Einführung in die digitale Signalverarbeitung - Teil 3

Was steckt eigentlich hinter PSK31?

Dipl.-Ing. Erich H. Franke, DK6II

erich.franke@afusoft.com

Wir haben uns in den vorangegangenen Teilen dieser Reihe durch die Grundbegriffe der Signalverarbeitung gearbeitet und danach einen einfachen Filterkonverter für RTTY in Software nachgebaut. Nun wollen wir uns an eine der moderneren digitalen Betriebsarten wagen, wie sie im Amateurfunk auf Kurzwelle angewandt wird. Wir sprechen von PSK31. Dazu werden wir uns die zugrunde liegende Modulationsart, nämlich die Phasenumtastung genau so anschauen wie einige der Tricks, die die Autoren von PSK31-Software in der Codierung angewandt haben.

Königsbach-Stein, im Juli 2011

Die Betriebsart PSK-31in Kurzform

Die Betriebsart PSK31 war ursprünglich von Peter Martinez, G3PLX im Amateurfunk eingeführt worden, um auf Kurzwelle die Übertragung von Texten mit geringer Bandbreite durchführen zu können und trotzdem eine gute Resistenz gegen Störungen (QRM, QRN) und Fading (QSB) zu erhalten.

G3PLX hat als Vorreiter bei den digitalen Betriebsarten im Amateurfunk eine lange Tradition. Er hatte beispielsweise früher einmal das AMTOR-Verfahren aus dem kommerziellen SITOR abgeleitet und dieses im Amateurfunk eingeführt.

PSK31 kann als Weiterentwicklung des Verfahrens SLOWBPSK angesehen werden, welches ursprünglich von Paweł Jałocha SP9VRC stammt. Paweł's grundlegende Idee dabei war, die normalerweise übliche Frequenzumtastung durch Phasenumtastung, also *PSK* zu ersetzen.

PSK31 ist kein „Protokoll“ im engeren Sinne. Es überträgt Daten zeichenweise, denn es ist als eine Weiterentwicklung des guten alte RTTY gedacht und wie das Funkfernschreiben für reine *Textübertragung* optimiert. PSK31 enthält lediglich in geringem Umfang eine Fehlerkorrektur, jedoch keine Maßnahmen für Datenwiederholungen im Fehlerfall wie SITOR/AMTOR, Packet Radio oder PACTOR.

Vergleicht man PSK31 mit RTTY, so bringen beide Betriebsarten in etwa den gleichen Durchsatz für geschriebene Texte.

Interessant ist jedoch der deutlich geringere Bandbreitebedarf, vergleich man PSK31 mit RTTY bei den im Amateurfunk typischen Parametern, also einer Schrittgeschwindigkeit von 45,45 Baud und einer Frequenzshift von 170 Hz.

Bei typischen Schreibgeschwindigkeiten belegt RTTY auf dem Funkkanal eine Bandbreite von etwas über 230 Hz.

PSK31 hingegen kommt bei gleichem Datendurchsatz mit einer Kanalbandbreite von deutlich unter 90 Hz aus. In der Theorie ergibt sich gar eine Bandbreite von lediglich 31.25 Hz.

Der Grund für den eklatanten Unterschied liegt in der verwendeten Modulationsart.

RTTY benutzt wie allseits bekannt die alt bewährte Frequenz-Umtastung (*Frequency shift keying*, FSK). Hierbei wird wie in [Abb. 1] gezeigt zu jedem Zeitpunkt jeweils eine von zwei Frequenzen ausgesendet und diese im Takt der zu übertragenden Datenbit hin- und hergeschaltet.

PSK31 hingegen benutzt wie der Name schon vermuten lässt die Phasen-Umtastung (*Phase shift keying*, PSK) eines in der Frequenz konstanten Trägersignals als zur Modulation.

Durch den Einsatz der PSK erhalten wir eine ganz ausgezeichnete Frequenzökonomie. Wir werden uns dies weiter unten näher ansehen.

Der geringere Bandbreitenbedarf gibt uns noch einen weiteren Bonus: Da sich bei PSK31 die gesendete Energie auf eine geringere Bandbreite verteilt, erhalten wir eine energieeffizientere Übertragung als bei der breiteren FSK.

Will heißen, dass wir bei PSK gegenüber FSK auf der Empfangsseite die gleiche Bitfehlerrate mit deutlich geringerer Sendeleistung erzielen können.

Dieser „Systemgewinn“ macht PSK für QRP-Anwendungen attraktiv und erlaubt damit auch Funkverkehr in Situationen mit schlechten Antennen und niedriger Sendeleitung.

PSK31 bedient sich noch eines weiteren Tricks um den effektiven Datendurchsatz zu steigern. Es führt nämlich die Textübertragung mittels Codes variabler Länge durch.

G3PLX spricht in seinem Papier von einem „Varicode“ [G3PLX].

Die Idee hinter diesem Verfahren besteht darin, häufig verwendete Zeichen mit einer geringeren Anzahl an Bit zu codieren als selten verwendete.

Klingt bekannt? Ja: Der gute alte Morsecode beruht auf dem gleichen Prinzip.

Ebenfalls werden Codes variabler Codewortlänge, die die statistischen Unterschiede in der Häufigkeit des Auftretens der zu codierenden Daten bei der drahtgebundenen Übertragung von Faksimile verwendet.

In der Literatur findet man sie häufig unter dem Begriff „Huffman-Code“

Auch diesen Aspekt werden wir detailliert betrachten.

Phasenumtastung, was ist das?

Die Idee der Einführung von BPSK in den Amateurfunk stammte ursprünglich von SP9VRC mit seinem SLOWBPSK. Eine ganze Reihe von Amateuren experimentiert inzwischen auf diesem Gebiet. Das CLOVER-System beispielsweise versuchte bereits vor vielen Jahren BPSK, QPSK und höhere Verfahren wie QAM im Amateurfunk einzuführen – jedoch ohne wesentlichen Erfolg.

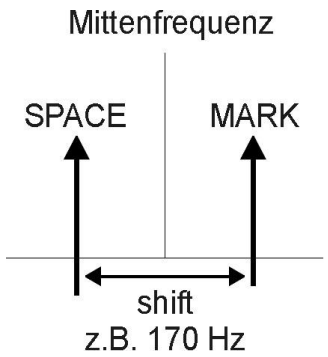

Abb.1: Lage der Kennfrequenzen bei RTTY

In anderen Systemen, beispielsweise bei der digitalen Fernsehübertragung hingegen ist die Phasenumtastung bereits seit langem Stand der Technik.

Proprietäre Verfahren und Systeme, welche eine teuere Hardwareausstattung benötigen waren bislang in diesem Segment nicht durchzusetzen, obgleich zahlreiche kommerzielle und militärische Verfahren PSK bereits seit langer Zeit nutzen. Ich habe darüber bereits an anderer Stelle ausführlich berichtet [UKW95].

Erst durch die Verbreitung von Public Domain Softwareprogrammen, die auf handelsüblichen PC laufen und die die meist sowieso verfügbare Soundkarte benutzen wurden derartige „digitale“ Verfahren in Amateurfunkkreisen salonfähig.

Das Programm WinPSK von AE4JY ist ein exzellentes Beispiel. Moe veröffentlicht nicht nur ein gut verwendbares Programm sondern was noch viel wichtiger ist, auch die Quelltexte. Des weiteren beschreibt er in seinem absolut lesenswerten technischen Handbuch [AE4JY] den Programmaufbau detailliert, so dass jeder Interessent eigene Arbeiten darauf aufsetzen kann.

Nun aber erst einmal zu den Grundlagen der Phasenumtastung:

Wir können uns diese am Beispiel eines NF-Signals anschauen. Die Modulation beruht darauf, zu bestimmten Zeitpunkten die Phasenlage um 180° zu verschieben.

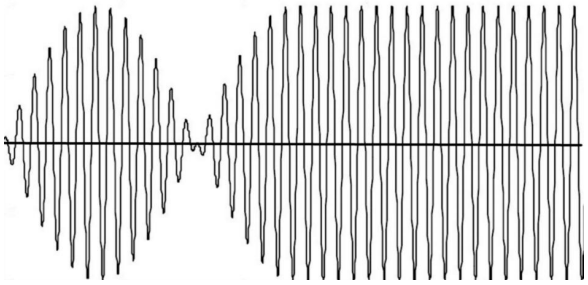
Abb.2: Amplituden/Phasenverlauf bei der Umtastung

Doch hat dieses Verfahren seine Tücken: Bereits im Teil 2 unserer Reihe haben wir gesehen, dass abrupte Phasensprünge die Bandbreite eines ausgesendeten Signals dramatisch erhöht.

Um die PSK für den Einsatz im Funk zu zähmen, tastet man die Phasenlage also nicht „hart“ und sprunghaft um, sondern lässt den Verlauf der Umtastung dem einer Kosinus-Funktion folgen, wodurch Sprünge vermieden werden. [Abb. 2] zeigt den Verlauf der Zeitfunktion während eines Phasenwechsels.

Nun ist aber die *absolute* Phasenlage ein schlechtes Kriterium zur Codierung binärer Werte. Es ist technisch nicht sinnvoll, der *absoluten* Phasenlage 0° die binäre NULL und der *absoluten* 180° eine EINS zuzuordnen. Das würde nicht funktionieren!

Man bräuchte nämlich ein unabhängiges Referenzsignal um erkennen zu können, welche *absolute* Phasenlage das modulierte Signal momentan gerade besitzt.

Da sich darüber hinaus die Phasenlage bedingt durch die Signallaufzeit dauernd verschiebt fällt diese Methode für den praktischen Einsatz aus.

Daher codieren PSK-Systeme die binären Modulationswerte in Wirklichkeit in die Phasen-*Differenz*, also an Hand der Tatsache, dass sich die Phase gerade verändert oder eben nicht. Man spricht daher korrekterweise von einer „Differenz-PSK“ oder kurz DPSK.

Jedes „Ereignis“, also die Tatsache eines Phasensprungs oder dessen Ausbleiben wird in der Literatur gerne als „Symbol“ bezeichnet. Ein Symbol ist aber nicht notwendigerweise einem Bit zugeordnet.

In einem binären System mit zwei Kennzuständen, also beispielsweise der FSK bei RTTY und BPSK (2-PSK) werden nur zwei „Symbole“ benutzt. Tritt das eine „Symbol“ auf, so würde dies einer EINS entsprechen. Die Tatsache, dass das andere Symbol erscheint repräsentiert dann die NULL.

Werden pro Kennzustand mehrere Bit codiert sieht dies anders aus:

Werden wie beispielweise bei der QPSK (4-PSK) mit vier Phasenzuständen vier Symbole pro Zeiteinheit übertragen, so können durch diese vier disjunkten Zustände zwei Bit gleichzeitig codiert und mithin übertragen werden.

In diesem Fall wäre die Bitrate das Doppelte der Symbolrate, also der Baudrate.

Die Symbolrate wird übrigens zu Ehren des Informationstheoretikers Emile Baudot auch als „Baudrate“ bezeichnet. Nur im Falle eines binären Systems entspricht die Baudrate der Bitrate. Bei einer vierwertigen Modulation, beispielsweise der QPSK werden ja zwei Bit quasi gleichzeitig übertragen und wir erhalten eine Bitrate, welche doppelt so hoch ist wie die Symbol- bzw. Baudrate.

Werden Symbole in der Literatur diskutiert, so zeichnet man ihre Lage gerne in der „komplexen“ Zahlenebene ein.

Erinnern wir uns an unsere Gymnasialzeit, so wurden schon damals viele von uns mit dem Begriff der „komplexen Zahlen“ konfrontiert. Physiker und Elektrotechniker haben damit spätestens durch die Theorie der Wechselströme zu tun bekommen.

Im Gegensatz zu reellen Zahlen, die wir hintereinander und „linear“ auf einer Geraden, dem alt bekannten „Zahlenstrahl“ anordnen können, erstrecken sich „komplexe“ Zahlen in die Ebene.

Man kann die Zahlenwerte anschaulich gut mit Punkten auf einer topographischen Landkarte vergleichen. Geoinformatiker und Kartographen unter Ihnen sowie Informationstheoretiker mögen nun kurz wegzuhören oder die Augen schließen. Mir kommt es bei meinen Erklärungen insbesondere auf die Anschauung an und nicht auf wissenschaftliche Vollständigkeit!

Jeder Punkt, der auf der Karte eine Position innehat besitzt eine geographische Länge, die sich mehr oder minder in Richtung der X-Achse erstreckt. Die geographische Breite hingegen folgt der Richtung der Y-Achse.

Eine komplexe Zahl besitzt genau wie ein Punkt auf einer Landkarte zwei Komponenten. Die eine wird durch den so genannte Realteil gekennzeichnet, den man mit der geographische Länge vergleichen könnte und folglich auf der der X-Achse aufträgt.

Die zweite Komponente der komplexen Zahl ist der Imaginärteil, der in Richtung der Y-Achse aufgetragen wird.

Jeder Punkt in der komplexen Ebene kann wie auf einer Landkarte durch seine X- und Y-Koordinate bezeichnet werden. Da diese Koordinaten auf einem Koordinatensystem beruhen, dessen Achsen senkrecht aufeinander stehen, spricht man von einem kartesischen Koordinatensystem.

Man kann nun jederzeit einen Vektor zeichnen, der sich vom Koordinatenursprung bis zu dem Punkt erstreckt, der die Zahl repräsentiert.

Bleibt seine Länge ℓ konstant, dann liegen alle Punkte, die der Vektor bei Veränderung seines Winkels zur X-Achse „erreichen“ kann auf einem Kreis um den Koordinaten-Ursprung.

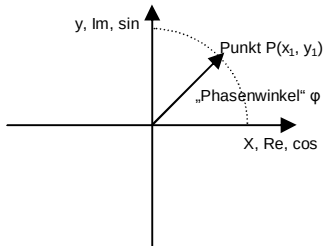

Abb.3a: Vektor-Darstellung

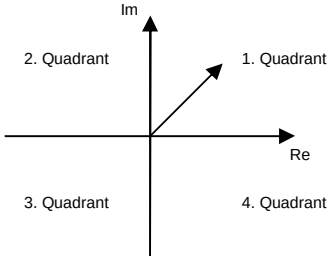

Abb.3b: Vier Quadranten

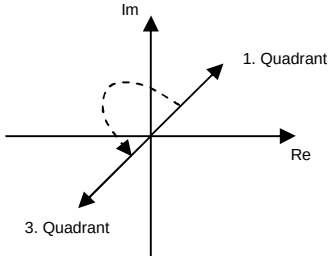

Abb.3c: Der Sprung um 180°

Wir können also grundsätzlich jeden Punkt in der XY-Ebene nicht nur über seine kartesischen Koordinaten, also x_1 und y_1, sondern auch über seine Länge ℓ und den Winkel φ ansprechen, den der Vektor zur X-Achse einnimmt. Die Kombination aus Länge und Winkel bezeichnen wir auch als seine Polarkoordinaten.

Wir können die Polarkoordinaten in die kartesischen Koordinaten mit den Gleichungen $x_1 = l * \cos(\varphi)$ und $y_1 = l * \sin(\varphi)$ umrechnen, wie man der Abb.3a mit etwas Trigonometrie unschwer entnehmen kann. Der Vektor kann wie in Abb.3b gezeigt in unterschiedliche „Quadranten“ zeigen, in die die Ebene durch die Koordinatenachsen aufgeteilt wird.

Diese Dosis an Mathematik ist erst einmal ausreichend, um die „differentielle“ Phasenmodulation zu verstehen:

Wir gehen dazu davon aus, dass unser Signal wie üblich zu festen, mithin äquidistanten Zeitpunkten abgetastet wird. Bei PSK31 wird eine Symbolrate von 31.25 Baud benutzt. Wir suchen demnach in diesem Zeitraster nach Phasenwechseln.

Im Falle der zweistufigen BPSK treten nun folgende Fälle auf:

- Springt zum Abtastzeitpunkt die Phase φ des Trägersignals um 180°, so wechseln sowohl der Sinus als auch der Kosinus seine Vorzeichen. Es ist dabei gleichgültig, ob der Ausgangspunkt der erste oder dritte Quadrant war.
    - Der Vektor springt vom ersten in den dritten Quadranten hinein [Abb. 3c].
      Der Sprung erfolgt also über *zwei* Quadranten.
      Diesem Fall ist das Symbol 2, oder binär ‚10' zugeordnet.
    - Gleiches passiert bei einem erneuten Sprung, dieses Mal vom dritten zurück in den ersten.
      Wieder springt der Vektor über *zwei* Quadranten. Das Symbol ist wieder ‚10'.
- Springt zum Abtastzeitpunkt die Phase φ des Trägersignals jedoch nicht, so können wir dies als einen – entarteten - Sprung über Null Quadranten sehen. Dem ist das Symbol 0, oder binär ‚00' zugeordnet.

Aus den beiden Symbolen ‚10' und ‚00' machen wir nun ein Binärsignal NULL und EINS wie folgt:

Sprung-Differenz	Bemerkung	n	Symbol	Bit
0° (kein Sprung)	kein Sprung	0	00	EINS
180°	Sprung über <n> = 2 Quadranten	2	10	NULL

Im Falle der vierstufigen QPSK haben wir es mit folgenden Fällen zu tun:

Sprung-Differenz	Bemerkung	n	Symbol
0° (kein Sprung)	kein Sprung	0	00
180°	Sprung über <n> = 2 Quadranten	2	10
+90°	Sprung entgegen dem Uhrzeigersinn, d.h. mathematisch positiv um einen Quadranten	1	01
-90°	Sprung im Uhrzeigersinn, d.h. mathematisch negativ um einen Quadranten. Man kann dies auch als einen mathematisch positiven Sprung um 270° und damit über drei Quadranten ansehen.	3	11

Bei der QPSK haben wir also *vier* Möglichkeiten für unterscheidbare Sprünge. Damit können wir pro Symbol *zwei* Nutzbit codieren und theoretisch die doppelte Bitrate bei gleicher Symbolrate übertragen.

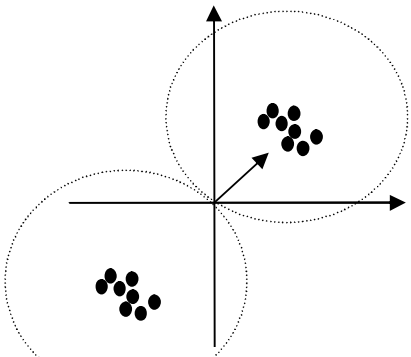

Abb.3d: Entscheider-Schwellen

PSK31 erlaubt grundsätzlich die Benutzung von 2-PSK und 4-PSK.

Doch auch ohne die Informationstheorie zu bemühten ist einzusehen, dass 4-PSK gegenüber 2-PSK nur einen geringeren Störabstand erzielen kann.

Die „Kreise", in denen empfange Signale in der Ebene liegen, können deutlich besser getrennt werden, wenn wir wissen, dass gültige Werte nur in zwei Quadranten liegen können. Man sagt, dass die „Entscheider-Schwellen" wesentlich günstiger liegen als wenn vier Quadranten unterschieden werden müssen.

Daher nutzt PSK31 das Extra an Übertragungsgeschwindigkeit bei 4-PSK nur zum Teil, denn es nutzt ein gehöriges Stück der Redundanz, um einen fehlerkorrigierenden Code (*Error Correcting Code, ECC; bzw. Forward Error Correction, FEC*) zu implementieren.

Dieser betrachtet fünf Nutzbit als Ganzes und bildet aus deren *Kombination* einen Phasensprung. Dadurch sieht so aus, als ob jedes Nutzbit über 160 Millisekunden hinweg „verschmiert" würde.

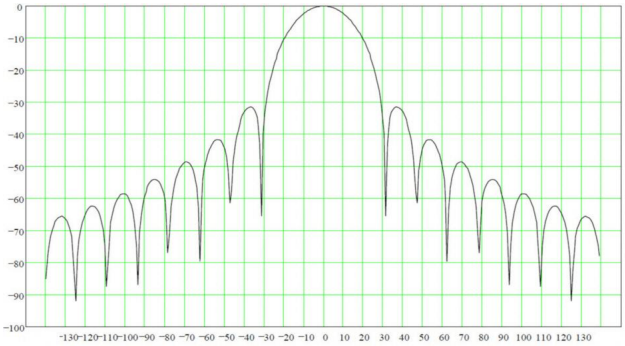

Abb.4: Idealisiertes BPSK-Spektrum

Das Verfahren ist jedoch im praktischen Einsatz im Kurzwellenkanal, in dem Burststörer vorherrschen weit weniger effizient als man annehmen möchte.

Ein hübscher Effekt aus der Verwendung der PSK zeigt sich, wenn man das Signalspektrum anschaut.

Die 3dB-Bandbreite beträgt in der Tat eben einmal besagte 32 Hz. Bei -30dB erweitert sich die Bandbreite auf ungefähr 60 Hz.

Damit ist dieses Modulationsverfahren für die dicht besiedelte Kurzwelle sehr gut geeignet.

Abb. 4 zeigt das Signal idealisiert durch MathCAD berechnet. Jedoch sind die Verhältnisse in der Praxis tatsächlich ähnlich geartet.

Auch sind wir bei dieser geringen Bandbreite hinreichend resistent gegen das frequenzselektive Fading, das uns in RTTY bei 2-FSK selbst bei einer Shift von 170 Hz manchmal Kummer macht.

Bei Bandbreiten kleiner 100 Hz sind die Frequenzlinien im Kanal im Allgemeinen ausreichend korreliert [Abb.5].

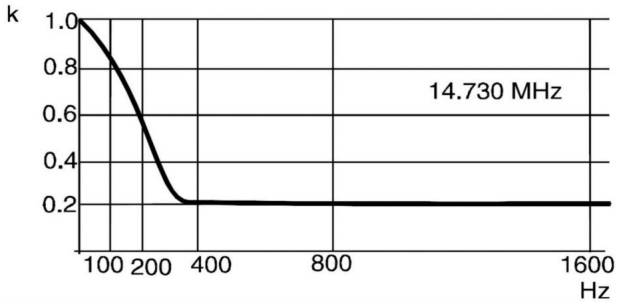

Abb.5: Korrelation benachbarter Frequenzen im Kurzwellenkanal (nach Wiesner)

Ein Nachteil der PSK liegt jedoch in der hohen Anforderung an Frequenzkonstanz und Wiederkehrgenauigkeit der verwendeten Funkgeräte.

Zur Einhaltung der spezifizierten Systemwerte müssen daher Vorkehrungen wie eine aktive Frequenzregelung (AFC) getroffen werden, um Datenübertragung über einen längeren Zeitraum aktiv zu halten.

Diese Aufgabe erledigt jedoch die Decoder-Software dadurch, dass sie intern synchron arbeitet und dadurch den Empfangschritt-Takt mit den Phasenwechseln des Eingangssignals korrelieren kann.

Das Gesamtsystem muss jedoch damit klar kommen, dass im „worst worst-case“, also im allerschlimmsten Fall bei einem gültigen Zeichen (Code 231, 0xE7) höchstens elf EINS aufeinander folgen was bedeutet, dass für die Dauer von 384 Millisekunden kein Phasenwechsel stattfindet, der Schrittakt jedoch völlig synchron weiter laufen muss, was aber zugegebenermaßen bei einem einigermaßen guten Design kein wirkliches Problem darstellt.

Im Bereich der ASCII-Zeichen fällt das Ausrufezeichen mit neun Einsen auf. Über diesen Zeitraum muss die Software also mindestens völlig synchron bleiben. Bei „häufig benutzten“ Zeichen hingegen ist dieser Zeitraum weitaus kürzer.

Daher ist es angebracht, bei der Auswahl der Texte die Varicode-Tabelle im Auge zu behalten.

Der Varicode und seine Auswirkungen

Da PSK31 für reine Textübertragung ausgelegt ist, kann sich das Programm eines weiteren Tricks bedienen um den Datendurchsatz zu steigern.

Die letztendlich für Menschen lesbaren Texte werden nämlich mittels eines Codes variabler Codewortlänge codiert. G3PLX spricht in diesem Zusammenhang vom „Varicode“.

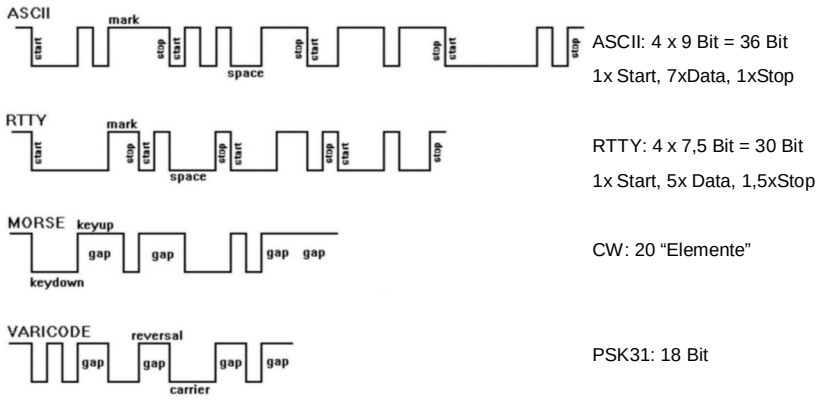

Abb.6: Unterschiedliche Datenmengen für die Übertragung des Wortes „ten_“. (Quelle G3PLX)

Die Idee dahinter besteht darin, häufig verwendete Zeichen mit einer geringeren Anzahl an Bit zu codieren als seltener verwendete [Abb.6].

Der Morsecode beruht beispielsweise auf diesem Prinzip.

Ebenso die spezialisierte Form des „Huffman-Code“ bei der drahtgebundenen Übertragung von Faksimile.

In gewisser Weise kann man auch die Codierung internationaler Zeichensätze in Unicode hiermit vergleichen.

Es ist nicht auf den ersten Blick ersichtlich, weshalb G3PLX einfach nicht einen „echten“ Huffman-Code zur Codierung genutzt hat.

Bei einem „echten“ Huffman-Code ist sichergestellt, dass man jedes einzelne Codewort aus dem *seriellen* Datenstrom in eindeutiger Weise extrahieren kann. Niemals tritt der Fall auf, dass ein gültiges Zeichen mit der gleichen Folge beginnt mit der ein anderes codiert ist.

Doch G3PLX hatte vermutlich eine asynchrone Datenübertragung im Sinn, bei der ein Mensch eine Tastatur bedient und eben mehr oder weniger lange Pausen beim Tippen macht.

Wie beim asynchronen RTTY dürfen zwischen den Zeichen ein beliebiger Leeraum („gap“) auftreten.

Doch anders als bei der klassischen asynchronen Übertragung benutzt PSK31 weder Start- noch Stoppbits.

Stattdessen wird bei PSK31 der „Ruhezustand“ durch zwei oder mehr aufeinanderfolgende Bit mit Wertigkeit NULL gekennzeichnet.

Diese NULL ergeben gemäß obiger Festlegung einen dauernden Phasenwechsel, mit der zwei Stationen ihre internen Timer auf einfache Weise synchronisieren können.

Eigentlich gar keine so schlechte Idee.

	0	1	2	3	4	5	6	7
0	NUL 1010101011	16 1011110111	space 1	0 10110111	@ 1010111101	P 11010101	` 1011011111	p 111111
1	1011011011	17 1011110101	! 111111111	1 10111101	A 1111101	Q 111011101	a 1011	q 110111111
2	1011101101	18 1110101101	" 101011111	2 11101101	B 11101011	R 10101111	b 1011111	r 10101
3	1101110111	19 1110101111	# 111110101	3 11111111	C 10101101	S 1101111	c 101111	s 10111
4	1011101011	20 1101011011	$ 111011011	4 101110111	D 10110101	T 1101101	d 101101	t 101
5	1101011111	21 1101101011	% 1011010101	5 101011011	E 1110111	U 101010111	e 11	u 110111
6	1011101111	22 1101101101	& 1010111011	6 101101011	F 11011011	V 110110101	f 111101	v 1111011
7	BEL 1011111101	23 1101010111	' 101111111	7 110101101	G 11111101	W 101011101	g 1011011	w 1101011
8	1011111111	24 1101111011	(11111011	8 110101011	H 101010101	X 101110101	h 101011	x 11011111
9	TAB 11101111	25 1101111101	) 11110111	9 110110111	I 1111111	Y 101111011	i 1101	y 1011101
A	LF 11101	26 1110110111	* 101101111	: 11110101	J 111111101	Z 1010101101	j 111101011	z 111010101
B	1101101111	ESC 1101010101	+ 111011111	; 110111101	K 101111101	[111110111	k 10111111	{ 1010110111
C	1011011101	28 1101011101	, 1110101	< 111101101	L 11010111	\ 111101111	l 11011	| 110111011
D	CR 11111	29 1110111011	- 110101	= 1010101	M 10111011	] 111111011	m 111011	} 1010110101
E	1101110101	30 1011111011	. 1010111	> 111010111	N 11011101	^ 1010111111	n 1111	~ 1011010111
F	1110101011	31 1101111111	/ 110101111	? 1010101111	O 10101011	_ 101101101	o 111	DEL 110110101

Abb.7: Varicodes für ASCII-Codes im Bereich zwischen 0x00 und 0x7F

Das bedeutet aber auch, dass innerhalb eines gültigen Codewortes niemals zwei Bit des Wertes NULL direkt aufeinander folgen dürfen.

Diese inhärente Synchronisation macht das Verfahren in der Praxis auch bei im „Adler-Suchsystem“ ausgeführten Fernschreibbetriebs ausreichend stabil.

Die Codetabelle in Abb.7 enthält ein paar interessante Details. So werden beispielsweise Kleinbuchstaben deutlich kürzer codiert als Großbuchstaben. Daran sollte man denken, wenn man in PSK31 Texte verfasst. Die Übertragung von Großbuchstaben und Sonderzeichen kosten deutlich mehr Zeit als die von Kleinbuchstaben.

Der Coderaum zwischen 128 und 255 enthält mithin die längsten Codes. Wie gesagt, PSK31 ist ein Fernschreibsystem und kein Verfahren für die Datenübertragung.

Ein wenig spaßig mutet an, dass G3PLX in seinem Text die Corierung von PSK31 mit der des Morsecodes vergleicht und dabei nachzuweisen versucht, sein Code sei „bit-effizienter“ ist als der bei der Morseübertragung verwendete [Abb.8]. Ich schreibe dies Peters Begeisterung zu.

Code		t		e		n		<leer>		n	Bemerkung
ASCII, 7Bit	0	0010111	10	1010011	10	0111011	10	0000010	1	36	
Morse	-	111	000	1	000	11101	000	00	-	20	Wortabstand = 5 ElAbst
Varicode	-	101	00	11	00	1111	00	1	00	18	

Abb. 8: Anzahl der Bit im Vergleich zwischen den Betriebsarten ASCII, Morse und PSK31

In dieser Tabelle werden die Bit von links nach rechts gesendet. G3PLX vergleicht seinen Varicode mit dem im U.S. Fernschreibbetrieb üblichen aber sonst eher exotischen 7-Bit-ASCII… Wir zählen die Start- und Stopbit zu den „Leerstellen“.

Selbstverständlich würde man leicht auch Gegenbeispiele finden. Trotzdem bleibt angemerkt, dass das Verfahren wirklich gute Ideen enthält und diese auch gut umsetzt. Es ist ein sehr positives Beispiel für gutes Engineering und Praxistauglichkeit im Amateurfunk.

Die Implementierung in der Software

Es gibt im Internet inzwischen eine ganze Sammlung von Programmen, welche digitale Betriebsarten auf handelsüblichen PC implementieren. Fast allen ist gemein, dass sie als Schnittstelle zum Sendeempfänger die Soundkarte benutzen.

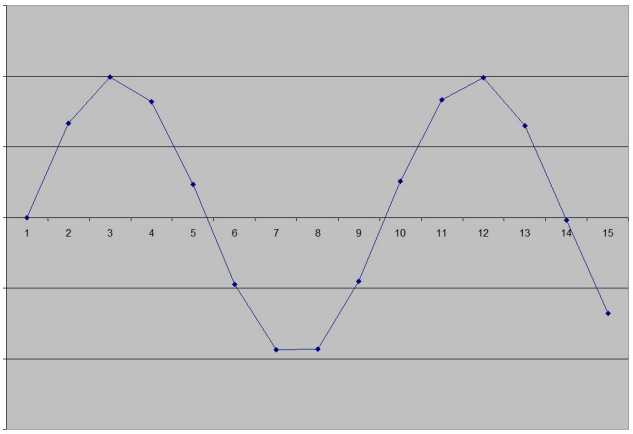

Abb.9: Signal, erzeugt aus diskreten Stützstellen

Auch wir haben uns bei unserem selbstgebauten Fernschreibkonverter dieser einfachen Möglichkeit bedient. Dieses Verfahren funktioniert immer, wenn wir ein Signal im *Zeitbereich* bilden können, welches wir dann über den Digital-Analog-Konverter der Soundkarte ausgeben können.

Sie erinnern sich: Zeitbereich heißt in diesem Fall, dass wir das Signal durch eine Folge von Stützstellen approximieren können.

Unsere Software muss die zum Argument der Sinus- oder Kosinusfunktion passenden Ordinatenwerte rechtzeitig zur Ausgabe bereitstellen.

Wie wir fernerhin gesehen haben, können wir auch Signalgemische erzeugen, indem wir die numerischen Werte im Computer addieren, subtrahieren oder gar multiplizieren.

Auf diese Weise haben wir auch amplituden- und frequenzmodulierte Signale erzeugt. Doch wie stellen wir nun *phasenmodulierte* Signale her?

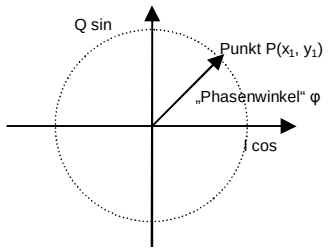

Abb.10: I und Q in Vektor-Darstellung

Einfach genug könnten wir den Phasenterm in der Sinusfunktion gemäß $y(t) = A \cdot \sin(\omega t + \varphi(t))$ manipulieren:

Verändern wir φ(t), so wackelt auch die Phase des erzeugten Signals.

Wenn wir uns die trockene Mathematik, die wir vorhin diskutiert haben nochmals näher anschauen, so stellen wir fest, dass es in der Praxis auch viel einfacher geht.

Erinnern Sie sich: Unser „Punkt“ in der komplexen Ebene konnte jederzeit durch Angabe des „Phasenwinkels“ und der Länge angesprochen werden, wenn wir daraus den Sinus- und Kosinusanteil errechnen und im Koordinatensystem gemäß Abb.10 in X- und Y-Richtung auftragen.

Nun verändern wir aber das Herstellungsverfahren ein wenig, indem wir annehmen, wir würden über einen Generator verfügen, der eine dauernde Sinus- und eine Kosinusschwingungen völlig synchron zueinander erzeugt.

Sie können sich vorstellen, als seien die Schwingungen seien einfach 90° zueinander versetzt.

Auf der digitalen Ebene müssten wir also zwei fortlaufende Zahlenfolgen erzeugen, deren Werte auf einer Sinus- und einer Kosinuskurve liegen. Und genau das klappt, wie wir bereits in Teil 1 und 2 gesehen haben, in Software ganz vorzüglich und auf einfachste Art und Weise durch $x(t) = I(t) \cdot \cos(\omega t)$ und $y(t) = Q(t) \cdot \sin(\omega t)$.

Durch Verändern der beiden zeitabhängigen *Modulations*-Terme ***I(t)*** und ***Q(t)*** können wir die Anteile an Sinus und Kosinus in den Komponentensignalen ganz feinfühlig steuern.

Addieren wir nun die Signalkomponenten *x(t)* und *y(t)* im Zeitbereich, so können wir das resultierende Summensignal $s(t) = I(t) \cdot \cos(\omega t) + Q(t) \cdot \sin(\omega t)$ einfach über den D/A-Wandler der Sound-Karte ausgeben.

Übrigens kommen die Namen für die beiden Modulationsfunktionen ***I(t)*** und ***Q(t)*** nicht von Ungefähr:

I(t) steht für „In-Phase“. Wir würden diese Komponente unter Umständen auch als den Realteil einer komplexen Funktion betrachten können. ***Q(t)*** steht für den „Quadratur“-Anteil, der orthogonal, also senkrecht zu ***I(t)*** steht und in gewisser Weise den Imaginärteil bildet.

Interessanterweise können wir aufgrund der Orthogonalität der Signale und des erfreulicherweise geltenden Überlagerungssatzes auf der Empfangsseite die Komponenten wieder trennen.

Wir benötigen dazu nur einen einzigen Analog-Digital-Wandler, den unsere Soundkarte jedoch sowieso mitbringt.

Das „Heruntermischen“ erfolgt dann wieder durch Multiplikation der Daten auf der digitalen Ebene mit synchron erzeugten Datenworten einer Sinus- und Kosinustabelle. Solch eine Struktur lässt sich sehr effizient implementieren.

Heute beinhalten praktisch alle moderneren Übertragungssysteme die Modulation und Demodulation nach diesem Schema, beginnend bei den Mobiltelefonen über das digitale terrestrische und Satellitenfernsehen und vielen anderen Anwendungen mehr.

Der Sendezweig

Wie macht das nun in typisches PSK31-Programm? Ich habe mir hierzu WinPSK angeschaut, dessen exzellente Beschreibung seiner Funktion für jeden Interessierten geradezu eine Pflichtlektüre darstellt.

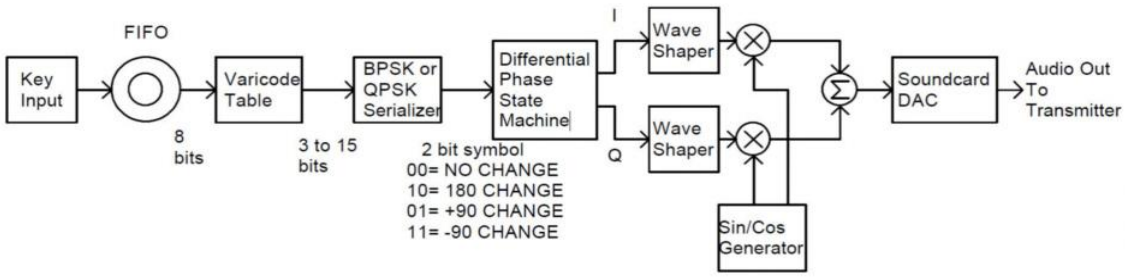

Abb.11: Der Sendezweig von WinPSK (Quelle AE4JY)

Der Sendezweig in Abb.11 kombiniert all die Techniken, die wir bis hierher beschrieben haben:

Die Daten von der Tastatur werden in einem Fifo entkoppelt – eine grundsätzlich empfehlenswerte Technik. Dann wird aus den intern im ASCII- bzw. Windows-Zeichencode vorliegenden Bytes die „Varicodes“ durch Nachschlagen in einer Tabelle generiert.

Damit Sie sich wegen der Anzahl der Bit nicht wundern: An jeden Varicode-Block werden gleich immer die beiden <NUL>-Bits angehängt, die sozusagen als Trennsymbole wirken. Der kürzeste Varicode, die „1“ für das Leerzeichen wird durch die beiden folgenden Pflichtwerte „00“ auf drei Bit aufgebläht.

Die Varicode-Bit werden nun aneinandergehängt. In längeren Schreibpausen werden einfach NULL-Bit eingestreut.

Aus dieser Bitfolge wird dann die passende Symbolfolge erzeugt, die letztendlich die Phasenwechsel steuert.

Im Falle der BPSK treten lediglich die Symbole ‚00’ und ‚10’ auf, die für Phasenwechsel von 0° und 180° stehen.

Bei QPSK liegt die Situation ein wenig komplexer, denn hier muss die Serialisation gleich die 5-Bit Fehlerkorrektur mit berücksichtigen. Hierbei können sämtliche der vier Symbole auftreten.

Die Symbolfolge triggert danach einen Zustandsautomaten, der die Modulationssignale I(t) und Q(i) generiert. Dieser führt auch die Signalformung mittels kosinusförmiger Hüllkurve aus.

Wenn Sie sich an Teil 2 meiner Vortragsreihe erinnern, so haben Sie dort eine ähnliche Formung bereits kennen gelernt. Der Trick besteht darin, dass wir es ja lediglich mit Zahlenwerten zu tun haben, die mit einem Vielfachen der Trägerfrequenz intern verarbeitet werden.

In unserem Beispiel arbeitet die Soundkarte mit 11025Hz, so dass wir genügend zeitlichen Abstand zur eigentlichen Trägerfrequenz besitzen.

Allerdings müssen wir in diesem Fall einen kompletten Zyklus in 90 µsec abschließen.

Daher verwendet das Programm den Trick, nur die wirklich kritischen Anteile mit hoher Frequenz zu bearbeiten und die anderen Vorgänge mit geringerer Rate aufzurufen.

Der Empfangszweig

Der Empfangszweig von WinPSK, wie er in in Abb.12 aufgeführt ist, sieht hingegen etwas stärker gewöhnungsbedürftig aus.

Das Signal kommt wie gewohnt vom Sendeempfänger und wird von der Soundkarte mit 11025 Abtastwerten pro Sekunde und eine Auflösung von 16 Bit digitalisiert.

Da wir auch mit der Hälfte der Abtastrate noch das Nyquist-Kriterium erfüllen, reduziert AE4JY die Datenrate erst einmal um den Faktor zwei, um den Aufwand im Programm zu verringern. Die Multiplikationen müssen dadurch nur noch alle 181 µsec ausgeführt werden.

Korrekterweise benutzt er dazu ein passendes Dezimationsfilter.

Sie erinnern sich an den Grundlagenteil: Wir müssen mit einem Tiefpass-Filter dafür sorgen, dass höher frequente Anteile entfernt werden, bevor wir danach einfach jeden zweiten Abtastwert wegwerfen können.

Das Signalgemisch enthält noch immer die I- und Q-Anteile. Wir multiplizieren mit synchron generierten Sinus- und Kosinuswerten um die Modulationsanteile zu extrahieren. Daraufhin dezimieren wir insgesamt durch neun um die interne Abtastrate von 612,5 Hz zu erhalten.

Wenn wir daran denken, dass die finale Baudrate nur 32,15 Bit/Sekunde beträgt haben wir immer noch genug „Abstand“, doch müssen Verarbeitungsschritte nun nur noch alle 1,6 Millisekunden und nicht mehr im Mikrosekundenraster erfolgen.

Das ist auch für einen modernen PC ein erheblicher Vorteil.

Das Programm erzeugt auf dieser Ebene Steuerwerte für die AFC, die die Sinus/Cosinus-Generatoren manipulieren kann. Rechnerisch funktioniert das einfach durch Einfügen und Weglassen einzelner Zählwerte und sieht im Blockschaltbild fast komplizierter aus als es im Programm später ist.

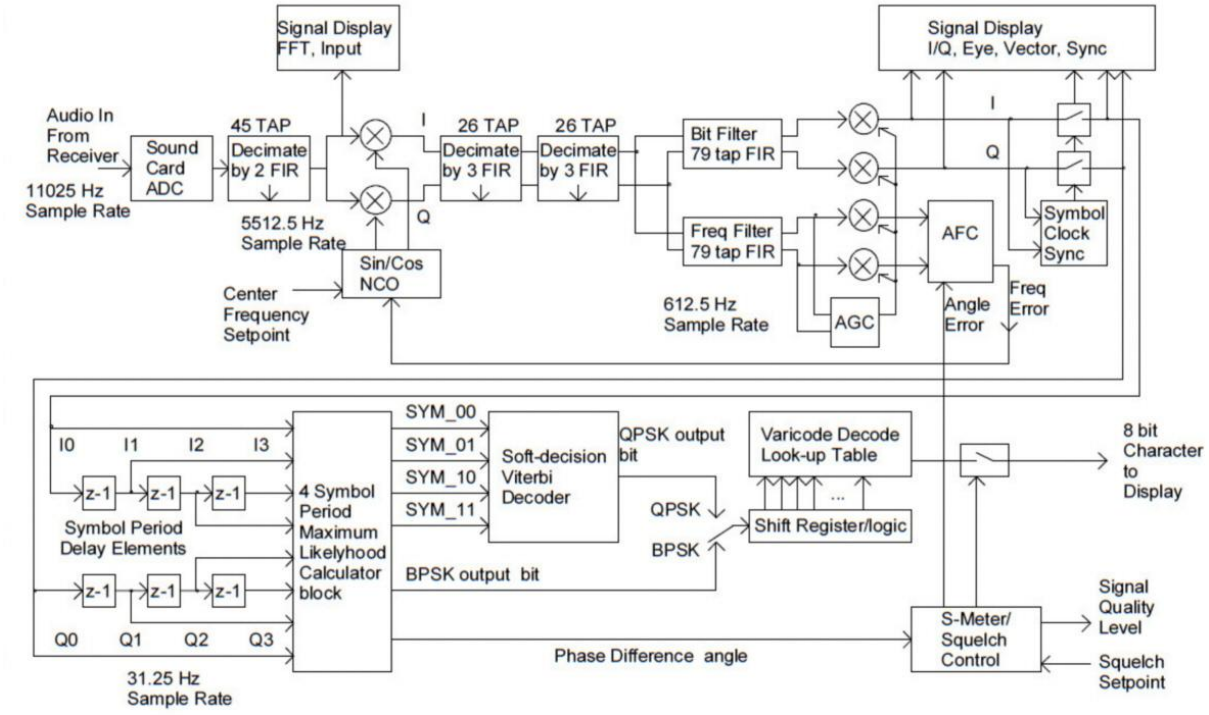

Abb.12: Der Empfangszweig von WinPSK (Quelle AE4JY)

Es folgt der Decoder-Block, der die beiden Fälle für DPSK und QPSK behandeln muss. Letzterer nutzt den Viterbi-Decoder für die Fehlerkorrektur, der aus der Symbolfolge „schätzt“ welche 5-Bit-Kombination wohl gesendet worden ist.

Am Ausgang dieses Gebildes taucht nun wieder der serielle Bitstrom als „Varicode“ auf, der über eine Codetabelle wieder zurück auf ASCII gemappt wird. Damit ist auch schon alles erledigt.

Schlussfolgerung

Ich gebe es unumwunden zu: Ganz so einfach, wie ich es gerne darstelle sind die Verfahren der digitalen Signalverarbeitung nicht wirklich.

Der Teufel steckt, wie in so vielen Dingen in den Details. Viele der raffinierten Details der Implementierung konnte ich in diesem Beitrag überhaupt nicht ansprechen.

Es wäre der Leistung von Peter, Moe und vielen anderen überhaupt nicht angemessen den Eindruck zu erwecken, dies alles wäre „primitive Technik“. Das genaue Gegenteil ist der Fall!

Wir habe es mit modernen Konzepten zu tun, denen wir in vielen Formen noch häufig begegnen werden.

Doch gleichzeitig möchte ich Ihnen ein Gefühl mit auf den Weg geben:

Diese Technik ist für Sie beherrschbar. Sie ist nicht nur etwas für einige wenige Spezialisten. Man kann die notwendigen Techniken erlernen. Man kann experimentieren und Spaß und Erfolgserlebnisse damit haben.

Genau das zu vermitteln ist das Ziel dieser Vortragsreihe.

Wir wollen gespannt drauf sein, was in der digitalen Welt als nächstes auf uns zukommen wird.

Abkürzungen und Begriffe

ADC	Analog Digital Converter; Analog Digital Wandler
AFC	Automatic Frequency Control; Automatische Anpassung an das empfangene Signal.
BPSK	Binary Phase Shift Keying, Umtastung mit zwei Phasen
CW	Continous Wave. Aussendung eines konstanten Trägersignals. Oft synonym für Morsetelegraphie benutzt.
DAC	Digital Analog Converter.

DFT	Discrete Fourier Transform; Fourier-Transformation von nicht-kontinuierlichen bzw. (zeit-) diskreten Signalen
EloKa	Elektronische Kampfführung
FEC	Forward Error Correction (Code); Verfahren, auf der Sendeseite Redundanz einzufügen, die es auf der Empfangsseite erlaubt, Bitfehler ohne Rückfrage zu korrigieren.
FFT	Fast Fourier Transform; Schnelle Fourier-Transformation
FSK	Frequency Shift Keying
HF	High Frequency. Hochfrequenz
MIDI	Musical Instrument Digital Interface
PSK	Phase Shift Keying
QAM	Quadratur-Amplituden-Modulation
QPSK	Quaternary Phase Shift Keying, Umtastung mit vier Phasen
SR	Sampling Rate; Abtastrate (eines ADC); Gibt die Anzahl der Konvertierungen pro Sekunde an.
SSB	Single Side Band (Modulation). Einseitenbandmodulation. Wird häufig zur Sprachübertragung auf Kurzwelle benutzt.
USB	In diesem Zusammenhang: „Upper Side Band“; bezeichnet die Betriebsart eines Einseitenbandsenders, in dem das obere Seitenband gesendet oder empfangen wird
VFO	Variable Frequency Oscillator. An diesem stellt man die Betriebsfrequenz ein.

Quellen

[AE4JY]	Wheatley, Moe; AE4JY, „WinPSK Technical Reference Manual“, ae4jy@mindspring.com
[DJ6HP]	Pietsch, H.J. DJ6HP, „Amateurfunk-Fernschreibtechnik RTTY“, Franzis Verlag
[G3PLX]	Martinez, Peter, G3PLX; “PSK31: „A new radio-teletype mode with a traditional philosophy”; verschiedene Speicherorte, z.B. http://www.pci-card.com/psk31.pdf
[HUFF]	Lang, H.W. „Codierungstheorie, Huffman-Code“; FH Flensburg; http://www.inf.fh-flensburg.de/lang/algorithmen/code/huffman/huffman.htm
[QAM]	http://en.wikipedia.org/wiki/Quadrature_amplitude_modulation
[SP9VRC]	Jalocha, Pawel, SP9VRC; Pawels Blog auf http://sp9jpa.blogspot.com/p/pawe-jaocha-sp9vrc.html
[TEIL2]	Franke, E.H., DK6II; „Digitale Signalverarbeitung für Einsteiger“; Skriptum zur 55. Weinheimer UKW-Tagung 2010 in Bensheim
[TIETZE]	U. Tietze, Ch. Schenk, „Halbleiter-Schaltungstechnik“; Springer-Verlag
[UKW95]	Franke, E.H. DK6II; „Digitale Modulationsverfahren in der drahtlosen Nachrichtenübertragung auf Kurzwelle“; Skriptum zur UKW-Tagung 1995; Weinheim
[WIKI]	http://de.wikipedia.org/wiki/PSK31