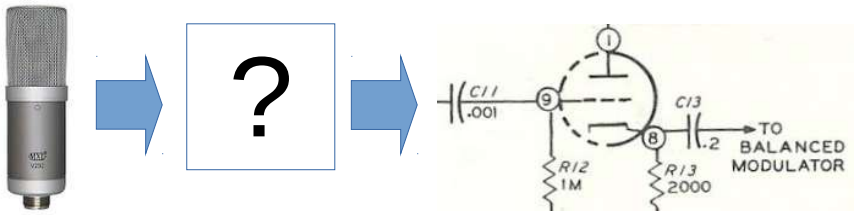


Andreas Bork
DM 4 AB

TX Audio Processing

Kompressoren
Equalizer
Filter

→ HW simulieren

Moderne Filter

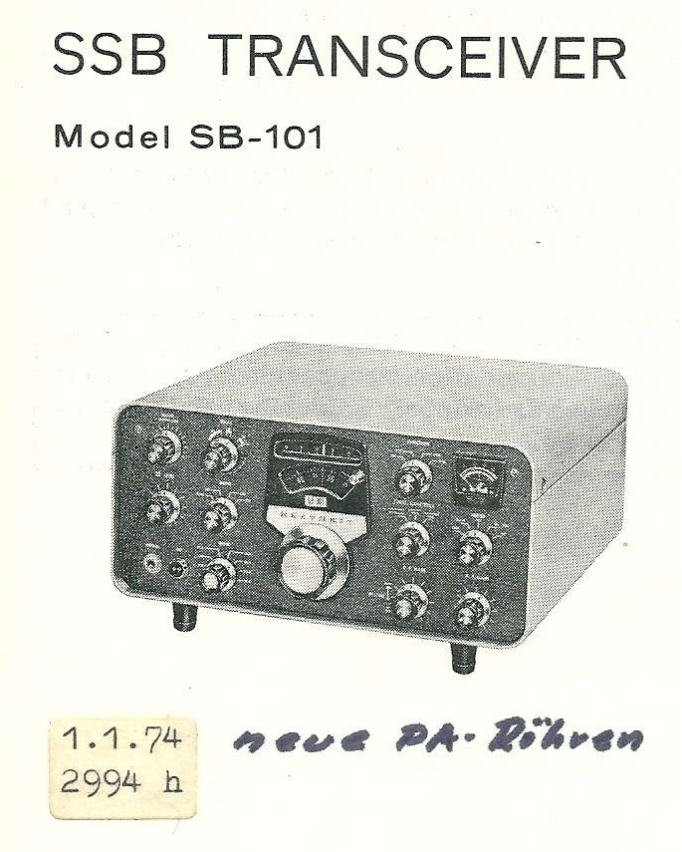
- in SW
- mit dem DSP

→ programmieren,
zusammenklicken, realisieren

Auslöser

Andreas Bork
DM 4 AB

- meine erste Amateurfunkliebe
- aus Schulzeiten
- die älter ist als ich selbst



- aber leider ging nach einigen Umzügen das Mikrofon verloren (Kristallmikrofon, high-Z)

Audio Eingangs-Stufe

Andreas Bork
DM 4 AB

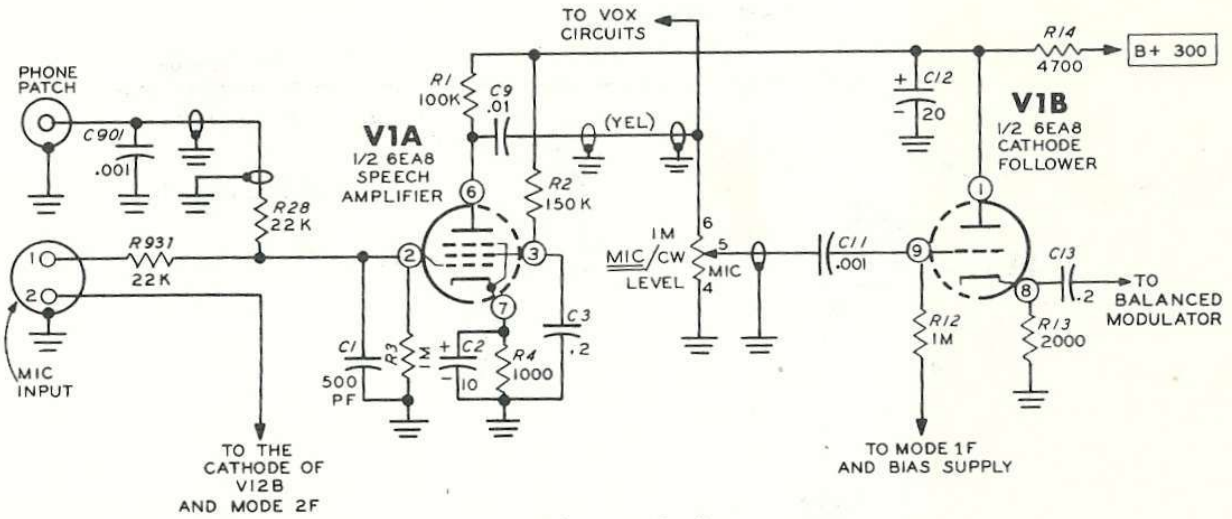


Figure 2-6

Audio Eingangs-Stufe

Probleme:

- damit wird man (heute) fast nicht mehr gehört („*You are a QRP station, right?*“)
- man muss sehr diszipliniert **s p r e c h e n**
- man braucht viel Sprechwasser
- und bekommt Ärger mit der XYL wegen Lärmbelästigung („*Sag mal, warum schreist Du hier so rum, die Kinder bekommen kein Auge zu…!*“)

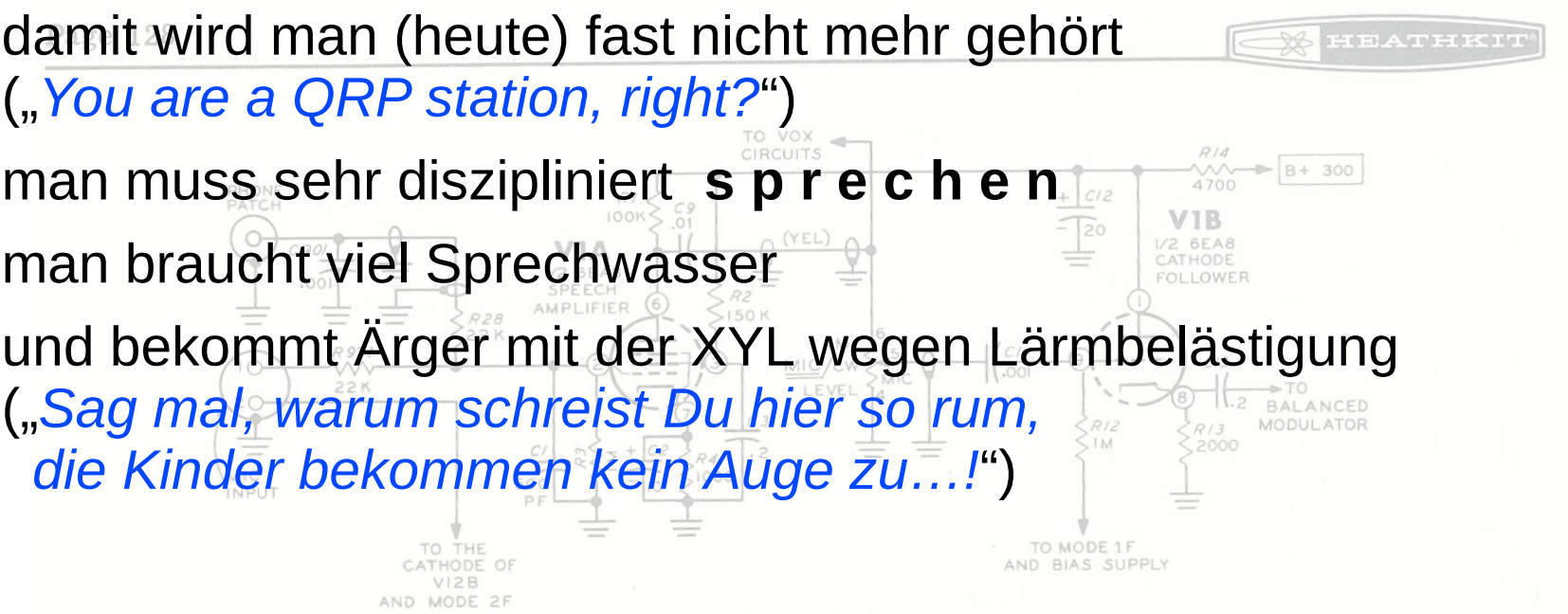
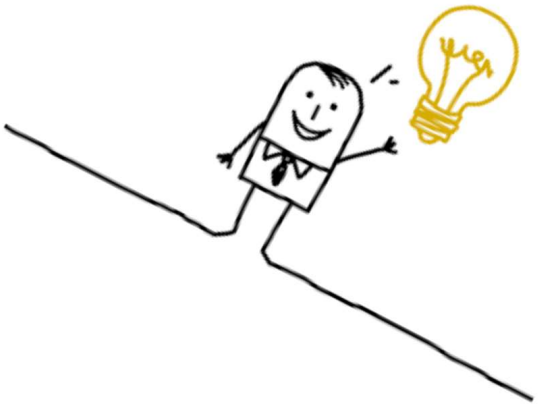


Figure 2-6

Hmmm… nicht so optimal.

1. Kompressor

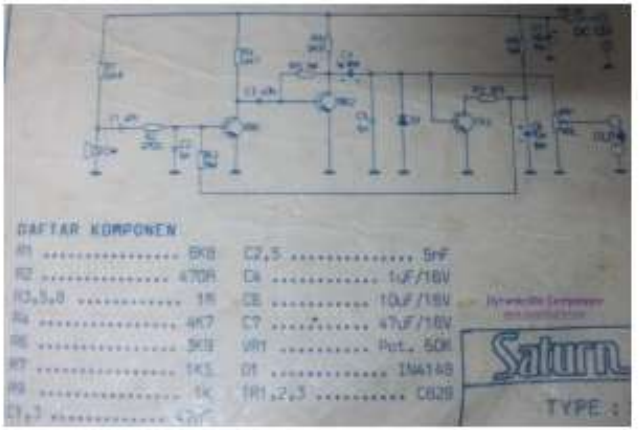


erste Idee: es muss ein **Kompressor** her !

- selbst ist der Mann, Google das Schweizer Messer
- und dann finden sich viele Schaltungen mit zweifelhaftem Aufbau oder mir unklarer Funktion

1. Kompressor 1

The circuit below is a **Dynamic Mic Compressor** circuit which is simple, easy to built, inexpensive but the results are quite satisfactory. Audio output sounded outstanding.

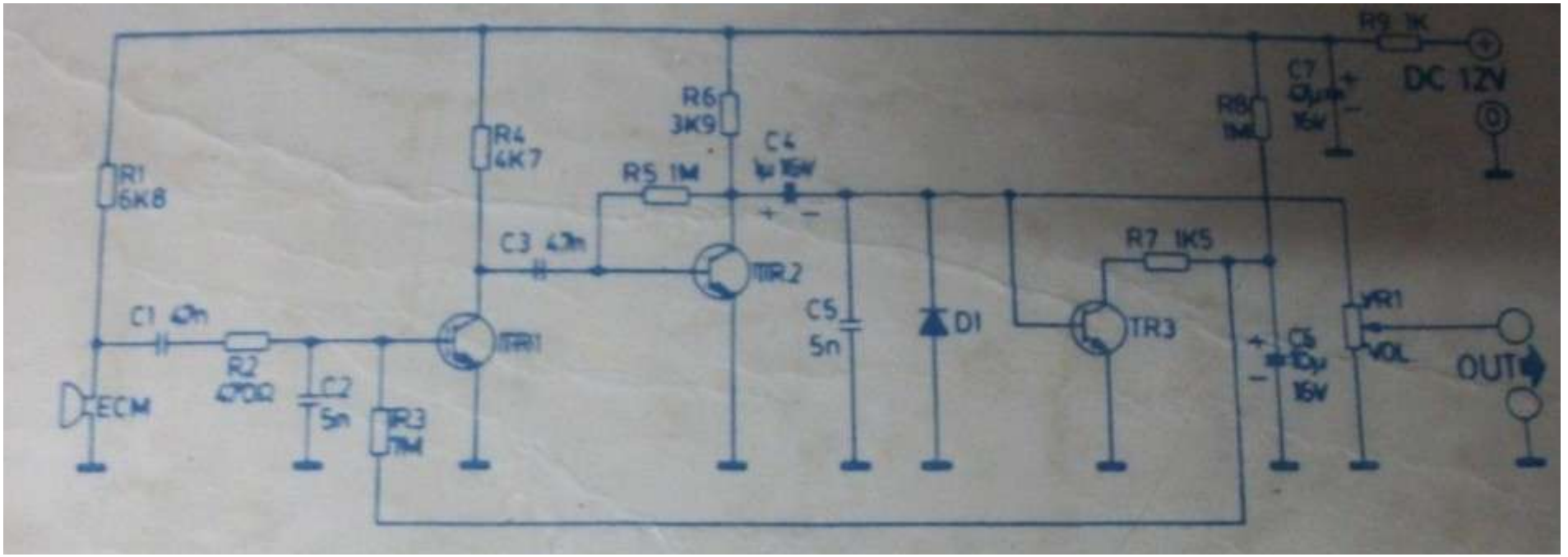


The principle of this circuit is very simple, the first transistor is used as microphone pre-amplifier. Then a second transistor used as a buffer. And the third transistor is a feedback circuit. A very sensitive electret microphone used in this circuit.

Supply voltage of this **Dynamic Mic Compressor** circuit must be 12 Volt. Highly recommended to use its own power supply, do not use a shared power supply from another electronic circuit module. Regulated circuit with fine/good cable and good grounding is recommended to minimize hum.

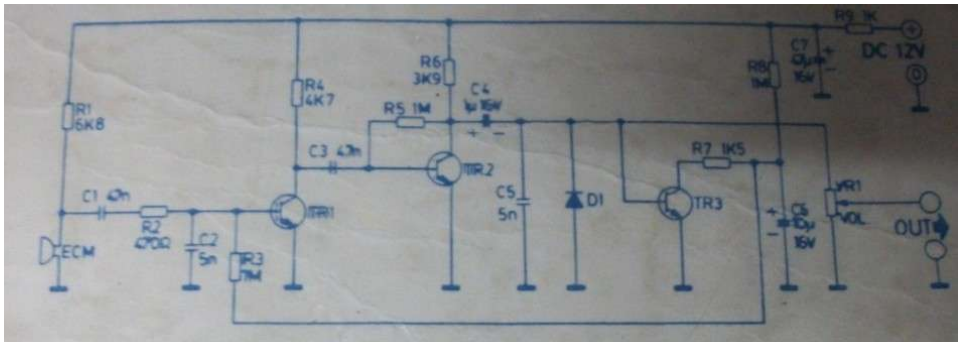
Quelle: http://circuitdiagram.net/dynamic-mic-compressor.html

1. Kompressor 1



Quelle: http://circuitdiagram.net/dynamic-mic-compressor.html

1. Kompressor 1



So, also aufbauen.

Oder… simulieren!

Wie ging das nochmal?

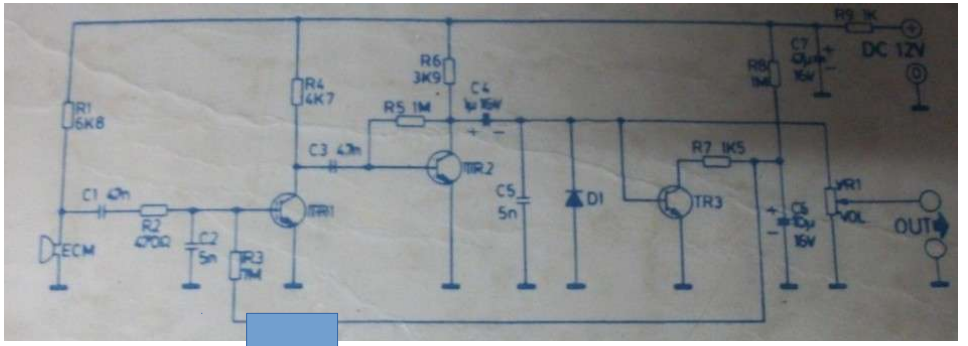
SPICE:

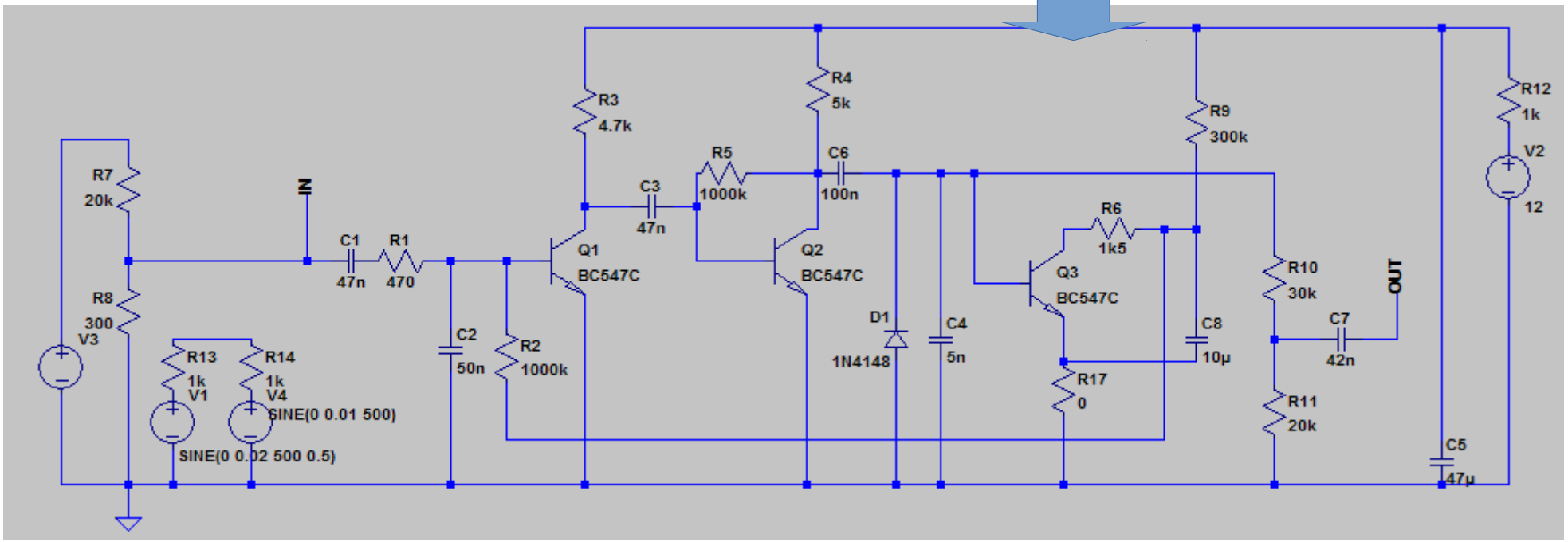
„*SPICE (**Simulation Program with Integrated Circuit Emphasis**) ist eine Software zur Simulation analoger, digitaler und gemischter elektrischer Schaltungen (Schaltungssimulation). […]*
SPICE1 wurde 1973 erstmals der Öffentlichkeit vorgestellt. 1975 erschien eine stark verbesserte zweite Version. 1989 wurde Version 3 von Thomas Quarles, einem Studenten von A. Richard Newton (der an Spice 2 mitgearbeitet hatte), herausgebracht, die in C geschrieben war.“

Quelle: https://de.wikipedia.org/wiki/SPICE_(Software)

1. Kompressor 1 simulieren

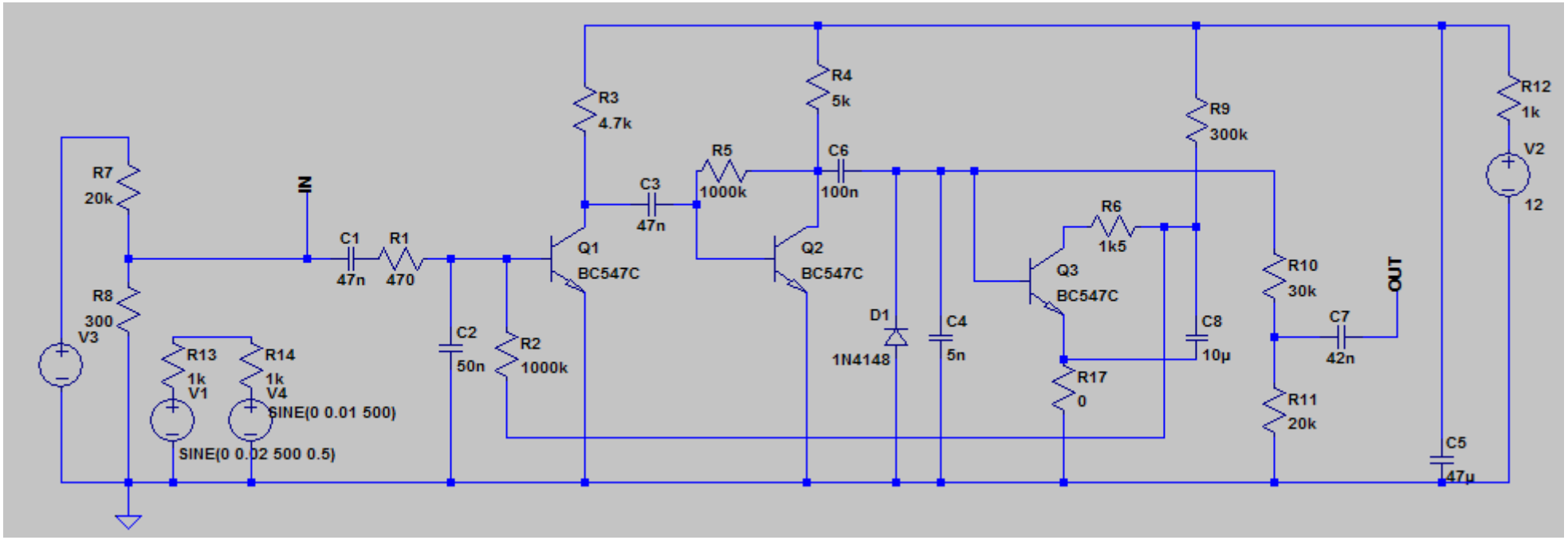
in meinem Fall mit „**ltspice**“, durch Linear Technology angepasste und gepflegte Spice Implementierung





1. Kompressor 1 durchspielen

1. Sinus am Eingang, Amplitudensprung
2. Frequenzgang
3. **Sprache**
4. Sprache mit **Bauteile-Variation**

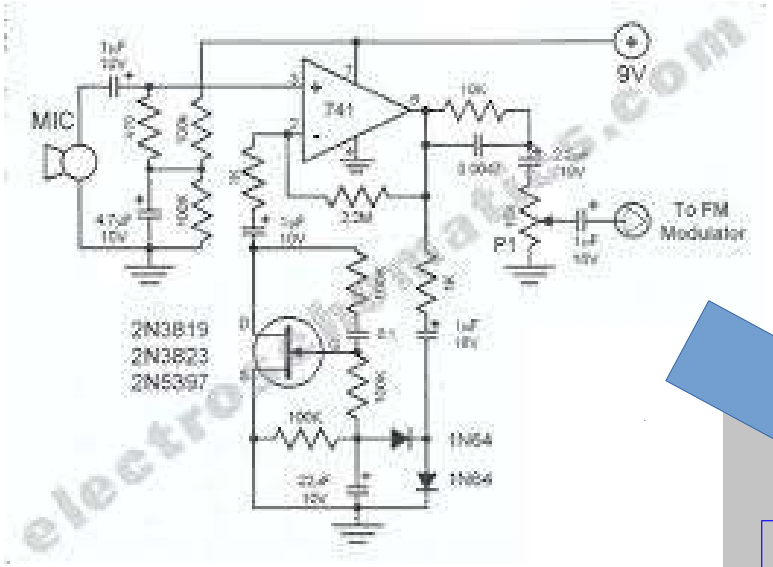


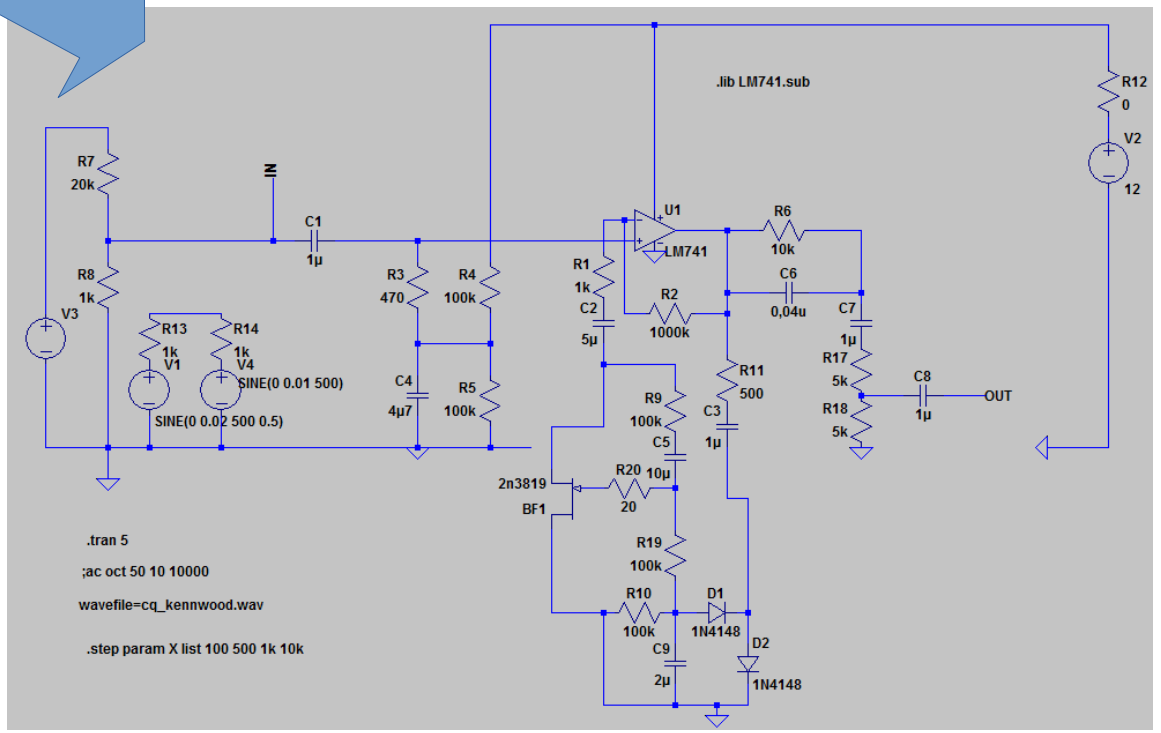
1. Kompressor 1 auswerten

Vergleichende Bewertung von Audio-Samples mit „**Audacity**“

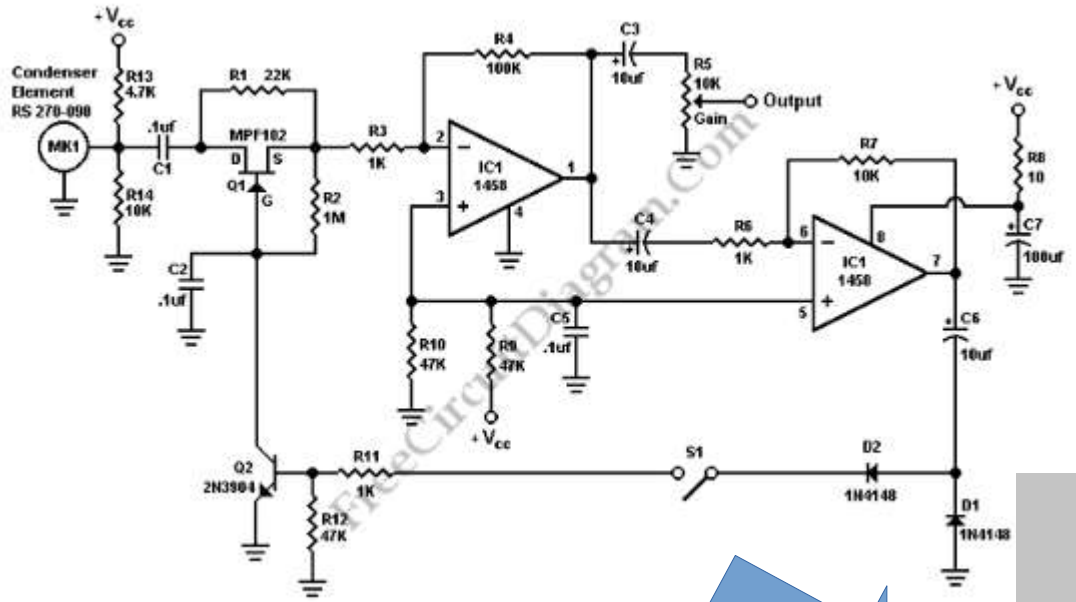
(kam auch mal in „CQ-DL“ oder dem „Funkamateur“ zur Sprache)

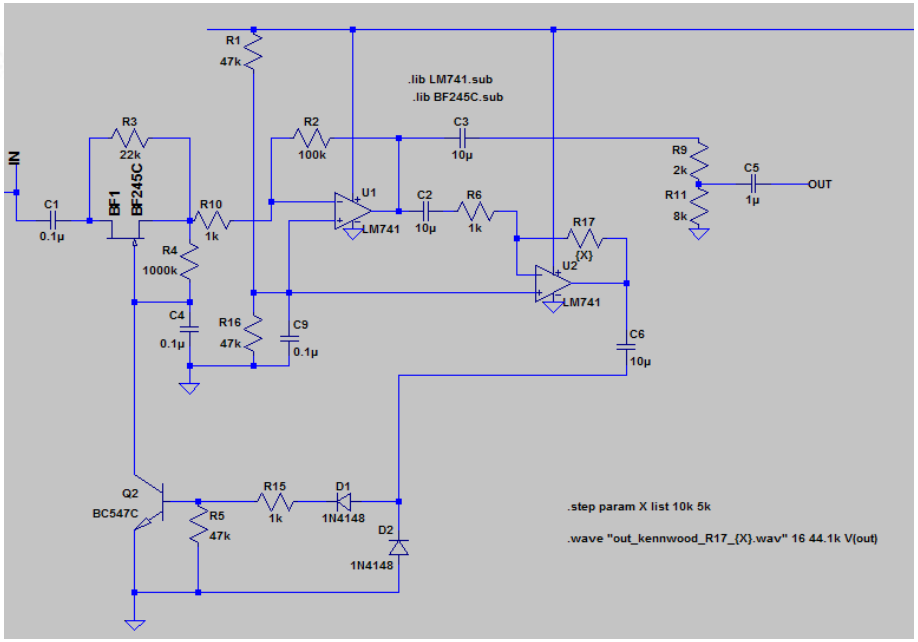
1. Kompressor 2



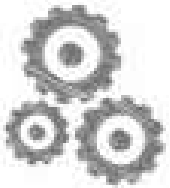


1. Kompressor 3





Zwischenergebnis



1. Kompressor ✓
2. Equalizer ?

(meine Stimme ist ja recht tief…)

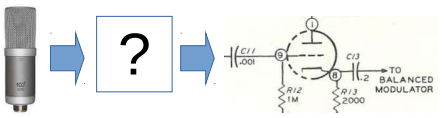
Ok, einen Equalizer zu bauen kann ja kein Hexenwerk sein. Kann man sicher vor dem Bauen

auch simulieren …?

2. Equalizer

- macht man für Audio „aktiv“ mit Operationsverstärker und entsprechender Filterschaltung,
- selektiv für Einzelfrequenzen als Bandpass,
- oder je ein Tief- und Hochpass,
- und am Ende alles wieder summieren.

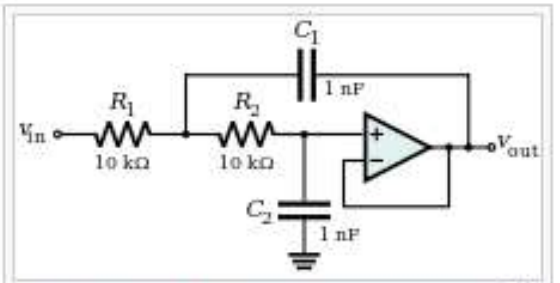
Alles klar, also los geht‘s!



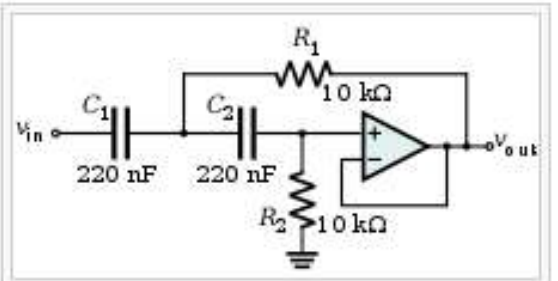
2. Equalizer

Wie realisiert man einen aktiven Tief-, Hoch bzw. Bandpass?

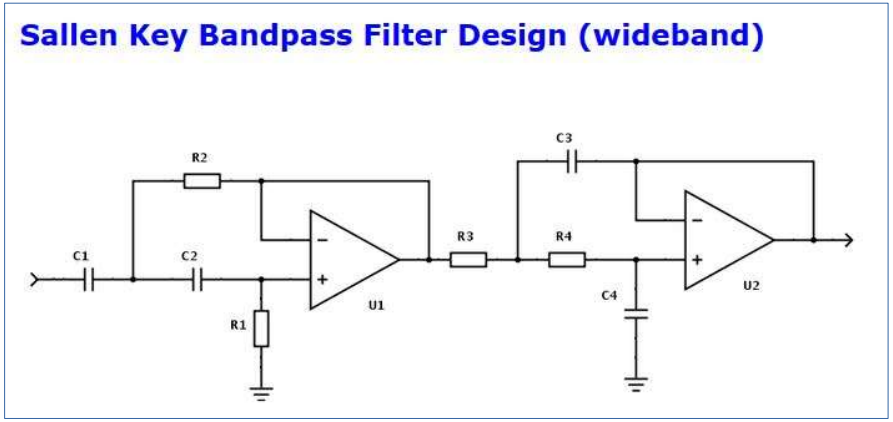
- idealerweise mit OP (=einfach)
- in Sallen-Key-Struktur
  (=gleiche Bauteilwerte)
- basierend auf den Einzelstrukturen auch als Bandpass



Sallen-Key-Filterstruktur eines Tiefpassfilters, Mitkopplungsstruktur



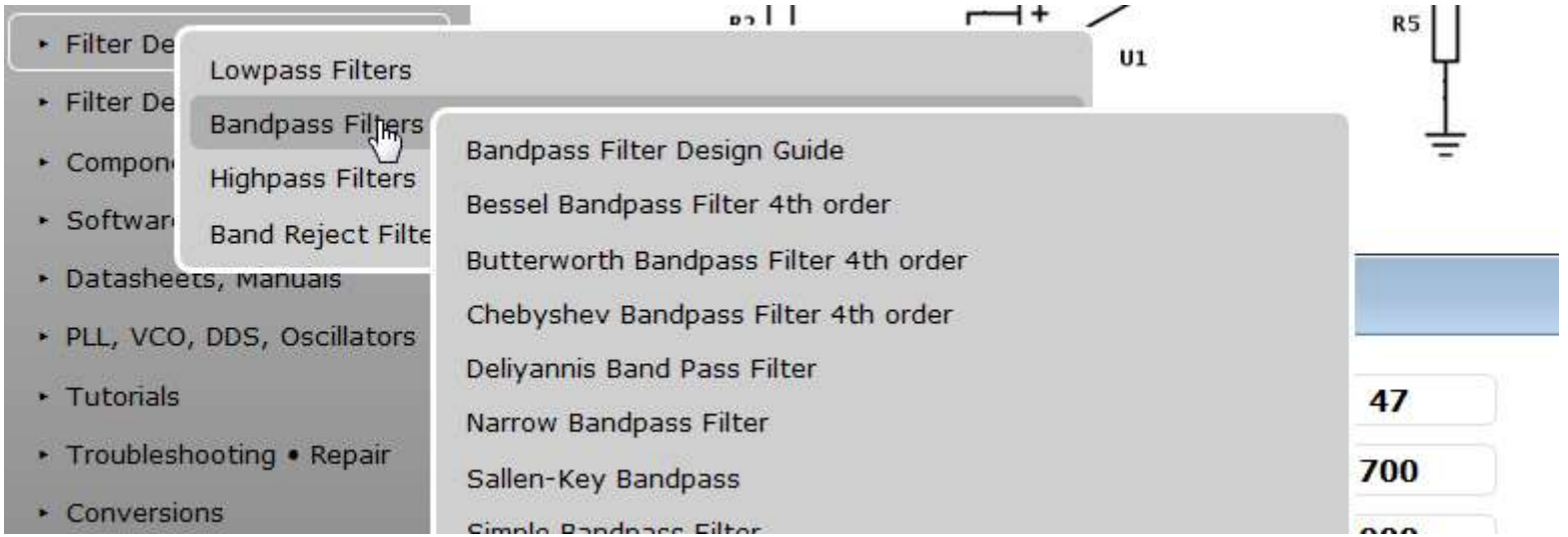
Sallen-Key-Filterstruktur eines Hochpassfilters, Mitkopplungsstruktur



Quelle: https://de.wikipedia.org/wiki/Sallen-Key-Filter

2. Equalizer

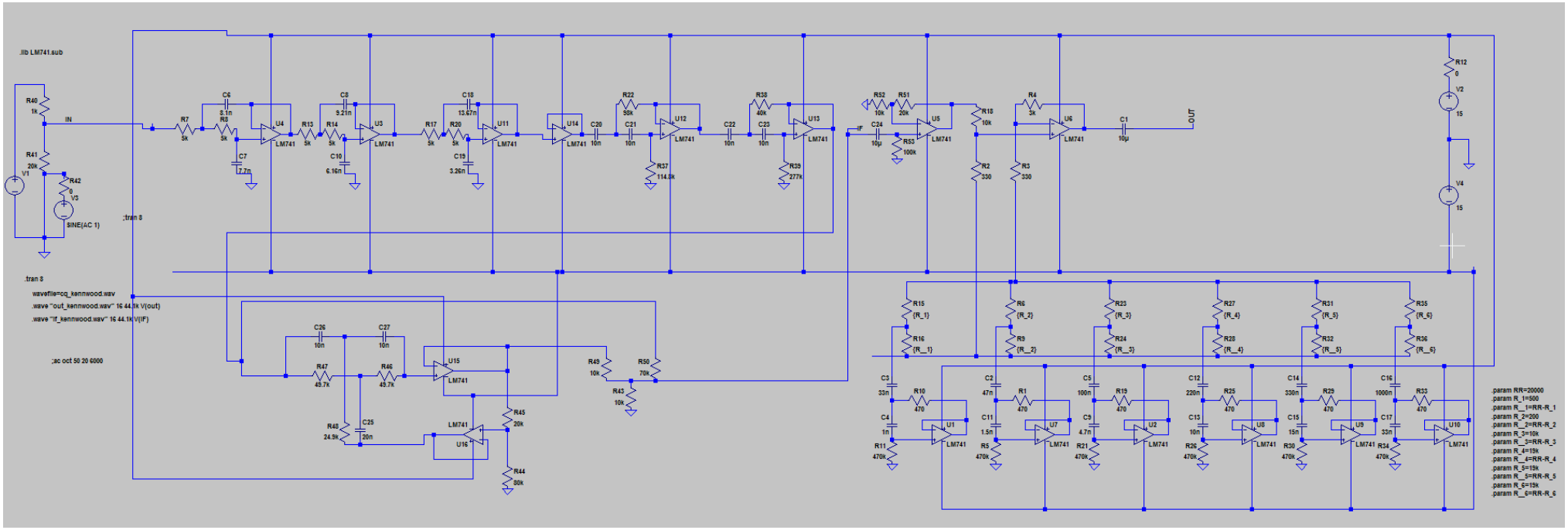
Tolle Web-Site (unter anderem) zur Berechnung von aktiven Filtern:
http://www.changpuak.ch



2. Equalizer

Aber warum mit einzelnen Bandpässen herum machen,
es gibt ja auch Vorschläge für diskrete Equalizer Schaltungen!

Bandpass



Mehrband-Equalizer

Rechenzeit für 10 Sekunden Analyse: etwa 20 Minuten @ 3GHz CPU

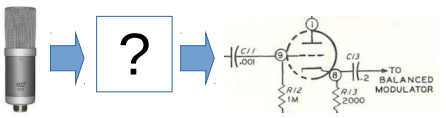
2. Equalizer

mehrere Erkenntnisse:

1. für einen guten Equalizer braucht es viele Frequenzbänder
2. schmale, selektive Frequenzbänder sind **nicht** mit einem Filter 1. Ordnung realisierbar (daraus nur 20dB/Dekade oder 12dB/Oktave)
3. höhere Ordnungen werden dann erforderlich (**n** * 20dB/Dekade)

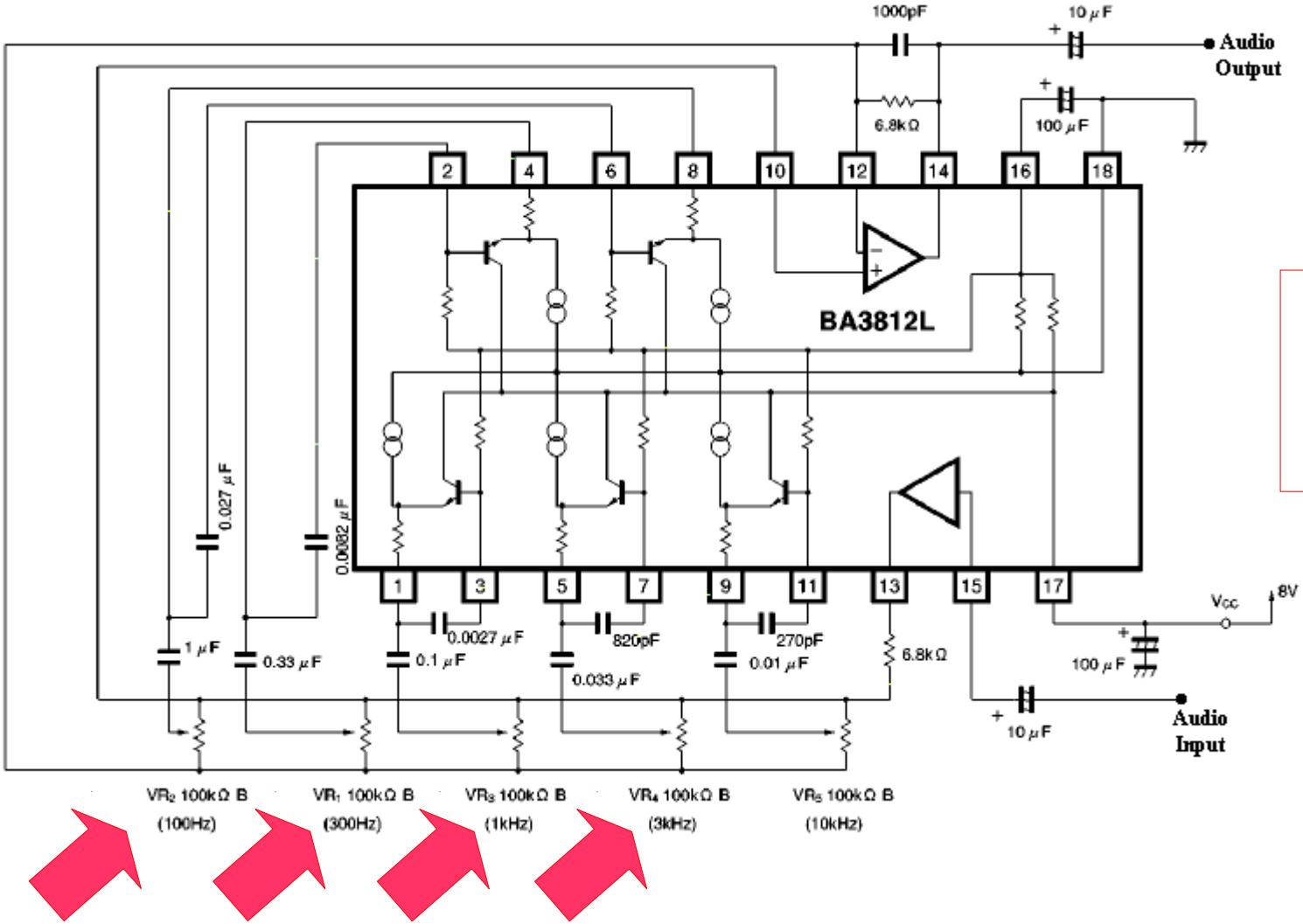
4. Aha! Da gibt es doch fertige IC‘s, oder nicht?

Rechenzeit für 10 Sekunden Analyse: etwa 10 Minuten @ 3GHz CPU



2. Equalizer

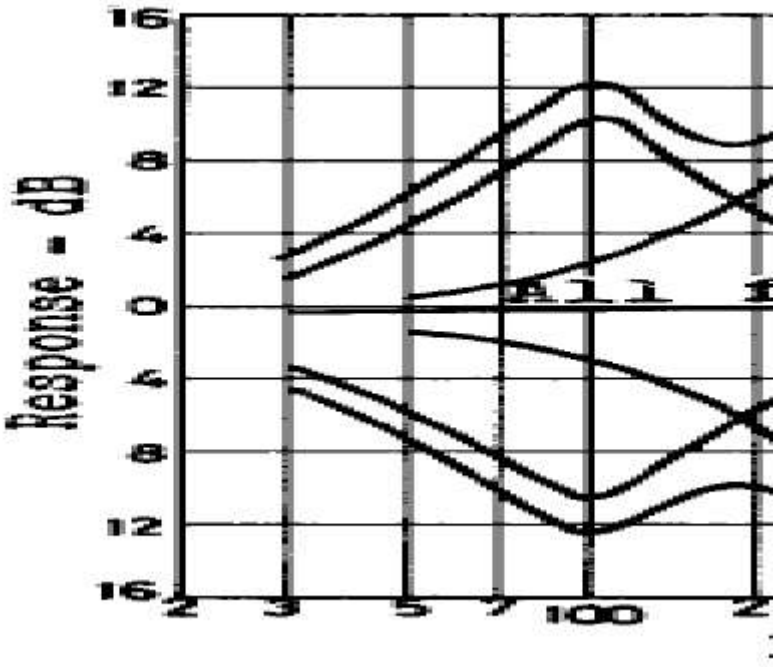
realisiert als IC-Schaltkreis BA3812L ?



nicht ausreichend selektiv

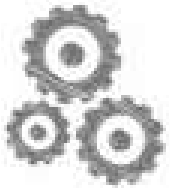
2. Equalizer

realisiert als IC-Schaltkreis LA3600?



nicht ausreichend selektiv

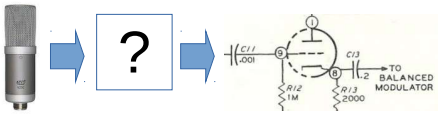
Zwischenergebnis



1. Kompressor ✓
2. Equalizer in HW –

Equalizer in HW, Erkenntnisse:

- funktionieren gut für breite Musikspektren
- funktionieren schlecht für präzises Sprach-Tuning
- Schaltkreise teilweise nur noch bei eBay verfügbar (Restbestände)
- und… wie macht das der Equalizer in meinem Winamp?



Zwischenergebnis

1. Kompressor ✓
2. Equalizer in HW –
3. Equalizer in SW ?

3. Equalizer in SW

- zu „**Digital Audio Processing**“ gibt es Kilometer von Papier-Büchern, Web-Sites, Hochschul-Praktika, Online-Büchern
- … und auch Tools und teilweise komplexen „Audio-Workbenches“, (z.B Stereo-Tool, nächste Seite)
- brauchbare „visuelle“ Bastelumgebungen für Audio Algorithmen scheint es nicht zu geben (ich habe zumindest nichts brauchbares gefunden)
- **Latenzprobleme** sind ein generelles Problem, wenn man direkt den Transceiver anschließen möchte; alle Algorithmen arbeiten mit Sample Puffern, und Win7 ist kein Echtzeit-Betriebssystem...

3. Equalizer in SW

(Quelle: http://www.stereotool.com/)

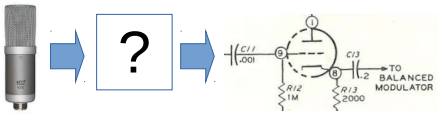
3. Equalizer in SW

(Quelle: http://www.stereotool.com/)

3. Equalizer in SW

- zu „**Digital Audio Processing**“ gibt es Kilometer von Papier-Büchern, Web-Sites, Hochschul-Praktika, Online-Büchern
- … und auch Tools und teilweise komplexen „Audio-Workbenches“, (z.B Stereo-Tool, nächste Seite)
- brauchbare „visuelle“ Bastelumgebungen für Audio Algorithmen scheint es nicht zu geben (ich habe zumindest nichts brauchbares gefunden)
- **Latenzprobleme** sind ein generelles Problem, wenn man direkt den Transceiver anschließen möchte; alle Algorithmen arbeiten mit Sample Puffern, und Win7 ist kein Echtzeit-Betriebssystem…

- aber es braucht **viel Strom** und der PC erzeugt viel **Warmluft!**



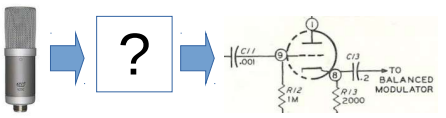
3. Equalizer in SW

Wie macht man das dann besser?

➔ in Hardware (=embedded) und mit effizienten Algorithmen

Grundlagen:

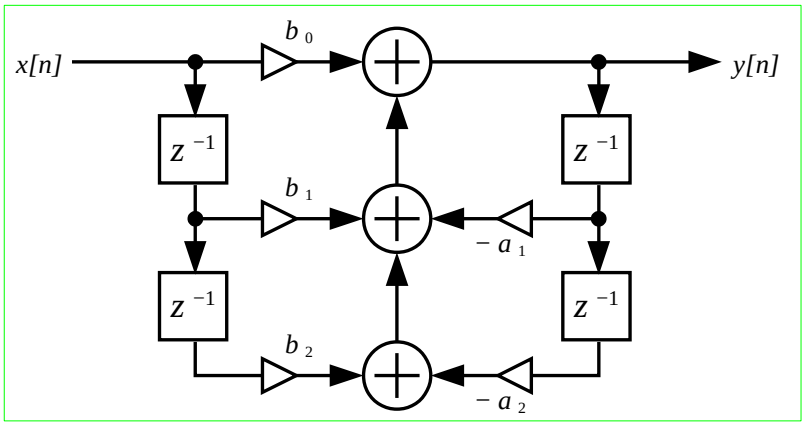
- Abtastung und Synthese bekannt ✓
- Filter-Algorithmen **?**

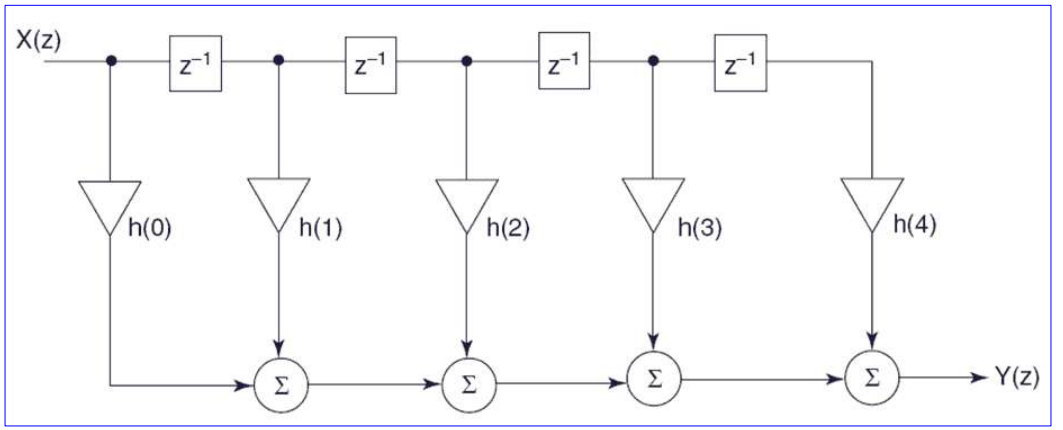


3. Equalizer in SW

Filter Algorithmen:

- wichtigstes Argument: einfach und schnell zu berechnen
- Herleitung mit viel Theorie und Mathematik
- Ergebnis: FIR und IIR Filter Algorithmen
  FIR: finite impulse response
  IIR: infinite impulse response



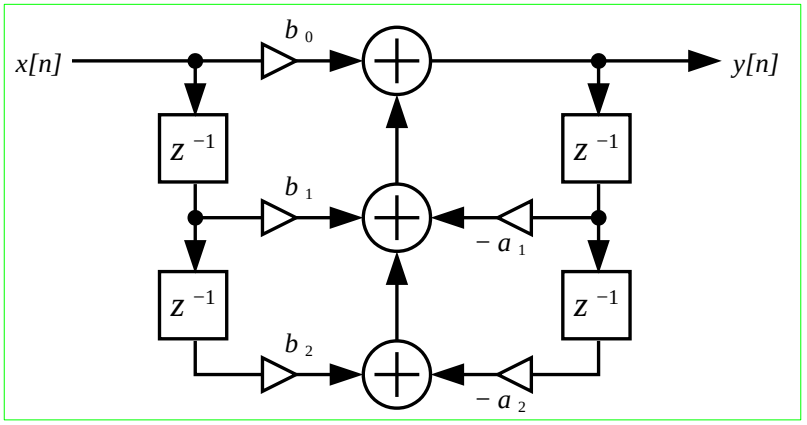


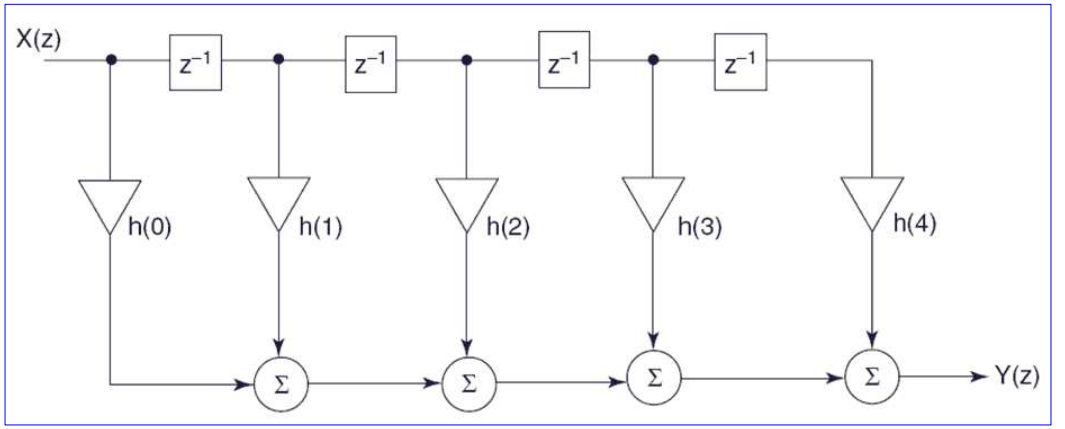
3. Equalizer in SW

Filter Algorithmen:

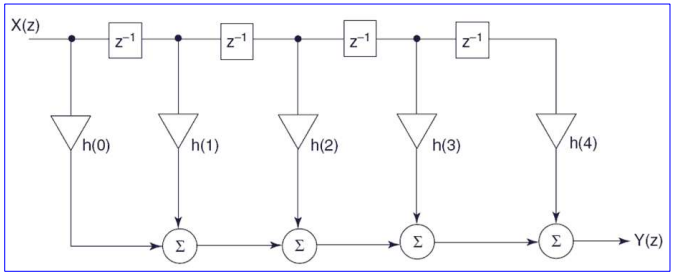
- viele Additionen
- viele Multiplikationen
- mit möglichst viel Bits
- mit möglichst wenig Rundungsfehlern

Rauschen !!





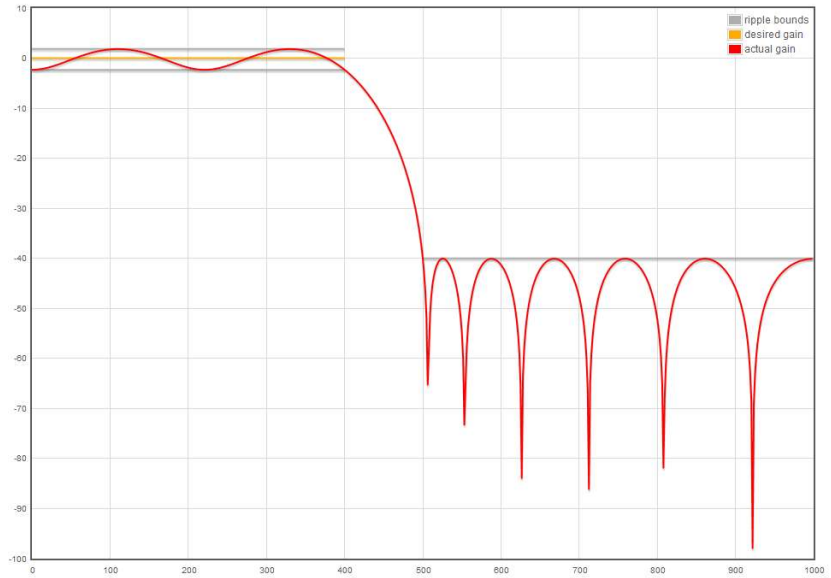
3. Equalizer in SW



Filter Algorithmen:

- Beispiel zur FIR Berechnung Tiefpassfilter, web-site: http://t-filter.engineerjs.com/

from	to	gain	ripple/att.	act. rpl	
0 Hz	400 Hz	1	5 dB	4.14 dB	
500 Hz	1000 Hz	0	-40 dB	-40.07 dB	



plain text | double

```
-0.02010411882885732
-0.0584279800435250 9
-0.061178403647821976
-0.010939393385338943
0.05125096443534972
0.033220867678947885
-0.05655276971833928
-0.08565500737264514
0.0633795996605449
0.31085440365663597
0.4344309124179415
0.31085440365663597
0.0633795996605449
-0.08565500737264514
-0.05655276971833928
0.033220867678947885
0.05125096443534972
-0.010939393385338943
-0.061178403647821976
-0.0584279800435250 9
-0.02010411882885732
```

plain text | int

fixed point precision (bits): 16

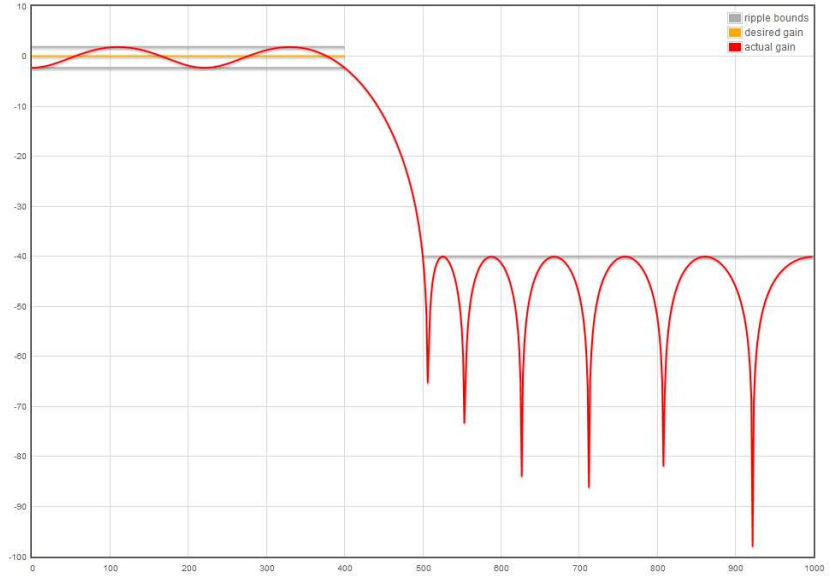
```
-659
-1915
-2005
-358
1679
1089
-1853
-2807
2077
10186
14235
10186
2077
-2807
-1853
1089
1679
-358
-2005
-1915
-659
```

3. Equalizer in SW

Filter Algorithmen:

- Beispiel zur FIR Berechnung Tiefpass
  web-site: http://t-filter.engineerjs.com/

from	to	gain	ripple/att.	act. rpl
0 Hz	400 Hz	1	5 dB	4.14 dB
500 Hz	1000 Hz	0	-40 dB	-40.07 dB

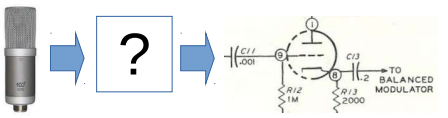



```
#include "SampleFilter.h"

static double filter_taps[SAMPLEFILTER_TAP_NUM] = {
  -0.02010411882885732,
  -0.05842798004352509,
  -0.061178403647821976,
  -0.010939393385338943,
  0.05125096443534972,
  0.033220867678947885,
  -0.05655276971833928,
  -0.08565500737264514,
  0.0633795996605449,
  0.31085440365663597,
  0.4344309124179415,
  0.31085440365663597,
  0.0633795996605449,
  -0.08565500737264514,
  -0.05655276971833928,
  0.033220867678947885,
  0.05125096443534972,
  -0.010939393385338943,
  -0.061178403647821976,
  -0.05842798004352509,
  -0.02010411882885732
};

void SampleFilter_init(SampleFilter* f) {
  int i;
  for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
    f->history[i] = 0;
  f->last_index = 0;
}

void SampleFilter_put(SampleFilter* f, double input) {
  f->history[f->last_index++] = input;
  if(f->last_index == SAMPLEFILTER_TAP_NUM)
    f->last_index = 0;
}
```

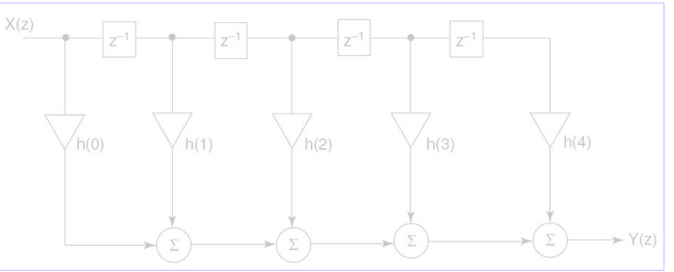
3. Equalizer in SW

```
#include "SampleFilter.h"

static int filter_taps[SAMPLEFILTER_TAP_NUM] = {
  -659,
  -1915,
  -2005,
  -358,
  1679,
  1089,
  -1853,
  -2807,
  2077,
  10186,
  14235,
  10186,
  2077,
  -2807,
  -1853,
  1089,
  1679,
  -358,
  -2005,
  -1915,
  -659
};

void SampleFilter_init(SampleFilter* f) {
  int i;
  for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
    f->history[i] = 0;
  f->last_index = 0;
}

void SampleFilter_put(SampleFilter* f, int input) {
  f->history[f->last_index++] = input;
  if(f->last_index == SAMPLEFILTER_TAP_NUM)
    f->last_index = 0;
}
```

```
#include "SampleFilter.h"

static double filter_taps[SAMPLEFILTER_TAP_NUM] = {
  -0.02010411882885732,
  -0.05842798004352509,
  -0.061178403647821976,
  -0.010939393385338943,
  0.05125096443534972,
  0.033220867678947885,
  -0.05655276971833928,
  -0.08565500737264514,
  0.0633795996605449,
  0.31085440365663597,
  0.4344309124179415,
  0.31085440365663597,
  0.0633795996605449,
  -0.08565500737264514,
  -0.05655276971833928,
  0.033220867678947885,
  0.05125096443534972,
  -0.010939393385338943,
  -0.061178403647821976,
  -0.05842798004352509,
  -0.02010411882885732
};

void SampleFilter_init(SampleFilter* f) {
  int i;
  for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
    f->history[i] = 0;
  f->last_index = 0;
}

void SampleFilter_put(SampleFilter* f, double input) {
  f->history[f->last_index++] = input;
  if(f->last_index == SAMPLEFILTER_TAP_NUM)
    f->last_index = 0;
}
```

3. Equalizer in SW



Filter Algorithmen:

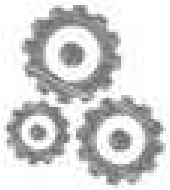
- Beispiel zur FIR Berechnung Tiefpassfilter, web-site: http://t-filter.engineerjs.com/

from	to	gain	ripple/att.	act. rpl
0 Hz	400 Hz	1	5 dB	4.14 dB
500 Hz	1000 Hz	0	-40 dB	-40.07 dB

Erkenntnisse:

- programmiert (Linux)
- Praxis folgt Theorie
- auch komplexe und steile Filter sehr gut realisierbar
- aber nicht einfach zum schnellen Rumprobieren
  (immer wieder Berechnung neuer Parameter erforderlich)

Zwischenergebnis



1. Kompressor ✓
2. Equalizer in HW –
3. Equalizer in SW ✓

Ist aber leider kompliziert und zeitraubend beim Finden der Parameter und im Optimieren…

Und eine „kleine“ Hardware fehlt dann auch noch.

Zwischenergebnis

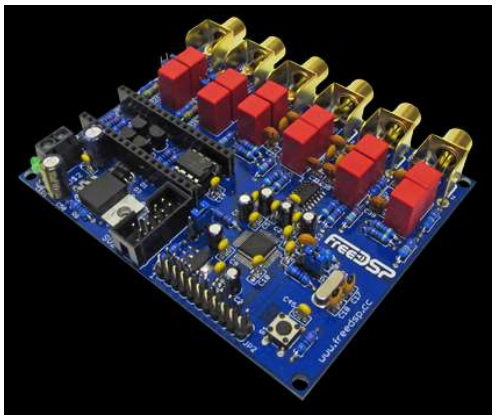
1. Kompressor ✓
2. Equalizer in HW –
3. Equalizer in SW ✓
4. Alles in einem DSP ?

Wieso nicht gleich alles zusammen in einem DSP realisieren? Weil DSP programmieren etwas für die **richtigen Cracks** ist!

Vielleicht aber doch auch für Otto-Normal-Funker?

4. Alles in einem DSP

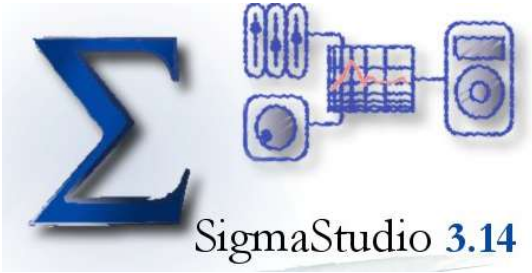
- da gibt es einige „freie“ DSP Projekte und Platinen
- z.B. http://www.freedsp.cc
- und es gibt viele DSP Hersteller
- aber das für Bastler beste Angebot scheint Analog Devices zu haben mit jede Menge verschiedener DSP‘s und
- **TOLLE SOFTWARE**



4. Alles in einem DSP

Analog Devices Software

- Sigma Studio



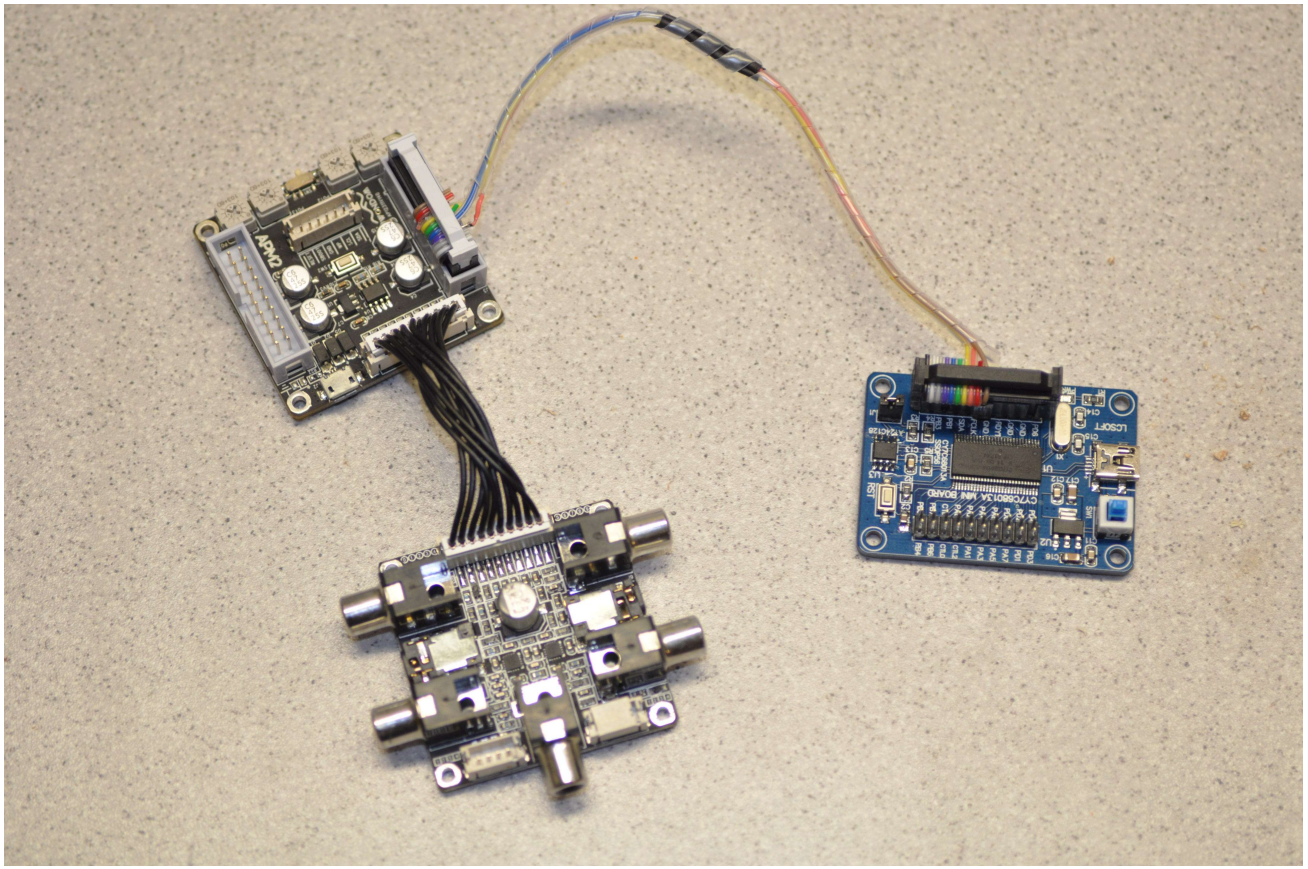
Meine Auswahl:

- Sure Electronics ADAU1701 Audio Digital Signal Processor Board
  + RCA Interface
  + freier Cypress Programmer (viiiiel billiger als das Original von Analog)

- für etwa 50€ inkl. Versand vom https://www.hobbyhifiladen.de/

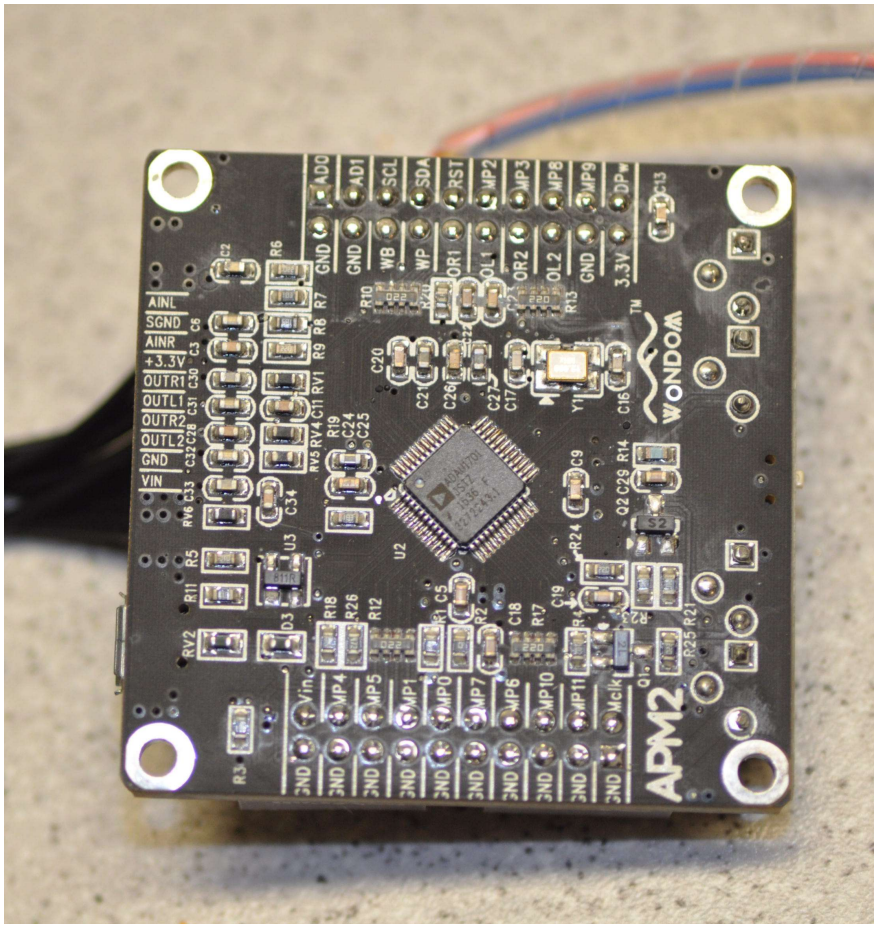
4. Alles in einem DSP

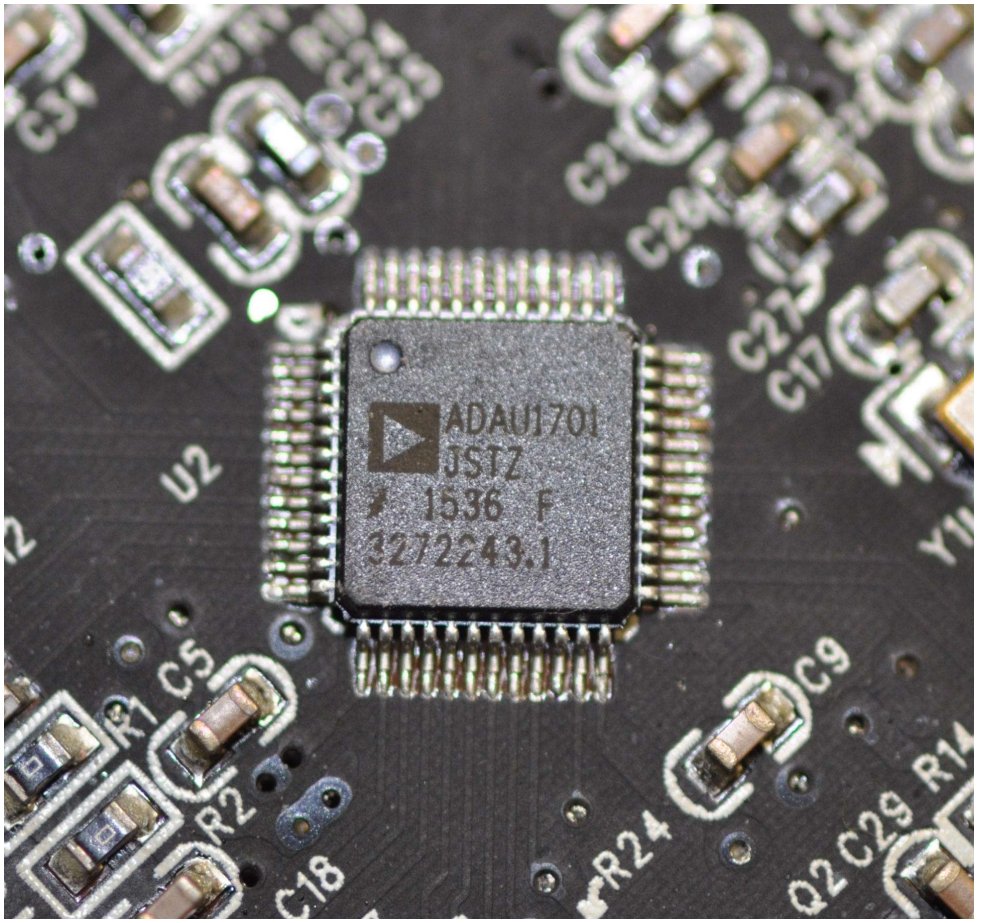
Sure Electronics ADAU1701 Audio Digital Signal Processor Board
+ RCA Interface
+ freier Cypress Programmer (viiiiel billiger als das Original von Analog)



4. Alles in einem DSP

Sure Electronics ADAU1701 Audio Digital Signal Processor Board

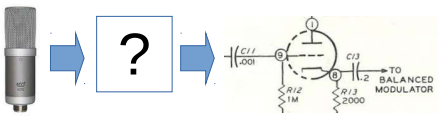




4. Alles in einem DSP

Sure Electronics ADAU1701 Audio Digital Signal Processor Board

Ausprobieren….!



Endergebnis

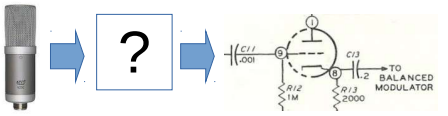
1. Kompressor ✓
2. Equalizer in HW –
3. Equalizer in SW ✓

4. Alles in einem DSP ✓

Jetzt muss ich‘s nur noch bauen…

(oder auch nicht – lehrreich war es jetzt schon :-)


Danke für Eure Aufmerksamkeit!


?
TO
BALANCED
MODULATOR

SW Setup

- Treiber Software für die Adapterplatine zwischen SigmaDSP Suite und DSP Board,
  vom Chip-Entwickler des USB Interface Devices (Cypress)
  CySuiteUSB_3_4_7_B204.exe (enthält CyConsole)
  CY3684 EZ-USB FX2LP Development Kit (enthält Treiber für Win7)
- Software zum Erstellen von „Baublock-Beschreibungen“ für einen SigmaDSP: SigmaStudio, Download von Analog Devices, ADI_SigmaStudio-Rel3.14-x64.exe
- Bootfile zur Verwendung in der CySuiteUSB, aus dem SigmaStudio ….\SigmaStudioDSP\USB drivers\x64\ADI_USBi.spt