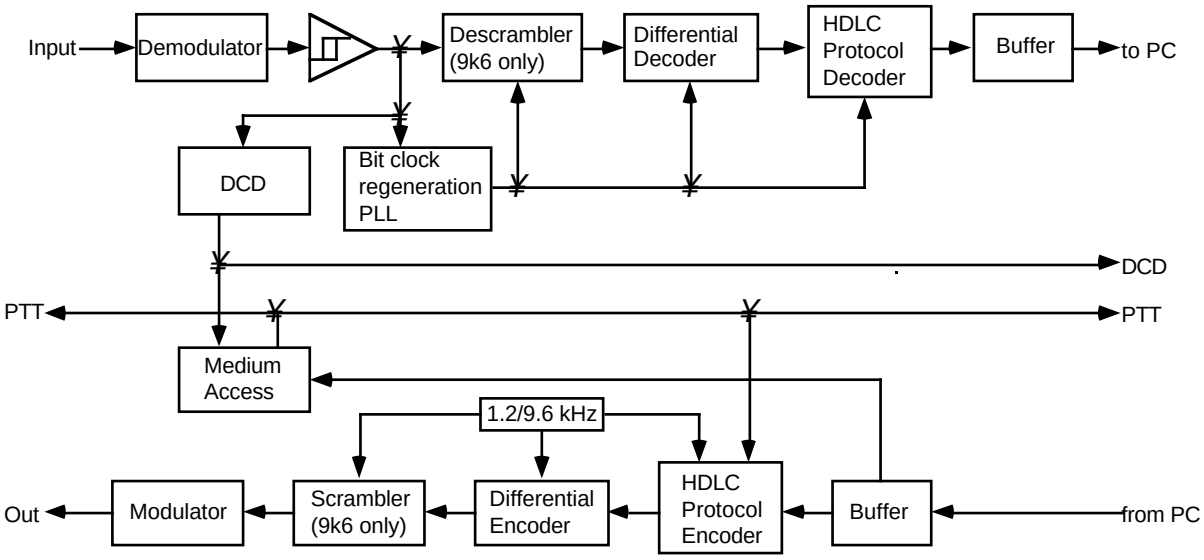
1. Einleitung

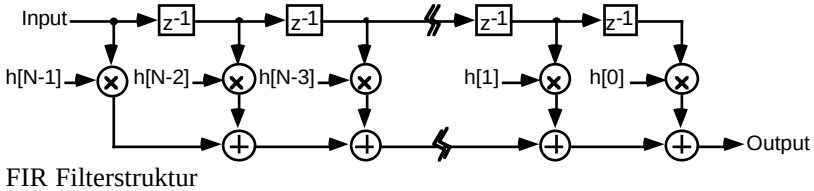
Ich möchte in diesem Vortrag die Algorithmen vorstellen, die es erlauben, Modems für gängige Amateurfunk-Betriebsarten, hier 1200Baud AFSK und 9600Baud G3RUH Packet, mit digitaler Signalverarbeitungshardware wie DSP-Boards, DSP-Soundkarten usw. zu realisieren. Auf mathematische Herleitungen und Beweise werde ich aber verzichten. Ich werde Programmbeispiele in einer C-ähnlichen Sprache bringen. In den Modems jedoch müßen diese Algorithmen aus Geschwindigkeitsgründen meist in Assembler programmiert werden.
Ich habe weitaus die meisten meiner DSP-Modems als PC/FlexNet-L1-Treiber realisiert, weil es hier einen sehr schönen "Entwickler-Kit" gibt. Der Treiber muß nun Packete von FlexNet entgegen nehmen und senden können, sowie empfangene Packete zwischenspeichern und an FlexNet abliefern. Vom AX25-Protokoll muß der Treiber nichts wissen. FlexNet übernimmt auch den Kanalzugriff, um Zugriffsprotokolle wie Optima oder DAMA realisieren zu können.
Die Linux-Treiber stellen einen KISS-Datenstrom zur Verfügung, der dann z.B. von WAMPES verarbeitet werden kann.

2. Modem-Blockschaltbild

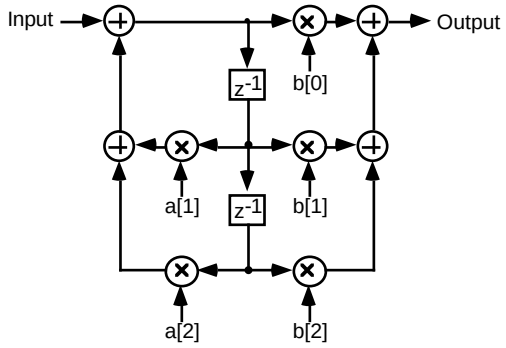


3. Modulator/Demodulator 9600 Baud

Der "Modulator" und der "Demodulator" besteht bei 9600Baud-G3RUH Packet lediglich aus einem Tiefpaßfilter mit einer Grenzfrequenz von etwa 5kHz, da die eigentliche FSK-Modulation durch den VCO im Sender und die Demodulation durch den FM-Detektor im Empfänger geschieht.



FIR Filterstruktur



IIR Filterstruktur (BiQuad)

Grundsätzlich gibt es zwei verschiedene Filtertypen:

Eigenschaft	FIR	IIR
Rückkopplung	– keine, daher nur eine endlich lange Antwort auf beliebige Eingangssignale (deshalb der Name: Finite Impulse Response)	– vorhanden, daher möglicherweise unendlich lange Antwort auf ein Eingangssignal (deshalb der Name: Infinite Impulse Response)
Komplexität	– mehr Multiplikationen und Additionen nötig – Struktur einfacher	– weniger Multiplikationen und Additionen nötig
Lineare Phase	– exakt möglich	– nur approximativ möglich
Stabilität	– unbedingt	– nur bedingt, d.h. bei ungeeigneter Wahl der Koeffizienten schwingt das Filter (bzw. kann als Oszillator gebraucht werden)
Empfindlichkeit gegenüber Quantisierungsfehlern, endlichen Wortbreiten	– klein	– gross, vor allem bei Polen mit hoher Güte
Design	– spezielle Algorithmen, aber z.T. einfach	– Algorithmen für analoge Filter werden verwendet, danach in den zeitdiskreten Bereich transformiert

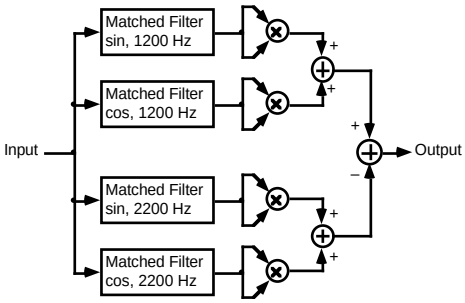
Vor allem wegen der Möglichkeit, exakt linearphasige Filter (d.h. alle Frequenzen werden genau gleich lang verzögert durch das Filter) zu bauen, sind die FIR-Filter bei Kommunikationssystemen sehr beliebt. Nichtlinearphasige Filter führen zu Verzerrungen des Augendiagramms und damit zu schlechteren Bitfehlerraten. Ich habe diese Filter deshalb als FIR-Filter implementiert (wie übrigens beim Original-G3RUH-Design auch das Sendefilter).
Beispiel-Implementation:

```
#define N 48
static int data[N];
static int ptr=0;
long out=0;
int i;
data[ptr]=input; ptr=(ptr+1)%N; /* % : Modulo */
for(i=0;i<N;i++) out+=coeff[i]*data[(ptr+i)%N];
return out;
```

4. Demodulator 1200 Baud

Der 1200-Baud-Demodulator muß zwei Wellenformen unterscheiden können, nämlich:
– 1/1200s lang eine Schwingung mit 1200 Hz
– 1/1200s lang eine Schwingung mit 2200 Hz

Dies kann durch permanenten Vergleich des empfangenen Signals mit beiden Wellenformen erreicht werden, wobei der Demodulator dann entscheiden muß, zu welcher Wellenform das empfangene Signal ähnlicher sieht. Da nun aber die Phasenlage der Schwingungen nicht vorgegeben ist, muß man das empfangene Signal je mit einer Sinus- und einer Cosinus-Schwingung beider Frequenzen vergleichen (siehe Grafik).
Vergleichen kann man zwei Signale mit einer Struktur wie ein FIR-Filter. Die Koeffizienten enthalten dann einfach das abgetastete Referenzsignal.

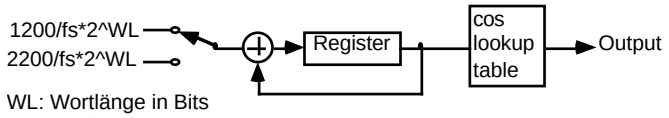


Der 1200 Baud AFSK Demodulator

```
#define N 8 /* das Verhaeltnis Sampling-Rate durch Baud-Rate */
static int data[N];
static int coeffloi[N]; /* Koeffizienten: sin 1200 Hz */
static int coeffloq[N]; /* Koeffizienten: cos 1200 Hz */
static int coeffhii[N]; /* Koeffizienten: sin 2200 Hz */
static int coeffhiq[N]; /* Koeffizienten: cos 2200 Hz */
static int ptr=0;
/* Annahme: int 16bit, long 32bit */
/* Initialisierung */
int i;
for(i=0;i<N;i++) {
  coeffloi[i]=32767*cos(2*3.1415*i/N);
  coeffloq[i]=32767*sin(2*3.1415*i/N);
  coeffhii[i]=32767*cos(2*3.1415*i/N*2200/1200);
  coeffhiq[i]=32767*sin(2*3.1415*i/N*2200/1200);
}
/* Der Teil, der fuer jedes Sample berechnet werden muss */
int i,d;
long outloi=0,outloq=0,outhii=0,outhiq=0,out;
data[ptr]=input; ptr = (ptr+1)%N; /* % : Modulo */
for(i=0;i<N;i++) {
  d = data[(ptr+i)%N];
  outloi += d*coeffloi[i];
  outloq += d*coeffloq[i];
  outhii += d*coeffihi[i];
  outhiq += d*coeffhiq[i];
}
/* Das >>15 schiebt das Komma wieder an seinen Platz */
out = (outhii>>15)*(outhii>>15)+(outhiq>>15)*(outhiq>>15)
     -(outloi>>15)*(outloi>>15)-(outloq>>15)*(outloq>>15);
return out;
```

5. Modulator 1200 Baud

Der Modulator kann analog einem DDS-Oszilator (Direct Digital Synthesis) aufgebaut werden.



Der 1200 Baud AFSK Modulator

```
#define WL 16 /* Wortlaenge des Phasenregisters */
#define COSTAB_WL 6 /* Wortlaenge der Cosinus-Tabelle */
static int costab[1<<COSTAB_WL];
static unsigned int dds_phase=0;
static unsigned int dds_increment[2];

/* Initialisierung */
int i;
for(i=0;i<(1<<COSTAB_WL);i++)
  costab[i] = 32767*cos(2*3.1415*i/(1<<COSTAB_WL));
dds_increment[0] = 1200.0/9600.0*(1<<WL);
dds_increment[1] = 2200.0/9600.0*(1<<WL);

/* Teil fuer jedes Sample */
dds_phase += dds_increment[input];
return costab[dds_phase >> (WL-COSTAB_WL)];
```

6. Hardlimiter

Der Komparator ist sehr einfach in Software zu realisieren: Man muß nur das Vorzeichen extrahieren.

7. DCD

Die DCD (Data Carrier Detect) arbeitet bei mir unabhängig von der Bittaktregeneration. Sie zählt die Anzahl der Impulse nach dem Komparator die etwa eine Bitlänge lang sind zur Anzahl der Impulse, die entweder ungefähr 0.5 oder 1.5 Bitlängen lang sind. Diese werden über eine Dauer von ungefähr 25ms gezählt. Die DCD ist immer ein Kompromiß: soll die Entscheidung möglichst fehlerfrei sein, braucht man eine lange Beobachtungszeit, was eine lange Verzögerungszeit bewirkt, bis die DCD detektiert wird. Umgekehrt bewirkt eine kurze DCD-Verzögerungszeit eine unsichere Entscheidung.

8. Bittaktregeneration

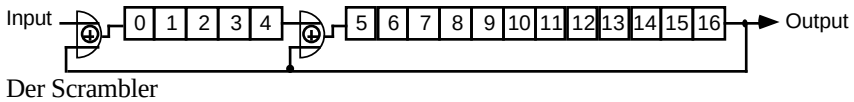
Der Bittakt wird von einer PLL regeneriert, die auf die Flanken am Demodulatorausgang synchronisiert. Meine PLL's verwenden üblicherweise ein 16-Bit Phasenregister, da dies die übliche Wortbreite bei den eingesetzten DSP's ist. Das Phasenregister wird bei jedem Sample so inkrementiert, daß es pro Baud einmal überläuft. Der PLL-Algorithmus ist nun bestrebt, Flanken im Eingangssignal bei 0x8000 stattfinden zu lassen. Damit ist der Abtastzeitpunkt bei 0x10000, was dem Überlauf des Phasenregisters entspricht. Der Korrekturwert, der zum Phasenregister addiert wird, wenn die Flanke zu früh oder zu spät kommt, ist ein Kompromiß. Ist er groß, dann rastet die PLL schnell ein, aber auch schnell wieder aus. Ausserdem ist dann dem Abtastzeitpunkt ein “Rauschen” überlagert. Ist er zu klein, dann rastet die PLL zu langsam ein, außerdem kann sie bei Frequenzabweichungen (Baudrate) weniger stark folgen

```
#define N 8 /* Verhaeltnis Samplerate/Baudrate */
#define PHASE_INC 65536/N /* 2^(Phasenregisterbits)/Samples per Baud */
#define PHASE_CORR PHASE_INC/2
```

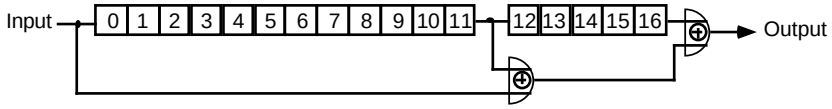
```
static unsigned int bit_phase;
/* dieses Register soll hier 16Bit breit sein */
/* der Algorithmus versucht, die Flanken auf einen bit_phase */
/* Wert von 0x8000 zu synchronisieren */
static char last_input;
long temp;

if (last_input != input) { /* Flanke detektiert */
  if(bit_phase<0x8000)
    bit_phase += PHASE_CORR; /* Flanke kam zu frueh */
  else
    bit_phase -= PHASE_CORR; /* Flanke kam zu spaet */
}
last_input = input;
temp = bit_phase + PHASE_INC;
bit_phase = temp & 0xffff;
if(temp>0xffff) { /* bit_phase hat soeben einen Overflow gehabt */
/* dies kann in Assembler eleganter programmiert werden, hier kann */
/* man dazu das Ueberlaufflag benutzen */
  Hier ein Bit dem Scrambler uebergeben
}
```

9. Scrambler

Der Scrambler hat zur Aufgabe, eine lange Folge gleicher Bits und damit sehr tiefe Frequenzen zu verhindern. Beim 1200Baud AFSK-Modem existiert kein Scrambler.
Der Scrambler wird als rückgekoppeltes Schieberegister realisiert (die Kästchen sind Einbit-Schieberegister), der Descrambler als nicht-rückgekoppeltes Schieberegister



Der Scrambler



Der Descrambler

```
/* Scrambler */
static unsigned long txshreg;

txshreg <<= 1;
if(input) txshreg |= 1;
if(txshreg & 0x20000) txshreg ^= 0x0021; /* ^ : XOR */
if(txshreg & 0x40000) return 1; else return 0;

/* Descrambler */
static unsigned long shreg;
unsigned char out;

shreg <<= 1;
if(input) shreg |= 1;
out = (shreg ^ (shreg >> 12) ^ (shreg >> 17)) & 1; /* ^ : XOR */
return out;
```

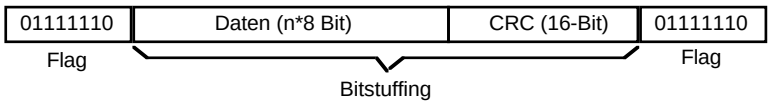
10. Differenzdecoder

Der Differenzcodierer wechselt den Ausgangspegel, wenn eine 0 am Eingang anliegt, bei einer 1 am Eingang bleibt der Ausgangspegel. Der Grund dafür ist, daß das Modem damit auch mit einer invertierenden Strecke zurechtkommt. Das HDLC Protokoll sorgt ja dafür, daß im Normalfall höchstens fünf Einsen nacheinander gesendet werden. Damit ist gewährleistet, daß das Ausgangssignal genügend Flanken enthält, worauf die Empfänger-Bittakt-PLL einrasten kann. Beim G3RUH-Modem verwürfelt der Scrambler die Bitfolge jedoch sowieso noch, wodurch diese Eigenschaft des HDLC-Protokolls keine Rolle spielt.

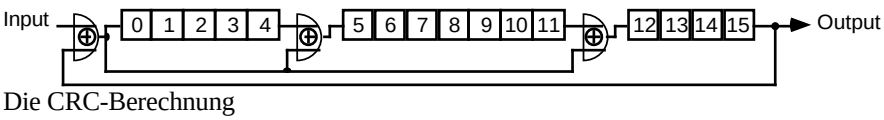
```
/* Codierer */
static unsigned char diffcoder;
if(!input) diffcoder = !diffcoder;
return diffcoder;

/* Decodierer */
static unsigned char lastbit;
unsigned char out;
if(input != lastbit) out = 0; else out = 1;
lastbit = input;
return out;
```

11. HDLC-Protokoll

Das HDLC-Protokoll ist ein bitsynchrones Protokoll, d.h. der Empfänger muß den Bittakt mittels einer PLL zurückgewinnen (siehe oben). Es werden keine Start- und Stopbits wie z.B. bei RS-232 gesendet. Damit die Bittakt-PLL einrasten kann, muß dafür gesorgt werden, daß genügend Flanken generiert werden. Da der Differenzcodierer 0 als Pegelwechsel (Flanke) überträgt und 1 als stabilen Pegel muß das HDLC-Protokoll also dafür sorgen, daß genügend Nullen gesendet werden. Dazu fügt es immer nach fünf Einsen eine Null ein, die der Empfänger dann wieder entfernen muß. Dies wird Bit-Stuffing genannt. Nun muß noch eine Möglichkeit gefunden werden, den Anfang und das Ende eines Packetes zu markieren. Dies passiert mit dem Bitmuster 01111110. Dieses Bitmuster wird HDLC-Flag genannt und kann im Datenstrom nicht vorkommen, da das Bitstuffing dafür sorgt, daß nach fünf Einsen eine Null eingefügt wird. Kommen im Empfangsdatenstrom mehr als sechs Einsen vor, soll der Empfänger dies als ABORT werten, d.h. das gerade empfangene Packet soll verworfen werden. Das HDLC-Protokoll kann eine beliebige Anzahl Bits pro Packet übertragen, wobei bei Packetradio aber immer ein Vielfaches von acht Bits übertragen wird, d.h. nur ganze Bytes. Das niederwertigste Bit des Bytes wird zuerst gesendet.



Das HDLC-Frameformat



Die CRC-Berechnung

Die Berechnung der CRC erfolgt gemäß obigem Schema, das dem des Scramblers sehr ähnlich sieht. Am Packetanfang muß das Schieberegister auf 0xffff (alle Bits auf 1) initialisiert werden. Dann werden alle Datenbits hineingeschoben. Die durch Bitstuffing erzeugten Nullen werden *nicht* in die Berechnung der CRC einbezogen. Wenn alle Datenbits gesendet sind, wird die CRC zuerst invertiert und dann gesendet, und zwar mit dem höchstwertigen Bit voran. Das Senden der CRC wird ebenfalls dem Bitstuffing unterzogen.
Der Empfänger setzt ebenfalls beim Empfang eines Flags (Framestart) das CRC-Register auf 0xffff. Der Empfänger füttert nun alle empfangenen Bits, außer den durch Bitstuffing entstandenen Nullen, in die CRC-Logik, auch die vom Sender generierten CRC-Bits. Wenn das Frame korrekt empfangen worden ist, enthält das CRC-Register nachher 0x1d0f.

						CRC Register
Input		1111	1111	1111	1111	FFFF
0	⊕ 1	1111	1111	1111	1110	
	XOR	0001	0000	0010	0001	
		1110	1111	1101	1111	EFDF
0	⊕ 1	1101	1111	1011	1110	
	XOR	0001	0000	0010	0001	
		1100	1111	1001	1111	CF9F
0	⊕ 1	1001	1111	0011	1110	
	XOR	0001	0000	0010	0001	
		1000	1111	0001	1111	8F1F
0	⊕ 1	0001	1110	0011	1110	
	XOR	0001	0000	0010	0001	
		0000	1110	0001	1111	0E1F
1	⊕ 0	0001	1100	0011	1110	
	XOR	0001	0000	0010	0001	
		0000	1100	0001	1111	0C1F
0	⊕ 0	0001	1000	0011	1110	
	XOR	0000	0000	0000	0000	
		0001	1000	0011	1110	183E
0	⊕ 0	0011	0000	0111	1100	
	XOR	0000	0000	0000	0000	
		0011	0000	0111	1100	307C
1	⊕ 0	0110	0000	1111	1000	
	XOR	0001	0000	0010	0001	
		0111	0000	1101	1001	70D9

Beispiel zur CRC-Berechnung. Als Eingangssignal wird das erste Datenbyte des untenstehenden HDLC-Beispiels verwendet.

fm HB9JNX-15 to HB9W ctl SABM+

	H	B	9	W	' '	' '	-0
Flag	90	84	72	AE	40	40	E0

01111110.00001001.00100001.01001110.01110101.00000010.00000010.00000111.

CRC Register	FFFF	70D9	93D4	CED0	C6B0	29C8	5D09	F2BF

H	B	9	J	N	X	-15	SABM
90	84	72	94	9C	B0	7F	3F

00001001.00100001.01001110.00101001.00111001.00001101.1111**0**1110.11111**0**100.

CRC Register	E174	AD4C	814D	79E2	AAC4	010D	13F0	FCC1

	CRC		
C0	7C	Flag	

00000011.00111111**0**0.01111110

CRC Register (nur Rx)	DFF0	1D0F

Ein Beispiel zum HDLC-Protokoll. Die Fett gedruckten Nullen sind diejenigen, die durch das Bitstuffing eingefügt wurden.

12. Medium Access

Der Medium-Access-Block muß entscheiden, wann der Treiber senden darf. Dieser Block ist bei PC/FlexNet-Treibern überflüssig, da dessen Funktion von PC/FlexNet übernommen wird. Bei andern Protokollen wie KISS [4] oder Packet Driver [5] muß er jedoch implementiert werden. Bei Fullduplex ist seine Aufgabe trivial: Sobald Daten im Sendepuffer vorhanden sind, muß er die PTT ziehen und damit die Sendung starten, bis keine Daten mehr im Sendepuffer sind. Bei Halbduplex wird üblicherweise der “p-persistent CSMA” Algorithmus [4] verwendet. Sobald Daten im Sendepuffer sind, testet der Algorithmus das DCD-Signal. Ist es an, dann ist der Kanal besetzt und das Modem wartet, bis er frei wird. Dann würfelt das Modem eine Zahl zwischen 0 und 1 (oder 0 und 255). Ist die Zufallszahl kleiner als der voreingestellte P-Persistence-Wert, beginnt es zu senden. Ansonsten wartet es SLOTTIME (üblicherweise 100ms), und testet wieder die DCD.
Für das Würfeln der Zufallszahl kann entweder einer der üblichen Algorithmen, die auch in den Compilerlibraries anzutreffen sind, oder auch denselben Algorithmus wie zur Berechnung der CRC im HDLC-Frame verwendet werden. Wird das Register mit einer Zahl ungleich 0 initialisiert und dann bei jedem Würfeln das Register so updated, als würde die CRC für ein 0-Eingangsbit berechnet, so kann ein Ausschnitt (8 Bit) des Registers als Zufallszahl verwendet werden. Die Sequenz wiederholt sich nach 65535 mal würfeln (die 0 wird nicht durchlaufen).

13. Ausblick

Digitale Systeme bieten viele Vorteile gegenüber analogen. Sie sind exakt reproduzierbar und simulierbar, besitzen keine Abgleichpunkte. Daher beginnen die digitalen Systeme, analoge zu verdrängen. Da digitale Bausteine in grossen Mengen produziert werden, sind sie auch preislich häufig attraktiv.
Mit frei programmierbaren DSP's ist allerdings die Signalisierungsgeschwindigkeit (Baudrate) beschränkt. Jedoch können durchaus mehr als zwei Symbole verwendet werden, d.h. mehrere Bits pro Symbol übertragen werden. Dies erlaubt einen Abtausch von Sendeleistung (d.h. Signal-Rauschabstand im Empfänger) gegen Übertragungsgeschwindigkeit. Der von den Amateuren eingeschlagene Weg war immer der Abtausch von Bandbreite gegen Übertragungsgeschwindigkeit. Weiter sollte vermehrt auch Kanalcodierung (fehlerkorrigierende Codes) eingesetzt werden, die es erlauben, mit weniger Leistung (d.h. weniger Signal-Rauschabstand) gute Ergebnisse zu erzielen.
Der Trend in der Industrie ist eindeutig, den A/D-Wandler näher zur Antenne zu kriegen. Heute wird bereits häufig die letzte Zwischenfrequenz digitalisiert und die Demodulation digital erledigt. Beispiele sind GSM (Mobiltelefon) und ADR (Astra Digital-Radio). Preislich sind heute recht günstige und sehr schnelle (Region 40 Megasamples/s) Analog-Digitalwandler erhältlich. Diese Sampleraten sind natürlich zu schnell für frei programmierbare DSPs. Hier müßen Spezial-IC's verwendet werden. Allerdings kann man mit FPGA's (Field Programmable Gate Arrays, programmierbare Logikbausteine) mit einigen Tricks auch sehr viel erreichen. Der Preis für das Entwicklungswerkzeug (Software) ist aber leider völlig untragbar für Amateuranwendungen.

14. Literatur

[1] Frerking, Marvin E., Digital Signal Processing in Communication Systems, New York, NY; Van Nostrand, 1993
[2] American Radio Relay League, AX.25 Amateur Packet-Radio Link-Layer Protocol Version 2.0, 1984
[3] Oppenheim, A.V., und R.W. Schafer, Digital Signal Processing, Englewood Cliffs, NJ; Prentice-Hall, 1975

[4] Karn, Phil KA9Q, und Mike Chepponis K3MC, The KISS TNC: A simple Host-to-TNC communications protocol, 1990
[5] Nelson, Russell, PC/TCP Version 1.09 Packet Driver Specification, Wakefield, MA; FTP Software, Inc., 1989

[3] ist eine eher theoretische Einführung in die Digitale Signalverarbeitung. [1] behandelt alle Aspekte eines Kommunikationssystems, die auf der untersten Ebene ablaufen, so eine kurze Einführung in die digitale Signalverarbeitung, Kriterien zum Beurteilen von A/D-Wandlern, Algorithmen zur Modulation und Demodulation aller gängiger Modulationsarten (einschliesslich die “analogen” wie FM, SSB, AM usw.). [2] beschreibt schliesslich die HDLC-Frame-codierung und den Aufbau eines AX.25-Frames.