55. Weinheimer UKW-Tagung, Bensheim

Digitale Signalverarbeitung für Einsteiger
Teil 2 – Praktische Anwendungen

Dipl.-Ing. Erich H. Franke, DK6II

erich.franke@afusoft.com

Im vorangegangenen Teil dieser Reihe hatten wir uns die Grundbegriffe aus der Signalverarbeitung angeschaut. Nun ist es an der Zeit, ein wenig zu experimentieren. Durch die Leistungsstärke von Heimcomputern ist es heute ohne spezielle Gerätetechnik möglich, einfache Anwendungen zu erstellen. Wir werden uns digitale Filter im Audiobereich, sowie die digitale Aufbereitung analoger Modulations- und Demodulationsverfahren an Hand von Programmbeispielen näher anschauen. Diese können sowohl als Kristallisationspunkt für eigene Arbeiten dienen und helfen Ihnen, die Vorteile und Grenzen der digitalen Verarbeitung analoger Signale besser verstehen zu können.

Königsbach-Stein, im Juli 2010

RTTY - Eine Testanwendung

Im ersten Teil unserer Reihe haben wir uns mit den Grundlagen beschäftigt und gesehen, wie einfach eigentlich die Gedanken sind, die der digitalen Verarbeitung analoger Signale zu Grunde liegen.

Heute wollen wir eine praktische Anwendung diskutieren und sehen, wie wir die einzelnen Parameter sinnvoll bestimmen müssen, dass das Ganze in der Praxis auch wirklich funktioniert.

Lassen Sie uns für diese Überlegungen ein bewusst einfaches Beispiel heranziehen.

Die wohl älteste digitale Betriebsart im Amateurfunk, dem Funkfernschreiben (Radio Teletype, RTTY) mutet heute durchaus nostalgisch an. Doch hört man sich auf der Kurzwelle um, so findet man eine ganze Reihe von Verfahren, die auf den gleichen Wurzeln basieren.

Selbstverständlich ist klar, dass die im Amateurfunk genutzten Parameter bereits vor Urzeiten im Hinblick auf mechanische Fernschreibmaschinen und geringe Bandbreite des ausgesendeten Signals festgelegt worden und die das verfügbare Spektrum nicht wirklich effizient nutzen.

Nichts desto trotz wird diese Betriebsart auch heute noch verwendet und die gewählten Parameter und deren Zusammenhänge sind nicht nur für Spezialisten recht einfach zu verstehen.

Deshalb ist RTTY für unsere ersten Einsteiger-Experimente bereits aus didaktischen Gründen gut geeignet.

Für diejenigen unter Ihnen, denen die Übertragungsparameter nicht mehr ganz geläufig sind, ein Schnellkurs: Im Funk-Fernschreiben ist auf der HF-Ebene Frequenzumtastung (F1B) üblich. Das heißt, dass der Träger auf dem Funkkanal zwischen zwei Frequenzen hin- und herspringt. Diese beiden „Kennzuständen“ können so als Werte eines „Bit“ gesehen werden. Sie werden mit MARK und SPACE bezeichnet.

Auf dem *Funkkanal*, also *auf der Hochfrequenz-Ebene* ist der Zustand *MARK* der jeweils *höheren* Frequenzlinie zugeordnet. MARK wird auch als „Ruhelage“ bezeichnet und ist in der Fernschreibtechnik mit dem Binärwert „1“ bzw. früher „fließendem Linienstrom“ verknüpft.

Hingegen spricht man vom Zustand *SPACE*, wenn auf dem *Funkkanal*, also *auf der Hochfrequenz-Ebene* die jeweils *niederere* Frequenz ausgesendet wird. SPACE wird auch als „Zeichenlage“ benannt und ist dem Binärwert „0“ bzw. einer Stromunterbrechung zugeordnet.

Die Frequenzen, die mit den Zuständen MARK und SPACE verknüpft sind, haben auf dem Funkkanal einen festen Abstand, der als „Shift“ bezeichnet wird. Da immer nur eine der beiden Frequenzen ausgesendet wird, spricht man anschaulich auch von „*frequency shift keying*“ oder kurz FSK.

Verwendet man einen Einseitenbandsender (*Single Sideband*, SSB) und stellt diesen auf den Betrieb im oberen Seitenband (*upper side band*, USB) ein, so erscheint im ausgesendeten Spektrum ein zur Modulationsfrequenz proportionales Hochfrequenz-Signal, da die Frequenzlinien auf der HF-Ebene durch lineare Mischung aus der Modulationsfrequenz entstehen.

Das heißt, dass wir zur Erzeugung eines FSK-Signals auf der HF-Ebene einfach einen SSB-Sender benutzen können, bei dem wir das Modulationssignal zwischen zwei Tönen mit passendem Abstand hin- und herschalten.

Zustand	Sendefrequenz auf dem Hochfrequenz-Kanal, Beispiel: f=14.075 kHz		Modulationsfrequenz bei Verwendung eines SSB-Senders mit Einstellung USB = („*upper Side Band*“)	
	Schmalband shift = 170 Hz	Breitband shift = 850 Hz	Schmalband shift = 170 Hz	Breitband shift = 850 Hz
MARK	14.075.085 Hz	14.075.425 Hz	1445 Hz	2125 Hz
SPACE	14.074.915 Hz	14.074.575 Hz	1275 Hz	1275 Hz

Tabelle 1: Zusammenhang zwischen Modulationsfrequenz und dem ausgesendeten Signal

Dieses so genannte „audio frequency shift keying“ (AFSK) macht uns das Leben wesentlich einfacher, denn wir können bereits vorhandenes Equipment benutzen.

Tabelle 1 gibt Beispiele für den Zusammenhang zwischen typischen Modulationsfrequenzen und den daraus erzeugten HF-Signalen.

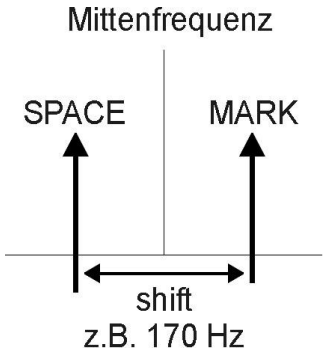


Abb.1: Lage der Kennfrequenzen im HF-Spektrum

Im praktischen Betrieb haben sich bestimmte Modulationsfrequenzen eingebürgert, obgleich theoretisch eigentlich jede Frequenzkombination mit passender Shift funktionieren würde. Doch diese liegen innerhalb der Bandbreite der Quarzfilter des SSB-Senders und liegen günstig bezüglich etwaiger In-Band-Mischprodukte.

Die von RTTY-Puristen bevorzugte Aussendung im unteren Seitenband (LSB) soll hier aus Gründen der besseren Verständlichkeit nicht näher beschrieben werden. Einen guten Artikel hierzu findet man beispielsweise unter [G3LDI].

Ein typisches RTTY-Signal auf Kurzwelle im klassischen Look mit shift=170 Hz und einer Schrittgeschwindigkeit von 45.45 Baud belegt rechnerisch eine Bandbreite von 246 Hz [UKW95].

Die Daten werden auf dem Funkkanal im „Asynchronformat“ übertragen. Jedes zu übertragene Zeichen bzw. Codewort beinhaltet eine bestimmte Anzahl von Nutzbit. Bei RTTY sind dies lediglich fünf.

Diesen Datenbit geht ein „Startbit“ in SPACE-Lage voraus und am Ende folgen ein oder mehrere „Stoppbit“ in MARK-Lage.

Die Baudrate oder „Schrittgeschwindigkeit“ wird im Amateurfunk historisch mit 45,45 Baud angegeben. Auf Kurzwelle sind jedoch Übertragungssysteme in F1B mit 300 Baud und mehr zu hören.

Eine Besonderheit stellt das Stoppbit dar. Es hat im klassischen Fernschreiben nämlich eine Länge von 1,5 Bitzeiten. Aus diesem Grund kann dieser exotische Wert auch heute noch bei seriellen Schnittstellenbausteinen eingestellt werden.

Das Datenformat ist direkt vergleichbar mit dem auf einer asynchronen seriellen Schnittstelle an einem Computer. Auch hier spricht man vom Zustand „MARK“ als der Ruhelage, welche gemäß V.28 (RS.232) einem Spannungspegel von -15..-5 Volt zugeordnet ist und der „Zeichenlage“ SPACE, welche sich auf dem Kabel als ein Spannungswert zwischen +5 und +15 Volt manifestiert.

Wir bauen den verbesserten Sendegenerator

Wie bereits im ersten Teil beschrieben, so können wir einen Signalgenerator für sinusförmige Signale recht einfach über eine Schleife und trigonometrische Funktionen erzeugen (Abb. 2).

Im praktischen Alltagseinsatz wird man jedoch das andauernde Berechnen der Werte der Sinusfunktion dadurch zu vermeiden suchen, dass man bereits „mundgerechte“ Amplitudenwerte vorab in einer Tabelle bereithält. Auf diese Weise lässt sich erheblicher Rechenaufwand sparen, da man zu einem Argument lediglich nachschlagen muss und Fließkomma-Rechnungen komplett entfallen.

Der zweite Griff in die Trickkiste der „low-Level“-Programmierung besteht darin, die Werte gleich beim Speichern auf passende Ganzzahlen zu skalieren. Da unsere Soundkarte ja sowieso nur 16 Bit verarbeitet, bietet sich an, pro Argumentwert lediglich ein 16-Bit-Wort abzuspeichern.

```
nAbtastrate    = 11025;
fFrequenz      = 1275.0;
fAmplitude     = 32767;

for (n=0; n<nAbtastrate; n++)
   {
   y[n] =
     (fAmplitude * sin((2*pi*fFrequenz*n) / nAbtastrate));
   }
```

Abb.2: Die grundlegende Schleife zur Tonerzeugung

Ist der Programmspeicherplatz ein Problem, beispielsweise bei der Implementierung auf Single-Chip-Mikro-prozessoren, so kann man den Tabellenplatz unter Nutzung der Symmetrie der Sinusfunktion sogar auf ¼ der ursprünglichen Größe eindampfen.

Man benötigt ja nur Amplitudenwerte, welche dem Argumentbereich $\{0 .. +\pi/2\}$ zugeordnet sind. Diese nehmen dann den Wertebereich {0 .. +1} ein. Bei Argumenten zwischen $\pi/2$ und π, also im „zweiten Quadranten“ wiederholen sich die Werte in umgekehrter Reihenfolge bis wieder 0 erreicht wird. Das Spiel setzt sich im dritten und vierten Quadranten fort. Hier liefert die Sinusfunktion die gleichen Werte nur eben mit negativem Vorzeichen.

```
short nSinusTabelle[MAXTAB] =
   {
     0,   130,   261,   392,   522,   653,   784,   914,  1045,  1176,  1306,  1437,  1568,  1698,  1829,  1959,
  2090,  2220,  2350,  2481,  2611,  2741,  2872,  3002,  3132,  3262,  3392,  3522,  3652,  3782,  3912,  4041,
  4171,  4301,  4430,  4560,  4689,  4819,  4948,  5077,  5206,  5335,  5464,  5593,  5722,  5850,  5979,  6107,
  6236,  6364,  6492,  6620,  6748,  6876,  7004,  7131,  7259,  7386,  7514,  7641,  7768,  7895,  8022,  8148,
  8275,  8401,  8528,  8654,  8780,  8906,  9031,  9157,  9282,  9408,  9533,  9658,  9783,  9907, 10032, 10156,
 10280, 10404, 10528, 10652, 10775, 10899, 11022, 11145, 11268, 11391, 11513, 11635, 11757, 11879, 12001, 12123,
          . . .
   }
```

Tabelle 2: Beispiel für die ersten Werte einer vor verarbeiteten Sinustabelle

Lediglich bei der Berechnung der Tabellenindices muss man mit den Vorzeichen operieren. Wenn wir jedoch genügend Platz haben, so können wir uns das Leben einfach machen. Ein kleines C- oder BASIC-Programm liefert uns die Sinustabelle, die wir direkt in unsere Senderoutine hinein compilieren können (Tab. 2).

Die Anzahl der Tabellenwerte hängt von der gewünschten Auflösung ab.

Für unsere Experimente verzichte ich an dieser Stelle aus Gründen der Klarheit auf das Tabellenverfahren, da wir momentan ja nur „offline“ Töne erzeugen.

Zeitstempel und Zählerstände

In unserem System ersetzen wir die kontinuierliche Zeit im Argument der Sinusfunktion durch einen Zähler, der in regelmäßigen Intervallen gemäß $\tau = \frac{1}{SR}$ fortgeschaltet wird. Hierdurch enthält eine Sekunde Echtzeit genau SR Abtastwerte, in unserem Beispiel hätten wir pro Sekunde 11025 Stützstellen mit einem Abstand von ca. 91 µs.

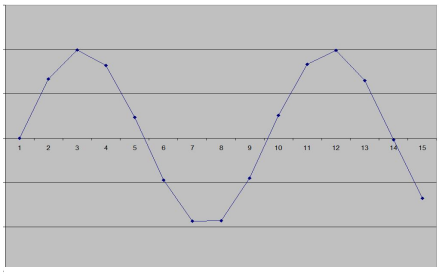


Abb.3: 1275 Hz bei SR=11025

Damit erhalten wir auf bequeme Weise eine Zeitbasis, mit der wir auch andere Aktivitäten des Programms synchronisieren können, doch im Gegenzug erhalten wir eben nur zu diskreten Zeiten Amplitudenwerte.

Wollen wir beispielsweise ein Signal mit einer Frequenz von 1275 Hz erzeugen um einen SPACE-Ton zu erhalten, so bestünde eine volle Schwingung dieses Signals aus ungefähr neun Abtastwerten. Genauer geht das nicht, weil 11025 nicht ganzzahlig durch 1275 teilbar ist.

Dies führt zu einem interessanten Problem, wenn wir das durch unser diskretes System erzeugte Signal modulieren oder umtasten wollen.

Zum Fernschreiben müssen wir schließlich das Signal im Takt der Datenbit zwischen den Kennfrequenzen für MARK und SPACE umschalten.

Das hört sich zwar recht einfach an, doch führt es uns zum Problem von Phasensprüngen an den Bitgrenzen, ein Problem mit dem sich Entwickler von Fernschreib-Sendekonvertern beschäftigen müssen, denn diese Sprünge sind höchst unerwünscht, da sie das Spektrum des Sendesignals unzulässig verbreitern.

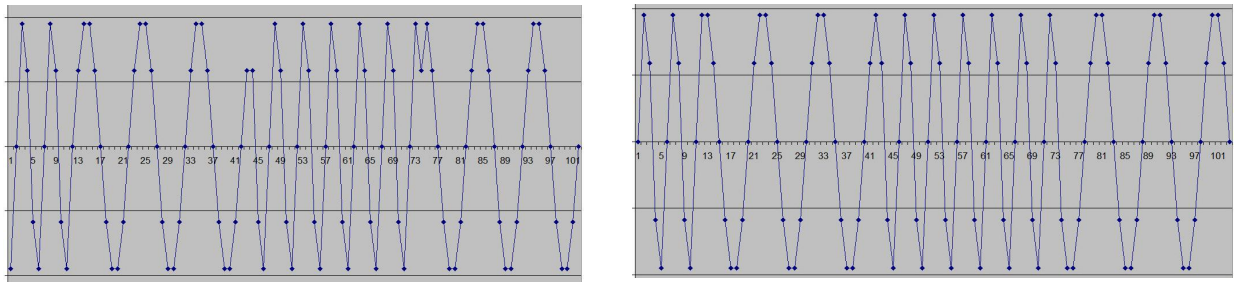


Abb.4: Tonerzeugung mit Phasensprüngen und phasenkorrigiert

Früher erleichterte der gute alte XR2206, eigentlich ein Schaltkreis für Funktionsgeneratoren [XR2206] Generationen von Entwicklern das Leben, denn er garantierte bereits durch seine interne Schaltung einen „glatten“ Phasenübergang beim Umschalten.

Doch in unserer Software-Implementierung müssen wir programmtechnisch selbst dafür sorgen, dass der Übergang im Umschaltzeitpunkt möglichst ohne lästige Phasensprünge erfolgt.

Der Trick dies zu erreichen liegt darin, den Zähler *n* im Argument der Sinusfunktion passend zu manipulieren.

Ziel ist es für Frequenz f_1 bei einem Zählerstand n_1 nach Umschalten auf f_2 einen passenden Zählerstand n_2 dergestalt zu finden, dass die Funktion den gleichen y-Wert zurückliefert. Dann entfallen die hässlichen Sprünge der Zeitfunktion.

$$y(n, f_1) = \sin\left(2\pi f_1 \cdot \frac{n_1}{N}\right) \overset{!}{=} y(n, f_2) = \sin\left(2\pi f_2 \cdot \frac{n_2}{N}\right)$$

Durch Koeffizientenvergleich und Kürzen erhalten wir die einfache Beziehung

$$f_1 \cdot n_1 = f_2 \cdot n_2 \rightarrow n_2 = \left(\frac{f_1}{f_2}\right) n_1$$

Also können wir zur Korrektur einfach den Wert des Zählerstandes proportional zum Verhältnis der umzuschaltenden Frequenzen verändern und passend runden, so dass der mit der neuen Frequenz berechnete Amplitudenwert glatt und ohne Sprünge auf den vorherigen folgt. Abb. 5 zeigt ein Beispiel in Pseudocode wie das funktionieren kann.

Nun zur Frage, wie wir zu unserem Bittakt kommen.

Prinzipbedingt besitzt unser Programm bereits einen Grundtakt, nämlich die Abtastrate. Da bereits die Signalerzeugung an diesem Takt hängt ist es sicher sinnvoll zu versuchen, auch den Schritt-Takt hiermit zu synchronisieren.

Bei der gewählten Abtastrate von 11025 $^{\text{samples}}/_{\text{sec}}$ erhalten wir mit $\tau = \frac{1}{SR}$ eine Auflösung von 90,7µs.

Um daraus die gewünschte Baudrate *B* von 45.45 Baud zu erzeugen, müssten wir eigentlich nur die Abtastschritte zählen und nach der passenden Anzahl den Bitwechsel vornehmen.

Wir müssten demnach N Abtastschritte warten: $N = \frac{SR}{B} = \frac{11025}{45{,}45} = 242{,}574$.

Runden wir auf den nächst höheren ganzzahligen Wert 243 auf, so beträgt der Fehler hierbei ca. 0,3%, was als „Verzerrung“ im Allgemeinen tolerabel ist.

Doch auch hier können wir noch etwas verbessern:

```
nAbtastrate = 11025;
fFrequenz0  = 1275.0;         // SPACE, bei Datenbit = 0 (FALSE)
fFrequenz1  = 1445.0;         // MARK,  bei Datenbit = 1 (TRUE)
fAmplitude  = 32767;          // Amplitude maximal
bNeuerWert  = TRUE;           // Anfangsbedingung: MARK
fFrequenz   = fFrequenz1;     // Anfangsbedingung: MARK-Frequenz
nZähler      = 0;
Zu jedem Abtastzeitpunkt
   {
   Wenn der Zeitpunkt für einen Bitwechsel gekommen ist, dann
      {
      Wenn sich der Wert des Datenbit geändert hat, dann
         {
         Merke Dir den neuen Wert des Datenbit in bNeuerWert

         Wenn bNeuerWert gerade eben TRUE geworden ist, dann
            Setze nZähler = (fFrequenz1 / fFrequenz0) * nZähler
         andernfalls
            Setze nZähler = (fFrequenz0 / fFrequenz1) * nZähler
         }
      // Den folgenden Schritt machen wir bei jedem Datenbit, ob geändert oder nicht!
      Wenn bNeuerWert derzeit TRUE ist, dann
         Setze fFrequenz = fFrequenz1; // MARK
      andernfalls
         Setze fFrequenz = fFrequenz0; // SPACE
      }
   // Das ist die Tonerzeugung, wie wir sie kennen
   y[n] = (fAmplitude * sin((2*pi * fFrequenz * nZähler) / nAbtastrate));
   nZähler = nZähler + 1;
   }
```

Abb.5: Pseudocode-Ausschnitt: Tonerzeugung mit Phasenkorrektur

Der Trick besteht darin, den Rundungsfehler über mehrere aufeinanderfolgende Bit zu verteilen und dadurch den Mittelwert genau zu halten (Abb. 6).

```
nBaudrate      = 4545;        // Skaliert mit Faktor 100, damit ...
nAbtastrate    = 1102500;     // ... alles ganzzahlig wird!
nZähler        = 0;
Zu jedem Abtastzeitpunkt
   {
   nZähler = nZähler + nBaudrate;
   Wenn nZähler größer oder gleich dem Wert von nAbtastrate geworden ist, dann
      {
      nZähler = nZähler - nAbtastrate;
      // Hierbei verbleibt ggf. ein Rest,
      // der in der nächsten Iteration wirksam wird!

      // Die Bitgrenze wurde erreicht.
      // Führe die gewünschten Arbeiten aus
      }
   }
```

Abb.6: Pseudocode-Ausschnitt: Korrektur von Rundungsfehlern

Das tun wir, indem wir die Division in fortlaufende Subtraktionen auflösen, genau so, wie wir das in der Grundschule gelernt haben und wie früher mechanische Rechenmaschinen funktionierten. Unserer Software leistet das Verfahren noch heute gute Dienste.

Wir benutzen dieses „Dithering“ immer dann, wenn wir Abläufe synchronisieren müssen, deren Zeiten nicht ganzzahlig dividierbar sind, beispielsweise bei Fax- und anderer Bildübertragung, um den „Schräglauf“ durch unvermeidliche Frequenzabweichungen zu kompensieren [MPSK].

Der Empfangszweig

Wir haben nun Bausteine für einen funktionstüchtigen FSK-Generator kennengelernt. Nun wollen wir schauen, wie die Technik auf der Empfangsseite aussieht.

Für RTTY hatte Hajo Pietsch, DJ6HP vor vielen Jahren seinen legendären '001-Filterkonverter veröffentlicht, der auch bei gestörten Übertragungskanälen auf Kurzwelle recht gut arbeitete und der mit ein paar wenigen Operationsverstärkern leicht nachzubauen war [DJ6HP] und auch kommerziell eingesetzt wurde [MINIX].

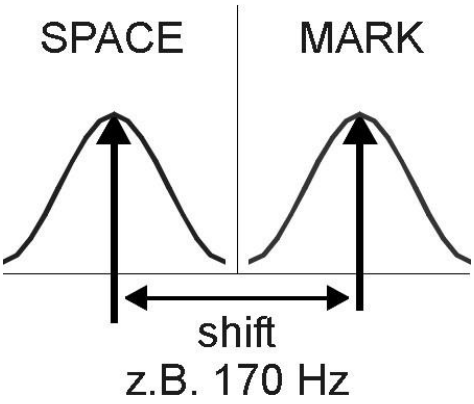


Abb.7: Schematischer Verlauf der Filterkurven

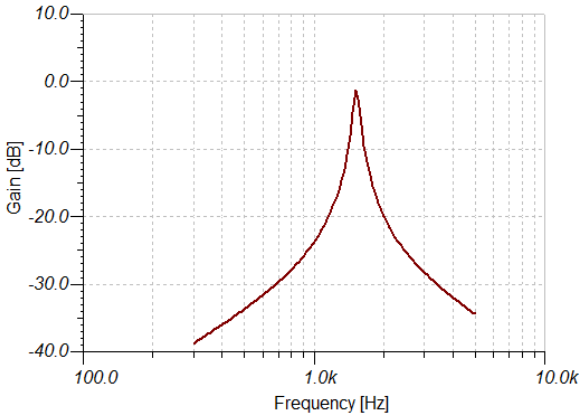


Abb.8: Verlauf des Frequenzgangs des Hardware-Filters

Der eigentliche Konverter besteht aus zwei parallelen Zweigen aus aktiven Filtern mit Mehrfach-Gegenkopplung. Die Auslegung derartiger Filter in Analogtechnik mittels Operationsverstärkern ist beispielsweise in [TIETZE] sehr gut beschrieben (Abb.7).

Die Filter im Originaldesign realisieren eine Bandbreite B=100Hz und eine Leerlaufverstärkung von $v_0 \approx 1$ (Abb.8).

Die Ausgangssignale der Filter werden durch Spitzenwertgleichrichter gleichgerichtet. Die gleichgerichteten Signale werden dann voneinander subtrahiert, integriert und schließlich einem Schmitt-Trigger zugeführt, der das Binärsignal zum Ansteuern des Endgerätes quantisiert.

Eigentlich steht nichts im Wege, das analoge Design probeweise einmal mit digitalen Filtern in Software nachzubauen.

Aus analog mach digital: Das Filterdesign

Für den Nachbau habe ich einen Chebyshev-Bandpass sechster Ordnung gewählt, der eine 3 dB-Bandbreite von ca. 100 Hz realisiert. Das Filter ist für diese Anwendung eher zu steil. Daher müssten die ausgewählten Filterparameter für einen praktischen Einsatz sicherlich noch einmal optimiert werden (siehe unten).

Doch zur Demonstration, was man mit aktiven Filtern anfangen kann, auf was man zu achten hat und wo die Grenzen liegen sind die Filter so wie sie sind völlig ausreichend.

Die Badpässe lassen sich gut in IIR-Struktur realisieren.

Den Begriff der „Unendlichen Impulsantwort" (*Infinite Impulse Response*, IIR) haben wir bereits im ersten Teil dieser Reihe kennengelernt. Filter mit dieser Struktur nutzen Rückkopplungen zur Erhöhung der Steilheit und sind damit kostengünstiger zu implementieren.

Sie tragen jedoch den inhärenten Mangel, dass sie in bestimmten Fällen instabil werden können.

Die als „*Direct Form*" bezeichneten Topologien implementieren eine Übertragungsfunktion wie folgt:

$$H(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}}$$

Wie einfach ersichtlich ist der Nenner des Bruchs verantwortlich für das latente Stabilitätsproblem nämlich dann wenn er den Wert Null annimmt.

Neben den „Direct Form"-Topologien gibt es andere bzw. verbesserte Strukturen, die in den verschiedenen Gebieten der Nachrichtentechnik Anwendung finden. Neben den „transponierten" Formen der

„Direct Form“ gibt es beispielsweise noch Kaskaden- (Cascade), Leiter- (Lattice) oder auch die Parallel-Topologien. Ein lesenswerter Artikel findet sich beispielsweise in [JONES].

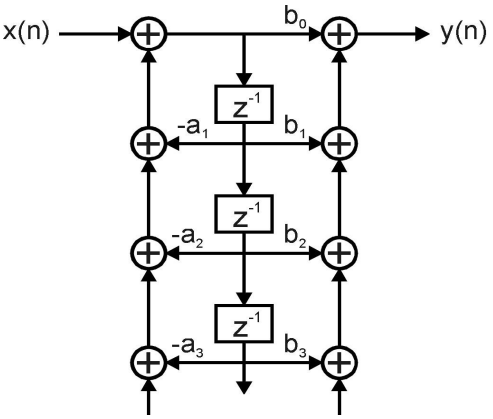


Abb.9: Struktur eines Direct-Form II IIR-Filters

Es ist sogar so, dass die in unserem Beispiel angewandte „Direct Form“ einen höheren Aufwand an Multiplikationen benötigt als andere Implementierungen – Man misst die Komplexität von Filterstrukturen gerne an den aufwändigen Multiplikationsoperationen. Außerdem ist das Design empfindlich bezüglich der Stabilität bedingt durch die Nullstellen im Nenner.

Doch können wir auf dem PC mit seinem großem Speicher und seiner hohen Rechenleistung ruhig etwas Performance verschwenden und erhalten dafür eine übersichtliche Implementierung.

Übrigens gibt es von der „Direct Form“ zwei Ausprägungen. Neben der „Direct Form Typ 1, dem Klassiker aus der Literatur, den ich im ersten Teil vorstellte besitzt die hier angewandte Form Typ 2 den Vorteil nur einen Verzögerungsspeicher zu benötigen (Abb. 9).

Wie üblich ist die Implementierung symmetrisch und erfordert neben massiven Multiplikationen und Additionen keine weiteren mathematischen Operationen.

Wieder werden gespeicherte Signalwerte mit passenden Koeffizienten multipliziert.

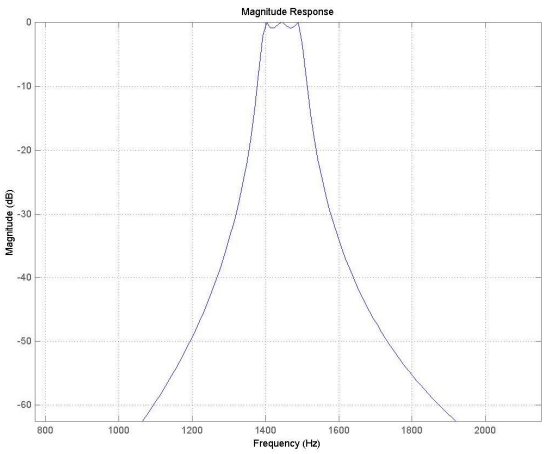
```
// Filterkoeffizienten für 1275 Hz (SPACE)
const double b[7] = {
   1.097405342129e-5, 0 ,-3.292216026386e-5, 0, 3.292216026386e-5, 0, -1.097405342129e-5};
const double a[7] = { 1, -4.449538053382,  9.542507295748, -11.99476181373,
                          9.365439640304, -4.285938052144,   0.9453646388629};

// Filterkoeffizienten für 1445 Hz (MARK)
const double b[7] = {
    1.09740534218e-5, 0, -3.292216026539e-5, 0, 3.292216026539e-5, 0,  -1.09740534218e-5 };
const double a[7] = { 1, -4.04848992365,  8.406189870221, -10.40224284676,
                          8.250212968327,-3.899635604713,   0.9453646388629};
```

Tabelle 3: Beispielhafte Filterkoeffizienten für die Realisierung

Hier gibt es gleich zwei Arten davon: Die Koeffizienten b_0 bis b_M gehören zum Zähler dieses Bruchs (Numerator), die Koeffizienten a_0 bis a_N sind dem Nenner (Denominator) zugeordnet [MTLB].

Es kann auf Zähler- und Nennerseite sogar eine unterschiedliche Anzahl von Koeffizienten existieren.



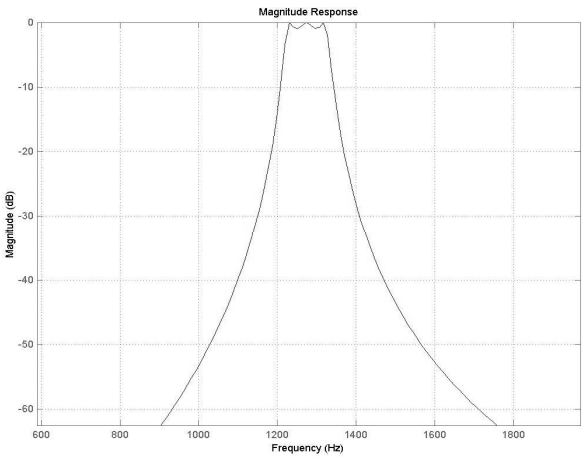


Abb.10: Frequenzgang der realisierten Filter (links MARK 1445 Hz, rechts SPACE bei 1275 Hz)

Der Nenner bewirkt die Rückkopplung. Die Koeffizienten werden hier negativ eingerechnet. Der Koeffizient a0 entartet zur „1“ ganz links in der Übertragungsfunktion.

Die Zählerseite wiederum sieht ganz ähnlich einer FIR-Topologie aus und trägt daher zur Instabilität nicht bei.

Da die Stabilität aus dem Nenner folgt, lässt sich bereits vor Implementierung rechnerisch nachweisen, ob das Design stabil ist oder nicht. Daher ist die Genauigkeit der Koeffizienten von entscheidender Bedeutung, beispielsweise, wenn bei „kleinen“ Mikroprozessoren mit Festkomma-Arithmetik gearbeitet werden muss.

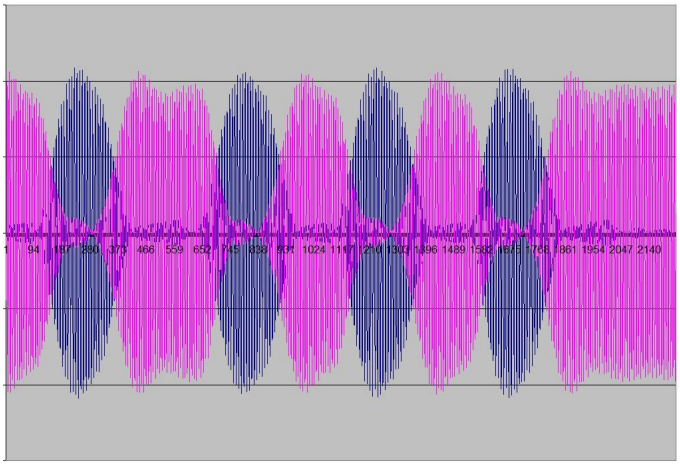


Abb.11: Signal im Zeitbereich nach den Filtern

Für unser Beispiel erwies sich ein IIR-Filter sechster Ordnung als völlig ausreichend. In diesem Fall war die Anzahl der Koeffizienten auf Zähler- und Nennerseite identisch. Wir benötigen jeweils sieben Stück. Anders als beim FIR-Filter sind die Koeffizienten jedoch nicht um den mittleren Koeffizienten herum symmetrisch.

Doch selbst die so einfach realisierten Frequenzgänge können sich sehen lassen (Abb. 10).

Da wir eine 16-Bit-Soundkarte benötigen, wodurch wir die Grenzdynamik von 96 dB im Eingangssignal prinzipbedingt nicht überschreiten können, habe ich weniger Wert auf Steilheit unterhalb von -60dB und mehr Wert auf ein gutmütiges Verhalten im Durchlass und Sperrbereich Wert gelegt. Im Durchlassbereich ergibt sich eine für diesen Fall exzellente Welligkeit von ca. 1dB.

Nicht ganz optimal ist die Sprungantwort in unserem Design. Rechnerisch benötigen die Filter ungefähr 100 Abtastwerte, also ca. 9 ms zum Einschwingen, was wir in der Zeitdarstellung (Abb. 11) an den verschliffenen Bitübergängen deutlich erkennen können.

Das klingt auf den ersten Blick nicht viel, doch müssen wir bedenken, dass die Dauer eines Bit bei RTTY nur ca. 22 ms beträgt.

Hier würden wir für einen praktischen Einsatz vermutlich Selektivität gegen ein besseres Einschwingverhalten abwägen.

```
Zu jedem Abtastzeitpunkt
   {
   fAkku = fAkku * fDecay;

   wenn fAkku kleiner als fNeuerWert ist, dann
      {
      fAkku = fNeuerWert;
      }
   Verarbeite fAkku
   }
```

Abb. 12: Spitzenwert-Gleichrichter mit Abfall-Zeitkonstante

Allerdings rettet uns, dass sich der Filterkonverter die Redundanz des FSK-Signals zunutze macht. Daher wird der regenerierte Bitstrom wieder deutlich verbessert.

Der hierzu notwendige Spitzenwert-Gleichrichter (Abb. 12) wird als Akkumulator realisiert, dessen Inhalt zu jedem Abtastzeitpunkt prozentual reduziert wird.

Die Abfallzeit berechnet sich zu $y(n) = x \cdot fDecay^n$ mit *fDecay*<1 als zeitbestimmendes Glied.

Wählt man beispielsweise bei einer Abtastrate von 11025 Samples pro Sekunde den Wert von *fDecay* zu 0,9 so sinkt das Signal des Spitzenwertgleichrichters in ca. 2 ms auf 10% des Ursprungswertes, gemäß der Beziehung $\tau = \frac{\log(0{,}1)}{\log(0{,}9)} \cdot \frac{1}{11025\frac{1}{s}} = 1{,}98 \cdot 10^{-3} s$. Kleinere Werte von *fDecay* erhöhen dabei die Steilheit. Werte größer 1 sind zu vermeiden, weil es sonst zu einer Aufschaukelung käme.

Im Übrigen kann das Ergebnis in *fAkku* nie negativ werden. Dieses Verhalten benötigen wir gleich.

In unserer Beispiel-Applikation schalten wir die Spitzenwert-Gleichrichter hinter die Filter-Ausgänge.

Die sich so ergebenden Werte ziehen wir voneinander ab. Wir dürfen das, da die Ausgangssignale der Gleichrichter nie negativ werden. Als Resultat erhalten wir ein Signal, das positiv ist, wenn das MARK-Filter einen Beitrag liefert und negativ wird, wenn das Ausgangssignal des SPACE-Filters überwiegt.

Da in unserem FSK-Signal im Sollzustand sich MARK und SPACE abwechseln, so muss das bipolare Differenzsignal nun diesem Wechsel folgen. Aus dieser Redundanz schöpft RTTY übrigens seine relativ hohe Störresistenz.

```
wenn der Wert von bBitValue FALSE ist, dann
   {
   wenn fAkku größer als fSchwellwert ist, dann
        Setze nBitValue auf TRUE
   }
andernfalls
   {
   wenn fAkku kleiner als -fSchwellwert ist, dann
        Setze nBitValue auf FALSE
   }
```

Abb. 13: Schmitt-Trigger

Um aus dem Differenzsignal nun ein echtes Binärsignal zu extrahieren lassen wir einen Schmitt-Trigger folgen.

Abb. 13 zeigt den Pseudocode der Softwarerealisierung.

Wir wenden den einfachen Trick an der besagt, dass wir das bipolare Signal in *fAkku* mit einem positiven und einem negativen Schwellwert symmetrisch vergleichen.

Damit lassen wir den Wert der binären Variablen *bBitValue* passend „springen“. Die Hysterese von ±*fSchwellwert* verhindert Störungen an den Bitgrenzen.

Doch wie würden sich unsere Filter in der Praxis verhalten?

Wenn wir unseren digital aufgepeppten Filterkonverter gegen ein Signal im QRM antreten lassen, so erhalten wir erwartungsgemäß eine ganz ordentliche Performance, vergleichbar zu der eines in Analogtechnik aufgebauten Konverters (Abb. 14).

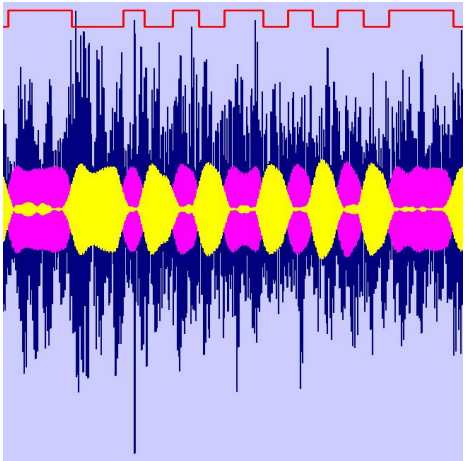


Abb.14: Selektivität im QRM

Der von der Software decodierte Bitstrom ist oberhalb des analogen „Oszillogramms“ dargestellt.

Aus der Darstellung des Signals im QRM sieht man, wie der Filterkonverter seine Selektivität ausspielen kann.

Zwar sind die Flanken der Bit bedingt durch die höhere Steilheit verschliffen, doch sieht man sehr schön, wie sich MARK und SPACE abwechseln.

Die Binärdaten sind gegenüber den Analogwerten in den Filtern ein wenig versetzt, da auf Grund der Hysterese des Schmitt-Triggers vom bipolaren Differenzsignal erst dessen Schaltschwelle überschritten werden muss.

Unser Testsignal besteht aus der für Fernschreibzwecke üblichen „RYRYRY“, dessen Nutzbit maximal viele Wechsel ausführen.

Die scheinbar verlängerten MARK-Perioden sind einfach Phasen, in denen das letzte Nutzbit eine „1“ ist, worauf das Stopbit mit Länge 1,5 folgt. Bei SPACE tritt eine Verlängerung dann auf, wenn das erste auf das Startbit folgende Datenbit den Wert Null besitzt.

Bereits hier sehen wir, dass unser Software-Konverter wie schon sein analoges Vorbild in der Lage ist, Signale aus dem QRM heraus zu detektieren.

Das ist nur der Anfang…

Das bestehende Design lässt sich nicht nur auf einem PC implementieren. Es funktioniert auch ganz problemlos auf einem Ein-Chip-Mikrocontroller mit eingebautem Analog-Digitalwandler. Die Anforderung an den Speicher und die CPU-Leistung sind selbst ohne Optimierungen moderat.

Bereits an diesem simplen Beispiel zeigt sich, dass sich digitale Filter grundsätzlich genau so wie ihre analogen Gegenstücke verhalten. Es gibt auch in der Softwaretechnik keine Wunder-Algorithmen, denn Mathematik und Physik sind überall auf der Welt gleich.

Selbstverständlich ist die Demodulation von RTTY mit digitaler Signalverarbeitung längst Stand der Technik. Es gibt eine große Anzahl von Programmen, die mit der Soundkarte solche Arbeiten durchführen. Filterkonverter sind auch nicht die einzige Methode, Fernschreibsignale mitzuschreiben.

Doch macht es sicher Spaß zu verstehen, wie solche Programme arbeiten und vielleicht weckt es die Lust, die bisher erlernten Techniken einmal selbst in der Praxis auszuprobieren.

Das war das eigentliche Ziel dieser Übung.

Ich wünsche Ihnen recht viel Spaß dabei.

Abbildungen

Abb. 1 Lage der Kennfrequenzen im HF-Spektrum
Abb.2: Die grundlegende Schleife zur Tonerzeugung
Abb.3: Stützstellen von 1275 Hz bei SR=11025
Abb.4: Tonerzeugung mit Phasensprüngen und phasenkorrigiert
Abb.5: Pseudocode-Ausschnitt: Tonerzeugung mit Phasenkorrektur
Abb.6: Pseudocode-Ausschnitt: Korrektur von Rundungsfehlern
Abb.7: Schematischer Verlauf der Filterkurven
Abb.8: Verlauf des Frequenzgangs des Hardware-Filters
Abb.9: Struktur eines Direct-Form II IIR-Filters
Abb.10: Frequenzgang der realisierten Filter (links MARK 1445 Hz, rechts SPACE bei 1275 Hz)
Abb.11: Signal im Zeitbereich nach den Filtern
Abb.12: Pseudocode-Ausschnitt: Spitzenwert-Gleichrichter mit Abfall-Zeitkonstante
Abb.13: Pseudocode-Ausschnitt: Schmitt-Trigger
Abb.14: Selektivität im QRM
Tabelle 1: Zusammenhang zwischen Modulationsfrequenz und dem ausgesendeten Signal
Tabelle 2: Beispiel für eine vorverarbeitete Sinustabelle
Tabelle 3: Filterkoeffizienten für die Realisierung

Abkürzungen und Begriffe

ADC Analog Digital Converter; Analog Digital Wandler

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

DVD Digital Video Disc

EloKa Elektronische Kampfführung

FFT Fast Fourier Transform; Schnelle Fourier-Transformation

FSK Frequency Shift Keying

HF High Frequency. Hochfrequenz

MIDI Musical Instrument Digital Interface

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 am Empfänger die Betriebsfrequenz ein.

VoIP „Voice over IP“; Technik zur Übertragung von Sprache über das Internet

Quellen

[DJ6HP] Pietsch, H.J. DJ6HP, Amateurfunk-Fernschreibtechnik RTTY, Franzis Verlag

[G3LDI] „RTTY INTRO by Roger Cooke - G3LDI”; http://www.southgatearc.org/data/rtty/

[JONES] Douglas L. Jones, „IIR Filter Structures“, http://cnx.org/content/m11919/1.2

[MINIX] MINIX RTTY-Konverter MSK-10; Schaltungsbeschreibung; Richter&Co, D-3000 Hannover

[MPSK] AFUSOFT MPSK, Prozessorgesteuerter Standardkonvwerter, TN 9631, AFUSOFT Kommunikationstechnik GmbH; D-75196 Remchingen

[MTLB] Mathlab; The MathWorks Inc.

[TEIL1] Franke, E.H., DK6II; „Digitale Signalverarbeitung für Einsteiger“; Skriptum zur 54. Weinheimer UKW-Tagung 2009 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

[XR2206] Datenblatt von EXAR XR2206, beispielsweise unter http://www.datasheetcatalog.com/datasheets_pdf/X/R/2/2/XR2206.shtml