopher D. Imlay, W3KD, documented the technical characteristics of PSK31 in a manner similar to how CLOVER, G-TOR and PACTOR were previously documented. There is no need for PSK31 to be mentioned specifically in the rules, because CLOVER, G-TOR and PACTOR are simply given as examples.—*Dave Sumner, K1ZZ*

The Convolutional Code

The left-most numbers in each column contain the 32 combinations of a run of five Varicode bits, transmitted left bit first. The right-most number is the corresponding phase shift to be applied to the carrier, with "0" meaning no shift, "1" meaning advance by 90°, "2" meaning polarity reversal and "3" meaning retard by 90°. A signal that is advancing in phase continuously is higher in radio frequency than the carrier.

00000	2	01000	0	10000	1	11000	3
00001	1	01001	3	10001	2	11001	0
00010	3	01010	1	10010	0	11010	2
00011	0	01011	2	10011	3	11011	1
00100	3	01100	1	10100	0	11100	2
00101	0	01101	2	1010	3	11101	1
00110	2	01110	0	10110	1	11110	3
00111	1	01111	3	10111	2	11111	0

As an example, the "space" symbol, a single 1 preceded and followed by zeros, would be represented by successive run-of-five groups 00000, 00001, 00010, 00100, 01000, 10000, 00000, which results in the transmitter sending the QPSK pattern 2,1,3,3,0,1,2.

Note that a continuous sequence of zeros (the Varicode idle sequence) gives continuous reversals, the same as BPSK.

but are now used in everyday written text generated on computers. These extra symbols are now standardized worldwide in the ANSI alphabet, the first 128 characters of which are identical to ASCII, and the second 128 contain all the special symbols. Since the *WINDOWS* operating system uses ANSI, and most PC programs are now written for *WINDOWS*, I have recently extended the PSK31 alphabet in a *WINDOWS* version.

It is very easy to add extra characters to the Varicode alphabet without backwards-compatibility problems. In the early PSK31 decoders, if there was no "00" pattern received 10-bits after the last "00", it would simply be ignored as a corruption. In the extended alphabet, I let the transmitter legally send codes longer than 10 bits. The old decoders will just ignore them and the extended decoder can interpret them as extra characters. To get another 128 Varicodes means adding more 10-bit codes, all 11-bit and some 12-bit codes. There seemed little reason to be clever with shorter common characters so I chose to allocate them in numerical order, with code number 128 being 1110111101 and code number 255 being 101101011011. The vast majority of these will never be used, so it hardly slows the transmission rate at all, but it would not be a good idea to transmit binary files this way!

Summary

This article has identified some of the characteristics of modern HF data-transmission modes that have contributed to the decline in live QSO operation on these modes, while traditional RTTY is still widely used. By concentrating on the special nature of live-QSO operation, a new RTTY mode (I don't call it a "data" mode) has been devised, which uses modern DSP techniques and uses the frequency stability of today's HF radios. The bandwidth is much narrower than any other telegraphy mode. Fig 4 shows the spectrum occupied by PSK31 and Fig 5 compares this to the bandwidth of a PACTOR signal.

At the time of writing (February 1999) PSK31 is available for the Texas TMS320C50DSK with software written by G0TJZ, the Analog Devices ADSP21061 "SHARC" kit with software by DL6IAK and my own software for the Motorola DSP56002EVM. For the SoundBlaster card, DL9RDZ has written a *LINUX*-based program for the PC. Some commercially available DSP-based multimode controllers have already been upgraded to include PSK31 and more will follow. However, the most popular implementation of PSK31 so far is the *WINDOWS*-based soundcard program, which I have written for the soundcard. The DSP algorithms for PSK31 are being made available free-of-charge to bona-fide amateur programmers, so there should be a wide choice of PSK31 systems in the future.

News of the latest PSK31 developments and activity can be found at **http://aintel.bi.ehu.es/psk31.html**. The site also contains a link to information for those who want to implement their own PSK31 modem.