RTTY-Demodulator

Zielstellung

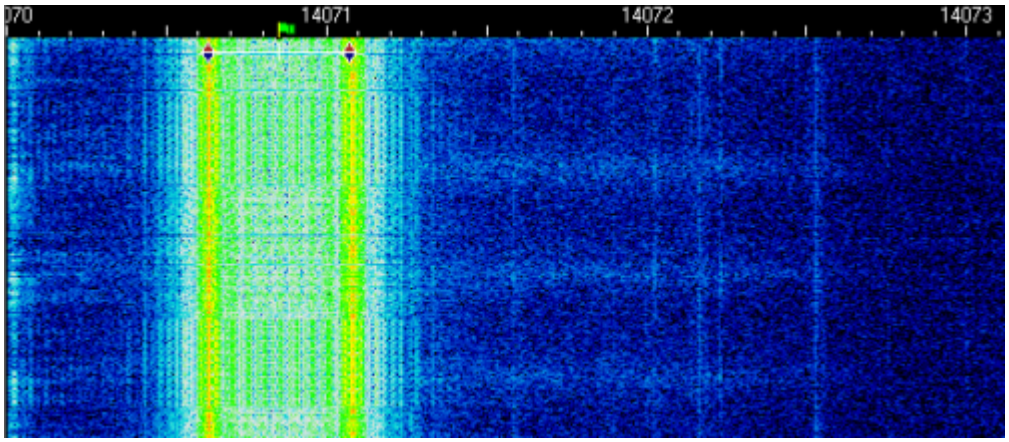
Es soll ein FSK-Demodulator entwickelt werden. Die Umsetzung soll in Assembler auf einem AtMega-Prozessor passieren, die dahinführende Entwicklung wird hier dokumentiert werden.

Einsatzzweck für das Gerät ist entweder RTTY der Funkamateure bzw des DWD. Als HF-Frontend zum praktischen Einsatz ist die Verwendung eines gewöhnlichen Transceivers geplant, sodass die Demodulation einer AFSK erfolgt, die Demodulation einer evtl. vorhandenen FM, SSB, AM usw wird vom Funkgerät durchgeführt.

Beschreibung der Betriebsart

RTTY - „Radio Teletype“ ist eine digitale Betriebsart, mithilfe welcher einfach Daten (v.A. Text) übertragen werden können. Es basiert auf Frequenzumtastung mit 2 Frequenzen. Je nach Betriebsmodus sind der Abstand beider Frequenzen und die Baudrate unterschiedlich. Absolute Frequenzen sind nicht festgelegt, weil im Rahmen der Demodulation (z.B. SSB) ohnehin eine Frequenzverschiebung auftreten kann.

Im folgenden Bild ist ein typisches Spektrum dieser Betriebsart zu sehen, deutlich erkennbar sind die 2 umgetasteten Frequenzen.



Häufig genutzte Modi:

- Funkamateure: Shift: 170Hz, 45.45 Baud
- DWD: Shift: 450Hz, 50 Baud (invertiert)
- SITOR (z.B. bei NAVTEX): 170Hz Shift, 100 Baud (kein normaler Baudot-Code)

Datenformat

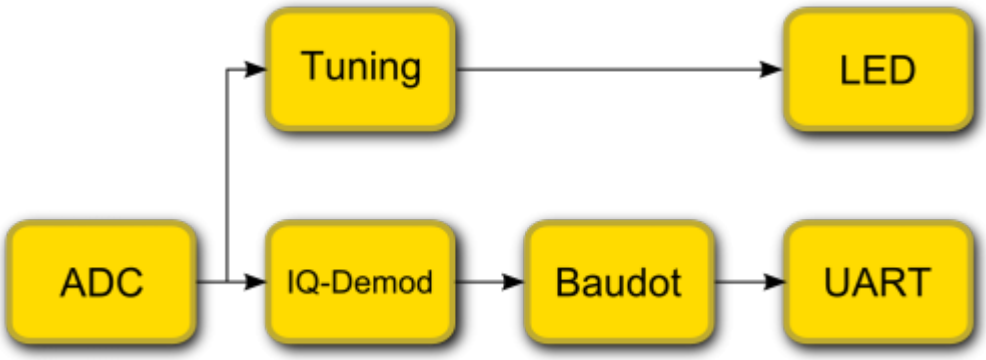
Es wird zur Übertragung sog. Baudot Code benutzt. Dieser kodiert jedes Zeichen mit 5 Bit. Es können alle Buchstaben (keine Unterscheidung zwischen Groß- und Kleinschreibung) kodiert werden. Durch ein Steuerzeichen kann auf eine zweite Gruppe umgeschaltet werden, in der diverse Sonderzeichen

und alle Zahlen vorhanden sind. Hier ist der verwendete Code übersichtlich zu finden. Weiß repräsentiert „Mark“, Schwarz steht für „Space“. Die Übertragung erfolgt LSB first.

Im Leerlauf (kein Zeichen wird übertragen) bleibt der Sender auf der Mark-Frequenz stehen. Jedes Zeichen beginnt mit einem Space-Startbit und wird mit 2 Mark-Stopbits abgeschlossen. Dies ist hier zu sehen.

Im normalen Modus ist Mark die höhere Frequenz und Space die niedrigere. Mark repräsentiert eine Binäre 0, Space eine 1. Im invertierten Modus ist zusätzlich zur Umkehrung Mark/Space auch die Zuordnung der Bits umgekehrt.

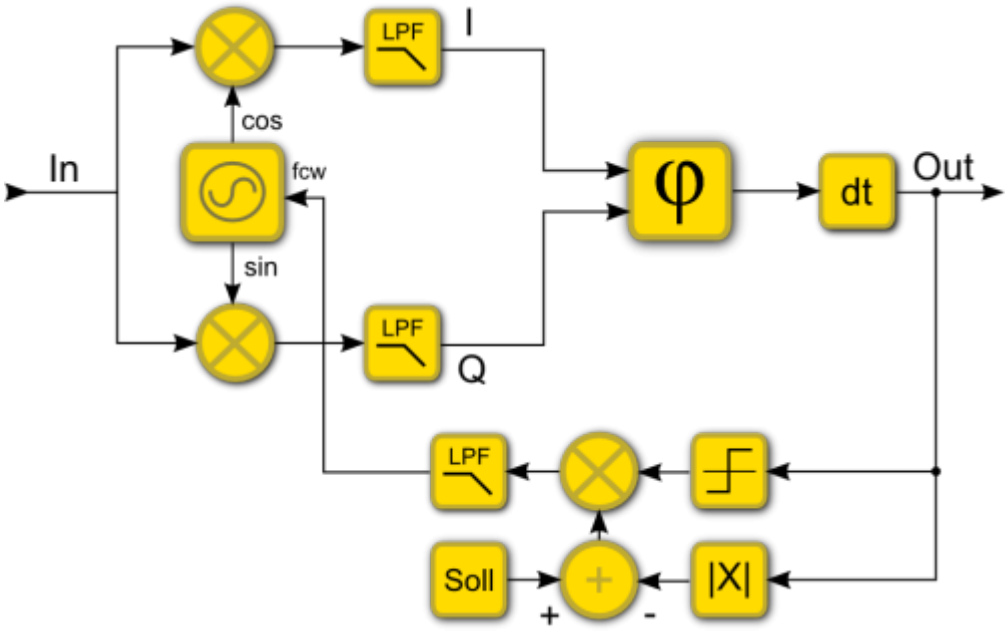
Konzept



Ausgegangen wird im Folgenden von der direkten Abtastung der NF eines FuG. Das NF-Signal vom FuG wird Tiefpassgefiltert und vom ADC abgetastet. Es wird intern dem FSK-Demodulator zugeführt, aus welchem die demodulierten, digitalen Daten herauskommen. Die Baudot-Erkennung wandelt die digitalen Daten des Demodulators in ASCII-Zeichen. Die Ausgabe dieser erfolgt per UART und (optional) auf einem kleinen Display (HD44780?).

Als Abstimmanzeige wird das abgetastete Signal einem steilflankigen IIR-Peakfilter zugeführt, dessen gleihgerichteter Ausgangswert eine LED in der Helligkeit moduliert. Man verstimmt das Funkgerät bis die LED hell zu leuchten beginnt, den Rest erledigt die AFC automatisch.

Demodulationskonzept



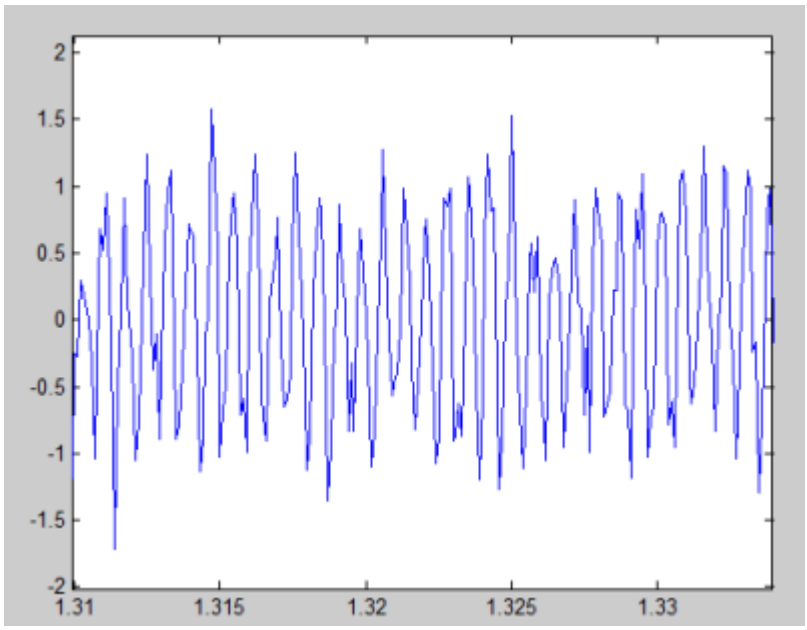
Das Eingangssignal wird mit einem komplexen NCO in der Mitte der umgetasteten Frequenzen multipliziert. Der dann entstehende komplexe Zeiger wird auf links- oder rechtsdrehung überprüft (positive bzw. negative Frequenz resultiert). Aus der Phasenänderung (Drehwinkel des Zeigers) zwischen 2 Samples kann die aktuell gesendete Frequenz abgelesen werden. Eine konstante Frequenz bedeutet eine Drehung des Zeigers mit konstanter Geschwindigkeit, also einen DC-Offset als Ausgangssignal. Aus dem DC-Offset kann direkt ein Fehlersignal abgeleitet werden, welches den NCO auf die korrekte Mittenfrequenz korrigiert. Da die Shift des Signals bekannt ist, kann die ideale Zeigerumlaufgeschwindigkeit berechnet werden und als Sollwert für die Regelung angenommen werden.

Ist der Ausgang positiv, führt ein zu großer DC-Offset zur Korrektur des FCW nach unten. Im umgekehrten Fall (y ist negativ) führt ein betragsmäßig zu großer Offset zur Korrektur nach oben. Ein Schleifenfilter bereinigt das Fehlersignal von Transienten (z.B. beim Umtasten der Frequenzen).

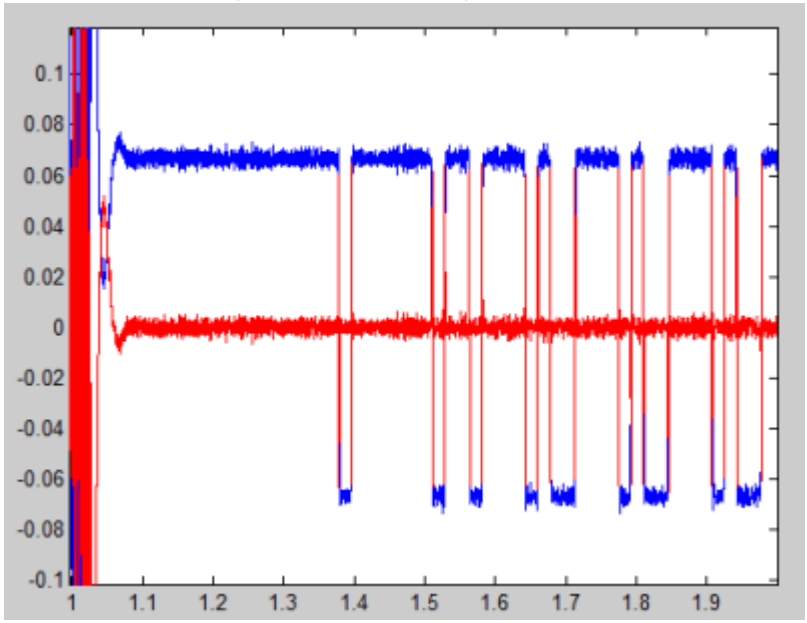
Der Ansatz, den Rechenaufwand (Drehwinkelbestimmung) zu minimieren, indem eine FM-Demodulation verwendet wird, wurde verworfen. Durch die dabei notwendige Normierung des Signals benötigt die Betragsbildung des Zeigers, welche hinsichtlich der Komplexität einer Winkelbestimmung gleichzusetzen ist (CORDIC).

Modulation,Demodulation, Symbolabtastung und Wandlung in einen ASCII-String ist funktionierend Matlab implementiert. rttymod.m rtty2.wav

Rohsignal, 30dB SNR:



Demoduliertes Signal und Fehlersignal, 30dB SNR - am Anfang ist der Einschwingvorgang erkennbar:



Dekodierter Text:

```
TSUQDKEFln???l THEl QUICKl BROWNl FOXl JUMPSl OVERl THEl LAZYl DOGl
n1234567890l n???l THEl QUICKcfBROWNl FOXl JUMPSl OVERl THEl LAZYl DOGl
n1234567890l l cccfC*BBCGX
```

Die implementierte Demodulation und Dekodierung funktioniert im Modell (auch bei schlechtem SNR, getestet z.B. 10dB) ohne Fehler. Die Regelung ist in ihren Parametern noch optimierbar. Als Parameter dafür sind in der Rückführung der Anteil des Tiefpasssignals, der ungefilterte Anteil und die Zeitkonstante des Tiefpass vorhanden.

Das Modell wurde außerdem (sowohl mit FIR als auch IIR-Filter bei der IQ-Mischung) mit real vom Funkgerät aufgenommener NF ausprobiert. Es wurde dafür das DWD RTTY-Wettersignal auf 10.100,8kHz genutzt. Die Erkennung war mit der Referenzimplementierung in MixW gleichgut. rtty_dwd_usb.wav , demoduliert mit rttymod_dwd.m.

```
RYRYRYccflCQ CQ CQ DE DDKn2 lDDHn7 lDDKn9ccflFREQUENCIES   n4583 lKHZ
n7646 lKHZ   n10100.8 lKHZ
```

Auch starke Frequenzschwankungen (Drehen am Abstimmknopf usw) sind kein Problem, die Regelung funktioniert sehr effizient, selbst ein ständiges Verstimmen des Signals bringt keine Empfangsfehler. Beispiel: rtty_dwd_moving_hard.wav Der Einfluss von Störsignalen (z.B. andere RTTY-Signale in der Empfängerbandbreite, Dauerträger) muss noch untersucht werden.

Andere Verfahren, wie die nichtkohärente Demodulation mit 2 Bandpässen (keine AFC möglich) oder einer PLL, die zwischen den FSK-Frequenzen im Ziehbereich hin- und hergezogen wird (schlechtes Einrastverhalten) wurden verworfen.

Symboltaktrückgewinnung

Die Symbolrückgewinnung ist dank des einfachen Modulations-Schemas relativ einfach zu bewerkstelligen. Dies ist durch ein paar positive Eigenschaften der Übertragung zu begründen:

- im Leerlauf sendet der Sender „Mark“
- jedes Zeichen wird mit einem „Space“ eingeleitet
- jedes Zeichen ist genau 5 Bit lang, dies ist kurz genug um keine Syncronisation auf den Symboltakt zu benötigen, die Flanke des Startbit reicht aus

Es wird also auf im demodulierten Signal auf eine Flanke von „Mark“ zu „Space“ gewartet. Daraufhin wird nach Vergehen von 0,5 T_bit auf vorhandensein des Start-Bit geprüft. Aller T_bit werden danach die 5 Baudot-Bits abgetastet. Um genau T_bit später ist der Sender im Stopbit (2x Mark) - Das Zeichen kann an die Baudot-Dekodierung weitergegeben werden und die Zeichenerkennung wartet auf das nächste Startbit.

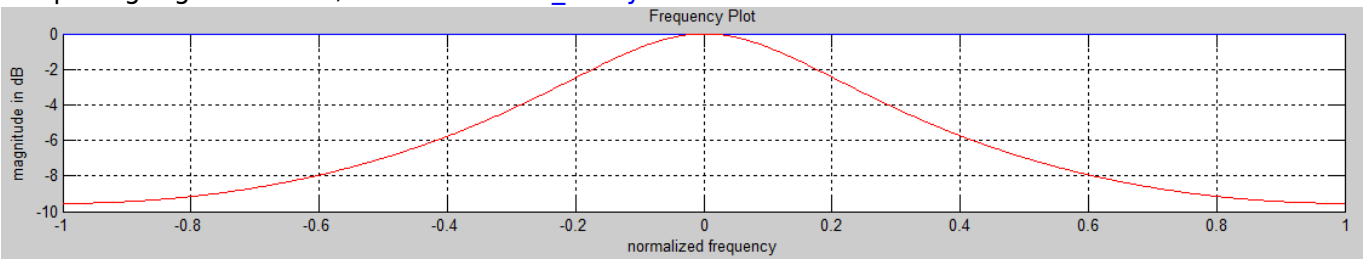
Im AtMega wird dieser Algorithmus als kleine State Machine umgesetzt.

Filterung

Schleifenfilter

Zur Filterung des Fehlersignals findet ein einfacher diskreter RC-Filter (mit Proportionalanteil) Anwendung. Eine Multiplikation kann hier eingespart werden, indem der Koeffizient als Zweierpotenz gewählt wird.

Frequenzgang des Filters, ermittelt mit rc_attiny.m:



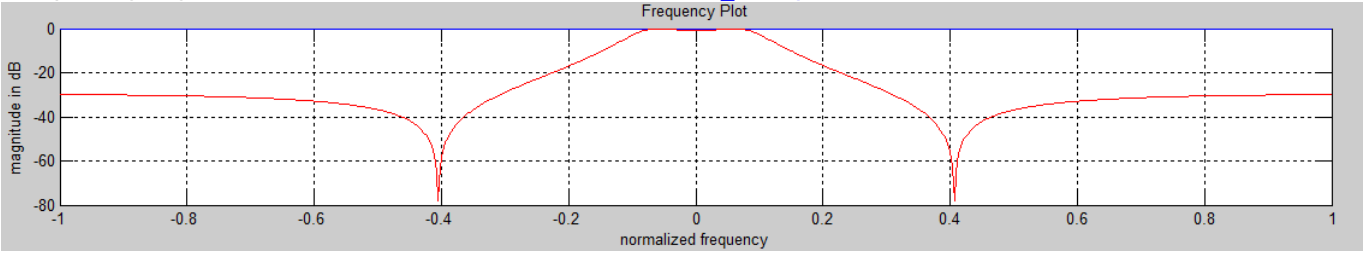
I/Q-Zweige

Im I und Q-Zweig wird das Tiefpassfilter als IIR-Biquadfilter ausgeführt. Es hat sich im Modell gezeigt, dass 2 Stufen eines solchen Biquad-Filters ausreichen. Es muss so dimensioniert werden, dass die Grenzfrequenz oberhalb der (halben) Shift liegt. Die nichtlineare Phase des Filters wirkt sich nicht negativ aus, weil die positive und negative Frequenz jeweils im I- und Q-Zweig die gleiche Frequenz repräsentieren. Nur, wenn die Regelung nicht eingeschwungen ist, hat diese einen eventuell störenden Einfluss.

Bei einer Audio-ZF von 1,5kHz und 850Hz Shift wird die untere FSK-Frequenz auf 2575 Hz hochgemischt, dort muss also bereits eine große Dämpfung vorhanden sein. Um Verzögerungsglieder einzusparen, wird das Filter als Direktform II implementiert.

Die sequentielle Implementierung wurde zuerst in Matlab durchgeführt, um Implementierungsfehler später zu verhindern.

Frequenzgang einer Stufe dieses Filters, erstellt mit iir_attiny.m:



Entwicklungsumgebung

Es wird mit dem internen ADC des AtMega gearbeitet, welcher als erstes auf eine feste Samplerate eingestellt wird. Zur Kontrolle sollen Signale „analog“ ausgegeben werden, es gibt hierfür 2 PWM-DACs, so ist die Darstellung zeitlich veränderlicher Größen am Oszilloskop möglich. Ein Pin wird zum Beginn des Rechenkerns und am Ende umgeschaltet, durch ihn ist am Oszilloskop die ungefähre Prozessorauslastung feststellbar.

Zur Validierung von Funktionsblöcken ist eine Anbindung an Matlab per UART vorgesehen, durch diese können die eigenen Ergebnisse mit verifizierten Funktionen in Matlab verglichen werden.

TODO für die HW:

- AtMega48, TQFP
- UART (RxD: PD0, TxD: PD1)
- 16MHz XTAL (PB6, PB7)
- 2xADC (PC0..5), Filterung 1.Ordnung @ 3kHz
- 2x DAC (PD5, PD6), Filterung 4. Ordnung @ 3kHz (PWM@61 kHz, DAC@6kHz)
- LED
- ISP

Arbeitspakete und Milestones

Es sind folgende, grundlegende Arbeitspakete zu bewältigen

- UART Inbetriebnahme - **code ok, testen!**
  - Loopback-Test
  - Einbindung in Matlab
- Timer Inbetriebnahme - **ok**
  - Pin-Toggle
- ADC Inbetriebnahme - **ok**
  - Loopback @ 12kHz
- Mathematikoperationen **todo**
  - saturierende Addition
  - saturierende Subtraktion
  - Fractional Multiplikation
  - Fraction Division
  - per Matlab testen (Eingabe: 2 Operanden, Ausgabe: 1 Ergebnis)
- Sinus/Kosinus (LUT mit Interpolation) **todo**
  - per UART und Matlab testen
  - Eingabe: phi, Ausgabe: sin(phi), cos(phi)
- Arkustangens (LUT mit Interpolation) **todo**
  - per UART und Matlab testen
  - Eingabe: x, Ausgabe: atan(x)
- NCO (Oszillator für IQ-Mischung) **todo**
  - Phasenakkumulator per UART validieren (Eingabe: step, Ausgabe: Wert –> Sinus in Matlab)
  - Phasenakkumulator am DSO validieren (FCW → Ausgangsfrequenz)
  - Phasenakkumulator mit Sinus und Kosinus am DSO validieren (FCW → Ausgangsfrequenz)
  - Phasenverschiebung des komplexen NCO bestimmen (frequenzunabhängig 90°)
- CORDIC (Phasenwinkelbestimmung) / Division **redefine**
  - per Matlab testen (Eingabe: I, Q, Ausgabe: phi)
- BiQuad-Filter (für I/Q-Daten) **todo**
  - passenden Koeffizientensatz berechnen
  - per DSO und Sig-Generator testen
- RC-Filter (für Fehlersignal) **todo**
  - per DSO und Sig-Generator testen
- Baudot-Code Dekodierung (2x LUT) **todo**
  - per UART testen (Eingabe: Baudot, Ausgabe: ASCII)

Interna

Systemdaten:

- verwendeter MCU: AtMega48
- Systemtakt: 16MHz
- ADC Samplerate (Timer): 12kHz
- Zur Verarbeitung bleiben 1333 Takte pro Sample

Bitbreite verschiedener Register:

- ADC-Input: 8bit, unsigned → zu 8 bit signed konvertieren
- NCO-PAC: 16bit, unsigned
- NCO-sin/cos: 8 bit, signed
- Mischer: Input 8bit signed * 8bit, signed, Output 16bit, signed
- RC-Filter: Input 16bit, signed, Output 16bit Signed (intern 24bit)
- BiQuad-Filter: Input 16bit, signed, Output 16bit Signed (intern 24 bit)

Rechenintensive Operationen:

- Sin/Cos/Atan jeweils max. 80 Takte (16bit)
- Division: 255 Takte (16/16 signed), 103 Takte (8/8 signed)

Skalierung:

- Sin(x)/Cos(x) 0x0000 .. 0x7fff (für 0..90)
- Atan(x) 0x000 .. 0x7fff (für x = 0..20, 0x7fff für >20)

NCO:

- Takt: 12kHz
- PAC: 16 bit
- FCW: 16 bit
- Frequenzauflösung: 12kHz/2^16 = 0,183Hz/Inkrement

Links

Datasheet AtMega8
Datasheet AtMega48
AVR Instruction Set
AVR200: Multiply and Divide Routines