Analog-Digital Umwandlung (Atmega328p)

https://www.mikrocontroller.net/articles/AVR-GCC-Tutorial/Analoge_Ein-_und_Ausgabe

Features (ATMEGA328)

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy
- 13 - 260 µs Conversion Time
- Up to 76.9 kSPS (Up to 15 kSPS at Maximum Resolution)
- 6 Multiplexed Single Ended Input Channels
- Temperature Sensor Input Channel
- Optional Left Adjustment for ADC Result Readout
- 0 - VCC ADC Input Voltage Range
- Selectable 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode
- Noise Canceler

Was ist Auflösung (Resolution)

N=10 bit → 2^N = 1024 Steps → U_{LSB}=VCC/2^N = 5/1024 ~ 5mV

d.h. der kleinste aufgelöste Spannungssprung ist 5mV bei 5V Versorgungsspannung und 10 Bit Auflösung. Das Signal wird quantisiert d.h. in 1000 Teile geteilt, das analoge Original wird verfälscht, dies kann man als Rauschen angeben. Der Signal-Rausch-Abstand beträgt ca. 6*N+1.8dB = 61.8dB

Was ist Integrale Nichtlinearität

Ein Maß für die Genauigkeit.

Nach dem Abgleich von Verstärkungs- und Offsetfehler können die Schaltschwellen vom Ideal abweichen (in nebenstehender Skizze schaltet der ADC schon bei 0.2V auf 1 und schon bei 1.2V auf 2 (ideal wäre 0.5 und 1.5), die Schaltschwelle liegt also bei (0.2V+1.2V)/2

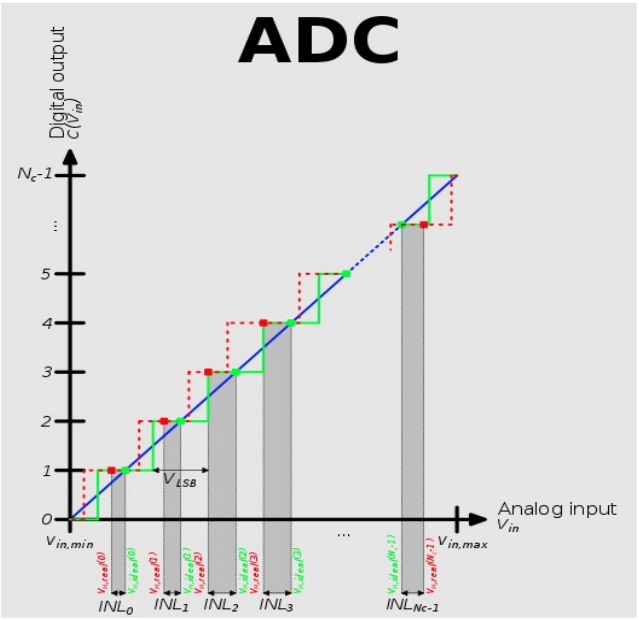


Figure 1: Wikipedia

=0.7V und nicht bei idealem 1V .

Was ist Absolute Genauigkeit

Ein weiteres Maß für die Genauigkeit. Beinhaltet alle Fehler (Offset, Verstärkung, Integrale Nichtlinearität …)

Wie berechnet man die Konvertierungsdauer

Für volle Auflösung benötigt der ADC eine Konvertierungsfrequenz von 50kHz bis 200kHz; ist die Taktrate höher, dann sinkt die Auflösung.

Einstellbar über den Prescaler. Z.B: Prescaler = 64, fosc=16MHz ==> fadc = 16MHz/128 = 125kHz

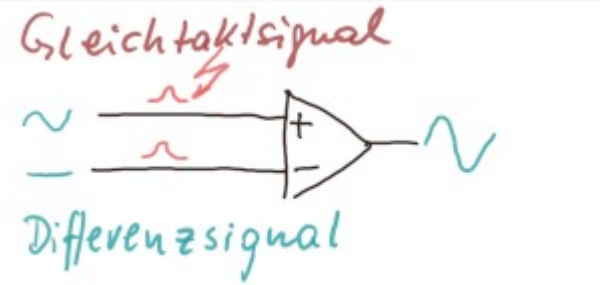
Was ist ein kSPS

Kilo Samples per Second = Anzahl der Konvertierungen pro Sekunde

Eine Konvertierung dauert 13 ADC Clock Zyklen → t = 13*10us = 130us bei 100kHz (13us bei 1MHz) d.h kSPS=7600

Multiplexed Single Ended Input Channels

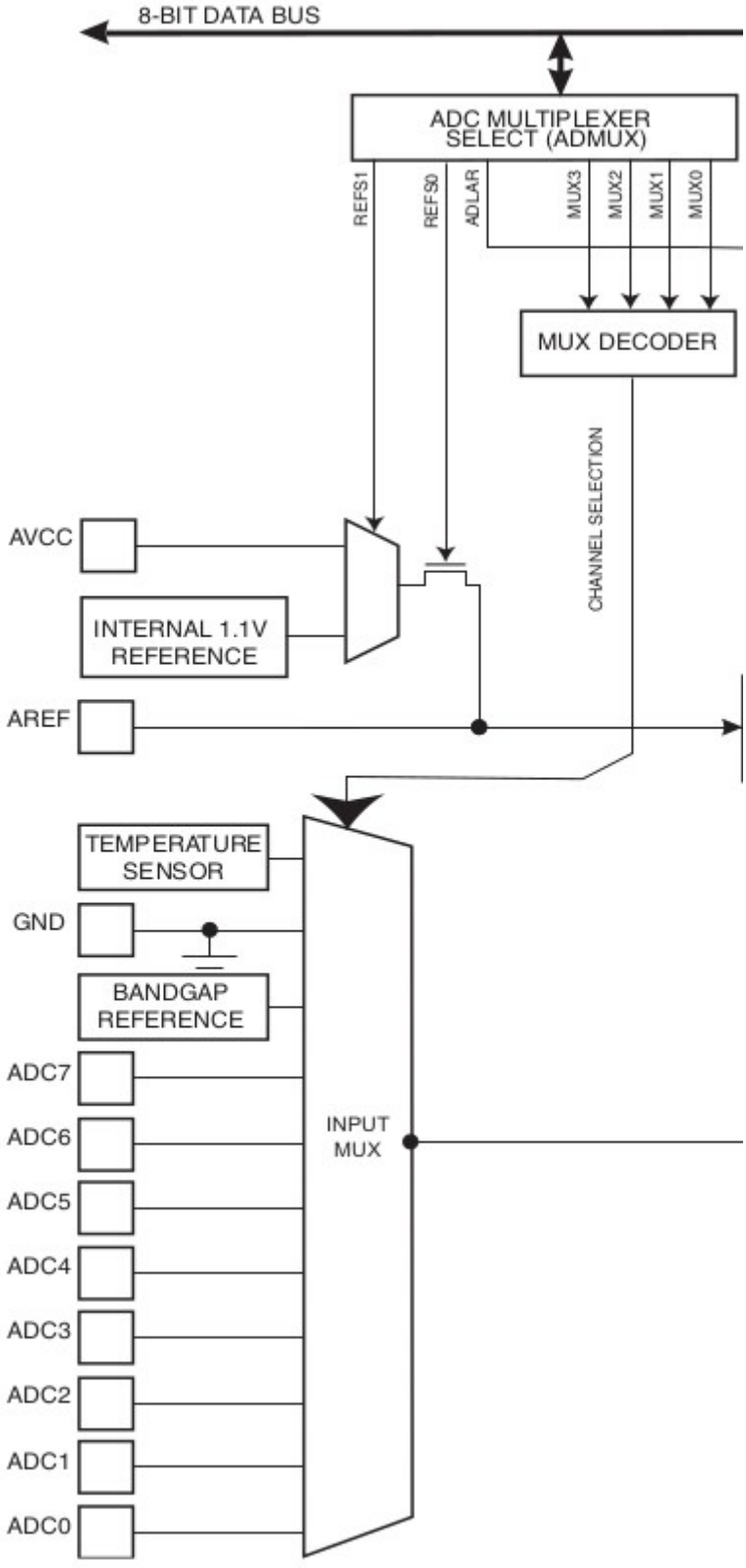
Differenzielle Eingänge sind weniger Störanfällig, weil Störsignale ein Gleichtaktsignal darstellen, das durch den differnziellen Eingang automatisch entfernt wird:



Single ended = not differnential

Referenzspannung und Analog Input Multiplexer

Left



Adjustment for ADC Result

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

Ergebnis der Konvertierung:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

Free Running, Auto Trigger oder Single Conversion

Single Conversion: Das Stromspar-Flag (PRADC power reduction) löschen und ADC starten (ADSC) ; ADSC wird nach dem Ende der Konvertierung automatisch gelöscht

Auto Trigger Mode: Auto Trigger enable (ADATE) startet eine Konvertierung, wenn eine der auswählbaren Quellen eine steigende Flanke liefert; die Quelle wird durch ADTS (trigger select) ausgewählt: Externer Interrupt, Timer Overflow, Timer Compare Match

Free Running: das ADIF (interrupt request flag) wird gesetzt, wenn die Konversion fertig ist und löst eine neue Konversion aus. Die erste Konvertierung muss durch ADSC (start of conversion) ausgelöst werden.

Interrupt on ADC Conversion Complete

Bei Auto-Trigger muss das Request-Flag manuell gelöscht werden um eine neue Konvertierung zu starten, bei Free-Running ist das nicht nötig, es wird automatisch zurückgesetzt.

Sleep Mode

Der ADC bleibt auch in den Prozessor Sleep Modi in Betrieb (außer IDLE und Noise Cancel Sleep Mode) und sollte daher vorher abgeschaltet werden (PRADC=1)

Noise Canceler

Störungen durch den Digitalteil des Prozessors sollen vermieden werden, daher wird der Prozessor schlafen gelegt, und nach einer Konvertierung wird der uC aufgeweckt.

```
#include <inttypes.h>
#include <avr/io.h>

/* Funktion übernommen aus

http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial#Der_interne_ADC_im_AVR

*/

uint16_t ReadChannel(uint8_t channel )
{
        uint8_t i;
        uint16_t result;

        // Frequenzvorteiler setzen auf 64 und ADC aktivieren
        ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);

        ADMUX = channel;                    // Kanal waehlen
        ADMUX |= _BV(REFS1) | _BV(REFS0); // interne Referenzspannung nutzen

        /* nach Aktivieren des ADC wird ein "Dummy-Readout" empfohlen,
           man liest also einen Wert und verwirft diesen,
           um den ADC "warmlaufen zu lassen"
        */
        ADCSRA |= _BV(ADSC);                // eine ADC-Wandlung
        while ( ADCSRA & _BV(ADSC) ) {
                ;     // auf Abschluss der Konvertierung warten
        }
        result = ADCW;  // ADCW muss einmal gelesen werden,
        // sonst wird Ergebnis der nächsten Wandlung
        // nicht übernommen.

        /* Eigentliche Messung
           - Mittelwert aus 4 aufeinanderfolgenden Wandlungen
        */
        result = 0;
        for( i=0; i<4; i++ )
        {
                ADCSRA |= _BV(ADSC);        // eine Wandlung "single conversion"
                while ( ADCSRA & _BV(ADSC) ) {
                        ;   // auf Abschluss der Konvertierung warten
                }
                result += ADCW;             // Wandlungsergebnisse aufaddieren
        }
        ADCSRA &= ~_BV(ADEN);               // ADC deaktivieren (2)
        result /= 4;           // Summe durch vier teilen = arithm. Mittelwert
        return result;
}
#define VREF 2.56 /* value of (interna) reference voltage) */

/*
 ** function prototypes
 */
uint16_t ReadChannel(uint8_t channel);

int main(void)
{
        int value;
        char buf[32];
        float f;
```

```
        DDRC &=~ (1 << PC3);        /* Pin PC3 input */

        /* now enable interrupt, since UART library is interrupt controlled */
        sei();

        while (1) {
                value = ReadChannel(3); // channel 3 is PC3
                f = value;
                f = (f*VREF)/1024.0;
                // 256 for 8 bits, 1024 for 10 bits; VREF=2.56 for internal ref.
                sprintf( buf, "Measuring %.3f Volt (raw value = %d)\r",
                        (double)(f), value  );
                dowait();
        }
}
```

Beim Linken muss eine zusätzliche printf-Bibliothek, die auch **Floats** ausgeben kann, hinzugebunden werden (libprintf_flt). Diese Bibliothek wird normalerweise nicht eingebunden, weil sie deutlich größere Executables produziert.

Die Register des ADC

ADC Control and **S**tatus **R**egister A.

In diesem Register stellen wir ein, wie wir den **ADC** verwenden möchten.
Das Register ist wie folgt aufgebaut:

Bit	7	6	5	4	3	2	1	0
Name	**ADEN**	**ADSC**	**ADFR**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initialwert	0	0	0	0	0	0	0	0

ADEN (**AD**C **En**able)

ADCSRA

Dieses Bit muss gesetzt werden, um den **ADC** überhaupt zu aktivieren. Wenn das Bit nicht gesetzt ist, können die Pins wie normale I/O-Pins verwendet werden.

ADSC (**AD**C **S**tart **C**onversion)

Mit diesem Bit wird ein Messvorgang gestartet. In der frei laufenden Betriebsart muss das Bit gesetzt werden, um die kontinuierliche Messung zu aktivieren.
Wenn das Bit nach dem Setzen des **ADEN**-Bits zum ersten Mal gesetzt wird, führt der Controller zuerst eine zusätzliche Wandlung und erst dann die eigentliche Wandlung aus. Diese zusätzliche Wandlung wird zu Initialisierungszwecken durchgeführt.

Das Bit bleibt nun so lange auf 1, bis die Umwandlung abgeschlossen ist, im Initialisierungsfall entsprechend bis die zweite Umwandlung erfolgt ist und geht danach auf 0.

ADFR (**AD**C **F**ree **R**un select)

Mit diesem Bit wird die Betriebsart eingestellt.
Ist das Bit auf 1 gesetzt arbeitet der ADC im "Free Running"-Modus. Dabei wird das Datenregister permanent aktualisiert.

ADIF (**AD**C **I**nterrupt **F**lag)

Dieses Bit wird vom **ADC** gesetzt, sobald eine Umwandlung erfolgt ist und das **ADC Data Register** aktualisiert wurde. Das Bit wird bei lesendem Zugriff auf **ADC(L,H)** automatisch (d.h. durch die Hardware) gelöscht.
Wenn das **ADIE** Bit sowie das **I-Bit** im AVR **Statusregister** gesetzt ist, wird der **ADC Interrupt** ausgelöst und die Interrupt-Behandlungsroutine aufgerufen.
Das Bit wird automatisch gelöscht, wenn die Interrupt-Behandlungsroutine aufgerufen wird. Es kann jedoch auch gelöscht werden, indem eine logische 1 in das Register geschrieben wird.

ADIE (**AD**C **I**nterrupt **E**nable)

Wenn dieses Bit gesetzt ist und ebenso das **I-Bit** im Statusregister **SREG**, dann wird der **ADC-Interrupt** aktiviert.

ADPS2...ADPS0 (**AD**C **P**rescaler **S**elect Bits)

Diese Bits bestimmen den Teilungsfaktor zwischen der Taktfrequenz und dem Eingangstakt des **ADC**.
Der **ADC** benötigt einen eigenen Takt, welchen er sich selber aus der CPU-Taktfreqenz erzeugt. Der **ADC**-Takt sollte zwischen 50 und 200kHz sein.
Der Vorteiler muss also so eingestellt werden, dass die CPU-Taktfrequenz dividiert durch den Teilungsfaktor einen Wert zwischen 50-200kHz ergibt.
Bei einer CPU-Taktfrequenz von 4MHz beispielsweise rechnen wir

$$TF_{min} = \frac{CLK}{200\,\mathrm{kHz}} = \frac{4000000}{200000} = \mathbf{20}$$

$$TF_{max} = \frac{CLK}{50\,\mathrm{kHz}} = \frac{4000000}{50000} = \mathbf{80}$$

Somit kann hier der Teilungsfaktor 32 oder 64 verwendet werden. Im Interesse der schnelleren Wandlungszeit werden wir hier den Faktor 32 einstellen.

ADPS2	ADPS1	ADPS0	Teilungsfaktor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8

1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADC Data Register

Wenn eine Umwandlung abgeschlossen ist, befindet sich der gemessene Wert in diesen beiden Registern. Von **ADCH** werden nur die beiden niederwertigsten Bits verwendet. Es müssen immer beide Register ausgelesen werden, und zwar immer **in der Reihenfolge: ADCL, ADCH**. Der effektive Messwert ergibt sich dann zu:

ADCL

ADCH

```
x = ADCL;         // mit uint16_t x
x += (ADCH<<8); // in zwei Zeilen (LSB/MSB-Reihenfolge und
                  // C-Operatorpriorität sichergestellt)
```

oder

```
x = ADCW; // je nach AVR auch x = ADC (siehe avr/ioxxx.h)
```

ADC **Mu**ltiple**x**er Select Register

Mit diesem Register wird der zu messende Kanal ausgewählt.
Das Register ist wie folgt aufgebaut:

Bit	7	6	5	4	3	2	1	0
Name	**REFS1**	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initialwert	0	0	0	0	0	0	0	0

MUX4...MUX0

Mit diesen 5 Bits wird der zu messende Kanal bestimmt. Wenn das Register beschrieben wird, während eine Wandlung läuft, so wird zuerst die aktuelle Umwandlung auf dem bisherigen Kanal beendet.

ADMUX

REFS1...REFS0 (**Ref**erence**S**election Bits)

Mit diesen Bits kann die Referenzspannung eingestellt werden. Bei der Umstellung sind Wartezeiten zu beachten, bis die ADC-Hardware einsatzfähig ist (Datenblatt):

REFS1	**REFS0**	**Referenzspanung**
0	0	Externes AREF
0	1	AVCC als Referenz
1	0	Reserviert
1	1	Interne 2,56 Volt

ADLAR (**ADC L**eft **A**djust **R**esult)

Das ADLAR Bit verändert das Aussehen des Ergebnisses der AD-Wandlung. Bei einer logischen 1 wird das Ergebnis linksbündig ausgegeben, bei einer 0 rechtsbündig. Eine Änderung in diesem Bit beeinflusst das Ergebnis sofort, ganz egal ob bereits eine Wandlung läuft. Linksbündig ausrichten hat den Vorteil, dass man – wenn man auf die 2 LSB verzichtet – nur das HIGH-Byte auszulesen braucht.