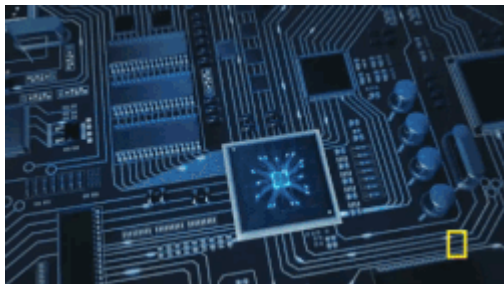


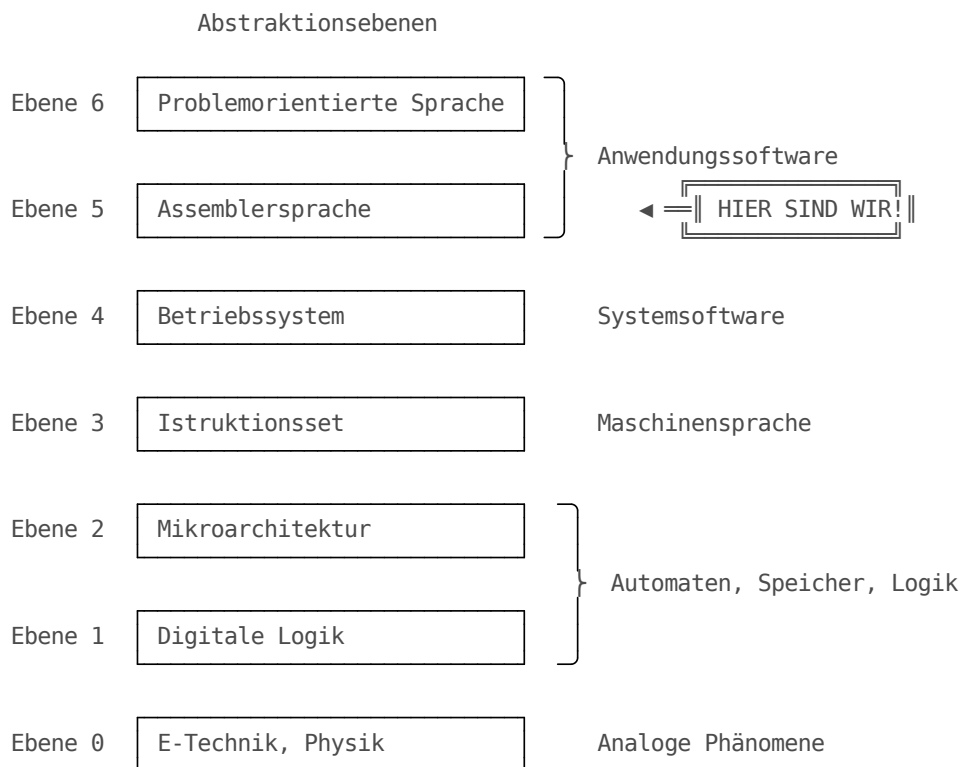
# Analog-Digital-Wandler

Parameter	Kursinformationen
Veranstaltung:	Digitale Systeme / Eingebettete Systeme
Semester:	Wintersemester 2025/26
Hochschule:	Technische Universität Freiberg
Inhalte:	Grundlagen und Verwendung des Analog-Digital-Wandlers
Link auf GitHub:	<a href="https://github.com/TUBAF-lfl-LiaScript/VL_EingebetteteSysteme/blob/master/14_ADC.md">https://github.com/TUBAF-lfl-LiaScript/VL_EingebetteteSysteme/blob/master/14_ADC.md</a>
Autoren:	Sebastian Zug & André Dietrich & Fabian Bär



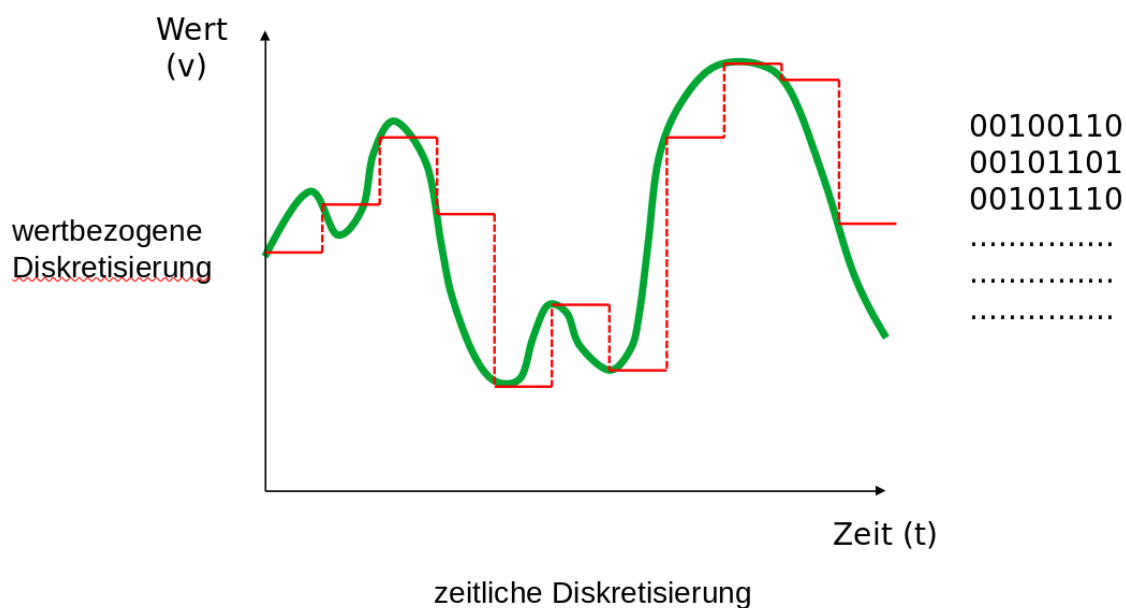
## Fragen an die Veranstaltung

- Was bedeutet der Begriff Auflösung im Zusammenhang mit einem Analog-Digital-Wandler?
- Welche Wandlungsmethoden kennen Sie, worin unterscheiden sich diese?
- Welches Wandlungskonzept ist im AVR-Core implementiert?
- Welche Störungen entstehen ggf. bei der Analog-Digital-Wandlung?
- Wovon hängt die Wandlungsdauer beim AVR ab? Durch welche Parameter können Sie diese beeinflussen?



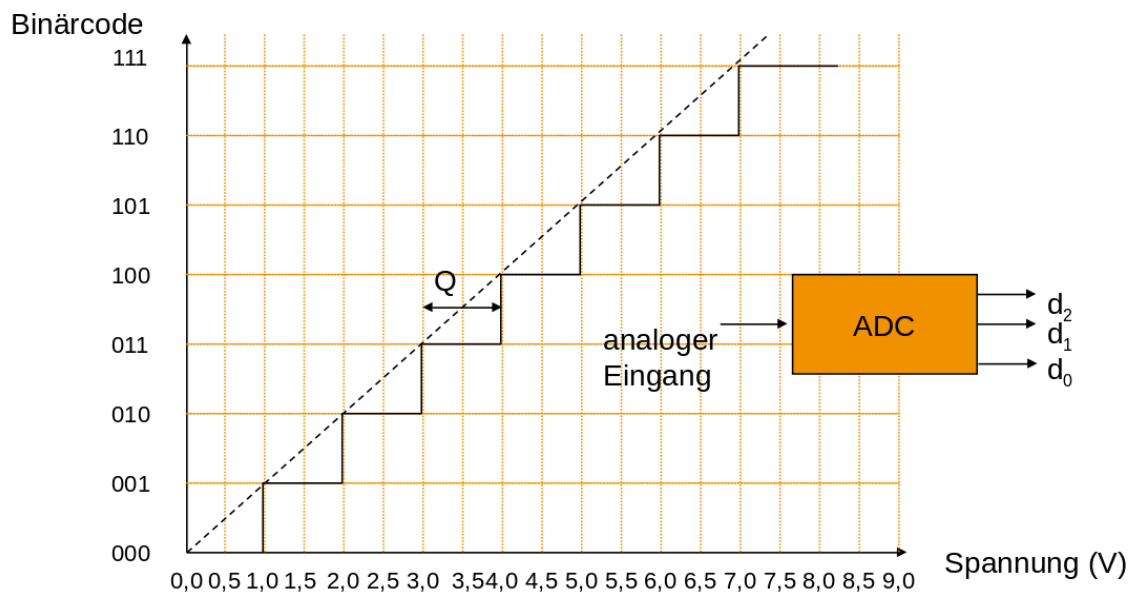
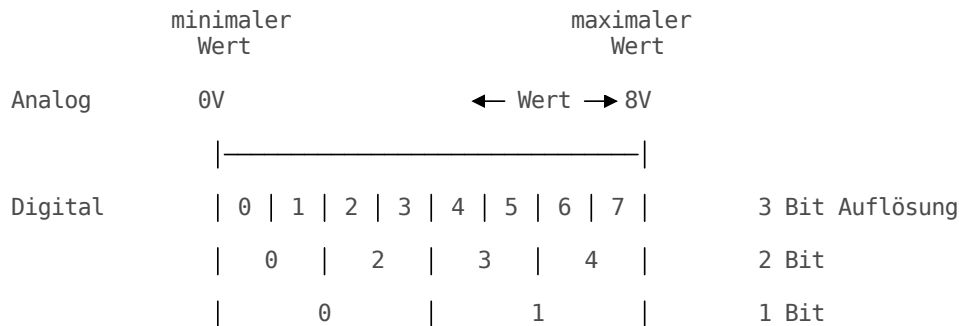
## Motivation

In der vorangegangenen Vorlesung sprachen wir insbesondere über die Erfassung von digitalen Signalen. Eine Erfassung von analogen Werten ist allerdings notwendig, um Phänomene der Umgebung mit dem notwendigen Detailgrad beobachten zu können.



Dabei wird das zeit- und wertkontinuierliche Eingangssignal in eine zeit- und wertdiskrete Darstellung überführt.

Die Idee besteht darin einem Spannungswert einer Zahlenrepräsentation zuzuordnen, die einen Indexwert innerhalb eines beschränkten Spannungswert repräsentiert.



Daraus ergibt sich die zentrale Gleichung für die Interpretation des Ausgaben eines Analog-Digital-Wandlers

$$ADC = \frac{V_{in}}{V_{ref}} ADC_{res}$$

oder

$$\frac{ADC \cdot V_{ref}}{ADC_{res}} = V_{in}$$

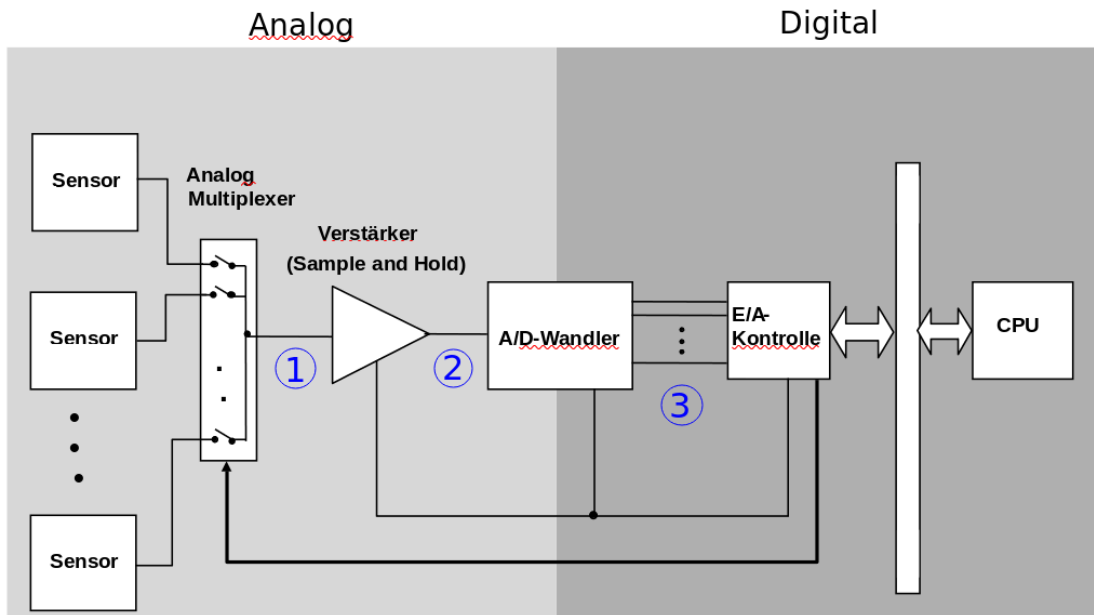
Für unser Beispiel aus der Grafik zuvor bedeutet die Ausgabe von "5" bei einem 3-Bit-Wandler, also

$$\frac{5 \cdot 8V}{2^3} = 5V$$

Der potentielle Quantisierungsfehler  $Q$  beträgt

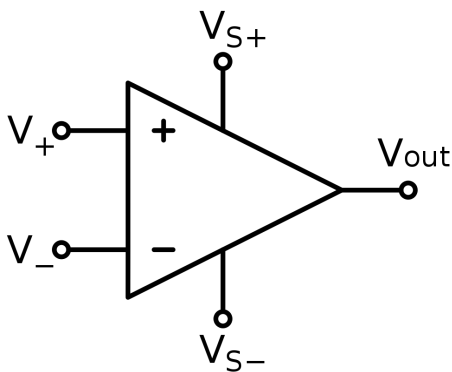
$$Q = \frac{8V}{2^3} = 1V$$

Dabei erfolgt die Wandlung in zwei generellen Schritten. Zunächst wird das Signal zeitlich diskretisiert und darauffolgend wertbezogenen gewandelt.

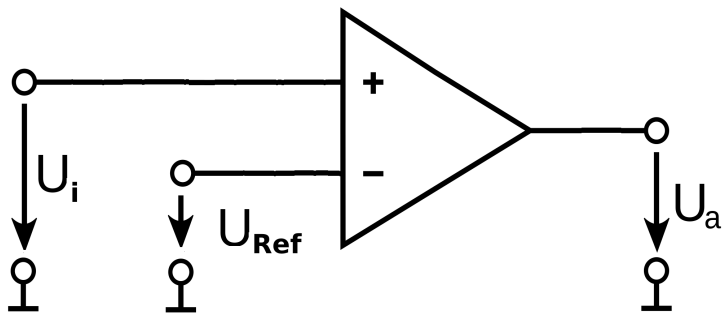


## Analog Komparator

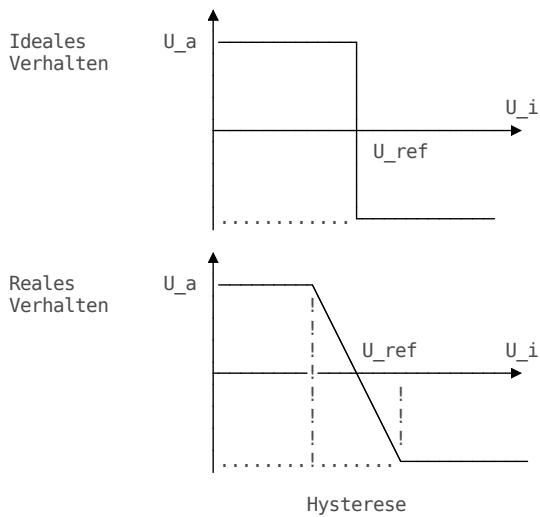
Ein Komparator ist eine elektronische Schaltung, die zwei Spannungen vergleicht. Der Ausgang zeigt in binärer/digitaler Form an, welche der beiden Eingangsspannungen höher ist. Damit handelt es sich praktisch um einen 1-Bit-Analog-Digital-Umsetzer.



Comparator Symbol [\[OPAMP\]](#)

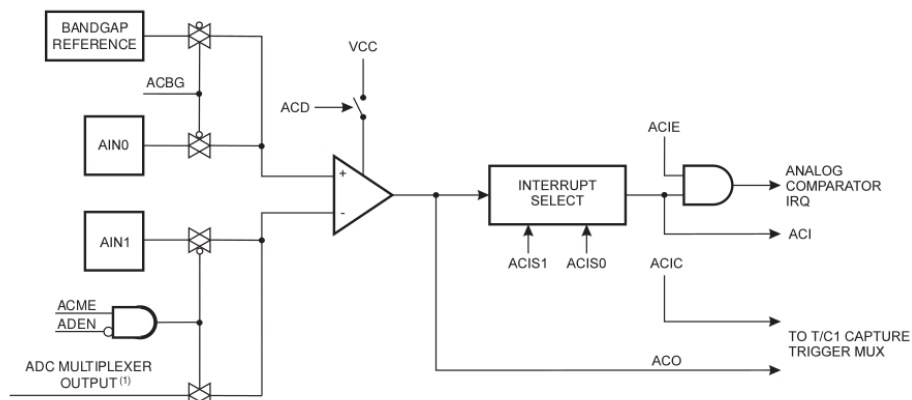


Comparator Funktionsprinzip [OPAMP]



Im AVR findet sich ein Komperator, der unterschiedliche Eingänge miteinander vergleichen kann: Für "+" sind dies die **BANDGAP Reference** und der Eingang **AIN0** und für "-" der Pin **AIN1** sowie alle analogen Eingänge.

Figure 23-1. Analog Comparator Block Diagram<sup>(2)</sup>



Analog Comperator, Seite 243 [megaAVR]

Die grundlegende Konfiguration erfolgt über die Konfiguration der Bits / Register :

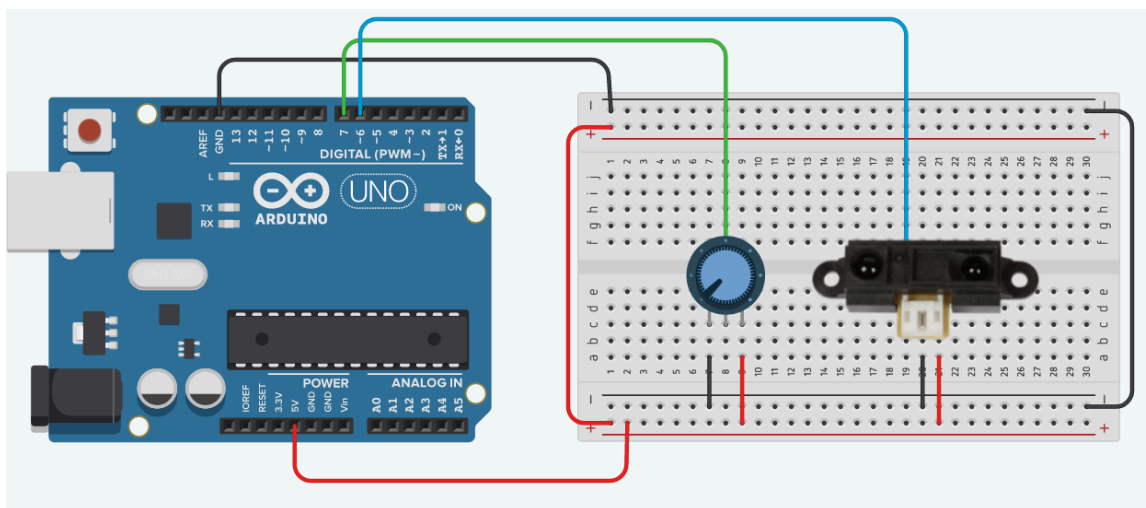
Bits	Register	Bedeutung
ACBG	ACSR	Analog Comparator Bandgap Select
ACME	ADCSRB	Analog Comparator Multiplexer Enable - Bit im Register
ADEN	ADCSRA	Analog Digital Enable
MUX2, MUX1, MUX0	ADMUX	Multiplexer Analog input

Dazu kommen noch weitere Parameterisierungen bezüglich der Interrupts, der Aktivierung von Timerfunktionalität oder der Synchronisierung.

Weitere Erläuterungen finden Sie im Handbuch auf Seite

**Aufgabe:** An welchen Pins eines Arduino Uno Boards müssen Analoge Eingänge angeschlossen werden, um die zwei Signale mit dem Komperator zu vergleichen. Nutzen Sie den Belegungsplan (Schematics) des Kontrollers, der unter [Link](#) zu finden ist.

Ein Beispiel für den Vergleich eines Infrarot Distanzsensors mit einem fest vorgegebenen Spannungswert findet sich im *Example* Ordner der Veranstaltung.



```
#define F_CPU 16000000UL
#include <avr/io.h>

int main()
{
    ADCSRB = (1<<ACME);
    // ...
}
```



```

DDRB = (1<<PB5);

while(1)
{
    if (ACSR & (1<<ACO)) /* Check ACO bit of ACSR register */
        PORTB &= ~(1 << PB5); /* Then turn OFF PB5 pin */
    else /* If ACO bit is zero */
        PORTB = (1<<PB5); /* Turn ON PB5 pin */
}
}

```

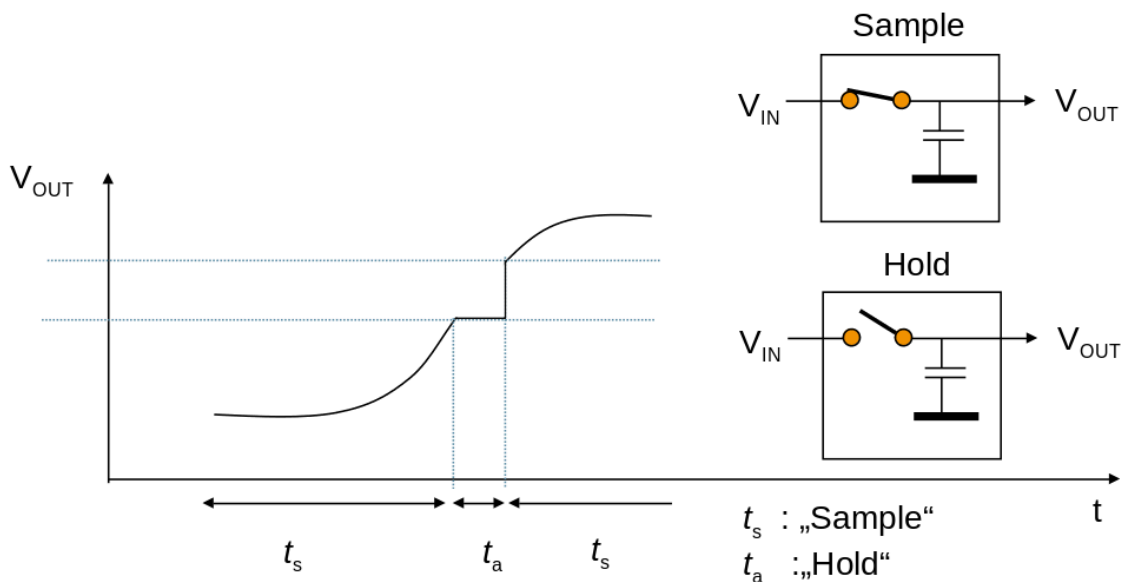
**Hinweis** Um der Diskussion der Interrupts nicht vorzugreifen wurde an dieser Stelle per Polling auf das Ergebnis des Comparators zugegriffen. Üblicherweise würde diese mit einer Interrupt Service Routine (ISR) realisiert.

[OPAMP] Wikipedia, Autor Omegatron - Eigenes Werk, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=983276>

[megaAVR] Firma Microchip, megaAVR® Data Sheet, [Link](#)

## Analog Digital Wandler

Voraussetzung für den Wandlungsprozess ist die Sequenzierung des Signals. Mit einer spezifischen Taktrate wird das kontinuierliche Signal erfasst.



Wie sollte die Taktrate für die Messung denn gewählt werden?



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Generierung der "Realen Messungen"
5 f = 40 # Hz
6 tmin = -0.3
7 tmax = 0.3
8 t = np.linspace(tmin, tmax, 400)
9 s = np.cos(2*np.pi*f*t)
10
11 # Abtastung mit einer Frequenz kleiner der Grenzfrequenz
12 T = 1/401.0
13 nmin = np.ceil((tmin) / T)
14 nmax = np.floor(tmax / T)
15 n = np.arange(nmin, nmax) * T
16 y = np.cos(2*np.pi*f*n)
17
18 # Fitting eines Cosinussignals anhand der Messungen
19 mfft=np.fft.fft(y)
20 imax=np.argmax(np.absolute(mfft))
21 mask=np.zeros_like(mfft)
22 mask[[imax]]=1
23 mfft*=mask
24 fdata=np.fft.ifft(mfft)
25
26 # Ausgabe
27 fig, ax = plt.subplots()
28 ax.plot(t, s)
29 ax.plot(n, y, '.', markersize=8)
30 ax.plot(n, fdata, '--', color='red', label=f"Estimated Signal")
31 ax.grid(True, linestyle='-.')
32 ax.tick_params(labelcolor='r', labelsz='medium', width=3)
33 ax.set_xlim([-0.15, 0.15])
34
35 #plt.show()
36 plt.savefig('foo.png')
```

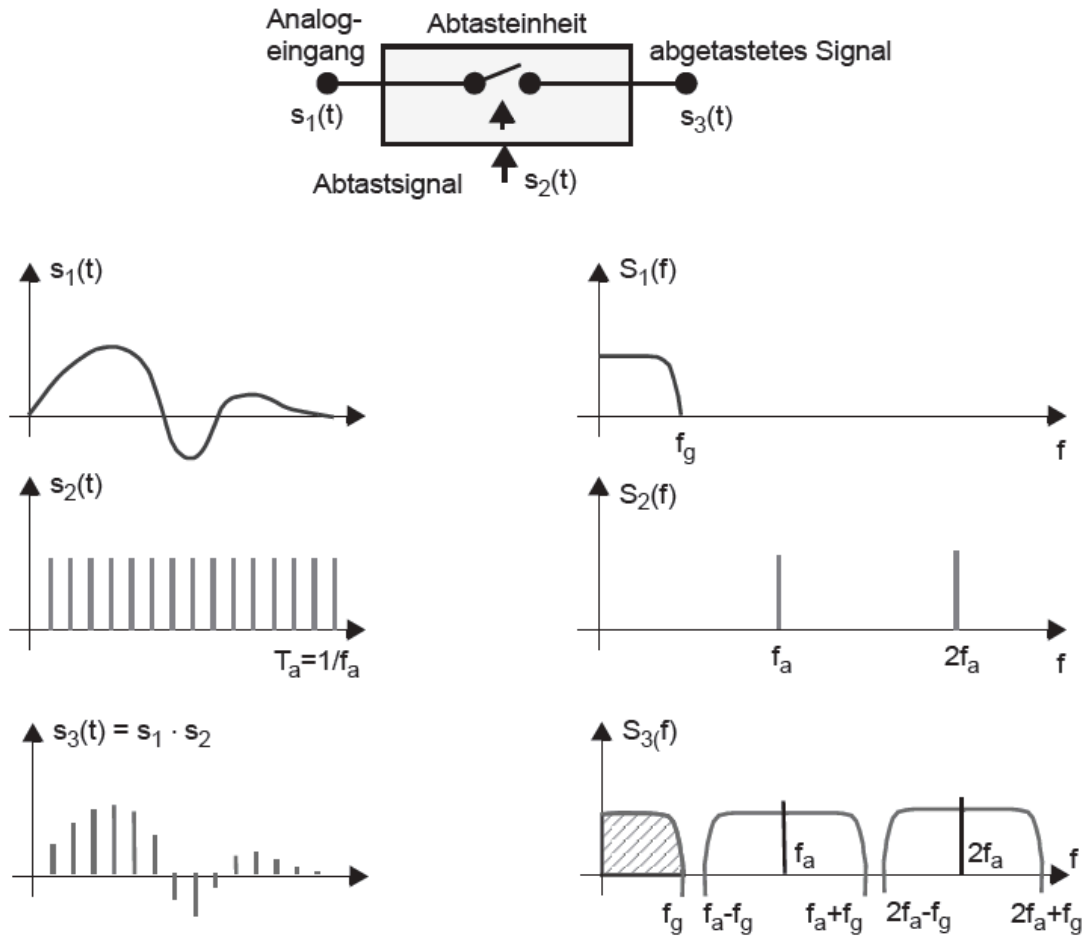
```
Waking up execution server ...
This may take up to 30 seconds ...
Please be patient ...
.....
```

s Wenn ein kontinuierliches Signal, das keine Frequenzkomponenten hat, die über einer Frequenz  $f_c$  liegen mit einer Häufigkeit von größer  $2f_c$  abgetastet wird, kann das Originalsignal aus den gewonnenen Punkten unverzerrt rekonstruiert werden.



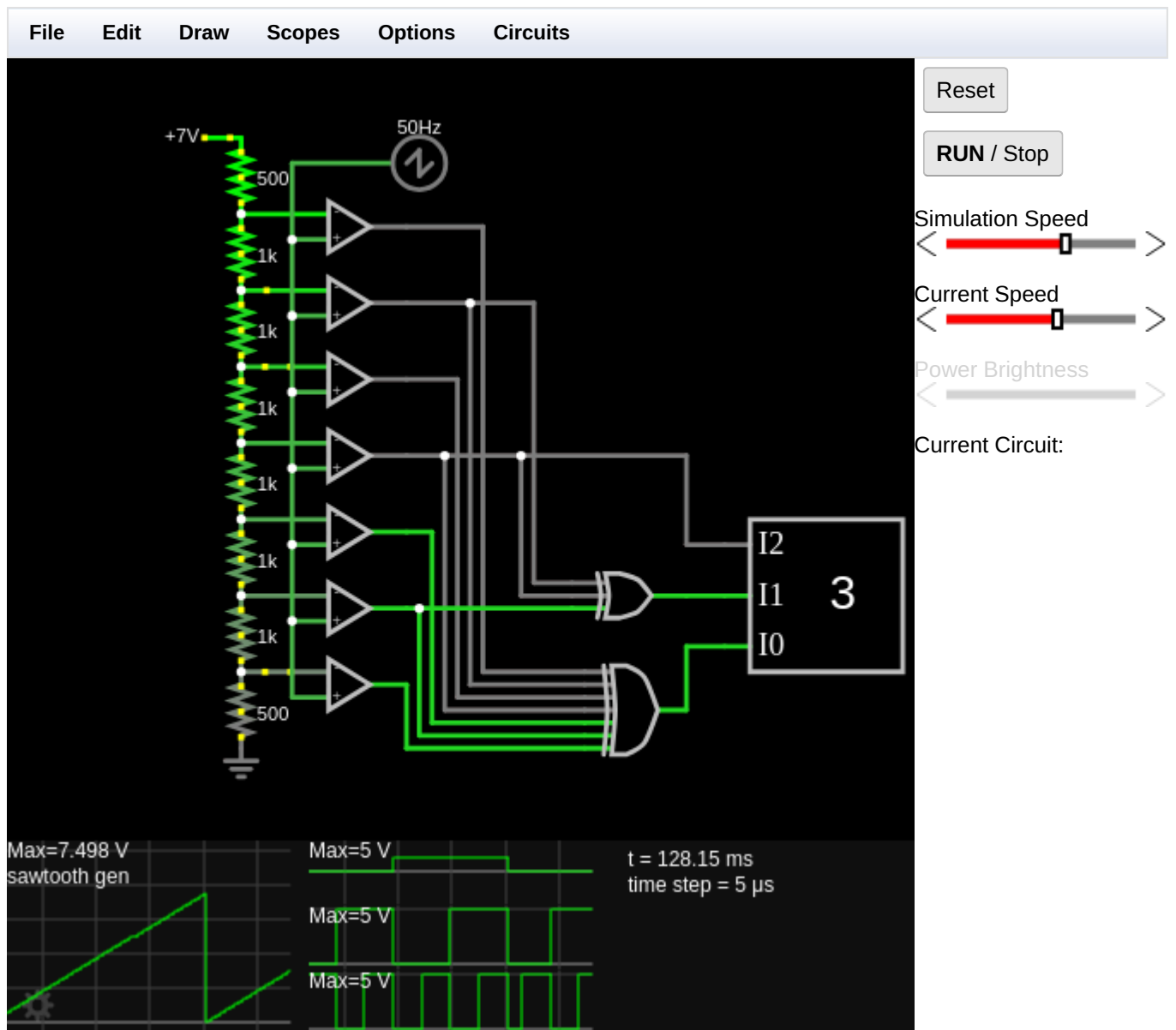
Falls dieses Kriterium nicht eingehalten wird, entstehen nichtlineare Verzerrungen, die auch als Alias-Effekt bezeichnet werden (vgl. Python Beispiel). Die untere Grenze für eine Alias-freie Abtastung wird auch als *Nyquist-Rate* bezeichnet.

Als Erklärung kann eine Darstellung des Signals im Frequenzbereich dienen:

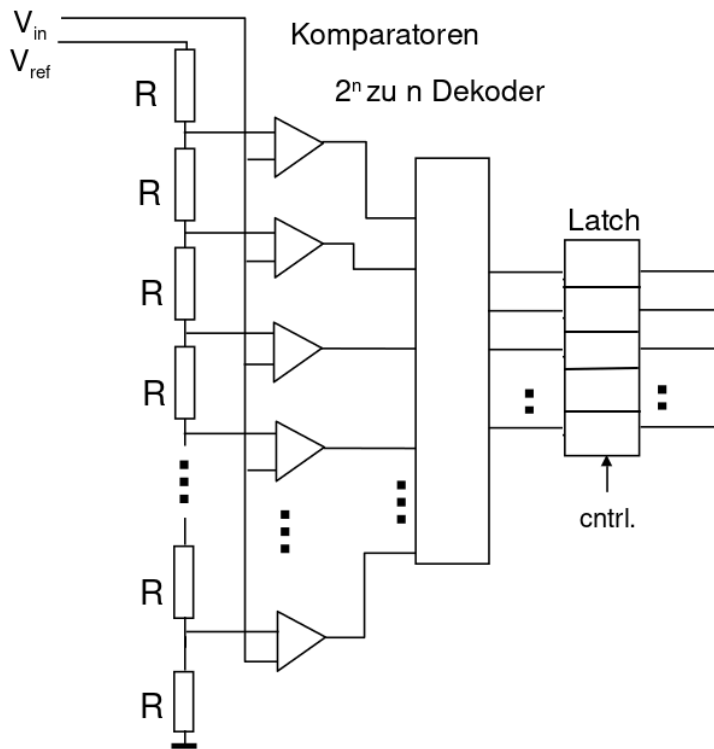


## Flashwandler

3 Bit Flashwandler



Verallgemeinerte Darstellung



Vorteil

- Hohe Geschwindigkeit

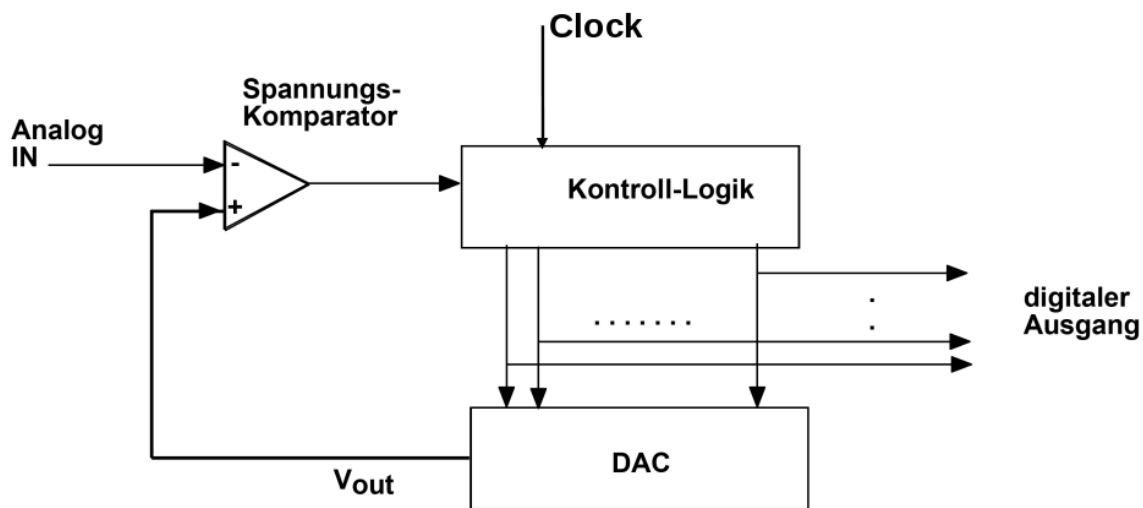
Nachteil

- Energieverbrauch größer
- Hardwareaufwand für höhere Auflösungen

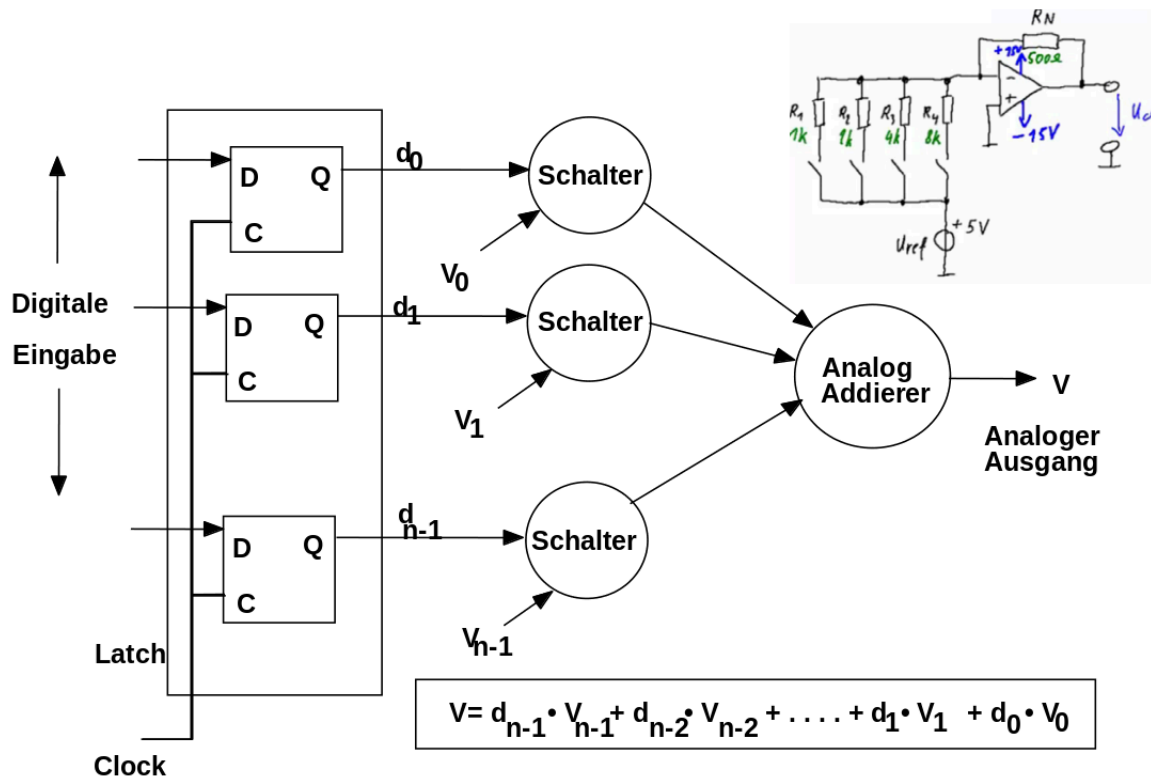
<https://www.youtube.com/watch?v=x7oPVWLD59Y>

## Sequenzielle Wandler

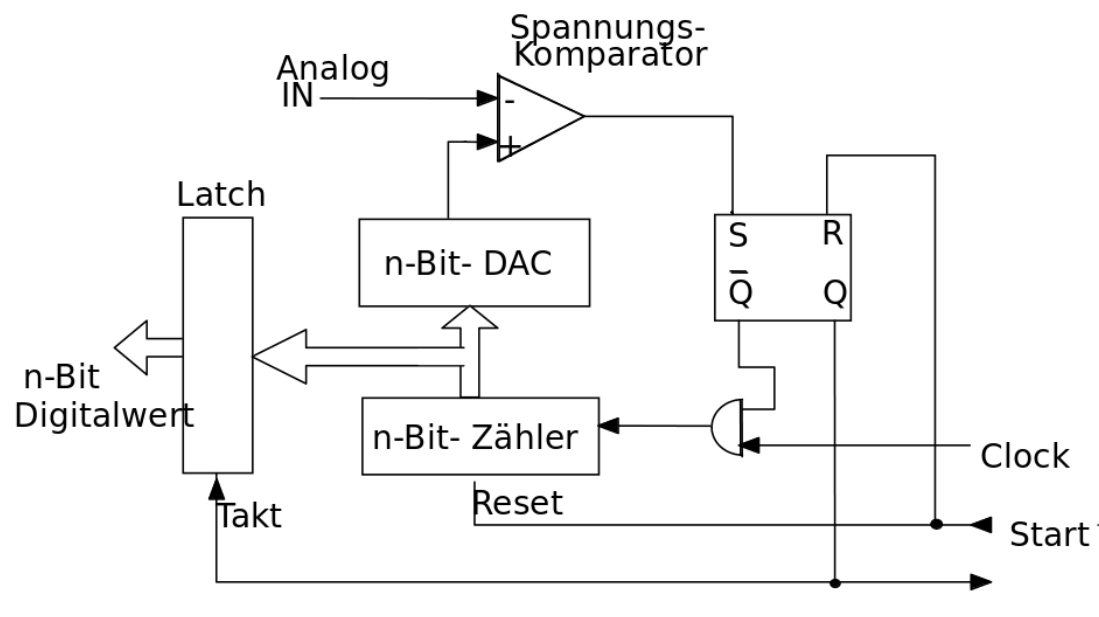
Sequenzielle Wandler umgehen die Notwendigkeit mehrerer Komperatoren, in dem das Referenzsignal variabel erzeugt wird.



Dafür ist allerdings ein Digital-Analog-Wandler nötig, der die Vergleichsspannung ausgehend von einer digitalen Konfiguration vornimmt.



Zählverfahren



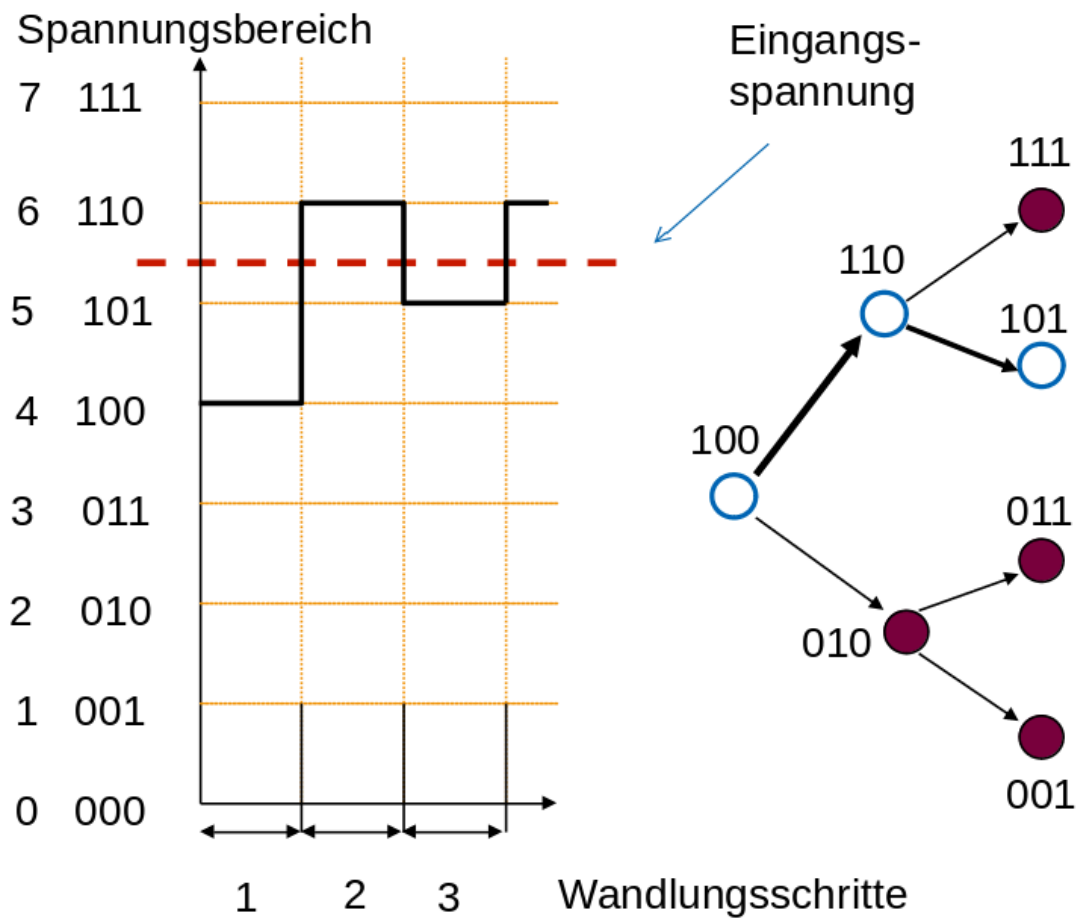
#### Vorteil

- sehr hohe Auflösungen möglich
- Schaltung einfach umsetzbar – kritisches Element DAC/Komperator

#### Nachteil

- Variierende Wandlungsdauer
- langsam (verglichen mit dem Flashwandler)

#### Sukzessive Approximation/Wägeverfahren

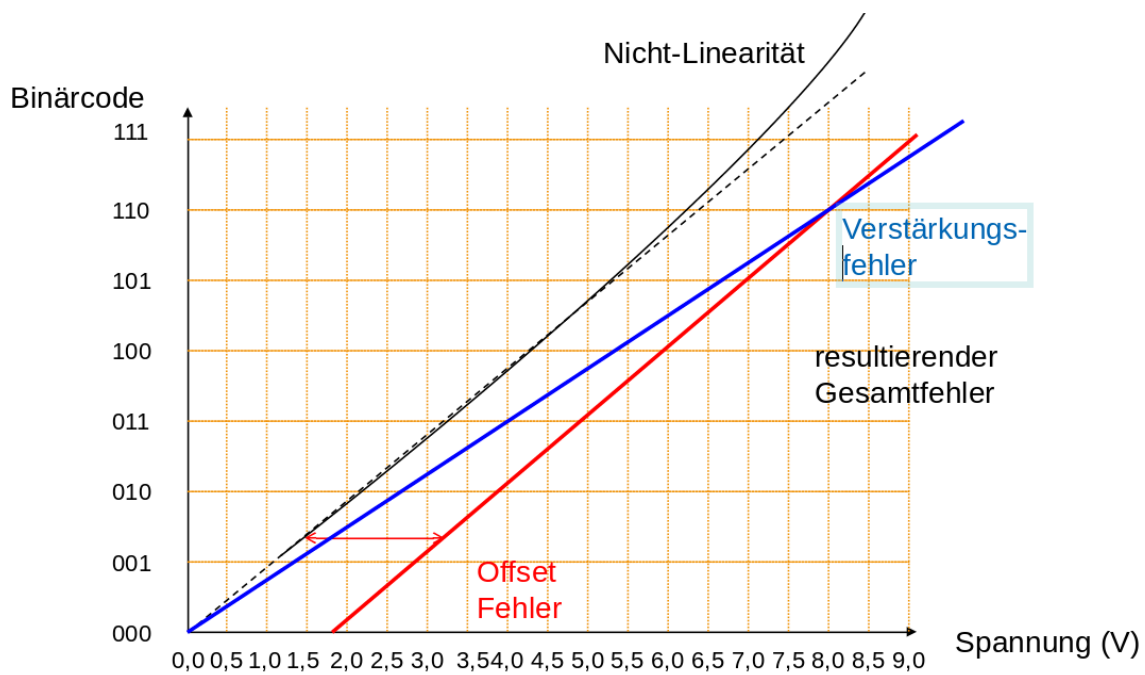


Vorteil

- Gleiche Wandlungsdauer

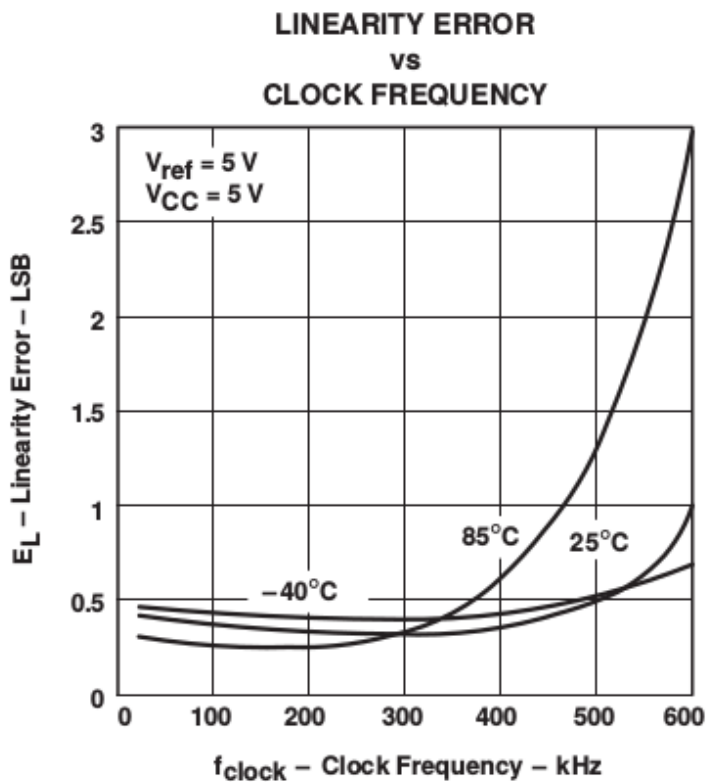
## Herausforderungen bei der Wandlung

Fehlertypen



- Quantisierungsfehler sind bedingt durch die Auflösung des Wandlers
- Offsetfehler ergeben sich aus einer Abweichung der Referenzspannung und führen zu einem konstanten Fehler.
- Verstärkungsfehler im Analog-Digitalwandler wirken einen wertabhängigen Fehler.
- Der Linearitätsfehler ist die Abweichung von der Geraden. Linearitätsfehler lassen sich nicht abgleichen.

**Merke:** Die Fehlerparameter hängen in starkem Maße von der Konfiguration des Wandlers (Sample Frequenz, Arbeitsbreite, Umgebungstemperatur) ab!



Datenblatt AD-Wandler 8 Bit DIL-8, TLC0831, TLC0831IP [\[Texas\\_TLC\]](#)

### Referenzspannung

Eine Herausforderung liegt in der stabilen Bereitstellung der Referenzspannung für den Analog-Digital-Wandler.

---

[Texas\_TLC] Firma Texas Instruments, Datenblatt AD-Wandler 8 Bit DIL-8, TLC0831, TLC0831IP

## Parameter eines Analog-Digital-Wandlers

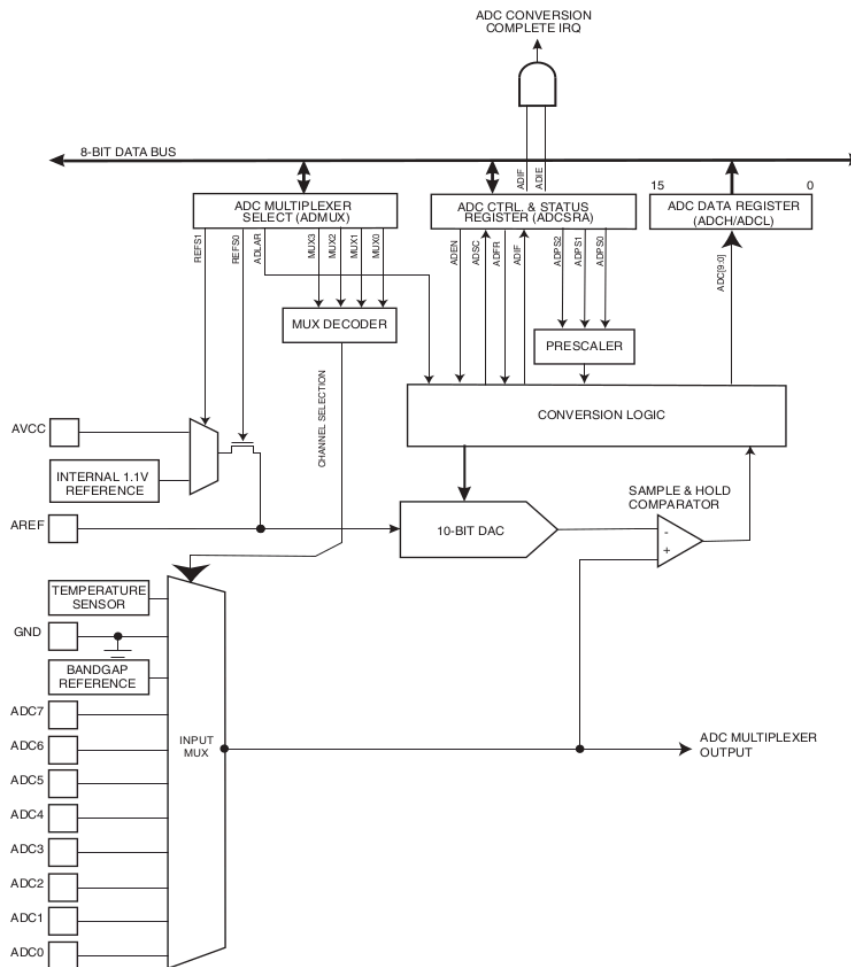
- Auflösung
- Messdauer
- Leistungsaufnahme
- Stabilität der Referenzspannung
- Unipolare/Bipolare Messungen
- Zahl der Eingänge
- Ausgangsinterfaces (parallele Pins, Bus)
- Temperaturabhängigkeit und Rauschverhalten (Gain, Nicht-Linearität, Offset)

## Umsetzung im AVR



Handbuch des Atmega328p	Bedeutung
10-Bit Auflösung	
0.5 LSB Integral Non-Linearity	maximale Abweichung zwischen der idealen und der eigentlichen analogen Signalverlauf am Wandler
+/- 2 LSB Absolute Genauigkeit	Summe der Fehler inklusive Quantisierungsfehler, Offset Fehler etc. (worst case Situation)
13 - 260µs Conversion Time	Die Dauer der Wandlung hängt von der Auflösung und der der vorgegebenen Taktrate ab.
Up to 76.9kSPS (Up to 15kSPS at Maximum Resolution)	
0 - V <sub>CC</sub> ADC Input Voltage Range	Es sind keine negativen Spannungen möglich.
Temperature Sensor Input Channel	
Sleep Mode Noise Canceler	Reduzierung des Streuquellen durch einen "Sleepmode" für die CPU

**Figure 24-1. Analog to Digital Converter Block Schematic Operation**



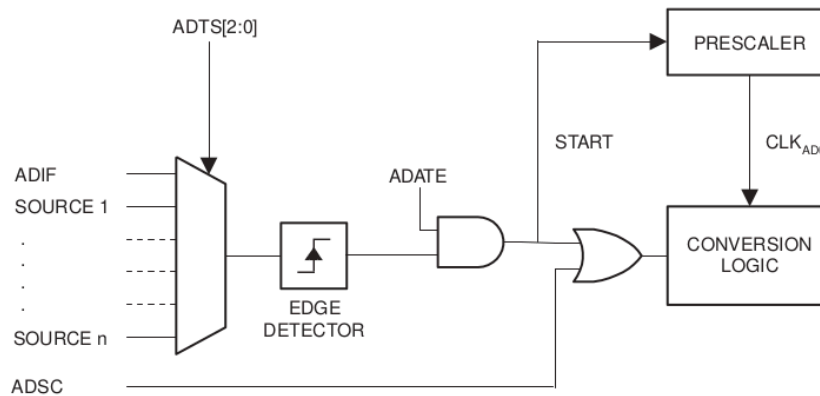
megaAVR® Data Sheet, Seite 247 [\[megaAVR\]](#)

## Trigger für den Wandlung

Grundsätzlich sind 3 Modi für die Wandlung möglich:

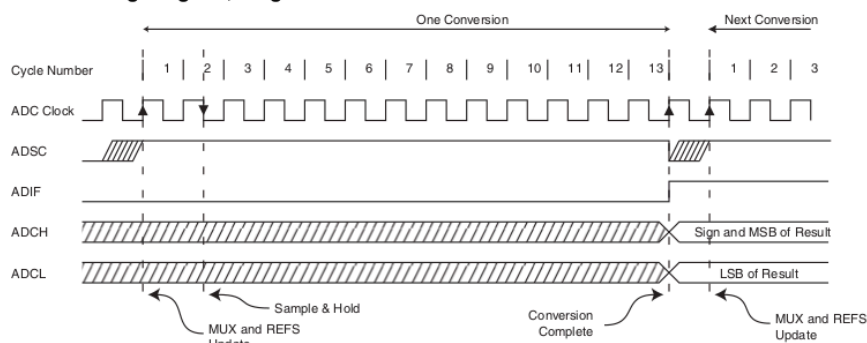
- Programmgetriggerte Ausführung der Wandlung
- Kontinuierliche Wandlung
- ereignisgetriebener Start

Figure 24-2. ADC Auto Trigger Logic



megaAVR® Data Sheet, Seite 247 [\[megaAVR\]](#)

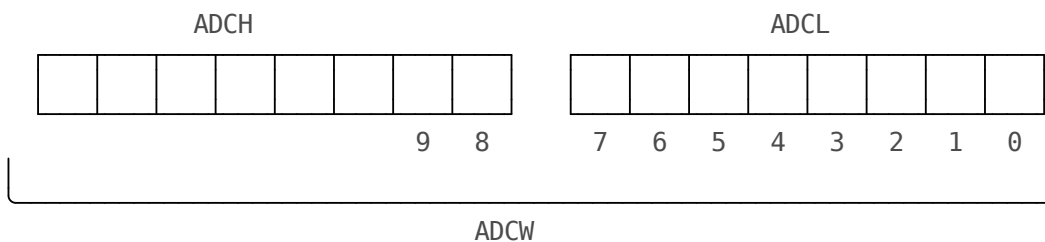
Figure 24-5. ADC Timing Diagram, Single Conversion



megaAVR® Data Sheet, Seite 250 [\[megaAVR\]](#)

## Ergebnisregister

Die Atmega Prozessoren bieten eine Auflösung von 10Bit oder 8Bit für die analogen Wandlungen. Entsprechend stehen zwei Register `ADCL` und `ADCH` für die Speicherung bereit. Standardmäßig (d.h. `ADLAR == 0`) werden die niederwertigsten 8 im Register `ADCL` bereitgehalten und die zwei höherwertigsten im Register `ADCH`.

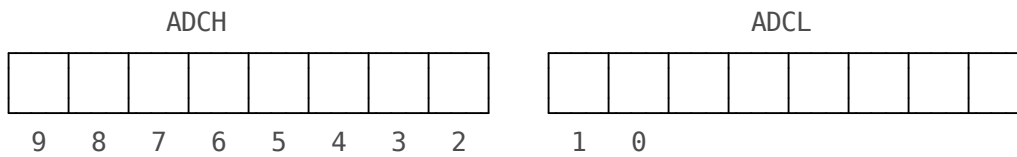


Das Ergebnis ergibt sich dann zu

```
uint8_t theLowADC = ADCL
uint16_t theTenBitResults = ADCH<<8 | theLowADC;
```



Ist keine 10-bit Genauigkeit erforderlich, wird diese Zuordnung durch das Setzen des **ADLAR** Bits im **ADMUX** Register angepasst. Auf diese Weise kann das ADC Ergebnis direkt als 8 Bit Zahl aus **ADCH** ausgelesen werden.



**Merke:** Immer zuerst ADCL und erst dann ADCH auslesen.

Beim Zugriff auf ADCL wird das ADCH Register gegenüber Veränderungen vom ADC gesperrt. Erst beim nächsten Auslesen des ADCH-Registers wird diese Sperre wieder aufgehoben. Dadurch ist sichergestellt, dass die Inhalte von ADCL und ADCH immer aus demselben Wandlungsergebnis stammen, selbst wenn der ADC im Hintergrund im Free-Conversion-Mode arbeitet.

[megaAVR] Firma Microchip, megaAVR® Data Sheet, [Link](#)

## Beispiele

Der Simulative Eingang x (0 bis 5V) ist an A0 angeschlossen.

Simulation time: 00:14.278

1.0 V

```
1 void setup() {  
2     pinMode(A0, INPUT);  
3     Serial.begin(9600);  
4     //analogReference(INTERNAL);  
5 }  
6 void loop() {  
7     Serial.println(analogRead(A0));  
8     delay(25);  
9 }
```





[illegible]









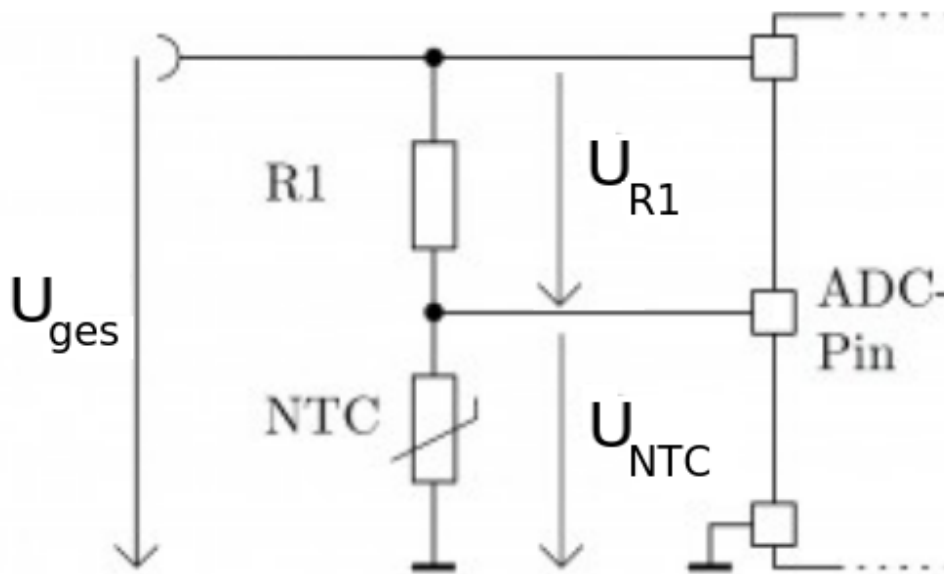
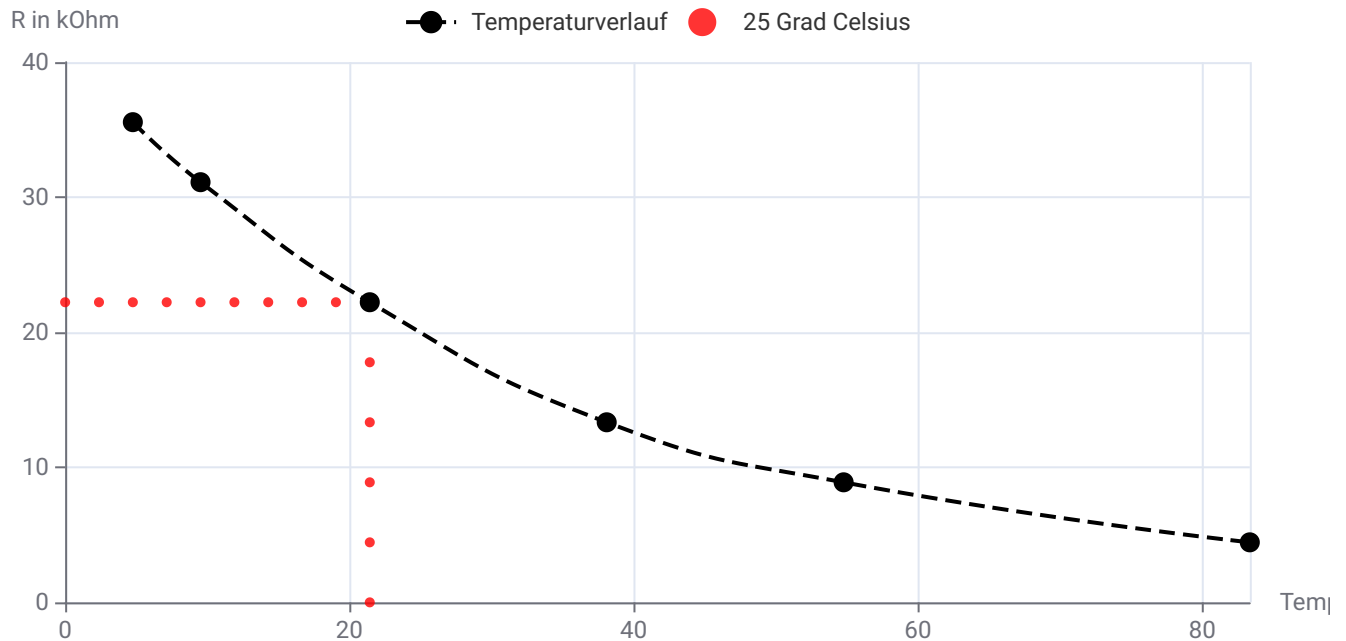
[illegible]

**Aufgabe:** Recherchieren Sie die Varianten von `analogReference()` für einzelne Vertreter der Arduino Familie. Warum unterscheiden sich die Parameter?

## Beispiel 1 - Heißleiter

NTC-Widerstände (*Negative Temperature Coefficient*) werden zur Messung der Temperatur eingesetzt. Wichtigster Kennwert eines NTCs ist der Nennwiderstand  $R_{25}$  bei einer Nenn­temperatur von 25 °C.

## NTC-Widerstand Kennlinie



$$\frac{R_{NTC}}{R_1} = \frac{U_{NTC}}{U_{R1}} = \frac{U_{NTC}}{(U_{ges} - U_{NTC})}$$

$$R_{NTC} = R_1 \cdot \frac{U_{NTC}}{(U_{ges} - U_{NTC})}$$

Wenn wir davon ausgehen, dass die Referenzspannung des AD-Wandlers gleich  $U_{ges}$  ist, generieren wir eine digitale Repräsentation  $U_{NTC_d}$  entsprechend

$$U_{NTC_d} = U_{NTC} \cdot \frac{U_{ges}}{ADC_{resolution}}$$

Wie interpretieren wir somit einen beispielhaften Wert von 628 am Ende des Wandlungsprozesses?

$$U_{NTC} = U_{NTC_d} \cdot \frac{ADC_{resolution}}{U_{ges}}$$

Mit diesem Spannungswert können wir nun den zugrundeliegenden Widerstand berechnen und letztendlich die Temperatur.

## Beispiel 2 - Lesen eines Analogen Distanzsensors

Für das Beispiel wird der AtMega2560 verwendet, der eine interne Referenzspannung von 2.56 V anstatt der des AtMega328 von 1.1 V bereit stellt.

### 26.8.1 ADMUX – ADC Multiplexer Selection Register

Bit (0x7C)	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in [Table 26-3](#). If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

**Table 26-3.** Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection <sup>(1)</sup>
0	0	AREF, Internal $V_{REF}$ turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Note: 1. If 10x or 200x gain is selected, only 2.56V should be used as Internal Voltage Reference. For differential conversion, only 1.1V cannot be used as internal voltage reference.

megaAVR® Data Sheet, Seite 281 [\[megaAVR2560\]](#)

Die Bedeutung ergibt sich beim Blick ins Datenblatt des Sensors GP2D, dessen Maximalwertausgabewert liegt bei etwa 2.55V

```
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz Taktrate
#endif

#include <avr/io.h>
#include <util/delay.h>

int readADC(int channel) {
    int i; int result = 0;
    // Den ADC aktivieren und Teilungsfaktor auf 64 stellen
    ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1);
    // Kanal des Multiplexers & Interne Referenzspannung (2,56 V)
    ADMUX = channel | (1<<REFS1) | (1<<REFS0);
    // Den ADC initialisieren und einen sog. Dummyreadout machen
    ADCSRA = (1<<ADSC);
```

```

ADCSRA |= (1<<ADSC);
while(ADCSRA & (1<<ADSC));
ADCSRA |= (1<<ADSC);
while(ADCSRA & (1<<ADSC)); // Auf Ergebnis warten...
// Lesen des Registers "ADCW" kümmert sich um das Lesen von ADCL und ADCH
result = ADCW;
// ADC wieder deaktivieren
ADCSRA = 0;
return result;
}

int main(void)
{
    Serial.begin(9600);
    while (1) // Unendliche Schleife
    {
        int result_individual = readADC(0);
        Serial.println(result_individual);
        Serial.flush();
        _delay_ms(10); // Verzögerung von 1 Sekunde
    }
    return 0; // wird nie erreicht
}

```

*The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result. Handbuch Seite 252*

[megaAVR2560] Firma Atmel, Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V, Seite 281, [Link](#)

## Beispiel 3 - Temperaturüberwachung des Controllers

*The temperature measurement is based on an on-chip temperature sensor that is coupled to a single ended ADC8 channel. Selecting the ADC8 channel by writing the MUX3...0 bits in ADMUX register to "1000" enables the temperature sensor. The internal 1.1V voltage reference must also be selected for the ADC voltage reference source in the temperature sensor measurement. When the temperature sensor is enabled, the ADC converter can be used in single conversion mode to measure the voltage over the temperature sensor. Handbuch Seite 256*

```

#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif

#include <avr/io.h>
#include <util/delay.h>

```



```

// include <avr/delay.h>

double getTemp(void)
{
    unsigned int wADC;
    double t;

    // Set the internal reference and mux.
    ADMUX = (1<<REFS1) | (1<<REFS0) | (1<<MUX3));
    ADCSRA |= (1<<ADEN); // enable the ADC

    // wait for voltages to become stable.
    delay(20);

    // Start the ADC
    ADCSRA |= (1<<ADSC);

    // Detect end-of-conversion
    while (ADCSRA & (1<<ADSC));
    wADC = ADCW;

    // The offset of 324.31 could be wrong. It is just an indication.
    t = (wADC - 324.31) / 1.22;

    // The returned temperature is in degrees Celsius.
    return (t);
}

int main(void)
{
    Serial.begin(9600);
    while (1)
    {
        Serial.println(getTemp());
        Serial.flush();
        _delay_ms(10); //1 second delay
    }
    return 0; // wird nie erreicht
}

```