

Projekt: Mobilfunk und Signalverarbeitung

Lukas Becker, Tobias Frahm

June 20, 2021

Date Performed:	June 20, 2021
Partners:	Lukas Becker, Tobias Frahm
Instructor:	Prof. Dr. Sauvagerd
University:	University of Applied Science

1 Projektbeschreibung und Ziel des Projektes

In diesem Projekt wird eine Continuous Phase Frequency Shift Keying (CPFSK) Demodulation von Wetterdaten (Seewetterbericht des Senders Pinneberg auf Kurzwelle) in Echtzeit durchgeführt. Diese CPFSK Demodulation wird in Form eines *Software Radios* auf einem Digitalen Signalprozessor (DSP) implementiert. Die CPFSK-kodierten Wetterdaten werden mit einer Draht-Antenne im Frequenzbereich 10.1MHz bis 11.1MHz empfangen und verstärkt. Bei dem so empfangenen Signal handelt es sich um CPFSK-Modulierte Seewetterdaten. Ziel des Projektes ist es, zunächst in *MATLAB* ein Modell zu erstellen, welches in der Lage ist, die CPFSK modulierten Seewetterdaten zu dekodieren und diese auszugeben. Für das *MATLAB* Modell wird hierfür zunächst ein CPFSK-Signal in *MATLAB* erzeugt um die ersten Schritte durchzuführen. Später in der Simulation kann hier auf ein zeitbegrenztes Signal als Grundlage für das *MATLAB* Modell genutzt zurückgegriffen werden. Sobald das *MATLAB* Modell steht, wird die Simulation in C-Code überführt und anschließend auf einem DSP implementiert.

2 Vorbereitung

Um ein grundlegendes Verständnis für das Projekt zu bekommen, werden zunächst einige Ansätze in *MATLAB* simuliert. Hierfür wird zunächst eine Rechteckfolge mit dem Wertebereich $D = \{0, 1\}$ generiert. Diese wird genutzt um zunächst ein CPFSK modulierte Testsignal zu erzeugen.

2.1 Rechteckimpuls $d(i)$

Mit Angabe der Symboldauer $T = 20\text{ms}$ lässt sich auf die minimale Abtastfrequenz nach Nyquist-Shannon schließen. Die minimale Abtastfrequenz f_A muss

mehr als doppelt so groß wie die höchste abzutastende Frequenz sein.
Aus

$$f_A > 2 * f_{max}$$

mit

$$f_{max} = \frac{1}{T_{max}}; T_{max} = T = 20ms$$

ergibt sich

$$f_A > 2 * \frac{1}{T}$$

Eingesetzt:

$$f_A > 2 * \frac{1}{20ms}$$

$$f_A > 100Hz$$

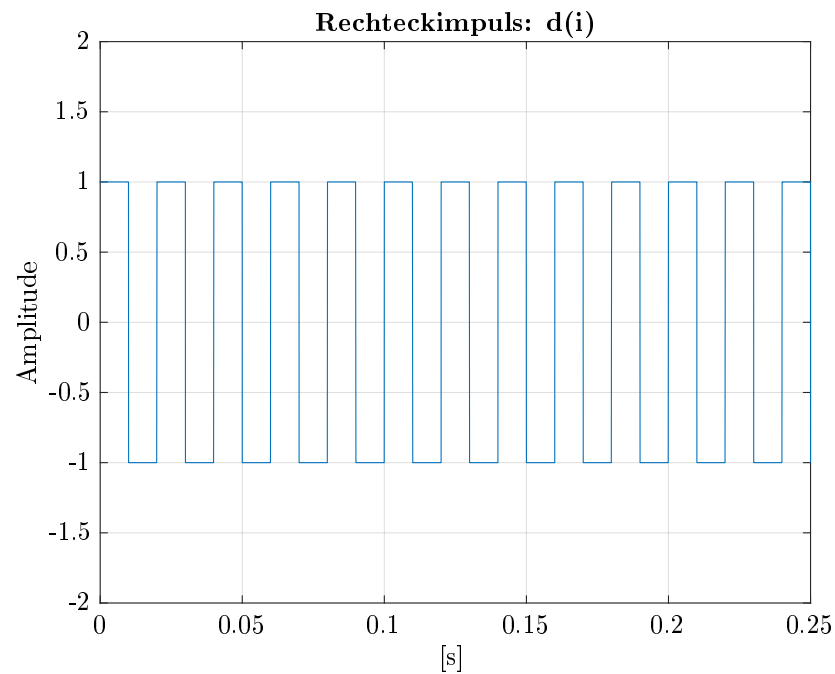


Figure 1: Rechteckimpuls zur CPFSK Simulation in *MATLAB*

2.2 Anfangsphase $\phi(iT)$

Für die CPFSK-Modulation muss die Anfangsphase $\phi(iT)$ bestimmt werden. Gegeben ist ein Integrator (IIR Filter 1.Ordnung) mit der Übertragungsfunktion:

$$H_I(z) = \frac{z^{-1}}{1 - z^{-1}}$$

Dieser kann hier anstelle der Summenbildung der komplexen Einhüllenden eingesetzt werden, da die Integration einer Aufsummierung diskreter Flächenelemente unter der Kurve entspricht.

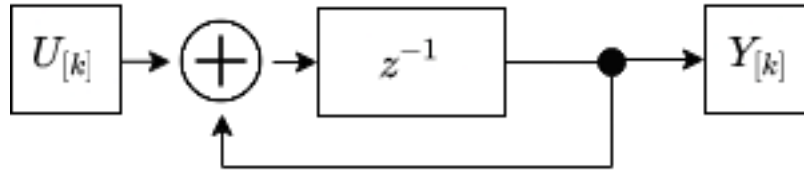


Figure 2: Signalflussdiagramm des Integrators 1.Ordnung

2.3 CPFSK Signal

Die CPFSK Modulation ist eine Methode um digitale Signale mithilfe einer Trägerfrequenz analog zu Übertragen. Bei der CPFSK Modulation handelt es sich um eine Frequenzmodulation ohne Sprünge im Phasenübergang. In Abb. 3 ist im oberen Bereich das binäre Signale, im mittleren Bild die Trägerfrequenz und unten das binäre Signal auf die Trägerfrequenze moduliert zu sehen. Der Phasenübergang ohne Sprung ist notwendig, da Sprünge ein theoretisch unendliches breites Band benötigen somit die Nachbarkanäle stören würde. Das in Abschnitt 2.1 generierte Rechtecksignal wird nun CPFSK Modeliert. Dafür wird die Formel aus Worksheet 1 verwendet:

$$CPFSK_{sig} = amp \cdot \sin(2\pi \cdot f_T \cdot n \cdot T_A + 2\pi \cdot \Delta F \cdot \frac{\phi(iT)}{f_A} + \varphi_0)$$

Zur bestimmung des Spektrums des Signals, wird in *MATLAB* die FFT verwendet. Bei einem $\Delta F = 225Hz$ wird hier eine Bandbreite von ca. $B > 450Hz = 2 \cdot \Delta F$ erwartet.

Zur Bandbreitenbestimmung wird das Parsevalsche Theorem genutzt, die Bandbreite des CPFSK Signals ist derjenige Bereich, in den 99% der Signalenergie von $0 \dots \frac{f_A}{2}$ fallen. Es ergibt sich hierbei eine Bandbreite von $742Hz$

$$\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=1}^{N-1} |X(k)|^2$$

```

1  % Die Bandbreite wird durch den Bereich beschrieben, in den 99%
2  % der Signalenergie fallen.
3
4  total_power = 0;
5  current_power = 0;
6  idx = 0;
7  idx_max = round(length(spectrum_sig)/2);
8  idx_start = idx_max/2; % fA/4
9
10 for i = 1:length(spectrum_sig)/2 - 1
11     total_power = total_power + abs(spectrum_sig(i))^2 /
    ↳ length(spectrum_sig);
12 end
13
14 N = length(spectrum_sig);
15
16 while current_power/total_power < 0.99
17     current_power = current_power + abs(spectrum_sig(idx_start -
    ↳ idx))^2 / N;
18     current_power = current_power + abs(spectrum_sig(idx_start +
    ↳ idx))^2 / N;
19     idx = idx + 1;
20 end

```

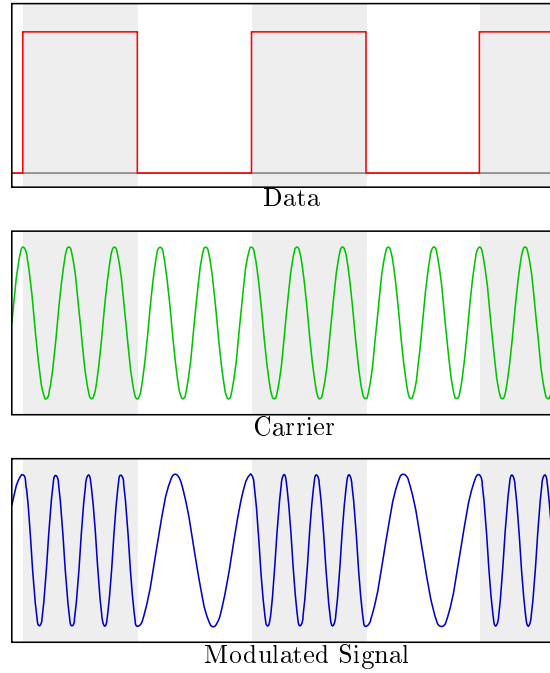


Figure 3: Beispielhaftes CPFSK moduliertes Trägersignal einer binären Information. Die Phasenübergänge sind ohne Sprung. Quelle: N.N (2021)

Das Weglassen der Start- und Stopbits spielt bei der Bestimmung der Bandbreite keine Rolle. Es werden nur binäre Daten übertragen $D = \{0, 1\}$, die Frequenzen $f_{0,1} = f_T \pm \Delta F$ repräsentieren. Da auch die Start/Stopbits durch D dargestellt werden, würde sich hier im Frequenzbereich nur eine Veränderung im Maximum der Amplitude ergeben, die repräsentierenden Frequenzen selbst bleiben unverändert.

3 Signalfluss Empfänger

Abb. 5 zeigt den Aufbau des Softwareradios. Der durch ein analoges Bandpassfilter extrahierte, zwischen 10.8MHz und 11.1MHz befindliche Kanal wird empfangen, und durch Unterabtastung mithilfe des ADCs in das auf das Band bezogene Basisband verschoben. Hierbei wird das Band auf 1.0096MHz Nyquistbandbreite geschmälert. Ein Dezimator mit Bandpass als Anti-Aliasingfilter, Kapitel 4, reduziert die Abtastfrequenz auf $3831,49\text{Hz}$. Der Durchlassbereich des Bandpasses wird hierbei jeweils auf den zu empfangenden Kanal angepasst. Die beiden Frequenzen für 1/Mark und -1/Space befinden sich nun im Basisband. Zur Symbolrückgewinnung durch Demodulation können unterschiedliche Methoden der Vorverarbeitung angewandt werden, in dieser Arbeit wurde zunächst

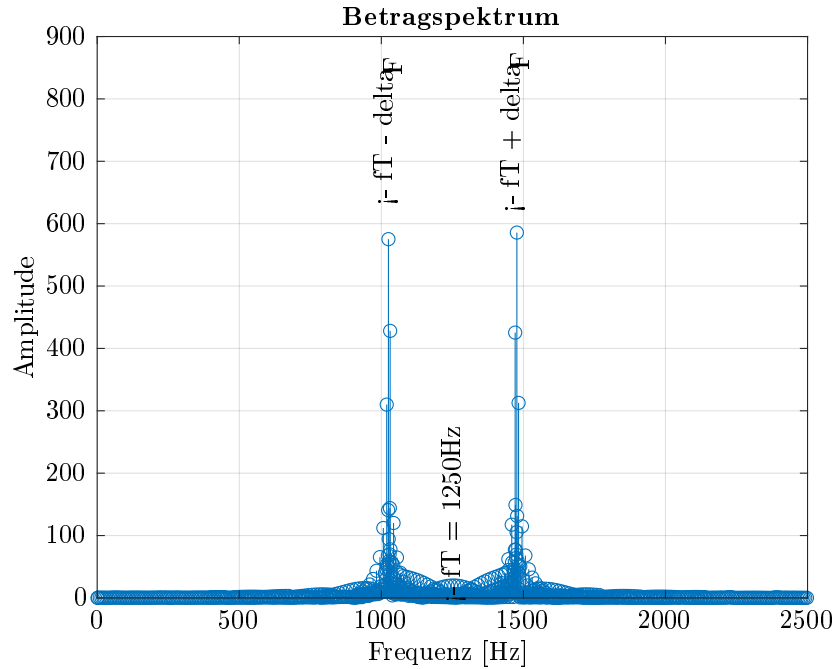


Figure 4: Betragspektrum des CPFSK-Modulierten Rechteckimpulses.

ein Ansatz über einen Hilberttransformator, Kapitel 5, verfolgt. Später hat sich jedoch die Verwendung eines komplexen Kammfilters als sinnvollere Alternative herausgestellt. Das Kammfilter in Kapitel 6 sorgt dafür, dass die Frequenzen deutlich voneinander zu differenzieren sind. Sobald das Signal durch das Kammfilter aufbereitet ist, kann die Demodulation durch einen FM-Verzögerungsdemodulator 7 mit anschließender Dekodierung erfolgen.

4 Dezimator mit Kanalselektion

Der Wetterdatensender sendet auf mehreren Kanälen in dem genannten Band. Nun muss der jeweils ein Kanal mithilfe eines Bandpasses selektiert werden. Der so ausgewählte Kanal wird nun mithilfe einer Unterabtastung in den Tiefpassbereich verschoben um eine Demodulation durchführen zu können. Zur Steigerung der Recheneffizienz wird eine Dezimationsstufe realisiert, die im Unterabtaster direkt mit den Polyphasen des Bandpasses filtert. So werden pro empfangenen Sample nur sehr wenige Multiplikationen durchgeführt. Das Filter wird in *MATLAB* ausgelegt und dann in Polyphasen zerlegt exportiert. Die exportierte Datei kann direkt als Feld in C verwendet werden.

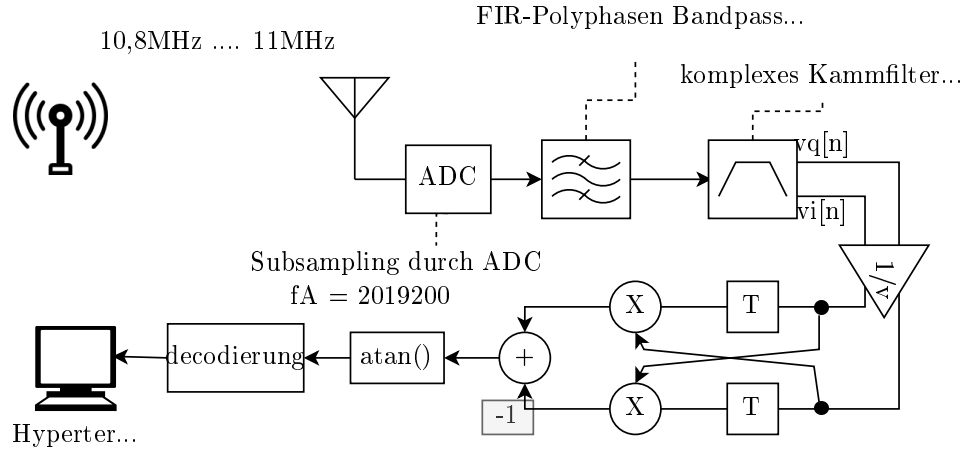


Figure 5: Durch den ADC findet direkt beim Empfang eine Unterabtastung statt. Durch eine weitere Unterabtastung im Bandpass, wird das Signal in das Basisband verschoben. Anschließend durch das Kammfilter für die Demodulation mit anschließender Dekodierung aufbereitet und auf dem Hyperterminal angezeigt.

4.1 Theorie

Der bei 10100.8 kHz befindliche, 5KHz breite Kanal wird durch die Unterabtastung in den Bereich von 4800Hz +/- 2500Hz verschoben. Da sich im schlechtesten Fall direkt daneben der nächste Kanal anschließt, muss vor einer Unterabtastung gefiltert werden um Aliasing zu vermeiden. Der Filter muss nun einen Durchlassbereich von mindestens der 99% Bandbreite bzw. 742Hz aufweisen, maximal jedoch einen Durchlassbereich von 5KHz weniger zwei mal dem Übergangsbereich, um zuverlässig danebenliegende Bänder zu dämpfen. Die Grenzen des Durchlassbereiches werden somit auf $f_u = 3800Hz$ und $f_o = 4800Hz$ festgelegt. Zudem wird der Bandpass als FIR-Filter realisiert. Der nachfolgende Unterabtaster wird mit dem Faktor 527 betrieben, woraus sich eine Abtastfrequenz von $f_{s_{neu}} = 3831,499Hz$ ergibt. Somit verteilen sich die 2056 Koeffizienten des Filters auf 527 Polyphasen mit jeweils vier Koeffizienten.

4.2 MATLAB

Zu Auslegung des FIR Filters in *MATLAB* listing 4.2, müssen zunächst die Grenz- und Stopfrequenzen Festgelegt werden. Es ergeben sich $f_u = 3800Hz$ und $f_o = 4800Hz$, die Stopfrequenzen $f_{uStop} = f_u - 1500$ und $f_{oStop} = f_o + 1500$. Bei den Stopfrequenzen gilt es Abzuwägen, umso steiler das Filter, desto mehr Koeffizienten, desto mehr Rechenzeit wird später in C benötigt. Der Amplitudengang des Filters, siehe Abb. 4.2, zeigt das gewünschte Verhalten. Es werden 2056 Koeffizienten benötigt.

```

1 function [outputArg1, outputArg2, outputArg3] = band(low, high,
    ↪ inputSig, sample_rate)
2 %FIR Bandpass
3 %
4 rp = 3;
5 rs = 40;
6 dev = [(10^(rp/20)-1)/(10^(rp/20)+1) 10^(-rs/20)];
7 low_stop = low - 1500;
8 high_stop = high + 1500;
9 [N_FIR,fo,mo,w] = firpmord([low_stop low high high_stop], [0 1
    ↪ 0], [0.05, 0.01, 0.05], sample_rate);
10 fprintf("N_FIR = %d\n", N_FIR);
11 coeff = firpm(N_FIR,fo,mo,w);
12
13 freqz(coeff,1, 200000, sample_rate)
14
15 xb = filter(coeff, 1, inputSig);
16 outputArg1 = xb;
17 outputArg2 = coeff;
18 outputArg3 = fo;
19 end

```

Listing 1: FIR-Bandpass Filterfunktion in *MATLAB*

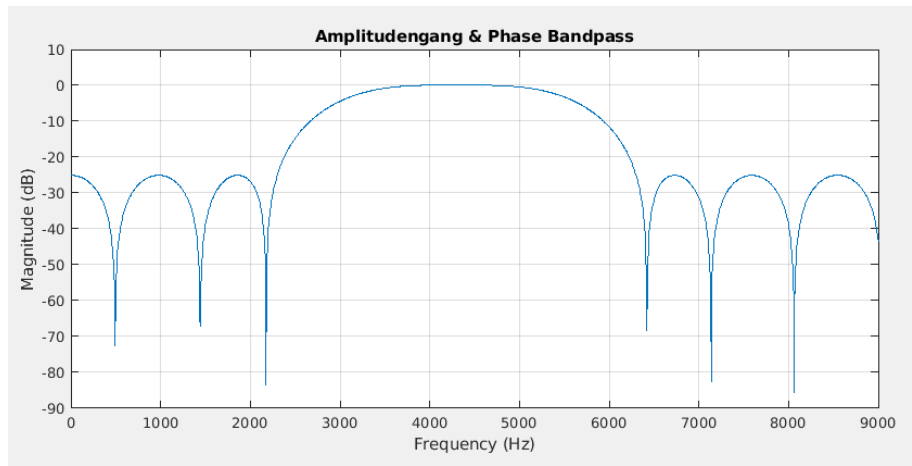


Figure 6: FIR-Bandpassfilter Amplitudengang. Das FIR Filter benötigt 2056 Koeffizienten.

```
1  const short FIR_BANDPASS_1[4] = {1304, 22, 186, -8,};
```

Listing 2: Beisphafte Polyphase in C, die Koeffizienten sind im Datentyp *short* abgelegt.

4.3 ANSI C

In der C-Implementierung wird in der endgültigen Version mit dem Datentyp *short* gearbeitet. Dies trägt zu einer besseren Performance auf dem DSP bei. Die Koeffizienten des FIR-Filters werden in 527 *const short* Arrays der Größe 4 aufgeteilt, siehe 4.3, und in den FIR Filter übergeben. Es wird Sample für Sample eingelesen, dezimiert und durch die Filterroutine verarbeitet. 4.3.

5 Hilbert Transformator

Der Hilbert Transformator wandelt ein reelles Zeitsignal $x(t)$ in ein analytisches Signal $x_+(t)$ um. Dies wird benötigt um das Signal anschließend mit dem FM-Verzögerungsdemodulator, beschrieben in Kapitel 7, zu demodulieren. Mit $\mathcal{H}(x(t))$, der Hilberttransformierten ergibt sich dies wie folgt:

$$x_+(t) = x(t) + j\mathcal{H}(x(t))$$

Wobei $y(t) = \mathcal{H}(x(t))$ sich mit $Y(f) = X(f) \cdot -j\text{sign}(f)$ berechnen lässt. Wird dieses analytische Signal jedoch dem FM-Verzögerungsdemodulator übergeben, ist aufgrund des geringen Frequenzhubs Δf eine eindeutige Demodulation jedoch nicht möglich. Der Abstand zwischen beiden Signalen ist so gering, dass

```

1  short FIR_filter_sc(short FIR_delays[], const short FIR_coe[],
   ↪  short int N_delays, short x_n, int shift) {
2      short i, y;
3      int FIR_accu32 = 0;
4      // read input sample from ADC
5      FIR_delays[N_delays-1] = x_n;
6      // clear accu
7      FIR_accu32 = 0;
8      // FIR filter routine
9      for( i = 0 ; i < N_delays ; i++ )
10         FIR_accu32 += FIR_delays[N_delays-1-i] * FIR_coe[i];
11
12     for( i = 1 ; i < N_delays ; i++
   ↪     )
13         FIR_delays[i-1] = FIR_delays[i];
14
15     y = (short) (FIR_accu32 >> shift);
16     return y;
17 }

```

Listing 3: FIR-Polyphasenbandpass Implementierung in C

sich die Frequenzanteile von w_{mark} und w_{space} teilweise überlappen. Daher wird dieser Ansatz auch nicht weiter verfolgt.

6 Komplexes Kammfilter

Eine Alternative zu dem in Kapitel 5 beschriebenen Hilberttransformator ist ein Kammfilter. Um die Frequenzen besser voneinander differenzieren zu können, wird ein Kammfilter genutzt. Dieser kann durch eine Anzahl N Verzögerer realisiert werden. Ziel in der Auslegung des Kammfilters ist es, die Nullstellen so zu legen, dass entweder w_{mark} oder w_{space} des Nutzsignals im positiven Frequenzbereich gedämpft wird, und das jeweils andere w im negativen Bereich. Zunächst wird das Filter dafür so ausgelegt, dass der Abstand zwischen Nullstelle und Hochpunkt in der Filterübertragungsfunktion ca. dem Frequenzhub Δf entspricht, anschließend muss es so verschoben werden, dass die Nullstellen wie beschrieben auf dem gewünschten $w_{mark/space}$ liegen. So wird der Frequenzunterschied zwischen den beiden Frequenzen von Δf auf $2 \cdot f_A$ angehoben. Die Frequenzen sind damit deutlich unterscheidbar und können im nächsten Schritt, der Demodulation eindeutig zugeordnet werden.

6.1 Theorie

Ausgehend von der Übertragungsfunktion

$$H(z) = 1 + z^{-u}$$

mit

$$z = e^{j\omega}$$

ergibt sich

$$H(j\omega) = 1 + e^{-j\omega u}$$

und dem ausklammern von $e^{-j\frac{\omega u}{2}}$:

$$H(e^{j\omega}) = e^{-j\frac{\omega u}{2}} \cdot (e^{j\frac{\omega u}{2}} + e^{-j\frac{\omega u}{2}})$$

$e^{-j\frac{\omega u}{2}} \neq 0$ daher wird nur $(e^{j\frac{\omega u}{2}} + e^{-j\frac{\omega u}{2}})$ weiter betrachtet
Es gilt:

$$(e^{-j\frac{\omega u}{2}} + e^{j\frac{\omega u}{2}}) = 2\cos(\frac{\omega u}{2})$$

Zur Bestimmung der Nullstellen, Argument des Kosinus betrachten.

Allg.: $0 = \cos(\frac{\pi}{2} + k\pi)$

$$\frac{w_0 u}{2} = \frac{\pi}{2} + k\pi$$

$$\Leftrightarrow w_0 = \frac{2\pi k + \pi}{u}$$

mit $k = 0$, $w_0 = 2\pi \cdot \frac{f}{f_A}$ und nach u aufgelöst, ergibt sich:

$$\Leftrightarrow u = \frac{\pi}{2\pi \frac{f}{f_A}}$$

eingesetzt:

$$u = \lfloor \frac{\pi}{2\pi \frac{450Hz}{3832Hz}} \rfloor = 4$$

Der Faktor u bestimmt die Anzahl der benötigten Verzögerer, damit der Abstand eines Maximums des Frequenzganges sowie eine Nullstelle genau dem Abstand der Frequenzen ω_{mark} und ω_{space} entsprechen. Die Übertragungsfunktion ergibt jetzt einen reellen Kammfilter der nun so verschoben werden muss, dass eine Nullstelle im positiven Frequenzbereich auf der hohen Symbolfrequenz liegt. Dies kann erreicht werden, indem das Maximum im Nullpunkt des Frequenzbereiches auf die untere Symbolfrequenz verschoben wird. Die Verschiebung ist somit ω_{space} . Aus der Verschiebungseigenschaft der Diskreten Fouriertransformation ergibt sich folgendes:

$$x(k) \cdot W_N^{kl} \longrightarrow X(n-l)$$

```

1 N = 5
2 fA_green = 3832;
3 freq_comb = (-fA_green:fA_green)/fA_green;
4
5 compl = (0.175 + 0.985j)
6 H_comb = freqz([1, zeros(1, N), compl], 1, 2*pi*freq_comb);

```

Listing 4: In MATLAB wird überprüft ob die Berechnung des N korrekt sind, indem das Filter mit den Frequenzen w_{mark} und w_{space} geplottet wird, siehe Abb. 6.2. Es wird kontrolliert ob die Nullstellen in den richtigen Punkten auf der Frequenzachse liegen.

Für die Verschiebung der bekannten Filterfunktion $h(k) = \{1, 0, 0, 0, 1\}$ mit $N = 5$ ergibt sich somit der Term

$$h(k) \cdot e^{-j\frac{2\pi l k}{N}}$$

Die Variable l bezieht sich jedoch auf den Bereich der Fouriertransformation und nicht auf die uns vorliegende, auf 2π normierte Frequenz $\omega_{space} = \frac{752Hz \cdot 2\pi}{3831.5Hz}$. Für die Umrechnung wird nun der Dreisatz verwendet:

$$\frac{l}{N} = \frac{\omega_{space}}{2\pi} \Rightarrow l = \frac{\omega_{space} \cdot N}{2\pi}$$

Ersetzt man nun das l in dem Verschiebungstheorem durch die errechnete Verschiebung, ergibt sich

$$h(k) \cdot e^{-j\frac{2\pi\omega_{space}kN}{2N\pi}} = h(k) \cdot e^{-j\omega_{space}k}$$

Der nun komplexwertige Filter berechnet sich zu:

$$h(k) = \{1 \cdot e^{-j\omega_{space}0}, 0 \cdot e^{-j\omega_{space}1}, 0 \cdot e^{-j\omega_{space}2}, 0 \cdot e^{-j\omega_{space}3}, 1 \cdot e^{-j\omega_{space}4}\}$$

$$h(k) = \{1, 0, 0, 0, (0.175 + 0.985j)\}$$

Dies ermöglicht die Verwendung eines Verzögerungsdemodulator mit einem Sinus, da im positiven Frequenzbereich ω_{mark} gegenüber ω_{space} um gedämpft wird und im negativen Frequenzbereich das invertierte Verhalten auftritt.

6.2 MATLAB

In *MATLAB* wird das Filter überprüft, Implementiert wird das Filter mithilfe der errechneten Übertragungsfunktion. Dafür wird in *MATLAB* ein mit N -Nullen gefülltes Array verwendet. In Abb. 6.2 ist zu sehen, dass die Nullstellen des Kammfilter gut auf den jeweiligen Frequenzen $-w_{mark}$ und w_{space} liegen.

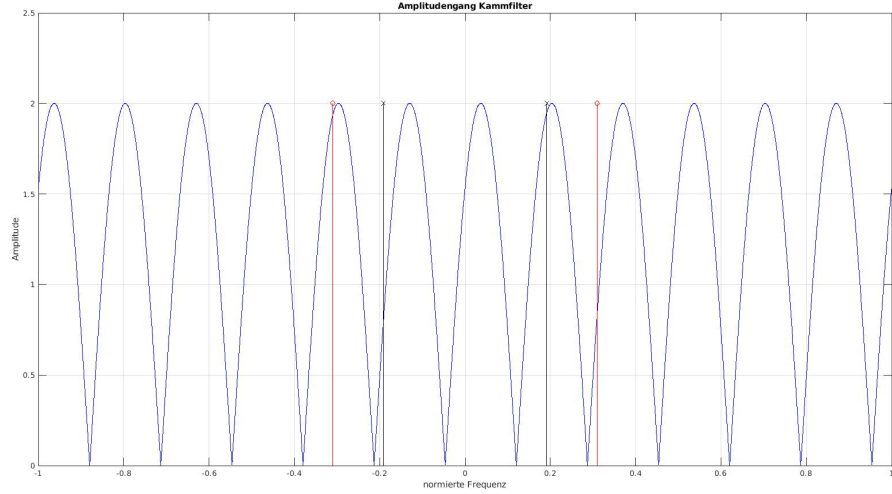


Figure 7: Kammfilter mit den beiden Frequenzen ω_{mark} und ω_{space} , das Kammfilter lässt jeweils eine der beiden Frequenzen passieren.

6.3 ANSI C

Für die C-Implementierung, siehe 6.3, wird jedes Eingabe Sample um $N = 5$ verzögert. Die Verzögerung wird über ein Buffer Array der Länge N realisiert:

7 Verzögerungsdemodulator

Der FM-Verzögerungsdemodulator wird im Worksheet 2 des Projekts behandelt. Der FM-Verzögerungsdemodulator soll das CPFSK-modulierte Signal demodulieren und eine Rechteckfolge ausgeben.

Um auf die Verwendung von komplexer Signalverarbeitung zu verzichten, wird der im Worksheet als "Verzögerungsdemodulator mit reeller Signalverarbeitung" referenzierte Aufbau verwendet. Das komplexe Signal $e^{jn\omega_0}$ auf zwei als reell zu betrachtende Anteile zerlegt:

$$v_I(n) = \Im\{e^{jn\omega_0}\} = \cos(\omega_0 n)$$

$$v_Q(n) = \Re\{e^{jn\omega_0}\} = \sin(\omega_0 n)$$

Aus dem reellen Verzögerungsdemodulator ergibt sich dann folgende Funktion:

$$y(n) = \cos(\omega_0(n-1)) \cdot \sin(\omega_0 n) - \sin(\omega_0(n-1)) \cdot \cos(\omega_0 n)$$

Um die Frequenz ω_0 zurückzugewinnen, wird dann der invertierte Sinus verwendet:

$$\omega_0 = \sin^{-1}(y(n))$$

```

1  short delayed_sample = 0;
2  short cnt = 0;
3  short *delay_iter = NULL;
4  short delay_line[4];
5  short *rotating_rw = delay_line;
6
7  static void process_comb_and_demod() {
8      // Comb filter
9      I_sig = dec_out_short + 175 * delayed_sample;
10     Q_sig = 984 * delayed_sample;
11     ....
12 }
13
14 static void output_sample() {
15     dec_out_short = dec_out >> 5;
16
17     // Delayline counter overflow management
18     if (rotating_rw == delay_line + 4)
19         rotating_rw = delay_line;
20
21     // Rotating delayed sample storage
22     delayed_sample = *rotating_rw;
23     *rotating_rw = dec_out_short;
24     rotating_rw += 1;
25
26     cnt++;
27 }

```

Listing 5: C-Implementierung des Kammfilters mithilfe eines Verzögerer Buffers

Doch zunächst muss $y(n)$ betrachtet werden, denn in dem hier dargestellten Fall, fällt die Trägerfrequenz im Basisband sehr nahe $\frac{f_s}{4}$ und somit $\frac{\pi}{2}$. An dieser Stelle weist der Sinus eine Symmetrie auf, was dazu führen kann, dass der Ausgang des Demodulators nur einen sehr geringen bis gar keinen Hub aufweist. Dies gilt es zunächst mathematisch zu prüfen:

$$\omega_{mark} = 2\pi \cdot \frac{\frac{f_s}{4} + \Delta f}{f_s} = \frac{\pi}{2} + \frac{2\pi\Delta f}{f_s}$$

$$\omega_{space} = 2\pi \cdot \frac{\frac{f_s}{4} - \Delta f}{f_s} = \frac{\pi}{2} - \frac{2\pi\Delta f}{f_s}$$

Zunächst wird der Term $y(n)$ anhand des Additionstheorems vereinfacht:

$$\begin{aligned} y(n) &= \{ \cos(\omega_0 n) \cos(\omega_0) + \sin(\omega_0 n) \sin(\omega_0) \} \cdot \sin(\omega_0 n) - \\ &\quad \{ \sin(\omega_0 n) \cos(\omega_0) - \cos(\omega_0 n) \sin(\omega_0) \} \cdot \cos(\omega_0 n) \\ y(n) &= \sin(\omega_0 n) \cos(\omega_0 n) \cos(\omega_0) + \sin^2(\omega_0 n) \sin(\omega_0) - \\ &\quad \sin(\omega_0 n) \cos(\omega_0 n) \cos(\omega_0) + \cos^2(\omega_0 n) \sin(\omega_0) \\ y(n) &= \sin^2(\omega_0 n) \sin(\omega_0) + \cos^2(\omega_0 n) \sin(\omega_0) \\ y(n) &= \{ \sin^2(\omega_0 n) + \cos^2(\omega_0 n) \} \sin(\omega_0) = \sin(\omega_0) \end{aligned}$$

Ein einsetzen in die Funktion ergibt für beide Fälle folgendes Ergebnis:

$$y(n)|_{\omega_{mark}} = \sin\left(\frac{\pi}{2} + \frac{2\pi\Delta f}{f_s}\right)$$

$$y(n)|_{\omega_{space}} = \sin\left(\frac{\pi}{2} - \frac{2\pi\Delta f}{f_s}\right)$$

Der Sinus an der Stelle $\frac{\pi}{2}$ ist jedoch symmetrisch, weswegen der Demodulator bis auf kleine Abweichungen stets für beide Frequenzen das selbe Ergebnis berechnen wird. Der komplexe Kammfilter führt durch das asymmetrische Dämpfungsverhalten jedoch zu folgenden Verhältnissen:

Leistung bei den Signalfrequenzen vor dem Filter:

$$|X\{-\omega_{mark}\}|^2 = |X\{-\omega_{mark}\}|^2 \leftrightarrow |X\{-\omega_{space}\}|^2 = |X\{-\omega_{space}\}|^2$$

Leistung bei den Signalfrequenzen nach dem Filter:

$$|X\{-\omega_{mark}\}|^2 >> |X\{-\omega_{mark}\}|^2 \leftrightarrow |X\{-\omega_{space}\}|^2 << |X\{-\omega_{space}\}|^2$$

Es gehen also $-\omega_{mark}$ und ω_{space} in den Demodulator ein. Somit ergibt sich für die obere Frequenz das Inverse des Ausgangs bei der unteren Signalfrequenz. Der Hub wird so maximal:

$$\sin(-\omega_{mark}) = -\sin(\omega_{mark}) = -\sin(\omega_{space}) \iff \sin(\omega_{space})$$

7.1 MATLAB

7.2 ANSI C

Es wird der reelle Verzögerungsdemodulator implementiert der jedoch aus Effizienzgründen auf die Bildung des invertierten Sinus verzichtet, da dieser letztlich einen nichtlinearen Verstärker darstellt, welcher beim Überschreiten des Wertebereichs von $[-\frac{\pi}{2}; \frac{\pi}{2}]$ nicht definiert ist. In den durchgeführten Simulationen reichte das SNR stets aus, um das Signal verlässlich zu dekodieren.

8 Decodierung

Der Dekodierer liest die Ausgangswerte des Demodulators binarisiert als $D = \{0, 1\}$ bitweise ein und schreibt diese in eine Puffer. Die Symbolfrequenz von $50Hz$ bestimmt die Abtastrate des Dekodierers. Da bei dem Start/Stop-Bit um das halbe Bit der Stop-Sequenz erkennen zu können liegt hier folgender Ansatz zugrunde:

$$f_{decode} = \lfloor f_A * \frac{1}{f_{symbol} \cdot 2} \rfloor$$

eingesetzt

$$f_{decode} = 3831,499Hz * \frac{1}{50Hz \cdot 2} = 38,31$$

So wird durch einen Takteiler die Abtastfrequenz erneut gesenkt. Der Dekodierer detektiert nun die Start/Stop folge und synchronisiert sich so auf den Datenstrom. Wird die Datenfolge $0x001F$ empfangen, werden die davor liegenden 10 Bits invertiert und um den Faktor zwei unterabgetastet. Der so entstehende Bitvektor wird unter Zuhilfenahme einer Lookuptable dann in Buchstaben oder Zahlen übersetzt. Das Umschalten zwischen zwei Tabellen geschieht bei den jeweiligen Zeichenfolgen.

8.1 MATLAB

In listing 8.1, der MATLAB Implementierung des Decoders wird die durch den Verzögerungsdemodulator binarisierte Ausgabe Blockweise aus dem Buffer *binarized* ausgelesen. Sobald die Start/Stop Bit-Sequenz [11100] erkannt wird, werden die Bits aus dem Buffer in der Reihenfolge umgedreht, da hier das MSB zuerst gesendet wird. Die Bits werden in den Datentyp String umgewandelt und anschließend als Dezimalzahl interpretiert. Die so ermittelte Zahl entspricht dem Index des Buchstaben der Lookuptable. Sollte der so erkannte Buchstabe das Umschalten zwischen Zahlen und Buchstaben repräsentieren, wird die Lookuptable entsprechend ausgetauscht.


```

1  for r = 1:length(binarized)-4
2      % get the next 5 bin. values
3      temp = binarized(r:1:r+4);
4      % look for start/stopbit
5      if temp == [1 1 1 0 0]
6          if r > 12
7              if r > prev_r + 14
8                  prev_r = r;
9                  % MSB first
10                 search = fliplr(binarized(r-10:2:r-1));
11                 search_str = num2str(search);
12                 search_str(isspace(search_str)) = '';
13                 % get index
14                 idx = bin2dec(search_str)
15                 umsch = lut2(idx);
16                 if umsch == '%'
17                     lut2 = let;
18                     continue
19                 elseif umsch == '#'
20                     lut2 = num;
21                     continue
22                 end
23                 y = [y lut2(bin2dec(search_str))];
24             end
25         end
26     end
27 end

```

Listing 6: *MATLAB*-Implementierung: Der Decoder in *MATLAB* arbeitet mit der Umwandlung der Binärdaten in das benötigte Zahlenformat. Zunächst in Strings, anschließend in eine Dezimalzahl welche den Index in einer Lookup Tabelle repräsentiert.

8.2 ANSI C

Der Taktteiler wird durch zwei Zähler realisiert, der Hauptzähler ruft die Dekodierer alle 38 Zeichen auf, alle 3 Perioden wird jedoch bis 39 gezählt um die $0.31Hz$ zu kompensieren. Der Dekodierer selbst verschiebt den bisher eingetroffenen Datenstrom nach links und verodert das neu erhalten Bit mit dem Puffer. Eine 16Bit Variable stellt hierbei den FIFO Puffer dar. Die 5 letzten Bits werden mit der Start/Stop Datenfolge verglichen. Wird diese erfolgreich detektiert, so wird der Puffer kopiert und die fünf Start/Stop Bits nach rechts aus dem Buffer geschoben. Um die Datenfolge zurückzugewinnen, werden die letzten 10 Bits des neuen Puffers maskiert, invertiert und unterabgetastet, sodass sich die fünf Symbolbits ergeben. Diese Werden dann als short-Typ als Feldindex für die Lookuptable verwendet. Mit der implementierten Prozedur wird jedoch das Symbol `xxxx111100` ebenfalls als Start/Stop-Folge interpretiert, weswegen ein weiterer Zähler den Abstand zur letzten detektierten Start/Stop-Folge prüft. Dies führt zu möglichen Synchronisierungsfehlern bis die erste Symbolfolge auftritt, die keine Start/Stop-Sequenz enthält.

References

N.N (2021). Online <https://commons.wikimedia.org/w/index.php?curid=635074>.

```

1 void decode(unsigned short bit) {
2
3     if (!current_lut)
4         current_lut = lookup_char;
5
6     buffer = buffer << 1;
7     buffer = buffer | bit;
8     startstop = buffer & 0x001F;
9
10    if (startstop_holdoff > 0) // Underflow protection
11        startstop_holdoff -= 1;
12
13    if (startstop == STOP_SEQUENCE && startstop_holdoff == 0) {
14        startstop_holdoff = 11;
15        // Shift over the start and stop section
16        // and mask the 10 message bits
17        index_pre = (buffer >> 5) & 0x03FF;
18        // Compress bit stream to half its size
19        // Turn this 1 1 1 1 0 0 1 1 0 0 into 1 1 0 1 1
20        real_index = (((index_pre >> 0) & 0x0001) << 4)
21            | (((index_pre >> 2) & 0x0001) << 3)
22            | (((index_pre >> 4) & 0x0001) << 2)
23            | (((index_pre >> 6) & 0x0001) << 1)
24            | (((index_pre >> 8) & 0x0001) << 0);
25
26        if (real_index == SWITCH_TO_CHAR) {
27            // Change to characters
28            current_lut = lookup_char;
29        } else if (real_index == SWITCH_TO_NUM) {
30            // Change to numbers
31            current_lut = lookup_num;
32        } else {
33            #ifdef USE_MSVC_ANSI_C_SIM
34            // Everything else is a valid character
35            printf(" >> %c <<\n", current_lut[real_index-1]);
36            #endif
37        }
38    }
39 }

```

Listing 7: C-Implementierung: Funktion des Decodierers