In[119]:=

```
(*DCF77 Decoding Mathematica Program*)
(*1. Importiere das Audio-File und filtere das 750 Hz Signal*)
                        Audio  Datei
audio = Import[NotebookDirectory[] <> "DCF77_Test_File_WebSDR_heute.wav", "WAV"];
        importi···  Notebook-Verzeichnis                    Datei
samplingRate = 7119;
threshold = 0.23;
thresholdMinutes = 0.55;
filAudio =
  BandpassFilter[audio, {Quantity[700, "Hz"], Quantity[800, "Hz"]}, samplingRate];
  Bandpassfilter              Größe                  Größe

(*2. Extrahiere die Audiodaten und flache die Liste ab*)
filteredAudioData = Flatten[AudioData[filAudio]];
                        ebne ein  Audiodaten

(*3. AM-Demodulation durch Absolutwertbildung*)
envelope = Abs[filteredAudioData];
           Absolutwert

(*4. Glättung durch Tiefpassfilter*)
demodulatedSignal = LowpassFilter[envelope, 2 * 10 / samplingRate];
                    Tiefpassfilter

(*Normalisiere die demodulierte Hüllkurve*)
normalizedEnvelope = demodulatedSignal / Max[demodulatedSignal];
                                         größtes Element

(*5. Visualisierung der normalisierten Hüllkurve und Thresholds*)
startSeconds = 1; endSeconds = 90; lenSeconds = endSeconds - startSeconds;
ListLinePlot[{Table[0.6, {i, 1, 60 * samplingRate}],
listenbezogene Li··  Tabelle
  Table[threshold, {i, 1, lenSeconds * samplingRate}],
  Tabelle
  Table[thresholdMinutes, {i, 1, lenSeconds * samplingRate}],
  Tabelle
  normalizedEnvelope[[startSeconds * samplingRate ;;
     Min[endSeconds * samplingRate, Length[normalizedEnvelope]]]},
     kleinstes Element                  Länge
 PlotStyle → {Green, Red, Yellow, Blue}, PlotLegends →
 Darstellungsstil  grün  rot  gelb  blau  Legenden der Graphik
  {"1 Minute", "Threshold: " <> ToString[threshold],
                Schwellenwert      als Zeichenkette
   "ThresholdMinutes: " <> ToString[thresholdMinutes], "demod. Envelope"},
                           als Zeichenkette
 PlotRange → All, PlotLabel → "Normalized Envelope and Threshold",
 Koordinatenb···  alle  Beschriftung de··  normalisiert          Schwellenwert
 AxesLabel → {ToString[lenSeconds] <> " * Samples", "Amplitude"}]
 Achsenbeschrif··  als Zeichenkette

(*6. Bitextraktion*)
bitDuration = samplingRate;
```

```
extractedBits = Table[If[Min[Take[normalizedEnvelope,
                Tabelle […  kle··  entferne
        {i, Min[i + bitDuration - 1, Length[normalizedEnvelope]]}]] < threshold, 1, 0],
           kleinstes Element          Länge
    {i, 1, Length[normalizedEnvelope] - bitDuration + 1, bitDuration}];
          Länge
Print["Anzahl der extrahierten Bits: ", Length[extractedBits]];
gib aus                                  Länge

ListPlot[extractedBits[[1 ;; Min[lenSeconds, Length[extractedBits]]]], PlotRange → All,
listenbezogene Graphik          kleinstes Element   Länge                 Koordinatenb···  alle
 PlotLabel → "Extracted Bits", AxesLabel → {"Time/Sekunden", "Bits"}]
 Beschriftung der Graphik              Achsenbeschriftung

(*7. Minutenmarker-Erkennung mit Verschiebung*)
minuteMarkerPositions =
  Table[If[Min[Take[normalizedEnvelope, {i, Min[i + bitDuration - 1,
  Tabelle […  kle··  entferne                    kleinstes Element
         Length[normalizedEnvelope]]}]] > thresholdMinutes, 1, 0],
          Länge
    {i, 1, Length[normalizedEnvelope] - bitDuration + 1, bitDuration}];
          Länge
minuteMarkerPositions = Drop[Prepend[minuteMarkerPositions, 0], -1];
                        entf···  stelle voran

Print["Gefundene Minutenmarker-Positionen (angepasst): ",
gib aus
  Position[minuteMarkerPositions, 1]];
  Position

ListPlot[minuteMarkerPositions[[1 ;; Min[lenSeconds, Length[minuteMarkerPositions]]]],
listenbezogene Graphik          kleinstes Element   Länge
 PlotRange → All, PlotLabel → "Minutenmarker-Erkennung",
 Koordinatenb···  alle  Beschriftung der Graphik
 AxesLabel → {"Time/Sekunden", "Marker"}]
 Achsenbeschriftung

(*Funktion zur Paritätsprüfung*)
CheckParity[bits_, parityBitIndex_, bitRange_] :=
 Module[{bitSegment, calculatedParity}, bitSegment = bits[[bitRange]];
 Modul
  calculatedParity = Mod[Total[bitSegment], 2];
                     M···  Gesamtsumme
  Return[calculatedParity == bits[[parityBitIndex]]]
  gib zurück

(*Funktion zur Formatierung und Dekodierung mit Paritätsprüfung*)
FormatDecodeDCF77[bitSegment_] := Module[
                                  Modul
  {minuteOnesBits, minuteTensBits, hourOnesBits, hourTensBits, dayOnesBits, dayTensBits,
   monthOnesBits, monthTensBits, yearOnesBits, yearTensBits, minutes, hours, day,
   month, year, minuteParity, hourParity, dateParity, parityErrors, datumZeit},
  (*Dekodiere die einzelnen Bitsegmente*)minuteOnesBits = bitSegment[[21 ;; 24]];
```

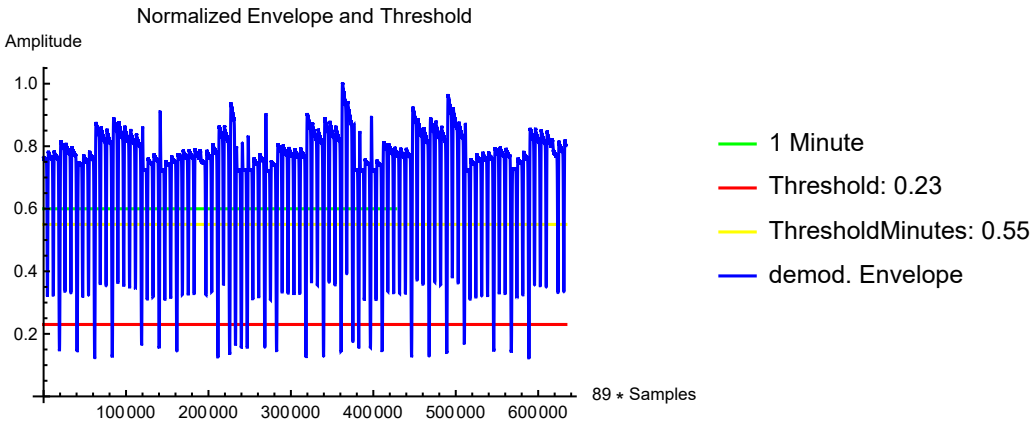
```
  minuteTensBits = bitSegment[[25 ;; 27]];
  hourOnesBits = bitSegment[[29 ;; 32]];
  hourTensBits = bitSegment[[33 ;; 34]];
  dayOnesBits = bitSegment[[36 ;; 39]];
  dayTensBits = bitSegment[[40 ;; 41]];
  monthOnesBits = bitSegment[[45 ;; 48]];
  monthTensBits = bitSegment[[49]];
  yearOnesBits = bitSegment[[50 ;; 53]];
  yearTensBits = bitSegment[[54 ;; 57]];
  (*Berechne Minuten,Stunden,Tag,Monat und Jahr*)
  minutes = FromDigits[Reverse[minuteOnesBits], 2] +
              Zahl aus Ziff··· kehre um
    10 * FromDigits[Reverse[minuteTensBits], 2];
         Zahl aus Ziff··· kehre um
  hours =
   FromDigits[Reverse[hourOnesBits], 2] + 10 * FromDigits[Reverse[hourTensBits], 2];
  Zahl aus Ziff··· kehre um                  Zahl aus Ziff··· kehre um
  day = FromDigits[Reverse[dayOnesBits], 2] + 10 * FromDigits[Reverse[dayTensBits], 2];
        Zahl aus Ziff··· kehre um                  Zahl aus Ziff··· kehre um
  month = FromDigits[Reverse[monthOnesBits], 2] + 10 * monthTensBits;
          Zahl aus Ziff··· kehre um
  year = FromDigits[Reverse[yearOnesBits], 2] +
         Zahl aus Ziff··· kehre um
    10 * FromDigits[Reverse[yearTensBits], 2] + 2000;
         Zahl aus Ziff··· kehre um
  (*Paritätsprüfung*)minuteParity = CheckParity[bitSegment, 28, 21 ;; 27];
  hourParity = CheckParity[bitSegment, 35, 29 ;; 34];
  dateParity = CheckParity[bitSegment, 58, 36 ;; 57];
  (*Überprüfung auf Fehler*)parityErrors = {};
  If[! minuteParity, AppendTo[parityErrors, "Minuten"]];
  wenn                hänge an bei
  If[! hourParity, AppendTo[parityErrors, "Stunden"]];
  wenn               hänge an bei
  If[! dateParity, AppendTo[parityErrors, "Datum"]];
  wenn               hänge an bei
  (*Ausgabe bei korrekten Daten oder bei Fehlern*)
  datumZeit = IntegerString[day, 10, 2] <> "." <>
                Zeichenkette der ganzen Zahl
    IntegerString[month, 10, 2] <> "." <> ToString[year] <> " - " <>
    Zeichenkette der ganzen Zahl          als Zeichenkette
    IntegerString[hours, 10, 2] <> ":" <> IntegerString[minutes, 10, 2];
    Zeichenkette der ganzen Zahl          Zeichenkette der ganzen Zahl
  If[
  wenn
   Length[parityErrors] > 0, <|"DatumUndZeit" → datumZeit <> " (Paritätsfehler bei: " <>
   Länge
      StringJoin[Riffle[parityErrors, ", "]] <> ")"|>, <|"DatumUndZeit" → datumZeit|>]
      vereinige Ze··· füge wiederholt ein

(*Synchronisierungs-und Dekodierungsfunktion mit schöner Ausgabe*)
FormattedSynchronizeAndDecodeDCF77[extractedBits_, minuteMarkerPositions_] :=
 Module[{decodedData, bitSegments, segmentStart},
 Modul
```

```
(*Synchronisiere Bit-Segmente mit Minutenmarkern*)
Print["Minutenmarker-Positionen: ", Position[minuteMarkerPositions, 1]];
bitSegments = Table[segmentStart = pos + 1;
  If[segmentStart + 58 ≤ Length[extractedBits],
   extractedBits[[segmentStart ;; segmentStart + 58]], {}],
  {pos, Flatten[Position[minuteMarkerPositions, 1]]}];
(*Dekodiere jedes Bit-Segment und formatiere das Ergebnis*)
decodedData = Map[If[Length[#] == 59, FormatDecodeDCF77[#], <|
     "Error" → "Unvollständiges Bit-Segment"|>] &, bitSegments];
(*Ausgabe der dekodierten Daten*)
Do[If[KeyExistsQ[decodedData[[i]], "DatumUndZeit"], Print["Segment ", i,
   ": ", decodedData[[i, "DatumUndZeit"]]], {i, Length[decodedData]}];]
```

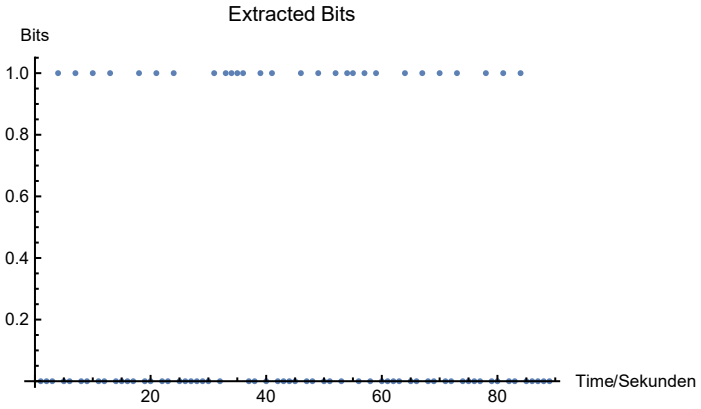
```
(*Beispielaufruf*)
FormattedSynchronizeAndDecodeDCF77[extractedBits, minuteMarkerPositions];
```

Out[129]=



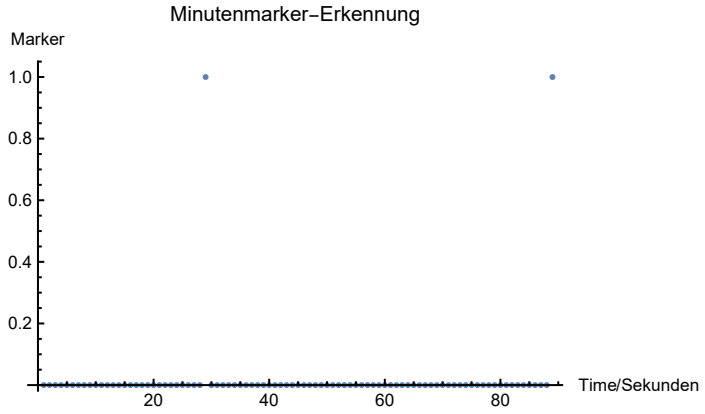

```
Anzahl der extrahierten Bits: 306
```

Out[133]=




```
Gefundene Minutenmarker-Positionen (angepasst): {{29}, {89}, {149}, {209}, {269}}
```

Out[137]=




```
Minutenmarker-Positionen: {{29}, {89}, {149}, {209}, {269}}

Segment 1: 24.10.2024 - 02:34

Segment 2: 24.10.2024 - 02:35

Segment 3: 24.10.2024 - 02:36

Segment 4: 24.10.2024 - 02:37
```