PSK31: A new radio-teletype mode with a traditional philosophy.

By Peter Martinez, G3PLX*

Part 1.

The writer has been active on RTTY since the 1960's, and was instrumental in introducing AMTOR to amateur radio at the end of the 70's. This improved the reliability of the HF radio link and paved the way to further developments which 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 is opening up 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 60's, 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 which I have developed specifically for live contacts, which is now becoming popular using low-cost DSP kits, and which could become even cheaper as the art of using PC sound cards is developed by amateur radio enthusiasts.

What is needed for a live contact?

I believe that it is the error-correcting process used in modern data modes which 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 450mS or 1.25 sec. or more, which will delay 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 out over a period of 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 is concerned with the way that the quality of the 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 your voice instinctively and even choose the topic of conversation to suit the conditions. In a digital mode the relationship between the signal-to-noise ratio on the air and the error-rate on 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 not linked, such as when calling CQ or listening to others. This makes it difficult to "getting to know" 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 (150mS), the flow of conversation is continuous and the error-rate is tolerable, and it is easy to listen-in and join-in.

How do we improve on RTTY?

How, then, do we go about using modern techniques that were not available in the 60's, to improve on traditional RTTY? First of all, 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. Secondly, modern transceivers are far more stable in frequency than they were in the 60's, so we should

be able to use much narrower bandwidths than in those days, and thirdly digital processors are much more powerful than the rotating cams and levers of the mechanical teleprinter, 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 the limitations of technology 30 years ago. We can do better now.

The PSK31 Varicode 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 synchronisation processes, is based firmly on another traditional, 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-synchronising: 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 corrupted. 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 keydown 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 analysed lots of English language text to find out how common were each of the ASCII characters, then allocated the shorter codes to the more common characters. The result is shown in the appendix 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 it's self-synchronising properties are working well.

The shortest code in Morse is the commonest letter "e", but in Varicode the shortest code is allocated to the wordspace. 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.

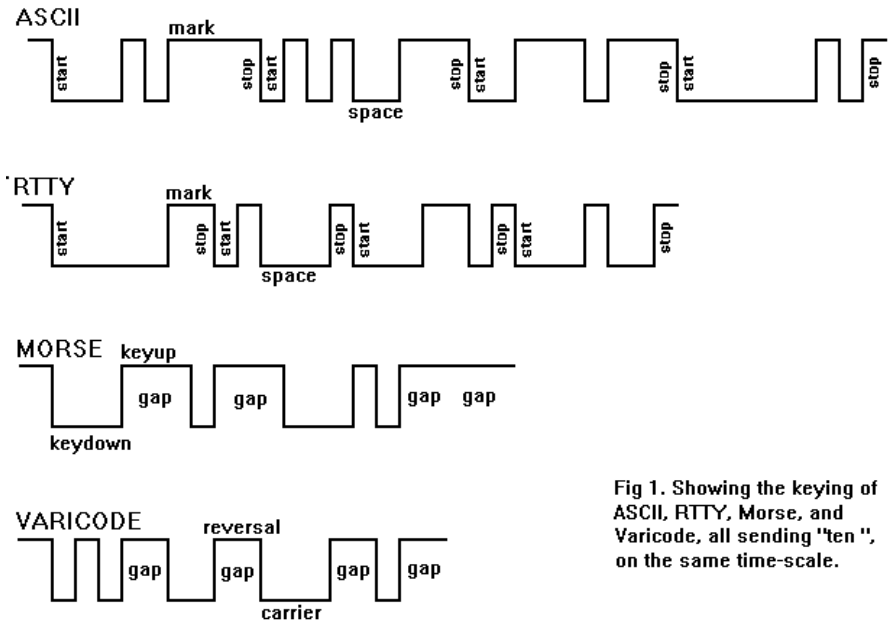


Fig 1. Showing the keying of ASCII, RTTY, Morse, and Varicode, all sending "ten ", on the same time-scale.

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 31.25Hz 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 signalled 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 sinewave 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 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.

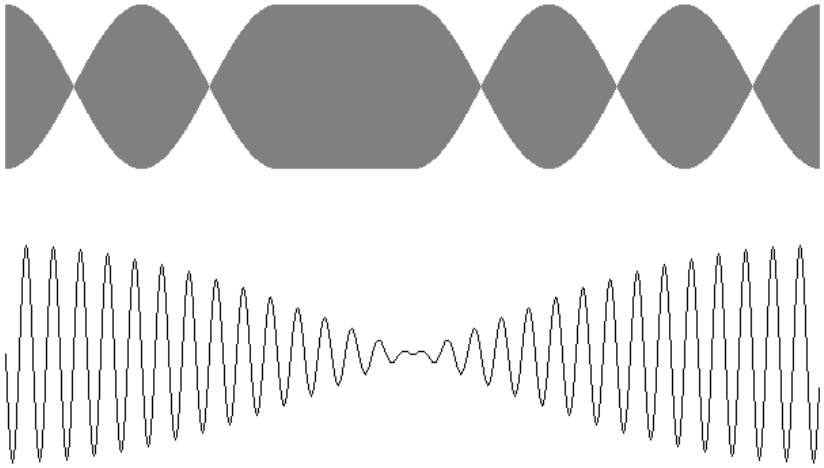


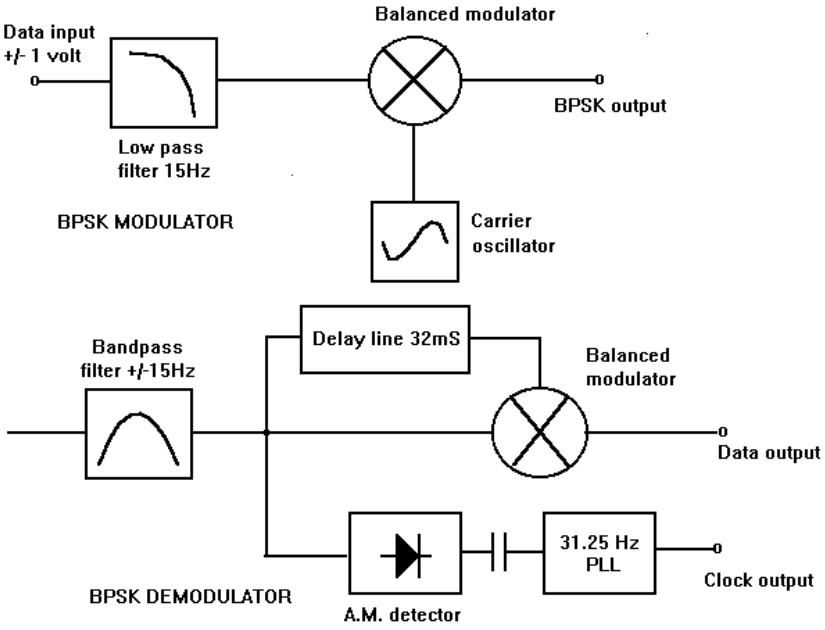
Fig 2. Showing the envelop of a BPSK carrier sending the Varicode "space' symbol, with a close-up of the phase-reversal at the cross-over.

To generate BPSK in it's simplest form we could convert our data-stream to levels of +/- 1 volt for example, take it through a low-pass filter, and feed it into a balanced modulator into which we also feed the desired carrier frequency. When sending continuous reversals, this looks like a 1 volt pk-pk sinewave going into a DSB modulator, so the output is a pure two-tone. In practice we use a standard SSB transceiver and perform the modulation at audio frequency or carry out the equivalent process in a DSP chip. We could signal a logic zero by continuous carrier and signal a one by a reversal, but I do it the other way round for reasons which will become clear shortly.

There are a variety of ways to demodulate BPSK, but they all start off with a bandpass filter. For the speed chosen for PSK31, this filter can be as narrow as 31.25 Hz in theory, but a "brick-wall" filter of precisely this width would be costly, not only in monetary terms but in the delay time through the filter, and we are trying to avoid delays. A practical filter might be twice the baud-rate (62.5Hz) wide at the 50dB-down point, and have a delay-time of two bits (64mS).

For the demodulation itself, since BPSK is equivalent to double sideband, the textbook method for demodulating DSB can be used, but another way is to delay the signal by one bit-period and compare it with the direct signal in a phase comparator. The output is negative when the signal reverses polarity and positive when it doesn't.

Although we could extract the information from the demodulated signal by measuring the lengths of the "dots and dashes" like we do by ear with Morse code, it will help to pick the data out of the noise if we know when to expect them. 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 synchronise 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, there is always a pure tone component in it at the baud-rate, and this can be extracted using a narrow filter or a phase-lock loop 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.



3. Block diagram of analogue BPSK modulator and demodulator.

For the synchronisation 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, and 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 10 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 pretty 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 so that the screen doesn't fill with noise when there is no signal.

Getting on PSK31

So much for the philosophy and the theory, but how do you get on the air with this mode? At the moment, the route to getting on PSK31 is 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. A number of 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 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 having done this yet, but probably the most promising hardware platform for the future will be the PC sound-card.

PSK31 operating

Since PSK31 performance is the same when calling, listening, or in contact, it's easy to progress from listening to others, to calling CQ, to two-way contacts and multiway nets, but the narrow bandwidth and good weak-signal performance do mean learning a few new tricks. It's usual to leave the radio dial on one spot and fine-tune the audio frequency, listening through the narrow audio filter rather than the loudspeaker, and using an on-screen phase-shift display to centre the incoming signal within a few Hz. 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 +/- 45Hz at a typical level of 36dB below P.E.P.

In part two of this article I will describe how I was persuaded to go back and take a second look at error-correction and describe how PSK31 has been extended to improve it's handling of languages other than English.

Appendix: The Varicode alphabet

ASCII	Varicode.				
0	1010101011	29	1110111011	:	11110101
1	1011011011	30	1011111011	;	110111101
2	1011101101	31	1101111111	<	111101101
3	1101110111	space	1	=	1010101
4	1011101011	!	111111111	>	111010111
5	1101011111	"	101011111	?	1010101111
6	1011101111	#	111110101	@	1010111101
7	1011111101	$	111011011	A	1111101
8	1011111111	%	1011010101	B	11101011
9	11101111	&	1010111011	C	10101101
l/feed	11101	'	101111111	D	10110101
11	1101101111	(	11111011	E	1110111
12	1011011101	)	11110111	F	11011011
c/ret	11111	*	101101111	G	11111101
14	1101110101	+	111011111	H	101010101
15	1110101011	,	1110101	I	1111111
16	1011110111	-	110101	J	111111101
17	1011110101	.	1010111	K	101111101
18	1110101101	/	110101111	L	11010111
19	1110101111	0	10110111	M	10111011
20	1101011011	1	10111101	N	11011101
21	1101101011	2	11101101	O	10101011
22	1101101101	3	11111111	P	11010101
23	1101010111	4	101110111	Q	111011101
24	1101111011	5	101011011	R	10101111
25	1101111101	6	101101011	S	1101111
26	1110110111	7	110101101	T	1101101
27	1101010101	8	110101011	U	101010111
28	1101011101	9	110110111	V	110110101

W	101011101	e	11	s	10111
X	101110101	f	111101	t	101
Y	101111011	g	1011011	u	110111
Z	1010101101	h	101011	v	1111011
[	111110111	i	1101	w	1101011
\	111101111	j	111101011	x	11011111
]	111111011	k	10111111	y	1011101
^	1010111111	l	11011	z	111010101
_	101101101	m	111011	{	1010110111
`	1011011111	n	1111	|	110111011
a	1011	o	111	}	1010110101
b	1011111	p	111111	~	1011010111
c	101111	q	110111111	127	1110110101
d	101101	r	10101		

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.

Part 2.

The first part of this article discussed the 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. In this part I will describe 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 in part 1, 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 works by adding redundant data bits. Suppose I devise an error-correcting system that doubles the number of transmitted bits. If I wanted to keep the traffic throughput the same, I would need to double the bit rate. But with BPSK that means doubling the bandwidth, so I lose 3dB of signal-to-noise ratio 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 SNR is good, the error-correction system does win, reducing the low error rate to very low levels, *but at the SNR 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 band space of course!

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. Forward error correction (FEC) seemed to deserve a second look provided that the transmission delay would not be too long.

I realised 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 it's bandwidth and speed. By adding a second BPSK carrier at 90 degrees at the transmitter and a second demodulator in the receiver, we can do the same trick that is used to transmit two colour-difference signals in PAL and NTSC television. I could call this quadrature polarity reversal keying, but everybody else calls it quaternary phase-shift keying or QPSK.

There is a 3dB signal-to-noise penalty with QPSK because we have to 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 SNR, and the multi-path performance would be exactly 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 which can be switched to any of four 90 degree phase-shift values. By the way, the clock recovery idea used for BPSK still works just as well on 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 which are organised in blocks of constant length such as ASCII codes, by transmitting longer blocks, but I know of nothing that covers error-correction of variable-length blocks like Varicode. However, there are ways of reducing errors in continuous streams of data which have no block structure and this seems a natural choice for a radio link since the errors don't have any block structure either. These are called convolutional codes, and one of the simplest forms does actually double

the number of data bits and 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, inter-twined 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 5 bits. The table that determines the phase shift for each pattern of 5 successive bits, is given in the appendix. The logic behind this table will not be covered here.

In the receiver, a device called a 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 5 transmitted data bits might have been. There are 32 different patterns of 5 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 like in a knock-out 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 are given marks out of ten 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". We therefore just keep a record of the family tree (the bit-guess sequence) of each survivor, and can trace back to find the transmitted bit-stream, although we have to wait at least 5 generations (bit periods) before all survivors have the same great great grandmother (who guessed right five bits ago). The whole point is that because the scoring system is based on the running total, the decoder always gives the most accurate guess *even if the received pattern is corrupted*, although we might need to wait a bit longer than 5 bits 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 4 times the time-spread, or 20 bits. We now have a 25 bit delay from one end to the other, (800mS), giving a round-trip delay to a two-way contact of 1.6 sec. 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 is sometimes disappointing. In bench tests with white noise, it is actually *worse* than BPSK, but in real band conditions with fading and interference, improvements of up to 5 times in the character error-rate have been recorded. This doesn't come free, however. Apart from the transmission delay, which can be a bit off-putting, QPSK with four phases instead of two, is twice as critical in tuning as BPSK, and needs to be within 4Hz. This could be a problem with some older radios. What tends to happen is that contacts start on BPSK and change to QPSK if both stations agree. There is one aspect of QPSK that has to be kept in mind - it is important for both stations to be using the correct sideband - on BPSK it doesn't matter.

Extending the alphabet

In the UK, our computer keyboards have a pound sign above the figure 3, and many people will have noticed that they can't reliably send pound signs, for example over Internet. This is because the Internet uses the 128-character ASCII alphabet, and the pound sign is not part of that set but part of the ANSI character-set which has an additional 128 characters and symbols. PSK31 as described so far is the same as Internet. It's a small problem in the UK but much more of a nuisance in other parts of the world where characters like the German umlauts, French accents, and Spanish tildes are also missing from the ASCII character set.. 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-compatability problems. In the earlier decoder, if there was no '00' pattern received 10-bits after the last '00', it would simply ignore it 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 ten-bit codes, all the eleven-bit ones, and some twelve-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. It would not be a good idea to transmit binary files this way!

Summarising

This article has tried to identify some of the characteristics of modern HF data-transmission modes that have contributed to the decline in "live QSO" operation on these modes, unlike traditional RTTY which is still widely used. By concentrating on the special nature of live-QSO operation, a new RTTY mode has been devised which uses modern DSP techniques and takes advantage of 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 standard FSK.

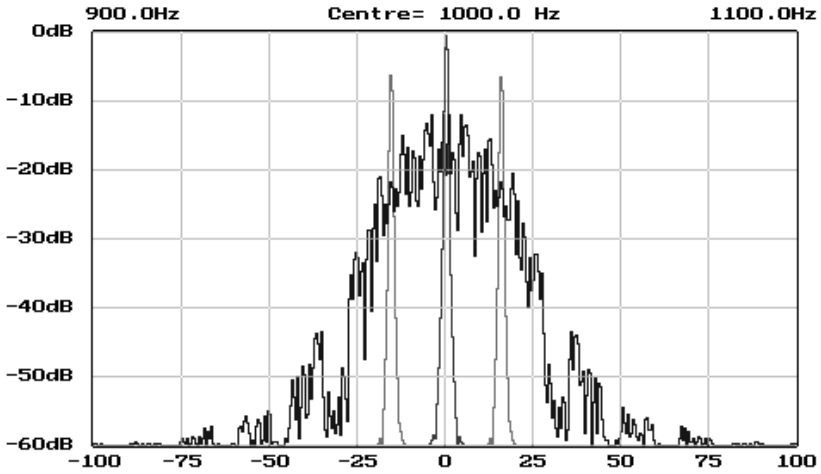


Fig 4. Spectrum of PSK31 sending carrier (red), reversals (green) and random data (blue)

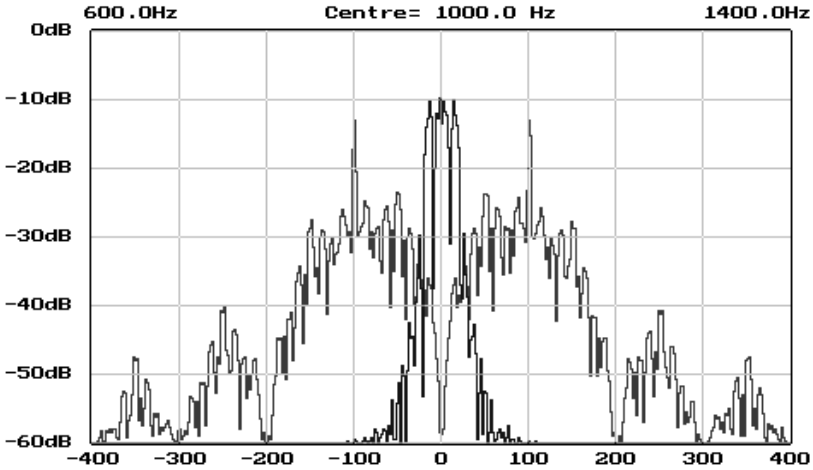


Fig 5. Comparing the spectrum of PSK31 with that of 200 Hz shift, 100 baud FSK (Amtor/Pactor)

At the time of writing (November 1998) PSK31 software is available for the Texas TMS320C50DSK, written by G0TJZ, the Analog Devices ADSP21061 "SHARC" kit, written by DL6IAK, and for the Motorola DSP56002EVM written by myself. For the Soundblaster card, DL9RDZ has written a LINUX-based program and I am in the process of writing a WINDOWS-based one. Two commercially-available DSP-based multi-mode controllers have already been upgraded to include PSK31.

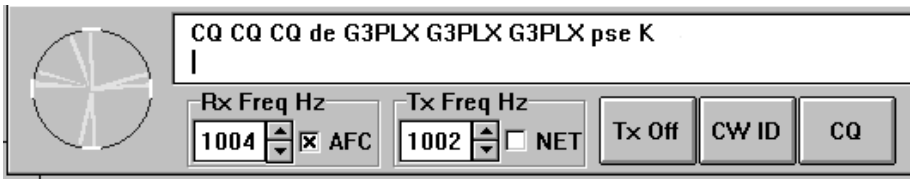


Fig. 6 Screenshot of the control panel of the PSK31 WINDOWS program, with the tuning display showing a slightly noisy QPSK signal, and showing the fine-tuning controls for the receive and transmit audio tones.

The two appendices contain sufficient information to define PSK31 for those that want to try it themselves. The available software and news of the latest developments and activity can be found on the PSK31 web page at http://aintel.bi.ehu.es/psk31.html.

Appendix: The convolutional code. The left column contains the 32 combinations of a run of five Varicode bits, transmitted left bit first. The right column 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 continuous phase advance is the same as an HF frequency shift.

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	10101 3	11101 1
00110 2	01110 0	10110 1	11110 3
00111 1	01111 3	10111 2	11111 0

For 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 idle sequence) gives continuous reversals, the same as BPSK.

Diagrams:

1. Showing the word "ten " keyed in ASCII, RTTY, Morse, and Varicode.
2. Showing the waveform of BPSK sending the Varicode "space" symbol.
3. Block diagram of analogue BPSK modulator and demodulator.
4. Showing the spectrum of the BPSK signal, idling and sending data, compared with an unmodulated carrier at the same signal level.
5. Comparison of spectrum of PSK31 with 100 baud, 200Hz shift FSK. <psk31f5.bmp>
6. Screenshot of the control panel of the PSK31 WINDOWS program, with the tuning display showing a slightly noisy QPSK signal, and showing the fine-tuning controls for the receive and transmit audio tones.

* High Blakebank Farm, Underbarrow, Kendal, Cumbria LA8 8HP