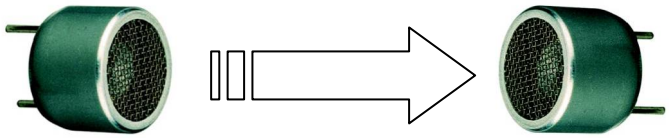
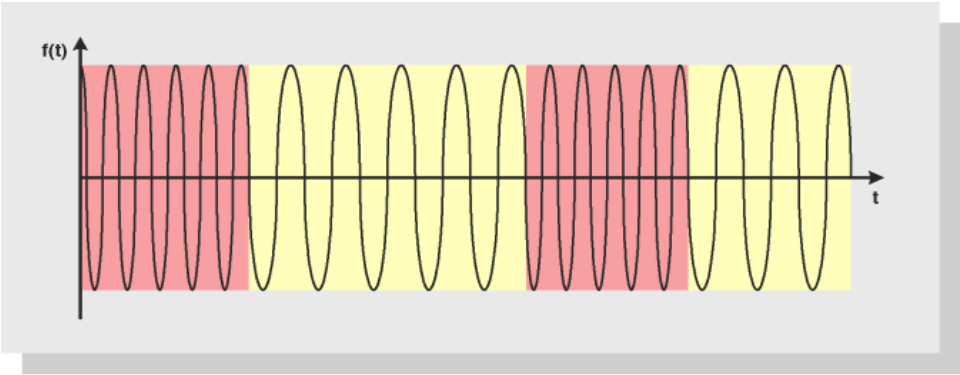
Funktionsbeschreibung
Ultraschall-Modem
Zoltan Gradwohl 15.04.04



Einleitung

Die Aufgabe ist es, binäre Daten mit Hilfe von Ultraschall zu übertragen. Ich habe die so genannte Frequenzumtastung (Frequency key shifting FSK) als Übertragungsmethode gewählt. Hier ist eine kurze Erläuterung:

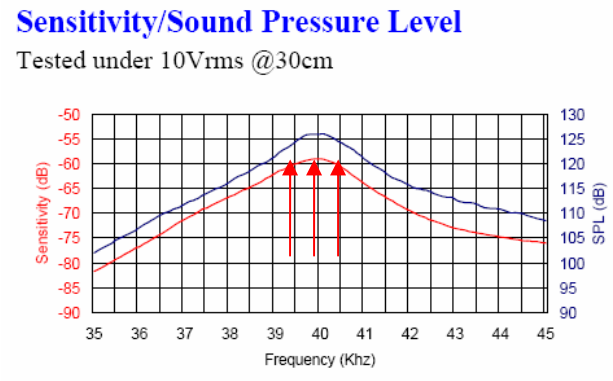
Frequenzumtastung
Engl. Frequency Shift Keying (FSK). Digitale Variante der Frequenzmodulation (FM), eingesetzt beispielsweise bei der elektrischen Datenübertragung im Modemverfahren. Hierbei werden die binären Kennzustände des Datensignals durch unterschiedliche Frequenzen dargestellt. Eine eigentliche Trägerfrequenz, die wie bei der FM mit einem analogen Signal moduliert würde, gibt es nicht. Man spricht stattdessen von einer Mittenfrequenz. Auch sie tritt nur als Rechenwert, nämlich als das arithmetische Mittel der Kennfrequenzen, auf. Als Ruhelage wird die Kennfrequenz für den Binärzustand 1 gesendet.



Nun ist es also möglich durch die Erzeugung von verschiedenen Frequenzen Informationen zu übertragen. In diesem Versuchsaufbau benutzte ich drei unterschiedliche Frequenzen für die Übertragung der einzelnen Daten-Bits (0 und 1) und des Synchronisationsbits (2):

- 37,7 kHz (theoretisch 39,6 kHz) = 0
- 38,1 kHz (theoretisch 40,0 kHz) = 1
- 38,5 kHz (theoretisch 40,4 kHz) = 2

Die Frequenzen werden so ausgewählt, dass sie möglichst nahe an der Resonanzfrequenz des Senders und des Empfängers liegen, um die el. Sendeleistung so gering wie möglich zu halten.



Sender

Die Aufgabe des Mikrokontrollers auf der Senderseite ist die variable Frequenz, entsprechend der zu sendenden Bits zu erzeugen. Dies geschieht mit einem Atmega8, der mit 8Mhz getaktet läuft. Mit Hilfe des PWM- Generators kann man sehr einfach eine beinahe beliebige Frequenz erzeugen. Hierfür wird der Register OCR1A mit einem Wert geladen, welche sich aus der folgenden Formel herleiten lässt.

$$OCR1A = \frac{clk_{CPU}}{2 \cdot f}$$

Z.B. muss bei einer CPU-Taktfrequenz von 8Mhz und einer gewünschten Frequenz von 40kHZ der OCR1A Register mit 100 geladen werden.

$$OCR1A = \frac{8000000\frac{1}{s}}{2 \cdot 40000\frac{1}{s}} = 100$$

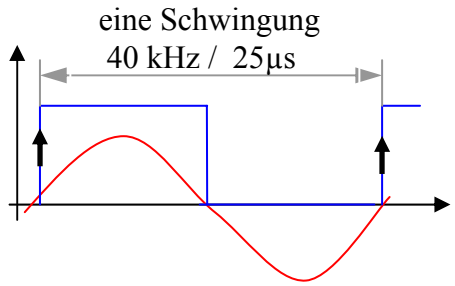
Auszug aus dem Sender-Programm:

```
// Einstellen der PWM-Modi
TCCR1B=(1<<WGM13)|(1<<WGM12)|(1<<CS10);// clkI/O/1 (No prescaling),WGM10,11,12,13-> Fast PWM (siehe S.97)
TCCR1A=(1<<WGM11)|(1<<WGM10)|(0<<COM1A1)|(1<<COM1A0); // COM1A0:Toggle OC1A on compare match (siehe S. 96,
S. 89)

OCR1A = 100;          // Beispiel: Frequenz einstellen (40,0 kHz)

...
...

      uint8_t out_= BIN8(0,0,1,1,0,1,1,1); );// dieser Wert (55 ) soll testweise gesendet werden
      uint8_t k;
      uint16_t x;

      // Sendet die 8 bit hintereinander
      for (;;)
      {
            for (k=0;k<8;k++)
            {
                  if(((out_>>k) & 1) ==1) OCR1A = 101;          // High (39,6 kHz)
                  else OCR1A = 100;                             // Low (40,0 kHz)
                  for(z=0;z<500;z++) asm volatile ("nop");      // Pause 4001 cyclen / 500 µs
            }
            OCR1A = 99;                                         // Synchronization  (40,4 kHz)
            for(z=0;z<1020;z++) asm volatile ("nop");           // Pause 8161 cyclen / 1020 µs
      }
```

Weil es Zeit braucht, bis sich die verschiedenen Frequenzen eingeschwungen haben müssen immer mehrere Schwingungen pro Bit gesendet werden. Aus Versuchen wurden ermittelt, dass ungefähr 500 µs lang eine bestimmte Frequenz gesendet werden muss, um den Sender und Empfänger das Einschwingen zu ermöglichen. Während dieser Zeit finden bei 40 KHz 20 Schwingungen statt. Daraus folgt, das für 10 Bit (8 Datenbits und 2 Synchronisationsbits) 20 * 10 = 200 Schwingungen oder 0,5 ms * 10 = 0,005s benötigt werden. Also braucht man für das Senden eines Bytes 0,005 s was einer Datenrate von 200byte/s oder 1600bps entspricht.

Empfänger

Das Sinus-Signal des US-Empfängers wird durch zwei OP-s verstärkt. Weiterhin wird in den Nulldurchgängen der Sinus-Schwingung eine Flanke generiert. Auf diese Art hat man den Sinus in eine Rechtecksignal umgewandelt. Die Breite der Rechteckschwingung muss nun vom MC erfasst werden. Deswegen wird die Zeit zwischen zwei positiven Flanken mit dem Timer1 gemessen. Da der MC auf der Empfängerseite mit 16 MHz arbeitet erhält hat man bei 40kHz 400 Maschinen-Cyclen, die zur Verarbeitung der Daten zur Verfügung stehen.



Da das Signal etwas verrauscht ist wird der gleitende Durchschnitt aus den jeweils letzten acht gemessenen Schwingungen gebildet (siehe Excel Diagram).

Auszug aus dem Empfänger-Programm:

```
////////////////////////////////////////////////////////////////////////////////////
// input, mesures puls lenght
////////////////////////////////////////////////////////////////////////////////////
SIGNAL(SIG_INTERRUPT0)                    // signal handler for external interrupt int0
{
          // Diese Funktion wird nach jeder positiven Flanke gestartet.
          // Aufgaben:          - Messen der Pulslängen
          //                    - Bilden des gleitenden Durchschnitts
          //                    - Erkennen der Frequenzen (diskretisierung)
          //                    - Zuordnen der Bits zu einem Byte
          //                    - Synchronisation

          Puls_l_original[zaehler2]=(uint8_t)(TCNT1-375);  // Liest den Timerzeit
                    // Begrünung  für -375 : Messwert variiert von 380..420.
                    // Nur der Bereich von 380..420 enthält Informationen
                    // Um nicht einen 16bit Wert verwenden zu müssen, wird hier die 375 subtrahiert.

          glDurchschnitt[zaehler0]=Puls_l_original[zaehler2];
                    // bilden des gleitenden Durchschnitts  aus 8 werten

          TCNT1=0;   // Timer zurücksetzten

          zaehler0++;
          zaehler0 = (zaehler0==8) ? 0 : zaehler0;
                    // Zähler für glDurchschnitt



          summe=0;
          for(zaehler1=0;zaehler1<8;zaehler1++) summe+=glDurchschnitt[zaehler1];
          Puls_l_gefiltert[zaehler2]=summe;
                    // bildet die Summe für den glDurchschnitt , es wir aber nicht durch die Anzahl
                    // geteilt, da dies nur Zeit kostet und keine Neune Informationen liefert.



                    // Zuordnen von verschiedenen bits entsprechend der Frequenz zu einer Variablen
                    // „signal_" Dieser enthält nun die binären Informationen
          if(Puls_l_gefiltert[zaehler2]>=GRENZE_H_u)
          {
                    signal_=1;
          }

          if(Puls_l_gefiltert[zaehler2]>=GRENZE_L_u  && Puls_l_gefiltert[zaehler2]<GRENZE_L_o  \
          && Puls_l_gefiltert[zaehler2-4]>=GRENZE_L_u  && Puls_l_gefiltert[zaehler2-4]<GRENZE_L_o)
          {
                    signal_=0;
          }
```

```
            if(Puls_l_gefiltert[zaehler2]<=GRENZE_S_o)
            {
                        signal_=2;
                        zaehler3++;
            }

            Puls_l_digital[zaehler2]=signal_;


                        // Synchronisation
            if(zaehler3>30 && Puls_l_digital[zaehler2]!=2)
            {
                        zaehler2=3;
                        zaehler3=0;
            }


                        // Zusammenfassen der gefundenen Bits zu einem Byte
                        // Abtastung der aktuellen Variable „signal_"
            switch(zaehler2)
            {
                        case(9):    {  eingang.e00 = (signal_==1) ? 1: 0;                break;}
                        case(28):   {  eingang.e01 = (signal_==1) ? 1: 0;                break;}
                        case(47):   {  eingang.e02 = (signal_==1) ? 1: 0;                break;}
                        case(66):   {  eingang.e03 = (signal_==1) ? 1: 0;                break;}
                        case(85):   {  eingang.e04 = (signal_==1) ? 1: 0;                break;}
                        case(104):  {  eingang.e05 = (signal_==1) ? 1: 0;                break;}
                        case(123):  {  eingang.e06 = (signal_==1) ? 1: 0;                break;}
                        case(142):  {  eingang.e07 = (signal_==1) ? 1: 0;
                                    memcpy((void*)&ausgang, (PGM_VOID_P)&eingang, 1) ;
                                                // struct"eingang.xx" wird in  "temp" kopiert
                                    break;
                                }
                        default:;
            }
}
```

Man erkennt an dem Diagramm die 200 Schwingungen, die für die Übertragung eines Byte mit Synchronisationsbits nötig sind. Blau ist der „reine" Messwert der Schwingungslänge, woraus man auf die Frequenz schließen kann. Rot ist der gefilterte Wert und unten, schwarz dargestellt sieht man die Bits, die mit Hilfe von verschiedenen vorbestimmten Grenzen aus der roten Kurve generiert wurden. Anschließend wurde die schwarze Kurve an vorher definierten Stellen abgetastet (siehe switch case…), und ein Byte erzeugt.
Der gesendete Wert 55, also binär 11101100 22 ist auch deutlich an der schwarzen Kurve zu erkennen: Mission accomplished!