

# GITEC-Meeting 2021

16. und 17. Oktober 2021, in der Music Academy Regensburg

Digitalisierte E-Gitarristen

Alexander Potchinkov

- 
- Digitalisierung
    - im allgemeinen und im speziellen
  - Analoge und digitale Signalverarbeitung
    - digitale Signalverarbeitung ist die Verarbeitung analoger Signale mit Digitalrechnern
    - Analogien zwischen analoger und digitaler Signalverarbeitung, Abtasttheorem und Signalkette
  - Elementare Algorithmen der digitalen Signalverarbeitung
    - Differenzengleichung, Digitalfilter,
  - Nichtlineare Digitale Signalverarbeitung: Was machen wir mit dem Abtasttheorem?
    - Nichtlinearitäten und Aliasing, Overdrive, Fuzz, Polynomauswertung für nichtlineare Verzerrer
  - Effekte und Aspekte ihrer digitalen Realisierung
  - Vorschläge zur Ausführung der Digitalisierung
    - Onlinebetrieb und Simulationsbetrieb, Signalcontroller und DSPs (Digitale Signalprozessoren)
    - Graphisches Programmieren, direktes Programmieren (die harte Tour), Simulation mit Python
  - Hardware für die ersten Schritte
  - Wie geht's weiter?

## Digitalisierung

Digitalisierung bezeichnet die Einbeziehung digitaler Technologien in geschäftliche / soziale Prozesse mit dem Ziel, diese zu verbessern.

Die Digitalisierung ist transformativ. Sie ändert die Art und Weise, in der Unternehmen mit ihren Kunden interagieren, und häufig auch ihre Einnahmequellen.

Ursprünglich bedeutet Digitalisierung das Umwandeln physischer Daten in digitale Daten, wie es beispielsweise beim Scannen eines Berichts geschieht.

[www.scrive.com](http://www.scrive.com)

Digitalisierung bezeichnet die Einbeziehung digitaler Technologien in die E-Gitarrensignalverarbeitung mit dem Ziel, diese zu verbessern.

Die Digitalisierung ist transformativ. Sie ändert die Art und Weise, in der E-Gitaristen mit ihren Effektgeräten interagieren, und häufig auch ihre Soundcharakteristika.

Ursprünglich bedeutet Digitalisierung das Umwandeln des analogen Gitarrensignals in ein digitales, wie es beispielsweise durch die Nutzung eines AD-Wandlers geschieht.

Digitale Gitarreneffekte sind naheliegend, da digitale Signalverarbeitung wesentlich leistungsfähigere Kontrolle der Effekteigenschaften erlaubt. Es gibt zahlreiche dokumentierte Experimente und Implementierungen, man muß nur mit der Auswahl einer Hardwareplattform beginnen...

## Analoge und digitale Signalverarbeitung

Gitarrentonabnehmersignal und Ansteuersignal des Gitarrenverstärkers sind Analogsignale.

Ein Gitarreneffektgerät, z.B. ein Pedal, ist ein System der Signalverarbeitung. Mit diesem System wird ein Eingangssignal  $x()$ , das Gitarrentonabnehmersignal, in ein Ausgangssignal  $y()$ , das Ansteuersignal des Gitarrenverstärkers überführt.

Diese Überführung erfolgt bei analoger Signalverarbeitung mit einer elektronischen Schaltung und bei digitaler Signalverarbeitung mit einem Algorithmus.



Der Algorithmus der digitalen Signalverarbeitung wird mit einem Digitalrechner ausgeführt.

Um digitale Signalverarbeitung anzuwenden bedarf es Wandler mit denen das Analogsignal in ein Digitalsignal, AD-Wandler, und das Digitalsignal in ein Analogsignal, DA-Wandler, gewandelt werden.



## Analoge und digitale Signalverarbeitung

Analogien zwischen analoger Signalverarbeitung mit Operationsverstärkern und digitaler Signalverarbeitung mit Signalprozessoren am Beispiel einer Frequenzweiche

**Analog:** Blockstruktur, Teilsysteme arbeiten zeitlich parallel und sind räumlich verteilt

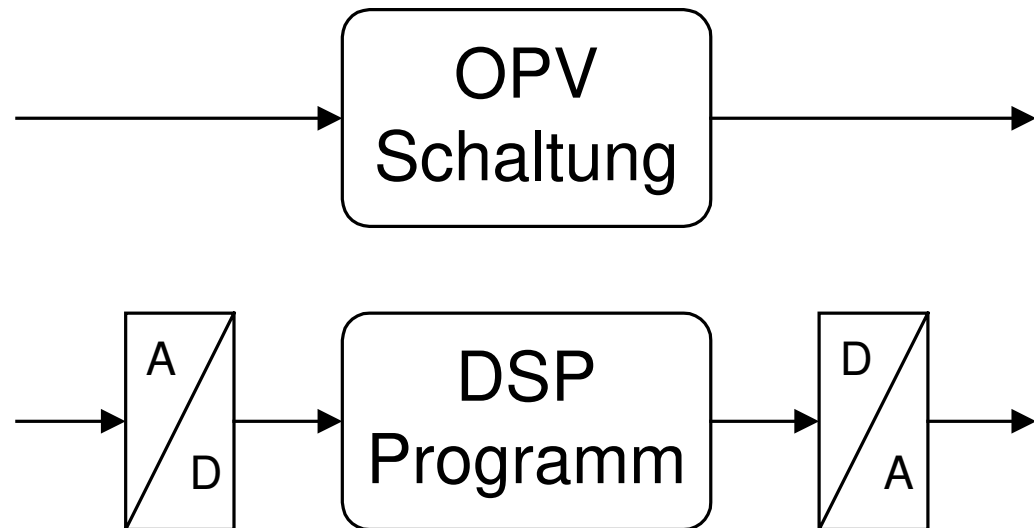
**Digital:** Blockstruktur, Teilsysteme arbeiten zeitlich sequentiell und sind im Rahmen der Audioprogrammschleife zusammengefasst

**Analog:** Filterfunktion mit Grundschtung, Filtereigenschaften werden mit den Bauelementewerten eingestellt

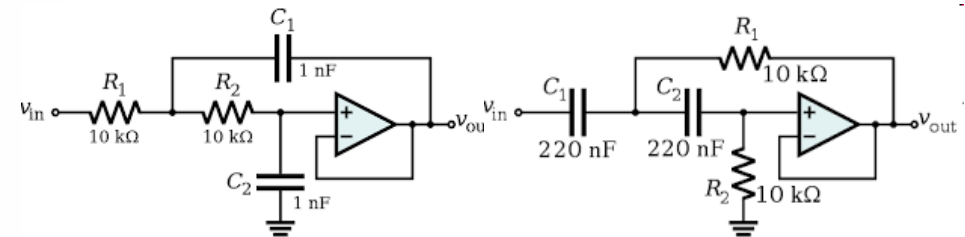
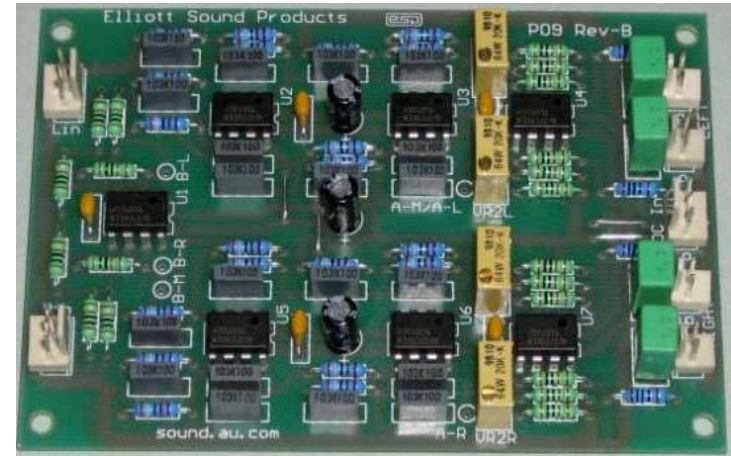
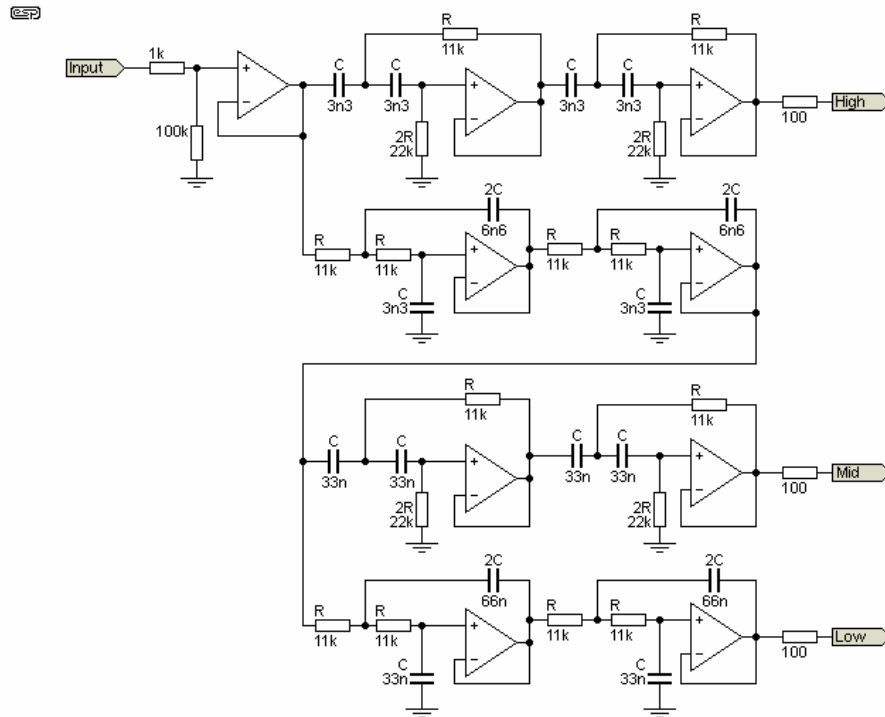
**Digital:** Filterfunktion mit Unterprogramm (auch Filtermakro), Filtereigenschaften werden mit den Koeffizientenwerten eingestellt

Analog: Filterschaltung, z.B.  
Sallen&Key-Topologie

Digital: Faltungssummen, z.B.  
Direktstruktur II



# Analoge und digitale Signalverarbeitung

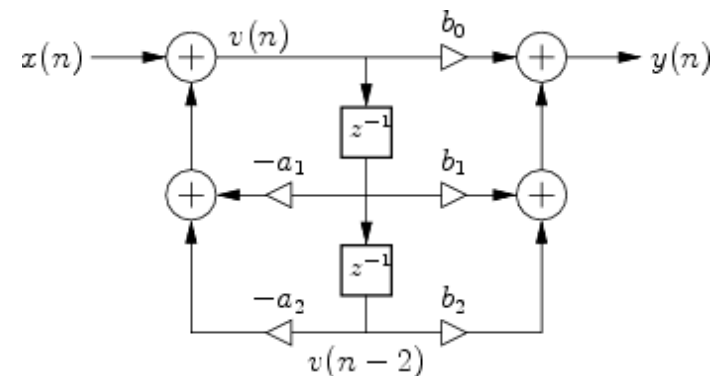


Schaltungen

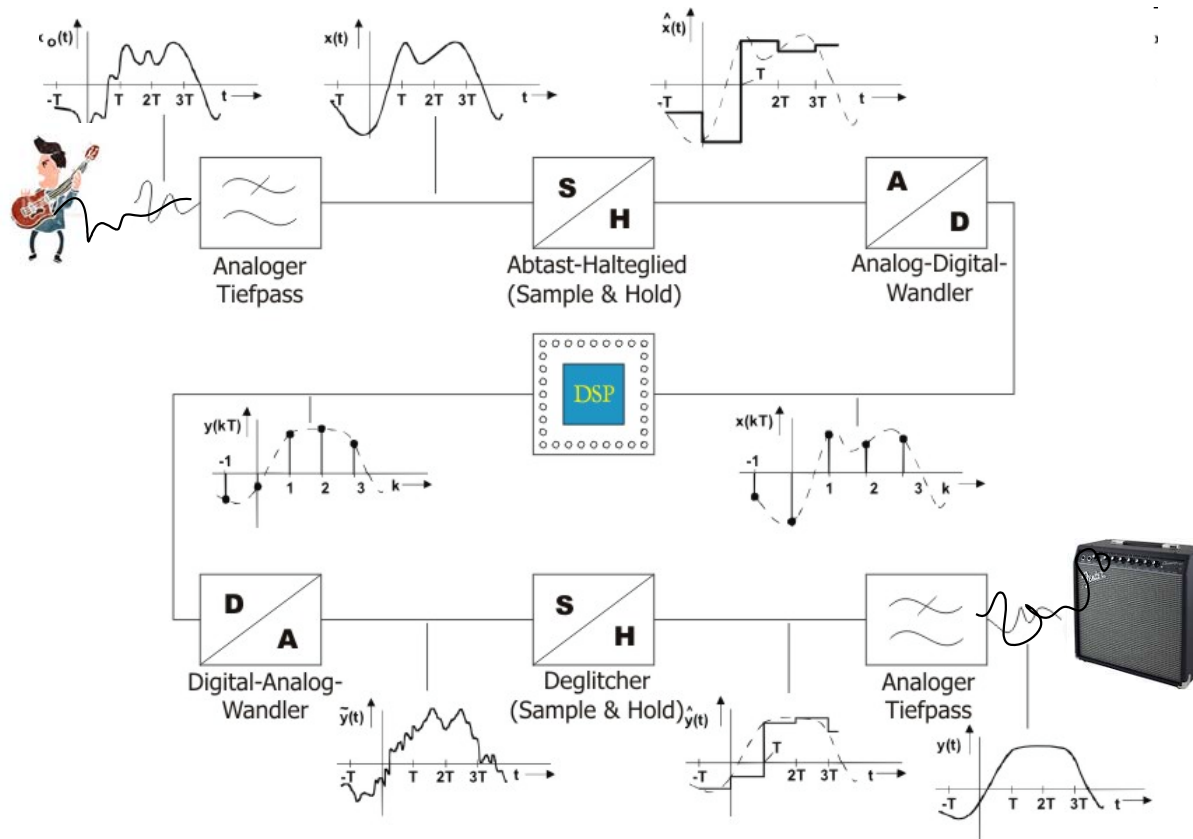
Algorithmen

- Universell / programmierbar
- Verstärker und passive Bauelemente / Algorithmen
- Grundsaltungen / Grundalgorithmen

Ein Signalcontroller/DSP ersetzt **Schaltungen** der analogen Signalverarbeitung durch **Algorithmen** der digitalen Signalverarbeitung. Signalcontroller/DSPs werden programmiert und Analogschaltungen verschaltet.

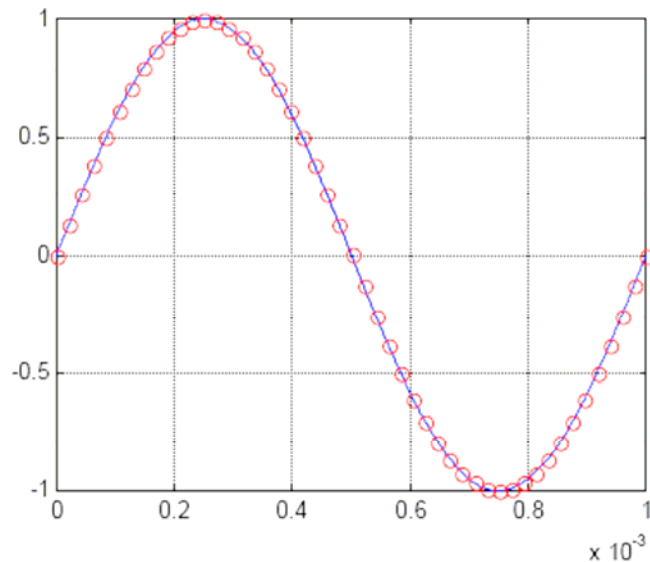


# Abtasttheorem und Signalkette Analog-Digital-Analog



Das **Abtasttheorem** besagt, daß ein kontinuierliches, bandbegrenztes Signal, mit einer Minimalfrequenz von 0 Hz und einer Maximalfrequenz  $f_{\max}$ , mit einer Frequenz größer als  $2 \cdot f_{\max}$  gleichförmig abgetastet werden muß, damit man aus dem so erhaltenen zeitdiskreten Signal das Ursprungssignal ohne Informationsverlust, aber mit unendlich großem Aufwand, exakt rekonstruieren oder – mit endlichem Aufwand – beliebig genau approximieren kann.  
(Wikipedia)

# Elementare Algorithmen der digitalen Signalverarbeitung



Wir gewinnen das Digitalsignal durch gleichmäßige Abtastung aus dem Analogsignal.

Analogsignal:

$$x(t) = \sin(2\pi ft), t \in [0, 1ms], f = 1kHz$$

$$x(n) = \sin\left(2\pi \frac{f}{f_T} n\right), n = 0, \dots, 48, f_T = 48kHz$$

$y(n) \leftarrow \text{Algorithmus} \{ x(n), n=0, \dots, N-1; y(n-1), n=0, \dots, M-1 \}$

$$y(n) = \frac{1}{5}(x(n) + x(n-1) + x(n-2) + x(n-3) + x(n-4))$$

Butterworth-Tiefpassfilter, 2. Ordnung, Grenzfrequenz  $f_T/200$

$$y(n) = 0,002414x(n) + 0,004827x(n-1) + 0,002414x(n-2) + 1,9556y(n-1) - 0,9565y(n-2)$$

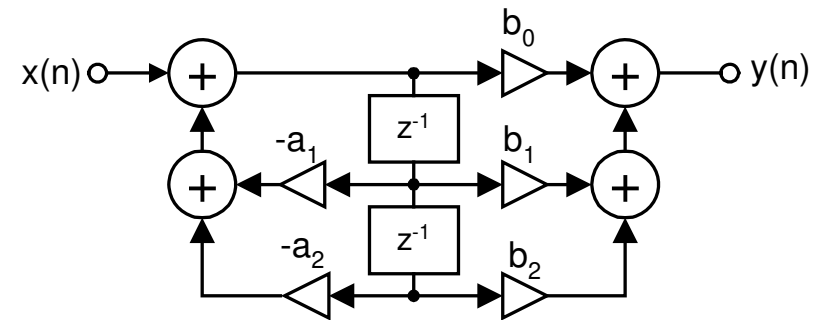
```
x(0) = 0.0
x(1) = 0.0042
x(2) = 0.0084
x(3) = 0.0126
x(4) = 0.0168
x(5) = 0.0210
x(6) = 0.0251
x(7) = 0.0293
x(8) = 0.0335
x(9) = 0.0377
x(10) = 0.0419
x(11) = 0.0461
```

Zeitindices,  
multipliziere mit  
 $1/48kHz = 20,83\mu s$

Samples

## Elementare Algorithmen der digitalen Signalverarbeitung

IIR-System, Grad 2, Direktstruktur II,  
Direkte Adressierung, MAC-Operation



```
s1:=0; s2:=0;
do <1>, *          # Schleife über Datenstrom
    x <- In-Dev;    # Lies Eingangsdatum
    hv0 := a1*s1;
    hv0 := hv0 + a2*s2; # MAC
    hv1 := x - hv0;
    y := b2*s2;
    y := y + b1*s1;      # MAC
    y := y + b0*hv1;     # MAC
    s2 := s1;
    s1 := hv1;
    y -> Out-Dev;
<1>
```

# Nichtlineare Digitale Signalverarbeitung: Was machen wir mit dem Abtasttheorem?

Die durch die Nichtlinearitäten entstehenden Oberschwingungen müssen auch das Abtasttheorem einhalten.

Eingang:  $u_1 = A \cdot \sin(\omega t) + B$

mit

$A$  - Schwingungsamplitude

$B$  - überlagerter Gleichwert

Kennlinie:  $u_2 = u_1^2$

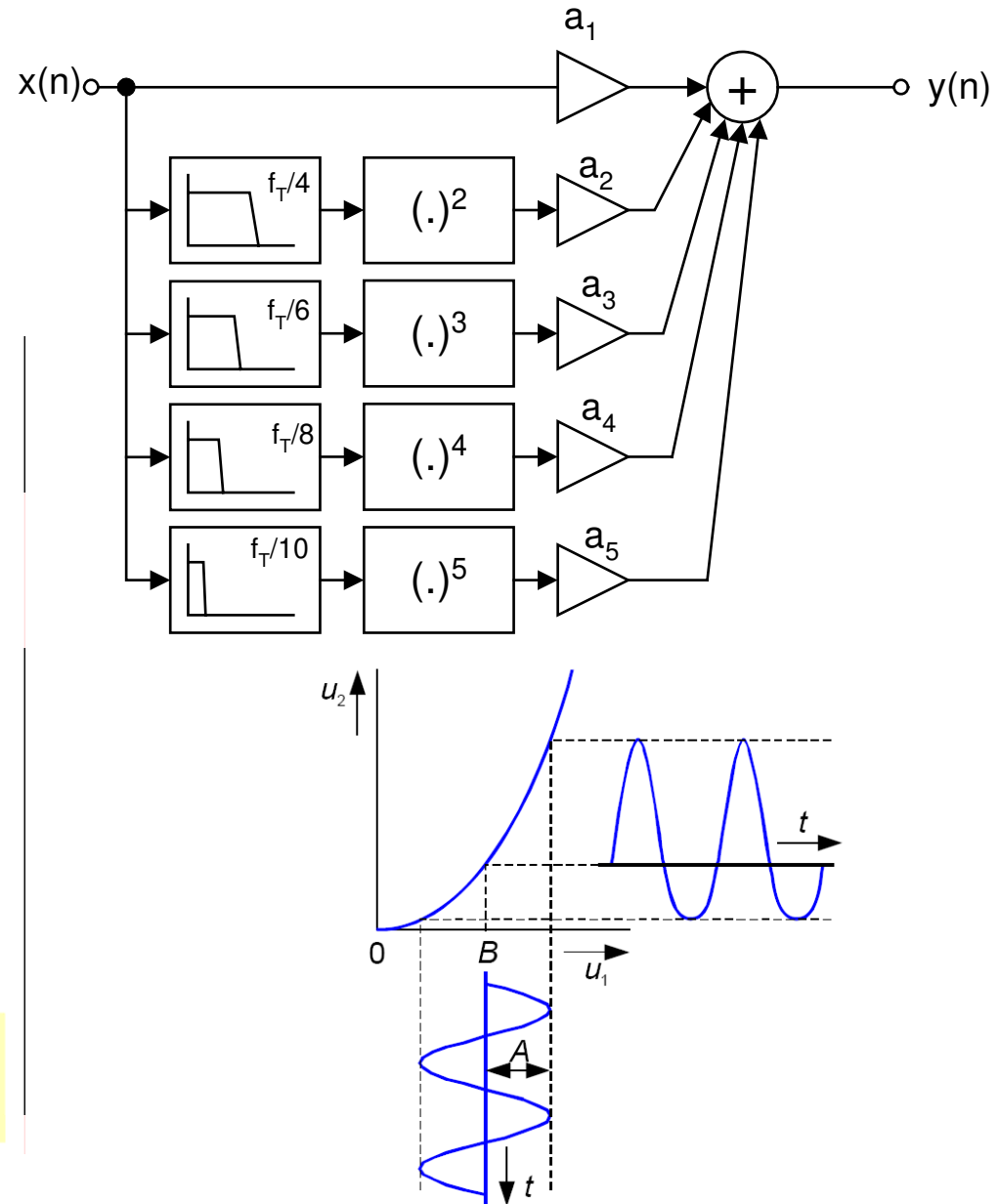
Ausgang:

$$u_2 = [A \cdot \sin(\omega t) + B]^2$$

$$u_2 = A^2 \cdot \sin^2(\omega t) + 2AB \cdot \sin(\omega t) + B^2$$

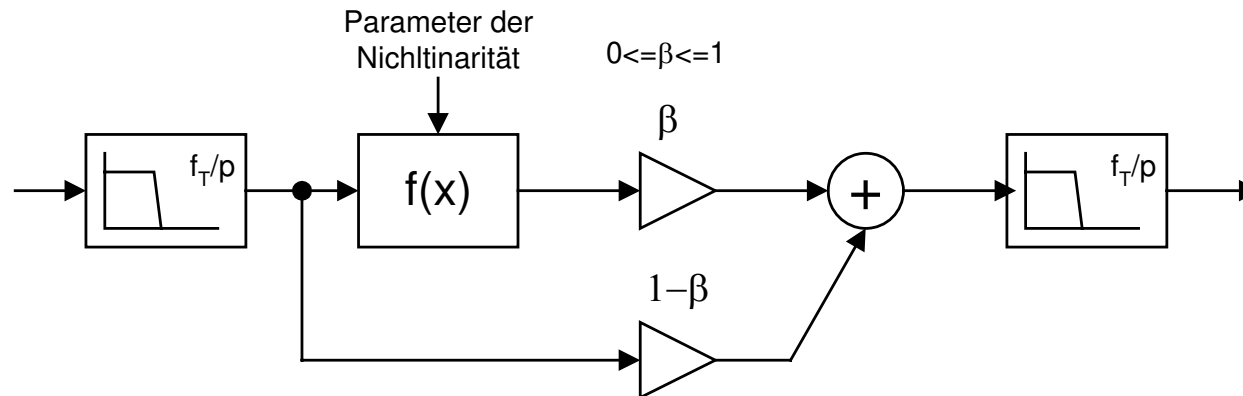
$$u_2 = A^2 \cdot \frac{1}{2} [1 - \cos(2\omega t)] + 2AB \cdot \sin(\omega t) + B^2$$

$$u_2 = \left[ \frac{A^2}{2} + B^2 \right] + 2AB \cdot \sin(\omega t) - \frac{A^2}{2} \cdot \cos(2\omega t)$$



# Nichtlineare Digitale Signalverarbeitung: Was machen wir mit dem Abtasttheorem?

Mischer von linearer und nichtlinearer Signalverarbeitung



Overdrive, Parameter  $th$ , default  $th=1/3$

$$f(x) = \begin{cases} 2x & \text{for } 0 \leq x < 1/3 \\ \frac{3-(2-3x)^2}{3} & \text{for } 1/3 \leq x < 2/3 \\ 1 & \text{for } 2/3 \leq x \leq 1 \end{cases}$$

Fuzz , Parameter  $\alpha$

$$f(x) = \frac{x}{|x|} (1 - e^{\alpha x^2 / |x|})$$

```
if abs(x(n)) < th, y(i)=2*x(i); end;
if abs(x(n)) >= th,
    if x(n) > 0, y(i)=(3-(2-x(i)*3).^2)/3; end;
    if x(n) < 0, y(i)=-(3-(2-abs(x(i))*3).^2)/3; end;
end;
if abs(x(n)) > 2*th,
    if x(n) > 0, y(n)=1; end;
    if x(n) < 0, y(n)=-1; end;
end;
```

```
q=x/max(abs(x(n)));
y(n)= sign(-q).*(1-exp(alpha*sign(-q).*q));
```

## Nichtlineare Digitale Signalverarbeitung: Polynome mit $N \geq 2$

Mithilfe des Horner-Schemas (William George Horner, 1786-1837) können Polynomwerte berechnet und Polynome durch Linearfaktoren dividiert werden. Wir betrachten ein Polynom vom Grad  $N$  mit

$$P_N(x) = \sum_{n=0}^N a_n x^n = a_0 + a_1 x + a_2 x^2 + \dots + a_N x^N$$

Das Polynom lässt sich (Horner) in maschinengerechter Form notieren

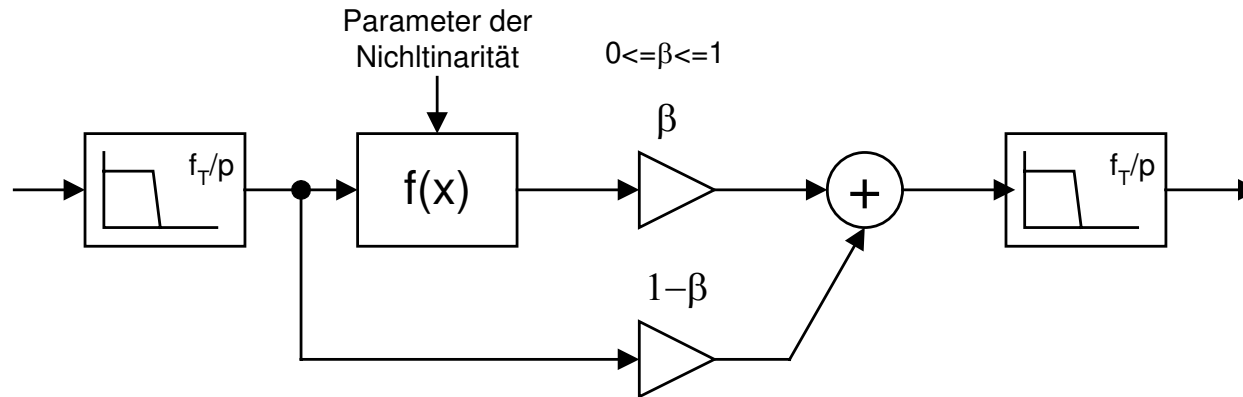
$$P_N(x) = (\dots((a_N x + a_{N-1})x + a_{N-2})x + \dots + a_1)x + a_0.$$

$$\text{Bsp. } P_3(x) = x^3 - 4x^2 + x + 6 = ((x-4)x+1)x+6 = x(x(x-4)+1)+6$$

### Algorithmus: Polynomauswertung mit Horner-Schema

Auswertestelle  $x$ , Polynomkoeffizienten  $a_n$ ,  $n=0, \dots, N-1$

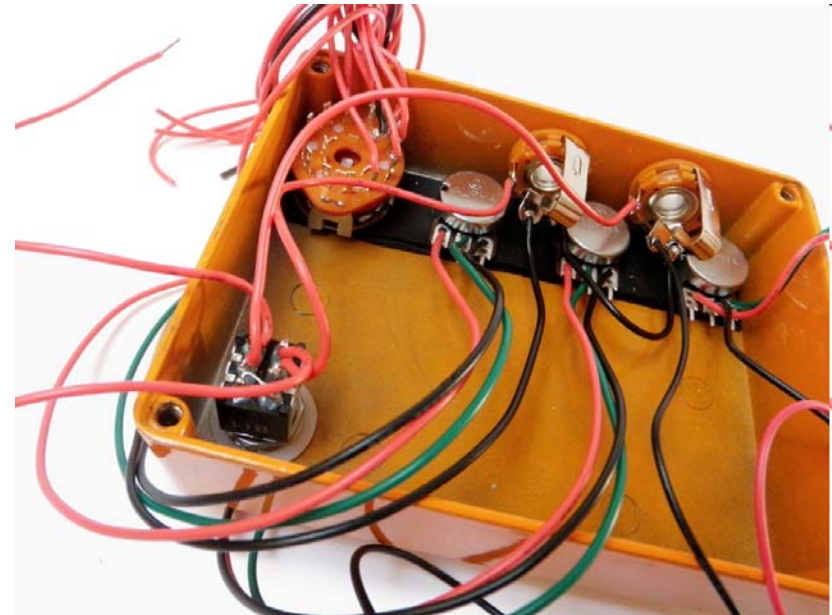
```
const a=[a0, a1, a2, ..., an]  
p:=0;                               Polynom-Wert  
do <1>, n=N-1, ..., 0,  
    p:=p*x+a(n);  
<1>
```



Polynomiale Nichtlinearität vom Grad  $N \geq 2$

$$f(x) = \sum_{n=1}^N a_n x^n = a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_N x^N$$

Für ein nichtlineares Effektgerät können die Polynomkoeffizienten z.B. mit Potentiometern, die vom AD-Umsetzer des Signalcontrollers abgefragt werden, eingestellt werden können. Man kann sich dann beliebige Kennlinien zurechtbiegen. Analog wäre so etwas nur mit absurdem Aufwand umsetzbar.



# Effekte und Aspekte ihrer digitalen Realisierung

## Kategorisierung im Hinblick auf eine Digitalisierung, Ampelsystem

- \* Lineare Effekte -> Sehr leicht digital zu realisieren
  - \* Lineare Effekte mit Signalspeicherung -> Die **absolute Stärke** digitaler Signalverarbeitung
  - \* Nichtlineare Effekte -> Abtasttheorem beachten (Aliasing). zusätzlicher Aufwand, hohe Abtastfrequenz und Tiefpassfilter nötig.
- 

- Modulationseffekte: Ringmodulator, Tremolo, Vibrato
- Verzögerungseffekte: Flanger, Chorus, Echo
- Zeitveränderliche Filter: Wah-Wah, Phaser
- Elementarfilter, zeitunveränderlich: Selektive Filter (Tief-, Hochpass- und Bandpassfilter), Shelvingfilter, Equalizer (Q-Filter)
- Dynamikprozessoren: Kompressoren, Limiter
- Nichtlineare Verzerrer: Overdrive, Distortion, Fuzz, Exciter/Enhancer
- Raumschall: Panning, Reverb

## Vorschläge zur Ausführung der Digitalisierung

Ziel: kostengünstige Lösungen  
leicht erhältliche Komponenten  
Sofort-Start -> kaufen, anschließen, loslegen

Voraussetzungen: Elementare Programmierkenntnisse, vorzugsweise C(++) und/oder Python  
Reduziertes C ohne komplizierte Datenstrukturen, keine dynamische Speicher-  
verwaltung ...  
Erweitertes C Zugriff auf die Signalcontroller-Schnittstellen, Interrupts ...

Heutzutage ist es beneidenswert einfach in die Technik einzusteigen.

Arduino, Signalcontroller Teensy mit Arduino-IDE, Beaglebones, Raspberries,  
IDE, graphische Programmierung

## Onlinebetrieb und Simulationsbetrieb

Für den Onlinebetrieb, d.h. das digitale Effektgerät wird statt des analogen Effektgeräts verwendet, verwenden wir i.a. geeignete Prozessoren **ohne** Betriebssystem. Dies können z.B. Signalprozessoren sein oder Mikrocontroller mit Signalprozessorfunktionen (Signalcontroller).

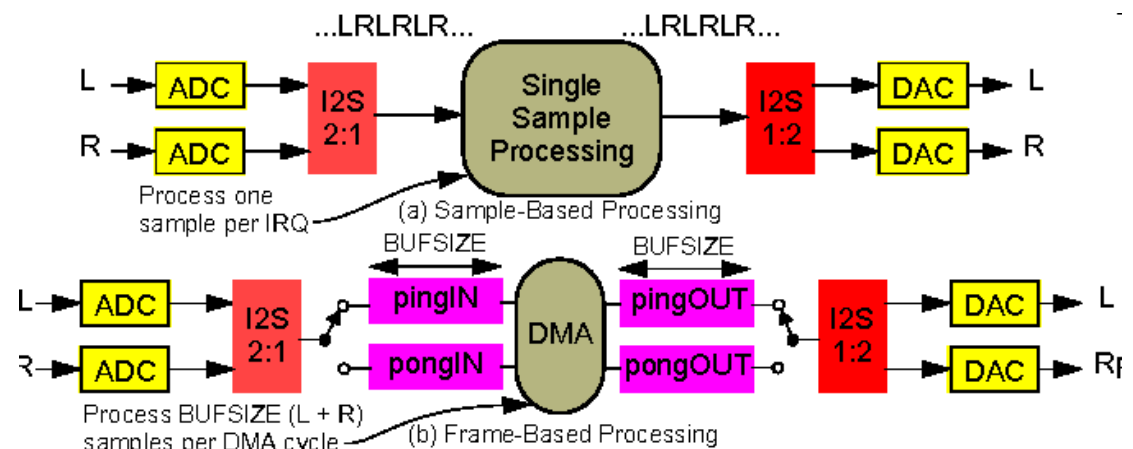
Rechner mit Betriebssystemen wie Windows oder Linux werden zur Simulation bzw. dem Test der Effekalgorithmen verwendet. PCs haben zwar genug Rechenleistung, aber das Betriebssystem verursacht eine recht hohe Latenz (Verzögerung) aufgrund der großen Datenpuffer.

Signalprozessoren und Signalcontroller werden entweder mit Maschinensprache (Assembler), mit einer imperativen Programmiersprache wie C(++) oder mit einer Skriptsprache wie Mikropython programmiert.

Simulationen werden meist mit Skriptsprachen wie Python oder Matlab programmiert.

DSP und Signalcontroller  
direkt

PC mit Betriebssystem  
und Signalcontroller mit CMSIS



# Onlinebetrieb mit DSPs und Signalcontrollern, Wie funktioniert's, die Audioloop

## Setup, Einstellen der (Audio-)Schnittstellen

### Beginn Loop

- \* Warte, bis Flag gesetzt ist
- \* Setze Flag zurück
- \* Lese Empfangsbuffer  $\Rightarrow x(n)$
- \* **Signalverarbeitung  $y(n) = f(x(n))$**
- \* Schreibe Sendebuffer  $\leftarrow y(n)$
- \* Springe zu Beginn Loop

**RCV-ISR:** Schreibe die seriell empfangenen Audiodaten kanalweise in den Empfangsbuffer  
Setze Flag, wenn alle Kanäle eingegangen sind

**TRX-ISR:** Schreibe die Sendebufferdaten kanalweise in die serielle Schnittstelle

Alle diese Vorgänge dürfen maximal  $T=1/f_T$  an Zeit verbrauchen

Die ISR (Interrupt Service Routinen) arbeiten sozusagen im Hintergrund und werden entsprechend der Anzahl der Audiokanäle mehrfach in einem Abtastzeitintervall  $T$  ausgeführt.

Das Beispiel zeigt eine solche Struktur für einen Teensy 4.0

-> Kompliziert ist **nur** das Einstellen der Audioschnittstellen !!! Hat man aber einmal einen passenden Rahmen programmiert, tauscht man lediglich den Signalverarbeitungsblock aus

# Online-Signalverarbeitung mit einem Signalcontroller vom Typ Cortex-M7

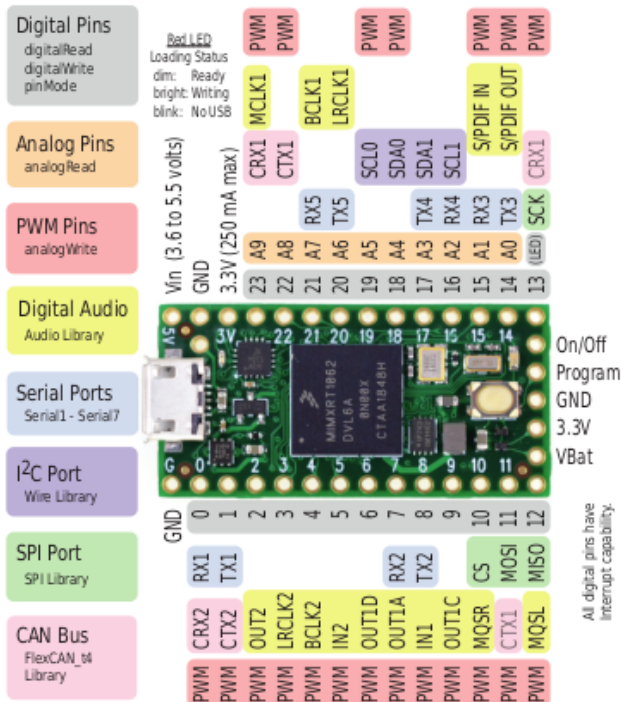


## Welcome to Teensy® 4.0

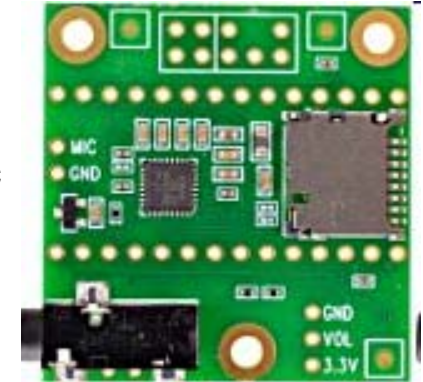
32 Bit Arduino-Compatible Microcontroller

To begin using Teensy, please visit the website & click [Getting Started](http://www.pjrc.com/teensy).

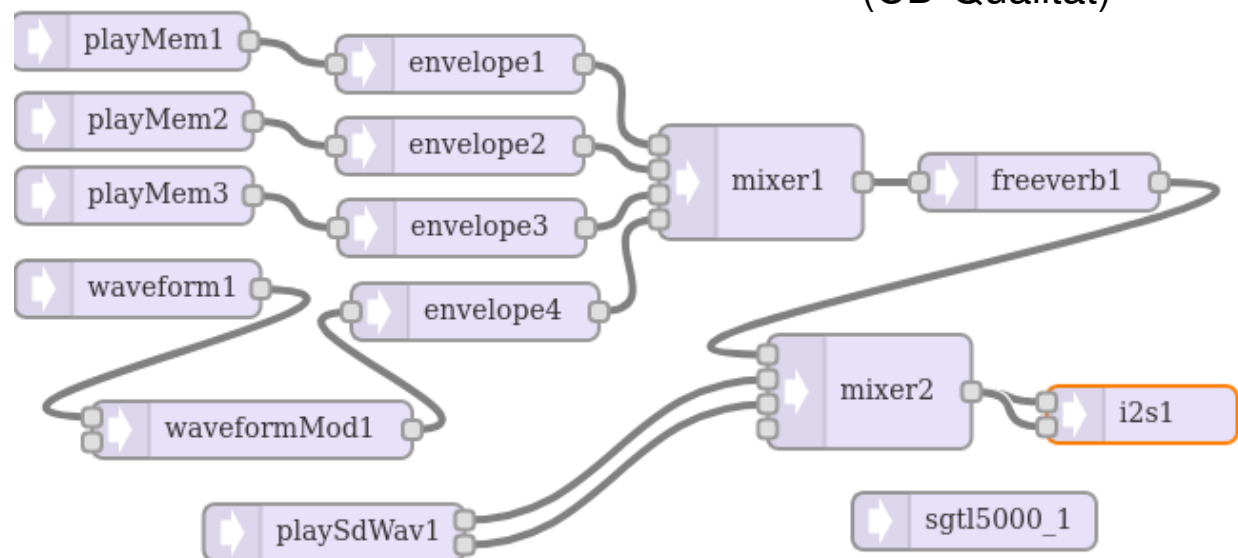
[www.pjrc.com/teensy](http://www.pjrc.com/teensy)



ARM Cortex-M7 at 600 MHz  
 Float point math unit, 64 & 32 bits  
 1984K Flash, 1024K RAM  
 USB device 480 Mbit/sec & USB host 480 Mbit/sec  
 40 digital input/output pins, 31 PWM output pins  
 14 analog input pins  
 7 serial, 3 SPI, 3 I2C ports  
 2 I2S/TDM and 1 S/PDIF digital audio port  
 32 general purpose DMA channels  
 Programmable FlexIO  
 Peripheral cross triggering



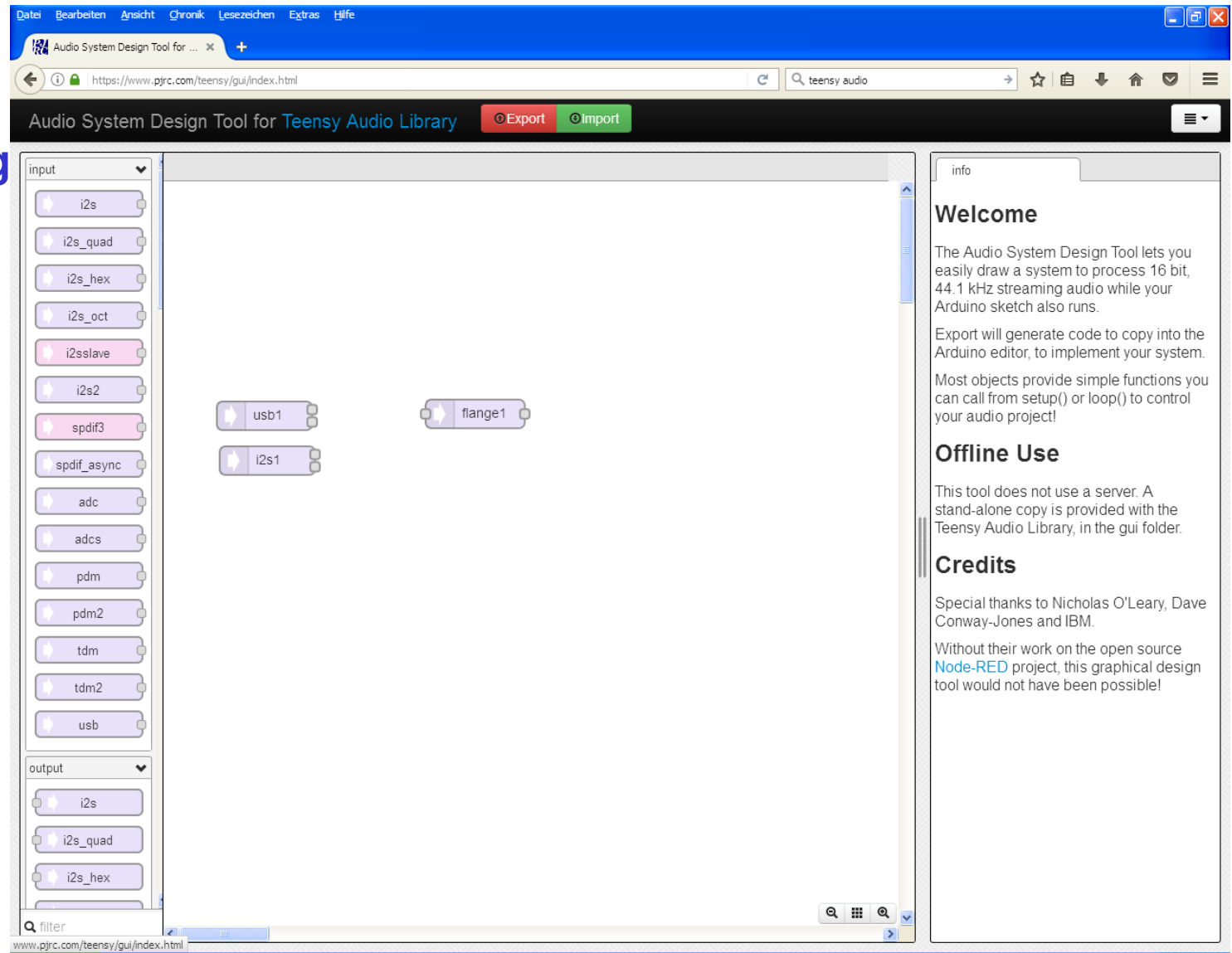
Audio-Interface,  
 Codec, 16Bit,  
 $f_t=44.1\text{kHz}$   
 (CD-Qualität)



Für die ersten Gehversuche ist es empfehlenswert, das sog. Audio System Design Tool zu verwenden und sich die Signalverarbeitung „zusammenzuklicken“. Gleich richtig loszulegen kann frustrierend sein, wenn man noch nicht so viel Erfahrung hat.

## Graphische Programmierung

Ist man weiter fortgeschritten, kann man selbst programmieren. Man ist dann nicht mehr auf eine mehr oder weniger umfangreiche Sammlung von Programmodulen angewiesen.



## Was ist ein Signalcontroller?

DSP-Funktionen kombiniert mit den IO-Peripherie von Mikrocontrollern. In vielen Fällen sind zum Betrieb keine weiteren Schaltkreise nötig. Die Hersteller bieten umfangreiche Programmierumgebungen, i.d.R. kostenfrei und Bibliotheken. Das Internet ist eine überbordende Quelle an Informationen (Foren, Youtube-Videos, GitHub (where the world builds software))

A digital signal controller (DSC) is a hybrid of microcontrollers and digital signal processors (DSPs). Like microcontrollers, DSCs have fast interrupt responses, offer control-oriented peripherals like PWMs and watchdog timers, and are usually programmed using the C programming language, although they can be programmed using the device's native assembly language. On the DSP side, they incorporate features found on most DSPs such as single-cycle multiply–accumulate (MAC) units, barrel shifters, and large accumulators.

DSCs, like microcontrollers and DSPs, require software support. There are a growing number of software packages that offer the features required by both DSP applications and microcontroller applications. With a broader set of requirements, software solutions are more rare. They require: development tools, DSP libraries, optimization for DSP processing, fast interrupt handling, multi-threading, and a tiny footprint.

[https://en.wikipedia.org/wiki/Digital\\_signal\\_controller](https://en.wikipedia.org/wiki/Digital_signal_controller)

The Cortex Microcontroller Software Interface Standard CMSIS includes a DSP component. The CMSIS DSP library is a collection of over 60 common DSP functions that support fixed and floating point data types. CMSIS-DSP is optimised to use the Single Instruction Multiple Data instructions that are available in the Cortex-M4, Cortex-M7 and Cortex-M33.

<http://www.hitex.com>

```
// LRCK Pin 20 Output, BCLK Pin 21 Output, DIN Pin 8 Input, DOUT Pin 7 Output, DIN Pin 8 Input, MCLK Pin 23 Output

#define I2S_TCSR_FRIE    ((uint32_t)(1<<8))    // FIFO Request Interrupt Enable
#define I2S_TCSR_FRF     ((uint32_t)(1<<16))    // FIFO Request Flag
#define I2S_TCSR_FEF     ((uint32_t)(1<<18))    // FIFO Error Flag
#define I2S_TCSR_SEF     ((uint32_t)(1<<19))    // Sync Error Flag

void config_i2s(void) {
    // if either transmitter or receiver is enabled, do nothing
    if (I2S1_TCSR & I2S_TCSR_TE) return;
    if (I2S1_RCSR & I2S_RCSR_RE) return;
    CCM_CCGR5 |= CCM_CCGR5_SAI1(CCM_CCGR_ON);
    IOMUXC_GPR_GPR1 = (IOMUXC_GPR_GPR1
        & ~(IOMUXC_GPR_GPR1_SAI1_MCLK1_SEL_MASK | ((uint32_t)(1<<20)) ))
        | (IOMUXC_GPR_GPR1_SAI1_MCLK1_DIR | IOMUXC_GPR_GPR1_SAI1_MCLK1_SEL(0));
    CORE_PIN23_CONFIG = 3;    //1:MCLK
    CORE_PIN21_CONFIG = 3;    //1:RX_BCLK
    CORE_PIN20_CONFIG = 3;    //1:RX_SYNC
    // TRANSMITTER
    I2S1_TMR = 0;
    I2S1_TCR1 = I2S_TCR1_RFW(1); // watermark is 2 samples
    I2S1_TCR2 = I2S_TCR2_SYNC(1) | I2S_TCR2_BCP; // transmitter synchronous; BCLK polarity
    I2S1_TCR3 = I2S_TCR3_TCE;
    I2S1_TCR4 = I2S_TCR4_FRSZ((2-1)) | I2S_TCR4_SYWD((32-1)) | I2S_TCR4_MF | I2S_TCR4_FSE | I2S_TCR4_FSP; // FSD = 0 (Slave)
    I2S1_TCR5 = I2S_TCR5_WNW((32-1)) | I2S_TCR5_WOW((32-1)) | I2S_TCR5_FBT((32-1));
    // RECEIVER
    I2S1_RMR = 0;
    I2S1_RCR1 = I2S_RCR1_RFW(1);
    I2S1_RCR2 = I2S_RCR2_SYNC(0) | I2S_RCR2_BCP; //
    I2S1_RCR3 = I2S_RCR3_RCE;
    I2S1_RCR4 = I2S_RCR4_FRSZ((2-1)) | I2S_RCR4_SYWD((32-1)) | I2S_RCR4_MF | I2S_RCR4_FSE | I2S_RCR4_FSP;
    I2S1_RCR5 = I2S_RCR5_WNW((32-1)) | I2S_RCR5_WOW((32-1)) | I2S_RCR5_FBT((32-1));
}

void beginI2S(void) {
    config_i2s();
    // OUTPUT
    CORE_PIN7_CONFIG = 3;    //1:TX_DATA0
    I2S1_TCSR = I2S_TCSR_TE | I2S_TCSR_BCE;
    // INPUT
    CORE_PIN8_CONFIG = 3;    //1:RX_DATA0
    IOMUXC_SAI1_RX_DATA0_SELECT_INPUT = 2;
    IOMUXC_SAI1_RX_BCLK_SELECT_INPUT = 1;
    IOMUXC_SAI1_RX_SYNC_SELECT_INPUT = 1;
    I2S1_RCSR = I2S_RCSR_RE | I2S_RCSR_BCE | I2S_RCSR_FR;
}
```

```

void isr(){
    if(!(I2S1_TCSR & I2S_TCSR_FRF) && !(I2S1_RCSR & I2S_TCSR_FRF)) {return;}
    in[0] = I2S1_RDR0;
    in[1] = I2S1_RDR0;
    I2S1_TDR0 = out[0];
    I2S1_TDR0 = out[1];
    readFlag = 1;
    if(I2S1_TCSR & I2S_TCSR_FEF) {
        I2S1_TCSR |= I2S_TCSR_FEF; // clear if underrun
    }
    if(I2S1_TCSR & I2S_TCSR_SEF) {
        I2S1_TCSR |= I2S_TCSR_SEF; // clear if frame sync error
    }
    if(I2S1_RCSR & I2S_TCSR_FEF) I2S1_RCSR |= I2S_TCSR_FEF; // clear if underrun
    if(I2S1_RCSR & I2S_TCSR_SEF) I2S1_RCSR |= I2S_TCSR_SEF; // clear if frame sync error
}

```

```

void setup() {
    //Serial.begin(9600);
    beginI2S(); // konfiguriere I2s
    _VectorsRam[IRQ_SAI1 + 16] = isr; // ISR eintragen
    NVIC_ENABLE_IRQ(IRQ_SAI1); // ISR aktivieren
    I2S1_TCSR = I2S_TCSR_TE | I2S_TCSR_BCE | I2S_TCSR_FRIE | I2S_TCSR_FR;
    I2S1_RCSR = I2S_RCSR_RE | I2S_RCSR_BCE | I2S_TCSR_FRIE | I2S_RCSR_FR;
}

```

## Die Audio-Loop

```

void loop() {

    while(readFlag == 0){ } // Warte bis Audio ISR aufgerufen wurde
    readFlag = 0;

    out[0] = in[0];
    out[1] = in[1];
}

```

Die beiden vorigen Folien sehen „grausam“ aus, aber diese Kärnerarbeit leistet man nur einmal, später fügt man seine Algorithmen in die Audioloop ein.

```
void loop() {  
  
    while(readFlag == 0){ } // Warte bis Audio ISR aufgerufen wurde  
    readFlag = 0;  
  
    out[0] = in[0];  
    out[1] = in[1];  
}
```

Ein Signalcontroller vom Typ Cortex-M7 ersetzt im Hinblick auf die „pure FIR-Leistung“ keinen DSP. Aber wenn nicht nur Audioschnittstellen und SPI/I<sup>2</sup>C gefordert werden, sondern auch beispielsweise WiFi/Ethernet oder QSPI für externes RAM, dann ist es ein solcher Signalcontroller in die engere Wahl zu ziehen.

Der ARM-Core wird von verschiedenen Herstellern verbaut. Eine interessante Familie sind die NXP-Crossover-Prozessoren.

Für die Signalcontroller vom Typ Cortex-M4/M7 gibt es die kostenfreie CMSIS-Routinensammlung, die man in die C-Programme einbindet. Man verliert so zwar den „Kontakt“ zum eigentlichen Prozessor und auch die Code-Effizienz sinkt, dennoch aber gelingt vor allem wenig erfahrenen Programmierern nach dem Nehmen der Hürde CMSIS der schnelle Einstieg in die Signalverarbeitung.

## Beispiel, Offlineverarbeitung mit Python

- \* Eine Zweikanal-Wavedatei wird eingelesen und verarbeitet
- \* Zwei digitale Tiefpassfilter vom Grad 2 filtern ausgewählte Kanäle
- \* Das verarbeitete Signal kann betrachtet werden, auf das Soundsystem des PC geschrieben werden um z.B. mit einem Kopfhörer abgehört zu werden und/oder in eine Wavedatei geschrieben werden, die dann auch zeitversetzt genutzt werden kann
- \* Python-Programme laufen sowohl auf Windows-PCs als auch auf Linux-PCs. Unterschiede sind nur bzgl. Der Verwendung des jeweiligen Soundsystems zu beachten
- \* Für manche Mikrocontroller ist mittlerweile auch ein abgespecktes Micropyton erhältlich. Ich bevorzuge die Programmierung in C(++) oder Assembler

## Offline-Betrieb mit Python

- (1) Import der notwendigen Libraries
- (2) Wavedatei einlesen
- (3) Audiodaten in Numpy-Array kopieren
- (4) Zerlegung in rechten und linken Kanal, unnötig bei einkanaliger Aufnahme

```
import wave
import numpy as np
import matplotlib.pyplot as plt
import sounddevice
import time

# Lies Wave-Datei ein
ifile =
wave.open("c:\\audio\\HaendelSarabandeVoicesOfMusic.wav")
NumOfChannels = ifile.getnchannels() #Anzahl der Audiokanaele
samples = ifile.getnframes()         #Anzahl der Abtastwerte
audio = ifile.readframes(samples)     #Audiosignal, Rohdaten

# Konversion nach float32 mit NumPy
audio16 = np.frombuffer(audio, dtype=np.int16)
audio32 = audio16.astype(np.float32)

# zerlegung in rechten und linken Kanal
left=audio32[0::2]
right=audio32[1::2]
```

## Offline-Betrieb mit Python

- (4) Filterkoeffizienten festlegen, m zwei Filter
- (5) Filterkoeffizienten in Numpy-Array kopieren
- (6) Zustandsspeicher Nullsetzen und Ausgangssignal vordefinieren

```
# tp, fg=1kHz @ fT=44.1kHz
b0_1k=0.004603998475022
b1_1k=0.009207996950045
b2_1k=0.004603998475022
a1_1k=-1.799096409484668
a2_1k=0.817512403384758
# tp, fg=0.4kHz @ fT=44.1kHz
b0_400=0.000780326282261
b1_400=0.001560652564521
b2_400=0.000780326282261
a1_400=-1.919444571521770
a2_400=0.922565876650812

# Filterkoeffizienten in ein Array schreiben

Koeff=np.array([b0_1k, b1_1k, b2_1k, a1_1k, a2_1k,
                b0_400, b1_400, b2_400, a1_400, a2_400])

s1l=0; s2l=0; s1r=0; s2r=0; # Zustandsspeicher Nullsetzen
yl=np.zeros(len(left))
yr=np.zeros(len(left))
```

## Offline-Betrieb mit Python

- (7) Auswahl der Filter für die Bearbeitung des linken und des rechten Kanals
- (8) Differenzengleichungen der Filter rechnen. Hierfür gibt es fertige Routinen in Python. Wir machen es aber „zu Fuß“, denn dies entspricht dann dem Vorgehen bei der Verwendung von DSPs oder Signalcontrollern, die direkt programmiert werden

Die Audio-Loop, getaktet von den Abtastwerten, die nacheinander aus der Wave-Datei gezogen werden

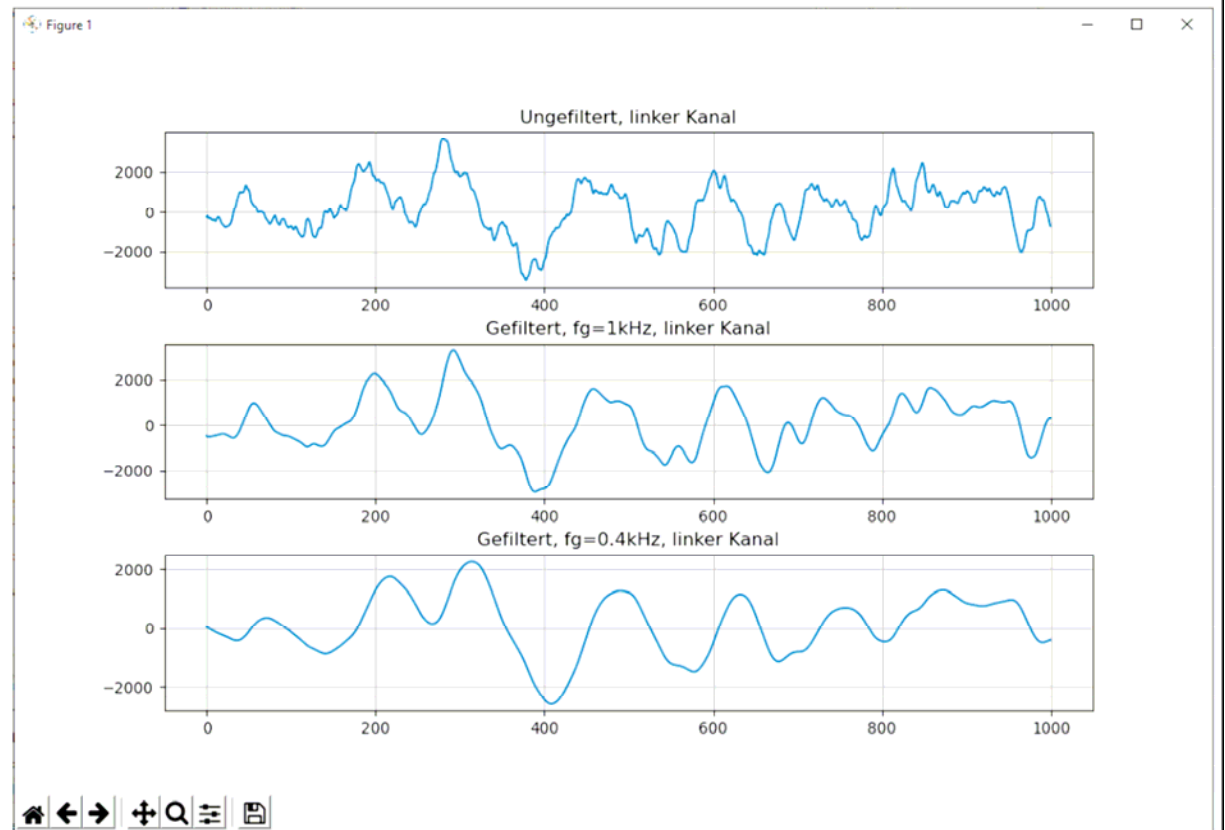
```
KoeffBaseLeft = 0    # so einfach wird ein Filter ausgewaehlt...
KoeffBaseRight = 5
for i in range(len(left)):
    w=left[i]-Koeff[KoeffBaseLeft+3]*s1l-Koeff[KoeffBaseLeft+4]*s2l
    y1[i]=w*Koeff[KoeffBaseLeft]+Koeff[KoeffBaseLeft+1]*s1l+Koeff[KoeffBaseLeft+2]*s2l
    s2l=s1l
    s1l=w

    w=left[i]-Koeff[KoeffBaseRight+3]*s1r-Koeff[KoeffBaseRight+4]*s2r
    yr[i]=w*Koeff[KoeffBaseRight]+Koeff[KoeffBaseRight+1]*s1r+Koeff[KoeffBaseRight+2]*s2r
    s2r=s1r
    s1r=w
# Zweimalige Linksfilterung um die unterschiedlichen Grenzfrequenzen zu verdeutlichen
```

## Offline-Betrieb mit Python

```
plt.subplot(311)
plt.plot(left[90000:91000])
plt.grid(True)
plt.title("Ungefiltert, linker Kanal")
plt.subplot(312)
plt.plot(y1[90000:91000])
plt.grid(True)
plt.title("Gefiltert, fg=1kHz, linker Kanal")
plt.subplot(313)
plt.plot(yr[90000:91000])
plt.grid(True)
plt.title("Gefiltert, fg=0.4kHz, linker Kanal")
plt.show()

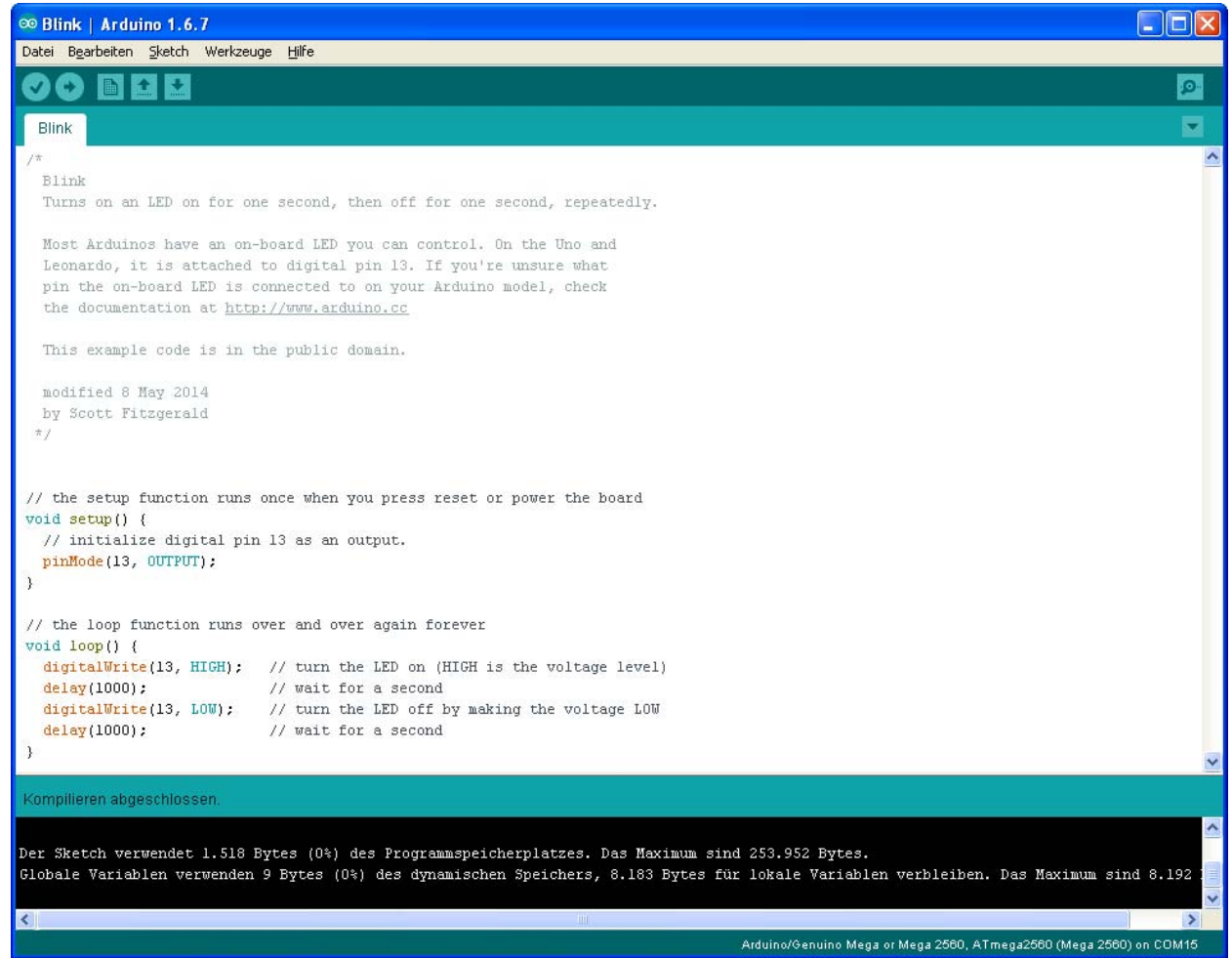
'''
ystereo = np.column_stack((y1,yr))
ystereo_normalised = ystereo / max_int16
sounddevice.play(ystereo_normalised, 44100)
'''
```



# Signalcontroller für Anfänger (der Appetit kommt beim Essen...)

## Wie geht's los?

- \* ca. 25€ investieren und ein Teensy 4.0 Board beschaffen. USB-Kabel mit Microstecker muß vorhanden sein.
- \* Kostenfreie Software unter Windows, MAC und Linux, Arduino IDE und Teensyduino installieren
- \* Blink-Programm unter Beispiele in den Editor laden, kompilieren und auf das Board via USB laden.
- \* Ggf. Audio Adaptor Board für Teensy 4.0 für ca. 15€ besorgen.
- \* Teensy Audio Library antesten, Video auf Teensy-HP erreichbar



```
Arduino IDE - Blink | Arduino 1.6.7
Datei Bearbeiten Sketch Werkzeuge Hilfe

Blink
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://www.arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
  */

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

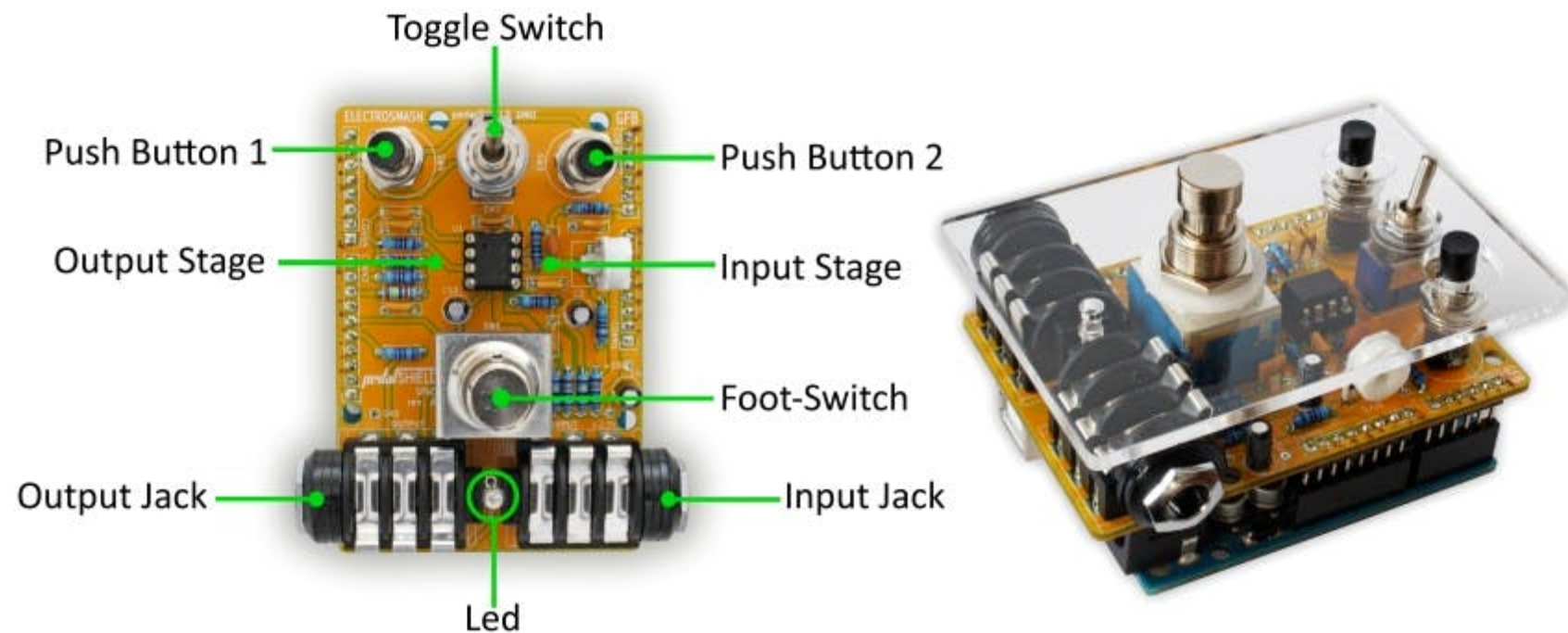
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}

Kompilieren abgeschlossen.

Der Sketch verwendet 1.518 Bytes (0%) des Programmspeicherplatzes. Das Maximum sind 253.952 Bytes.
Globale Variablen verwenden 9 Bytes (0%) des dynamischen Speichers, 8.183 Bytes für lokale Variablen verbleiben. Das Maximum sind 8.192 Bytes.

Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM15
```

[https://create.arduino.cc/projecthub/electrosmash/arduino-uno-guitar-pedal-b2ba96?ref=tag&ref\\_id=guitar&offset=0](https://create.arduino.cc/projecthub/electrosmash/arduino-uno-guitar-pedal-b2ba96?ref=tag&ref_id=guitar&offset=0)



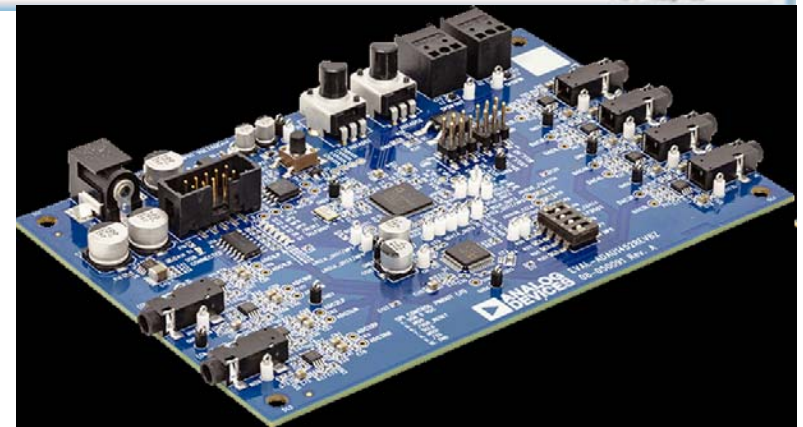
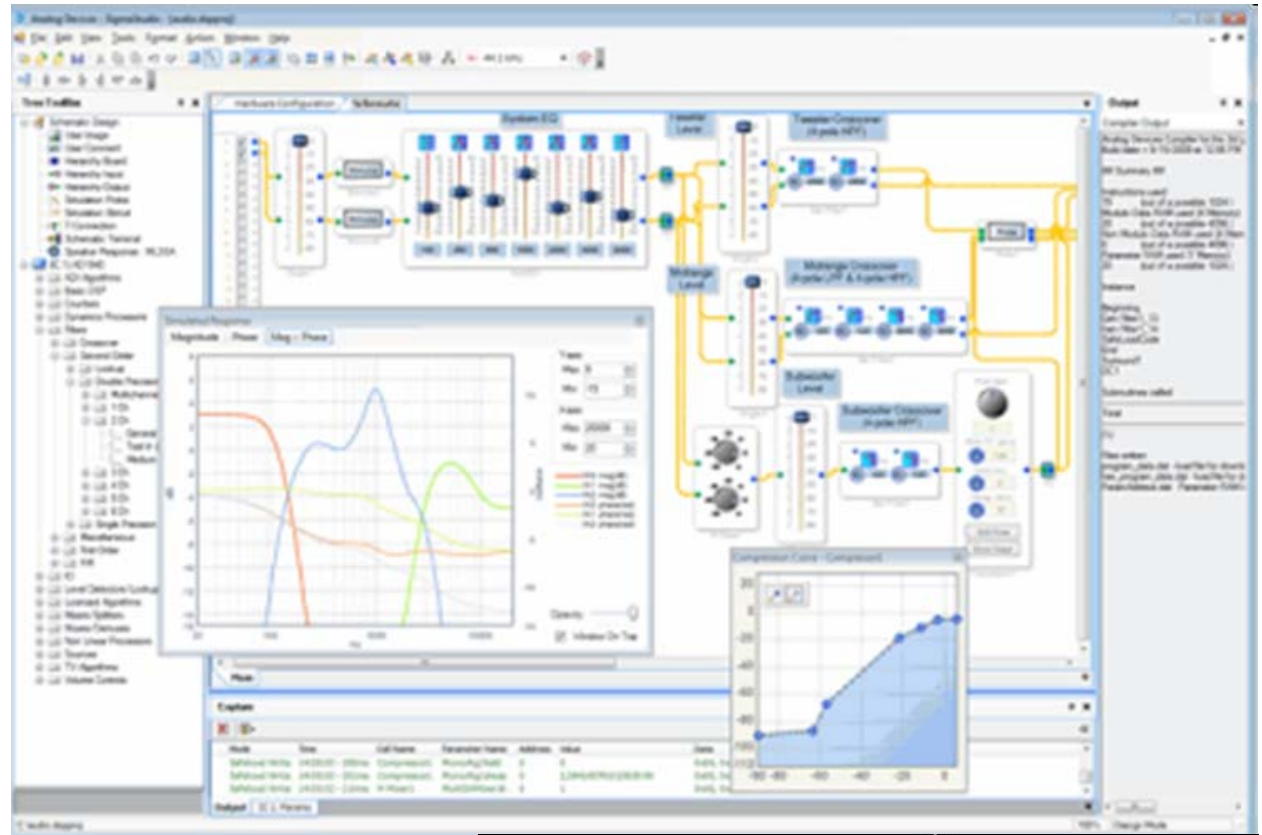
## Signalprozessor für Anfänger (der Appetit kommt beim Essen...)

### Wie geht's los?

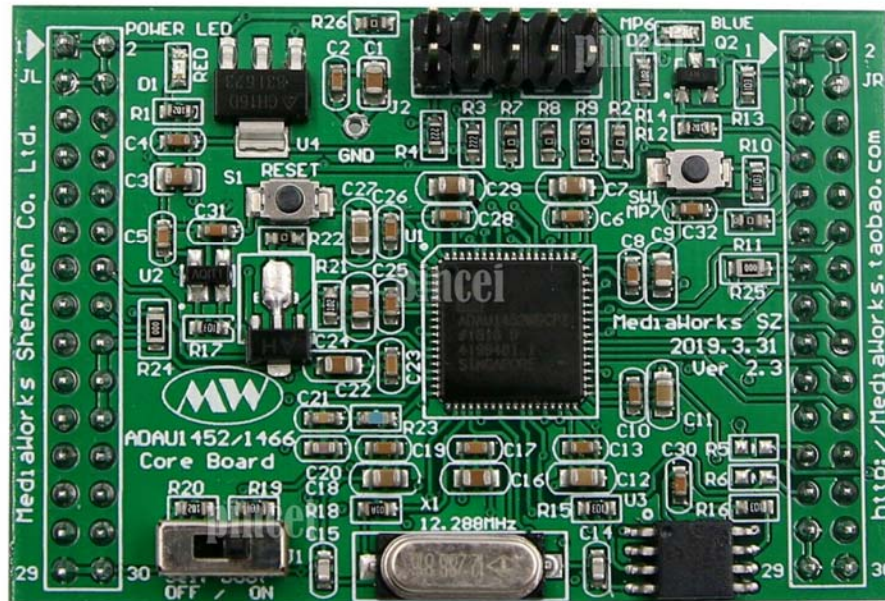
- \* ca. 200€ investieren und ein EVAL-ADAU1452RevBZ besorgen. Spannungsversorgung und Programmieradapter USB/I<sup>2</sup>C liegen bei.
- \* Über Ebay vom Chinamann ein Low-Budget-Board besorgen. Programmieradapter nicht vergessen.
- \* Sigma-Studio Software herunterladen (kostenfrei)
- \* Youtube-Video, z.B.

<https://www.youtube.com/watch?v=abl1r4i8hHI>

anschauen.

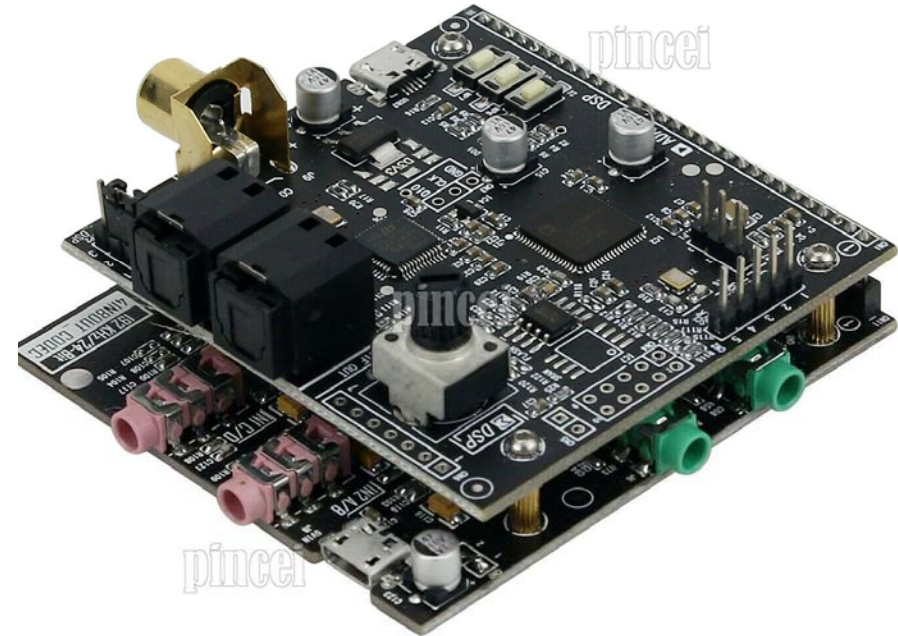


## Online-Signalverarbeitung mit einem Signalprozessor vom Typ Sigma-DSP



DSP-Board mit ADAU1452, billiges  
China-Board, erhältlich über ebay

Ohne Analogschnittstelle



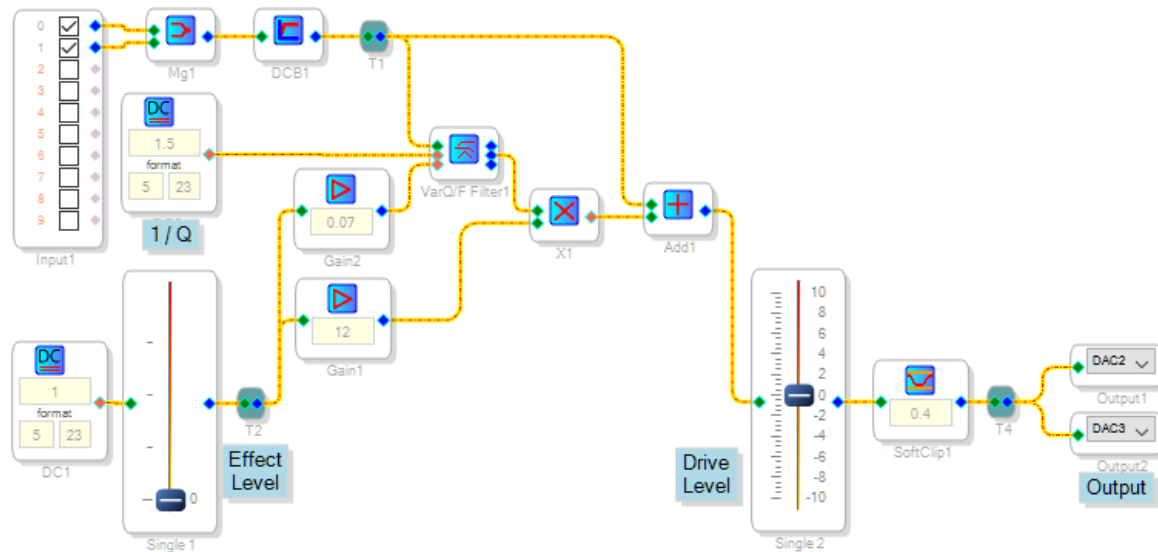
DSP-Board mit ADAU1452, billiges  
China-Board, erhältlich über ebay

Mit Analogschnittstelle Codec AD1938  
und Microcontroller

## Online-Signalverarbeitung mit einem Signalprozessor vom Typ Sigma-DSP

Tube Screamer mit Sigma-DSP, Analog Devices

<https://ez.analog.com/dsp/sigmadsp/f/q-a/65279/tubescreamer-guitar-effect-on-aida-dsp>



Wie geht's weiter?

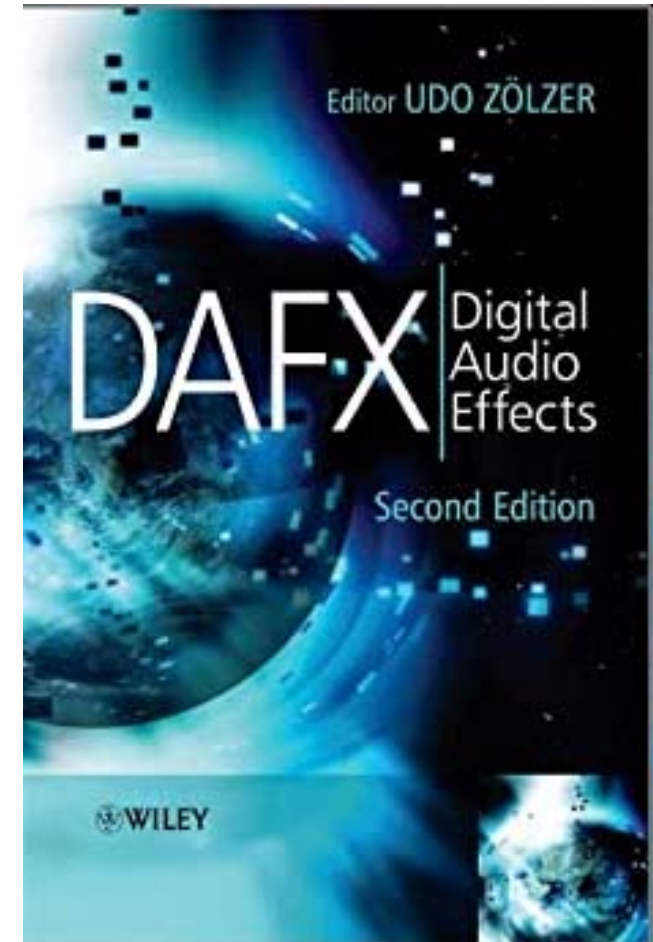
Wissen ist Macht



The rapid development in various fields of Digital Audio Effects, or DAFX, has led to new algorithms and this second edition of the popular book, DAFX: Digital Audio Effects has been updated throughout to reflect progress in the field. It maintains a unique approach to DAFX with a lecture-style introduction into the basics of effect processing. Each effect description begins with the presentation of the physical and acoustical phenomena, an explanation of the signal processing techniques to achieve the effect, followed by a discussion of musical applications and the control of effect parameters. Topics covered include: filters and delays, modulators and demodulators, nonlinear processing, spatial effects, time-segment processing, time-frequency processing, source-filter processing, spectral processing, time and frequency warping musical signals.

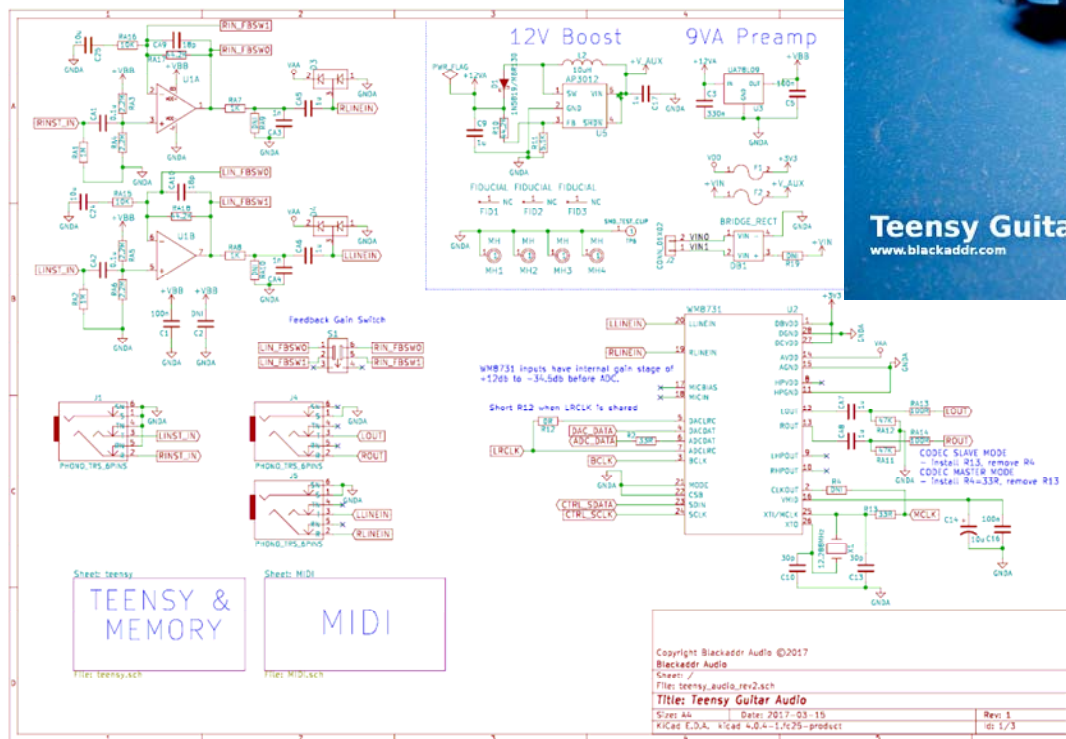
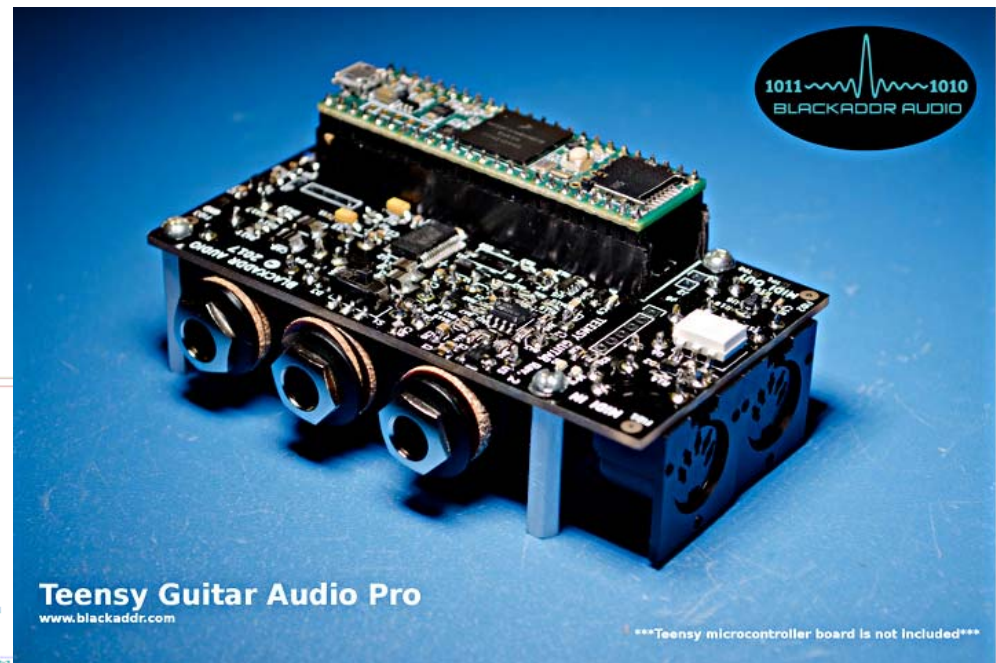
Updates to the second edition include:

- \* Three completely new chapters devoted to the major research areas of: Virtual Analog Effects, Automatic Mixing and Sound Source Separation, authored by leading researchers in the field .
- \* Improved presentation of the basic concepts and explanation of the related technology.
- \* Extended coverage of the MATLAB(TM) scripts which demonstrate the implementation of the basic concepts into software programs.
- \* Companion website ([www.dafx.de](http://www.dafx.de)) which serves as the download source for MATLAB(TM) scripts, will be updated to reflect the new material in the book.



## Wie geht's weiter?

Suche im Internet, wo z.B. dieses AddOn zum Teensy 4.1 zu finden ist (mit Ethernet!). Der Autor vertreibt auch das Audioshield.



<https://www.tindie.com/products/Blackaddr/arduino-teensy-guitar-audio-shield/>

## Wie geht's weiter?

Mach es selbst ...

Do it yourself...

fais le toi-même...

fallo da solo...

gör det själv...

hazlo tu mismo...

Faça Você Mesmo...

doe het zelf...