C1 A/D- und D/A-Wandler

Analog-Digital-Wandler[4] (*Analog-to-Digital Converter (ADC)*) setzen analoge Signale in digitale Daten um.

Digital-Analog-Wandler[5] (*Digital-to-Analog Converter (DAC)*) tun genau das Gegenteil.

Die Theorie zu A/D-Wandlern füllt ganze Bücher. Es soll hier nur auf die in den ATmega-Mikrocontrollern integrierten A/D-Wandler eingegangen werden.

A/D-Wandler

A/D-Wandler wandeln eine analoge Spannung in digitale Zahlenwerte um. Wichtige Kriterien sind hierbei die **Wandlungszeit**, die **Auflösung** in Bit und die **Genauigkeit**.

A/D-Wandler mit sehr hoher Auflösung benötigen viel Wandlungszeit. Ein solcher Wandler nach dem Dual-Slope-Verfahren (Zählverfahren) kann relativ einfach mit dem internen Komparator des ATmega32A und ein wenig externer Beschaltung aufgebaut werden.

Schnelle A/D-Wandler, welche Signale mit hohen Frequenzen wandeln können, haben eine relativ geringe Auflösung. Einen guten Kompromiss ermöglichen die in den AVR-Controllern eingesetzten A/D-Wandler, welche nach dem Wägeverfahren[6] arbeiten.

Die Genauigkeit eines A/D-Wandlers wird durch Quantisierungsfehler, Nullpunktfehler, Verstärkungsfehler, Nichtlinearität, Monotoniefehler und dynamische Fehler bestimmt. Im Datenblatt des ATmega32A findet man Angaben zur Genauigkeit des integrierten Wandlers. Durch ein eingebautes Abtast- und -Halte-Glied ist der dynamischen Fehler des integrierten Wandlers klein. Es können sogar Wandlungen, während sich der Controller in einem Schlafmodus befindet, durchgeführt werden um das eingestreute Rauschen zu minimieren. Wichtig für eine hohe Genauigkeit ist aber auch das Layout und die Beschaltung beim Anschluss an den A/D-Wandler (siehe Datenblatt).

Der im ATmega32A integrierte A/D-Wandler besitzt viele unterschiedliche Konfigurationsmöglichkeiten und ist deshalb flexibel einsetzbar. Er eine **Auflösung von 10 Bit** und eine schnellste Wandlungszeit nach Datenblatt von 13 μs (bei geringerer Genauigkeit), womit Frequenzen von einigen kHz verarbeitet werden können. Ein **Multiplexer** am Eingang ermöglicht es 8 verschiedene Eingangsspannungen zu wandeln. Es können auch mit Hilfe von je zwei Eingängen **Differenzsignale** verarbeitet werden. Ein **Vorverstärker** kann die Eingangsspannung um den Faktor 10 verstärken (Faktor 200 möglich bei SMD-Bauformen und entsprechendem Layout). Alle Eingänge des A/D-Wandlers werden über die Pins von Port A eingelesen.

4 A/D-Wandler, oder auch noch Analog-Digital-Umsetzer (ADU)

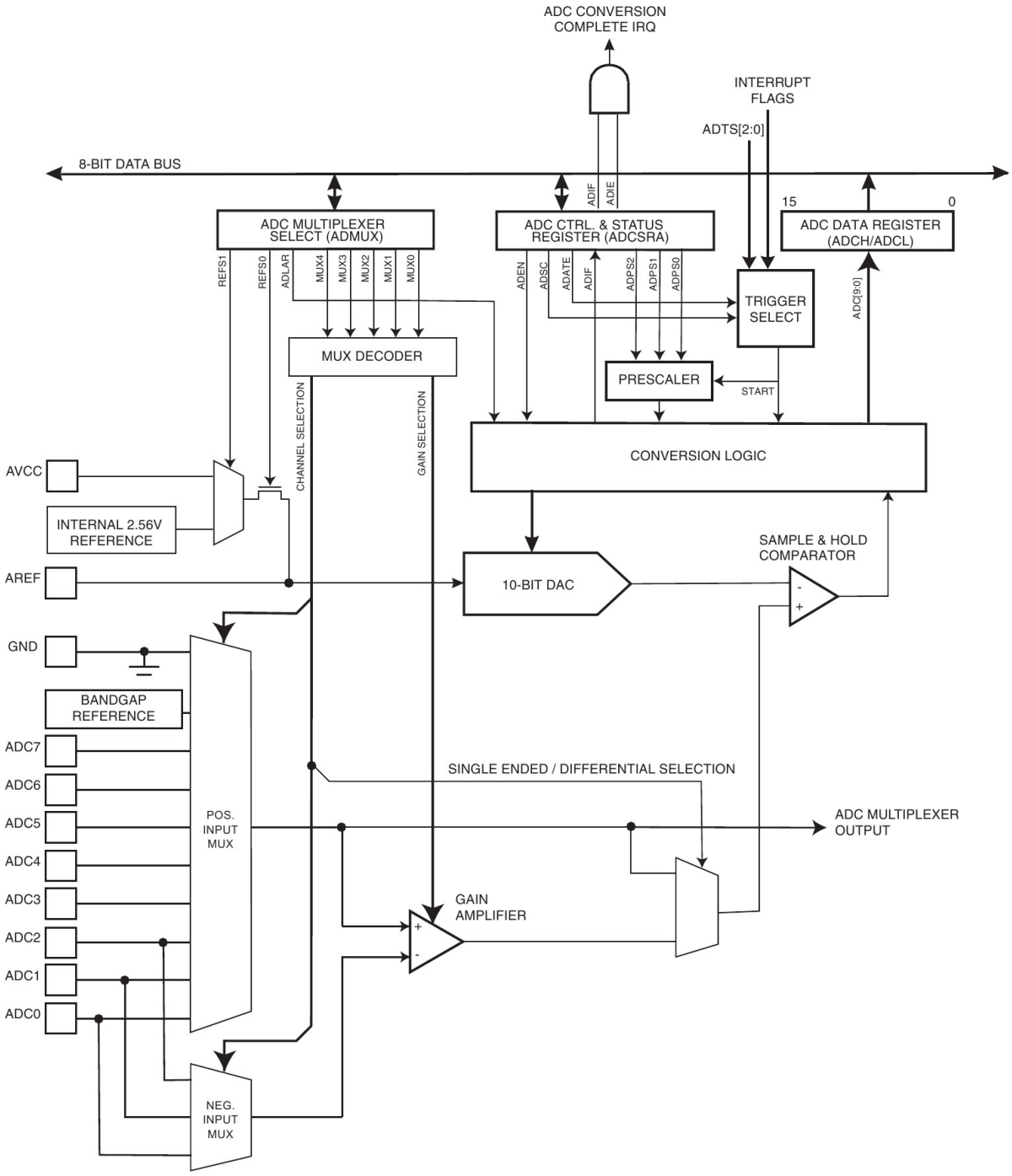
5 D/A-Wandler, oder auch noch Digital-Analog-Umsetzer (DAU)

6 Beim Wägeverfahren handelt es sich um ein Kompensationsverfahren. Die Annäherung an die Eingangsspannung erfolgt in gewichteten Schritten (Verfahren der sukzessiven Approximation).

Als Referenzspannung kann die Betriebsspannung des Controllers verwendet werden, eine externe Referenzspannung oder eine interne Referenzquelle von 2,56 V. Mit einer 10 Bit-Auflösung und einer Referenzspannung von zum Beispiel 5 V ist dann die feinste Auflösungsstufe 5 V / 2^{10} = 4,883 mV.

In der folgenden Abbildung aus dem Datenblatt des ATmega32A sieht man die vollständige A/D-Wandler-Baugruppe im Blockschaltbild.

Über die SF-Register **ADMUX**, **ADCSRA** und **SFIOR** wird der Wandler konfiguriert. Im 16-Bit Datenregistern **ADC** (**ADCL** und **ADCH**) wird das Ergebnis der Wandlung abgelegt.



Die Initialisierung des A/D-Wandlers

Zur **Initialisierung und Statusabfrage** des A/D-Wandler dienen die drei Register:

- **ADMUX** In diesem Register werden die Referenzspannungsquelle, die Anordnung der 10 Datenbit im Datenregister (links- oder rechtsbündig) und die zu verwendeten Eingangspins mit ihrer Betriebsart und Verstärkung festgelegt.
- **ADCSRA** Das Kontroll- und Statusregister schaltet auf Wunsch den Wandler, den ADC-Interrupt und den Auto Trigger ein, kann eine neue Wandlung einleiten, initialisiert den Vorteiler und enthält das Interrupt-Flag.
- **SFIOR** In diesem Spezialregister werden nur drei Bit für den A/D-Wandler benötigt. Sie entscheiden welche Interrupt-Quelle den Auto Trigger auslösen soll

Daten stehen nach der Wandlung im Doppelregister:

- **ADC** (**ADCH**, **ADCL**) zur Verfügung.

Die Referenzspannungsquelle (ADMUX)

Es kann zwischen mittels zweier Bit (**REFS1**, **REFS0**) im Register **ADMUX** zwischen drei Referenzspannungsquellen gewählt werden:

1. Es wird die i**nterne Referenzspannungsquelle von 2,56 V** benutzt.
2. Es wird eine **externe Referenzspannung** am Pin **AREF** des Chip angelegt.
3. Die **Betriebsspannung des Wandlers** am Pin **AVCC** wird als Referenz benutzt.

Bemerkung: Beim MICES-Board sind **AVCC** und **AREF** mit der Betriebsspannung des Controllers verbunden. Für die folgende Versuche wird die Spannung des Wandlers (5 V) am Pin **AVCC** als Referenz benutzt.

Anordnung im Datenregister (ADMUX)

Das Bit **ADLAR** (*ADC Left Adjust Result*) entscheidet über die Ausrichtung des 10-Bit-Ergebnisses der A/D-Wandlung.

Werden alle 10 Bit des Ergebnisses benötigt, so macht nur eine rechtsbündige Anordnung im 16-Bit-Datenregister **ADC** Sinn. Das Ergebnis ist eine 16-Bit-Dualzahl.

ADLAR = 0

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

Beim Einlesen des Resultats muss ADCL vor ADCH eingelesen werden!!

Wird keine so hohe Präzision benötigt, oder soll die Wandlung mit möglichst hoher Geschwindigkeit erfolgen, so kann mit nur 8-Bit gewandelt werden. Die beiden niederwertigsten Bits werden verworfen. Dies vereinfacht auch die Verarbeitung der Daten, da dann nur mit dem 8-Bit-SF-Register **ADCH** gearbeitet wird.

Die Anordnung erfolgt linksbündig.

ADLAR = 1

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

Es wird nur **ADCH** eingelesen.

Der Kanalmultiplexer (ADMUX)

Bei eingeschalteter A/D-Wandler-Baugruppe sind die ausgewählten Pins des Port A keine digitalen Ein-bzw. Ausgangspins mehr. Es können nun analoge Signale eingelesen werden. Ein Multiplexer ermöglicht es aus den 8 Eingangspins auszuwählen.

Oft wird ein analoger Eingang unsymmetrisch betrieben. Die analoge Spannung liegt gegen die analoge Masse **AGND** des Chip an. Die Auswahl des Pins geschieht durch Einschreiben der dem Pin entsprechenden Dualzahl in die 5 **MUX**-Bits des **ADMUX**-Register (**MUX0** - **MUX4**). Soll zum Beispiel PIN **PA4** ausgewählt werden, so ist **0b00100** zu initialisieren (siehe Tabelle des **ADMUX**-Registers im Kapitel „Die SF-Register des A/D-Wandlers“).

Soll eine symetrische Spannung eingelesen werden, so werden zwei Eingangspins benötigt. Die vielen Kombinationsmöglichkeiten des Anschlusses ($2^5 = 32$) zeigt die oben erwähnte Tabelle. Interessant ist auch die Möglichkeit den A/D-Wandler-Eingang zur Fehlersuche oder zur Kalibrierung mit einer Referenzspannungsquelle oder mit Masse zu verbinden. Das Messen einer Referenzspannung ermöglicht zum Beispiel die Bestimmung der Betriebsspannung des Chips. Besonders interessant ist dies bei mobilen Geräten mit Akku.

Der Vorteiler (ADCSRA)

Wie im großen Blockschaltbild ersichtlich benötigt der A/D-Wandler eine Taktfrequenz, welche von einem Vorteiler (*prescaler*) geliefert wird. Der Vorteiler teilt den Systemtakt und muss laut Datenblatt eine Taktfrequenz zwischen 50 kHz und 200 kHz liefern, damit der A/D-Wandler seine maximale Auflösung erreicht. Die Taktfrequenz darf allerdings auch höher sein, wenn nur mit 8 Bit gearbeitet wird. Der Vorteiler wird mit 3 Bit (**ADPS0** - **ADPS2**) im Kontrollregister **ADCSRA** eingestellt. Die beiden ersten Kombinationen liefern den gleichen Teilungsfaktor. Um zum Beispiel mit einem 16 MHz Quarz bei voller Auflösung zu arbeiten bleibt nur die Teilung durch 128 (16 MHz/128 = 125 kHz).

Das Taktsignal wird nur geliefert, wenn der A/D-Wandler-Baustein eingeschaltet ist.

ADPS2 ADPS1 ADPS0 2^2 2^1 2^0	Teilungs-faktor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

✎ **C100**

a) Eine normale Wandlung (*free running mode*) benötigt 13 Taktzyklen des A/D-Wandlertakts. Berechne die Wandlungszeit, falls bei 16 MHz der Vorteiler auf 128 eingestellt wird.

b) Im Datenblatt ist eine minimale Wandlungszeit von 13 μs angegeben. Mit welcher höchsten Taktfrequenz kann der Wandler noch arbeiten (wenn auch nicht mit voller Auflösung)?

c) Berechne die maximale Signalfrequenz bei einer A/D-Wandlungszeit von 13 μs nach Nyquist[7].

Weitere Einstellungen

Wie bei allen anderen Baugruppen ist der A/D-Wandler nach einem **RESET** abgeschaltet um Strom zu sparen. Das Einschalten der Baugruppe geschieht mit dem Bit **ADEN** (**ADCSRA**).

7 Das Nyquist-Shannonsche Abtasttheorem besagt, dass ein kontinuierliches, bandbegrenztes Signal, mit einer Minimalfrequenz von 0 Hz und einer Maximalfrequenz f_{max}, mit einer Frequenz größer als 2 f_{max} abgetastet werden muss, damit man aus dem so erhaltenen zeitdiskreten Signal das Ursprungssignal ohne Informationsverlust (aber mit unendlich großem Aufwand) exakt rekonstruieren und (mit endlichem Aufwand) beliebig genau approximieren kann (Quelle Wikipedia).

Das Einleiten einer Wandlung geschieht entweder softwaremäßig durch das Setzen des Bit **ADSC** (*start conversion*) im Register **ADCSRA** oder aber hardwaremäßig durch einen Interrupt.

Polling

Beim **Polling** (ständige Abfrage durch die Software) kann das Bit **ADSC** oder das Flag **ADIF** benutzt werden um das Ende der Wandlung festzustellen. **ADSC** bleibt während der Wandlung auf Eins und wird gleich nach der Wandlung durch die Hardware gelöscht, **ADIF** muss nach der Wandlung manuell durch Setzen einer Eins gelöscht werden.

Man unterscheidet den ***Single Conversion Mode*** bei dem jede einzelne Wandlung durch erneutes Schreiben des **ADSC**-Bits eingeleitet wird und den ***Free Running Mode*** (Freilaufmodus), bei dem jedes Ende einer Wandlung automatisch eine neue Wandlung auslöst.

Für den ***Free Running Mode*** muss der **Auto-Trigger** mit dem Bit **ADATE** (**ADCSRA**) eingeschaltet werden. Durch (einmaliges) Setzen des **ADSC**-Bits wird die erste Wandlung eingeleitet. Diese erste Wandlung dauert etwas länger (25 statt 13 A/D-Taktzyklen). Der Auto-Trigger leitet automatisch nach der ersten erfolgten Wandlung eine neue Wandlung ein, wenn eine positive Flanke einer Trigger-Quelle entdeckt wird[8]. Die Trigger-Quelle wird mit drei Bit (**ADTS0** - **ADTS2**) im SF-Register **SFIOR** festgelegt. Im *Free Running Mode* (**ADTS0** = **ADTS1** = **ADTS2** = 0) ist die Trigger-Quelle der A/D-Wandler selbst. Jedes Ende einer Wandlung triggert also eine erneute Wandlung.

Der **Auto-Trigger** funktioniert im *Free Running Mode* mit und ohne freigeschaltete Interrupts! Auch bei gesperrten Interrupts wird das Interrupt-Flag **ADIF** gesetzt. Um weitere Trigger zu ermöglichen, und damit Wandlungen auszulösen, muss das Flag dann aber softwaremäßig gelöscht werden.

Interrupts

Außer dem A/D-Wandler selbst kommen als Trigger-Quellen noch unterschiedliche Interrupt-Quellen in Frage. Besonders interessant ist das Auslösen eines Interrupts durch ein externes Signal (externer Interrupt) oder durch einen der Timer. Damit ist das Wandeln in festen Zeitabständen zum Beispiel fürs Datalogging möglich.

Das interruptgesteuerte Wandeln ermöglicht es den Mikrocontroller, während der Wandlungszeit des A/D-Wandlers, für andere Aufgaben zu nutzen und sollte bevorzugt verwendet werden.

8 Tritt diese Flanke während einer laufenden Wandlung auf, so wird sie wird sie ignoriert.

Beispiel für eine Initialisierung (Polling, Single Conversion Mode)

```
;ADC initialisieren
ldi     Tmp1,0b01100000 ;REFS = 01 AVCC als Referenzspannung
out     ADMUX,Tmp1      ;ADLAR = 1 linksbuendig (obere 8 Bit in ADCH)
                        ;MUX = 00000 unsymetrisch PA0 (ADC0)
ldi     Tmp1,0b10000111 ;ADEN = 1 (ADC einschalten)
out     ADCSRA,Tmp1     ;ADPS = 111 (16MHz/128 = 125kHz)
                        ;andere Bits per default = 0
```

Die SF-Register des A/D-Wandlers

Das ADMUX Register

Das **ADMUX**-Register befindet sich auf der SRAM-Adresse **0x0027** (SF-Register-Adresse **0x07**). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** dürfen verwendet werden, da es sich um eines der unteren 32 Register handelt.

ADMUX = ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
ADMUX 0x07	**REFS1**	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

REFSn ***Reference Selection Bits*** ***REFS1, REFS0***

Mit diesen beiden Bit wird die **Quelle für die Referenzspannung ausgewählt** (3 Möglichkeiten). Änderungen an diesen Bits werden erst nach einer eventuell stattfindenden Wandlung berücksichtigt. Liegt eine externe Referenzspannung an Pin **AREF**, so soll nicht softwaremäßig auf interne Spannungsquelle geschaltet werden!

00 **Externe Referenzspannung am Pin AREF des Chip (zwischen 2 V und VCC, interne Spannungsquelle VREF abgeschaltet)**
01 **Die Spannung des Wandler am Pin AVCC wird als Referenz benutzt*.**
10 **reserviert**
11 **Eine interne Referenzspannung VREF von 2,56 V wird benutzt*.**

* Der offene Eingang **AREF** soll zur Störunterdrückung mit einem Kondensator (z.B. 100 nF) gegen Masse geschaltet werden. Weitere Informationen zur Störunterdrückung im Datenblatt unter "*Analog Noise Canceling Techniques*".

ADLAR ***ADC Left Adjust Result***

0 Das 10-Bit-Ergebnis der A/D-Wandlung wird **rechtsbündig** im Doppelregister ADC abgelegt (siehe **ADCL** und **ADCH**).
1 Das 10-Bit-Ergebnis der A/D-Wandlung wird **linksbündig** im Doppelregister ADC abgelegt (siehe **ADCL** und **ADCH**).

MUXn ***Analog Channel and Gain Selection Bits MUX0-MUX4***

Mit diesen fünf Bit wird der bzw. die **Eingangspins PAx** am Port A (Eingangskanal), die **Betriebsart** (unsymmetrisch (unipolar, Spannung gegen Masse) oder symmetrisch (differenziell,

bipolar, Spannung zwischen zwei Pins)) und der **Verstärkungsfaktor**[9] nach der folgenden Tabelle **ausgewählt**. Änderungen an diesen Bits werden erst nach einer eventuell stattfindenden Wandlung berücksichtigt.

MUXn	UNSYMETRISCH	SYMETRISCH		
43 210 $2^4 2^3 2^2 2^1 2^0$	(Pin gegen Masse)	Positiver (nicht-invert.) Eingang	Negativer (invert.) Eingang	Verstärkung
00 000	PA0 (ADC0)			
00 001	PA1 (ADC1)			
00 010	PA2 (ADC2)			
00 011	PA3 (ADC3)			
00 100	PA4 (ADC4)			
00 101	PA5 (ADC5)			
00 110	PA6 (ADC6)			
00 111	PA7 (ADC7)			
01 000		PA0 (ADC0)	PA0 (ADC0)	10x
01 001		PA1 (ADC1)	PA0 (ADC0)	10x
01 010		PA0 (ADC0)	PA0 (ADC0)	200x
01 011		PA1 (ADC1)	PA0 (ADC0)	200x
01 100		PA2 (ADC2)	PA2 (ADC2)	10x
01 101		PA3 (ADC3)	PA2 (ADC2)	10x
01 110		PA2 (ADC2)	PA2 (ADC2)	200x
01 111		PA3 (ADC3)	PA2 (ADC2)	200x
10 000		PA0 (ADC0)	PA1 (ADC1)	1x
10 001		PA1 (ADC1)	PA1 (ADC1)	1x
10 010		PA2 (ADC2)	PA1 (ADC1)	1x
10 011		PA3 (ADC3)	PA1 (ADC1)	1x
10 100		PA4 (ADC4)	PA1 (ADC1)	1x
10 101		PA5 (ADC5)	PA1 (ADC1)	1x
10 110		PA6 (ADC6)	PA1 (ADC1)	1x
10 111		PA7 (ADC7)	PA1 (ADC1)	1x
11 000		PA0 (ADC0)	PA2 (ADC2)	1x
11 001		PA1 (ADC1)	PA2 (ADC2)	1x
11 010		PA2 (ADC2)	PA2 (ADC2)	1x
11 011		PA3 (ADC3)	PA2 (ADC2)	1x
11 100		PA4 (ADC4)	PA2 (ADC2)	1x
11 101		PA5 (ADC5)	PA2 (ADC2)	1x
11 110	1,22 V (Bandgap-Referenz V_{BG})			
11 111	0 V (GND)			

9 Die 200-fache Verstärkung ist nicht für PDIP-Gehäuseformen getestet.

Das ADC Kontroll- und Statusregister A

Das **ADCSRA**-Register befindet sich auf der SRAM-Adresse **0x0026** (SF-Register-Adresse **0x06**). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** dürfen verwendet werden, da es sich um eines der unteren 32 Register handelt.

ADCSRA = ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
ADCSRA 0x06	**ADEN**	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ADEN *ADC Enable*

0 schaltet den A/D-Wandler aus (Baugruppe verbraucht keinen Strom!). Geschieht dies während einer Wandlung, so wird diese abgebrochen.

1 **schaltet den A/D-Wandler ein**.

ADSC *ADC Start Conversion*

0 wird nach einer Wandlung automatisch wieder auf Null gesetzt. Das Löschen des Bit während der Wandlung hat keinen Einfluss.

1 **startet die A/D-Wandlung** und kann dazu benutzt werden festzustellen, ob momentan eine Wandlung durchgeführt wird.
Single Conversion Mode: Jede einzelne Wandlung wird durch erneutes Schreiben des Bits eingeleitet.
Free Running Mode: Durch (einmaliges) Setzen des Bit wird die erste Wandlung eingeleitet. Die erste Wandlung dauert länger (25 statt 13 A/D-Taktzyklen), da ein zusätzlicher Umwandlungszyklus vorangestellt wird, der zur Initialisierung des Wandlers dient.
ADSC und **ADEN** können gleichzeitig gesetzt werden.

ADATE *ADC Auto Trigger Enable*

0 beendet den Auto-Trigger.

1 **startet den Auto-Trigger**.
Der Auto-Trigger kann auf eine von acht Interrupt-Quellen (Trigger-Quellen) reagieren. Der A/D-Wandler startet eine Wandlung, wenn eine **positive Flanke** der Interrupt-Quelle erkannt wird. Die Interrupt-Quelle wird mit drei Bit (**ADTS0** - **ADTS2**) im **SFIOR**-Register ausgewählt.

ADIF *ADC Interrupt Flag*

0 keine aktuelle Daten im Datenregister **ADC**

1 wird **Eins** sobald der **Wandlungszyklus beendet** ist und die Daten im Doppelregister **ADC** (**ADCL**, **ADCH**) aktualisiert wurden. Gleichzeitig wird ein „*ADC Complete Interrupt*“ ausgelöst, wenn Interrupts global erlaubt sind (**I** im **SREG**) und das **ADIE**-Bit gesetzt wurde.
Wurde ein Interrupt ausgelöst, so wird das Bit hardwaremäßig nach dem Ausführen des Interrupt gelöscht.
Im Polling-Betrieb kann das Bit manuell durch Schreiben einer Eins gelöscht werden!

ADIE *ADC Interrupt Enable*

0 der A/D-Wandler Interrupt ist gesperrt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo die A/D-Wandlung beendet ist und neue Daten anliegen (**ADI**-Flag im **ADCSRA**). Interrupts müssen dazu global frei gegeben sein (**I**=1 im **SREG** mit "**sei**").

ADPSn *ADC Prescaler Select Bits ADPS2, ADPS1, ADPS0*

Mit diesen drei Bit wird der Teilungsfaktor des Vorteilers ausgewählt. Der kleinste Teilungsfaktor ist

2.
Die Taktfrequenz des A/D-Wandlers soll bei voller Auflösung (10 Bit) zwischen 50 und 200 kHz liegen! Bei 8 Bit kann sie bis zu 1 MHz betragen. Sie errechnet sich mit:

$$Wandlungstakt = \frac{Systemtakt}{Teilungsfaktor}$$

ADPS2 ADPS1 ADPS0 2^2 2^1 2^0	Teilungs-faktor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Das Sonderfunktionsregister SFIOR

Das **SFIOR**-Register befindet sich auf der SRAM-Adresse **0x0050** (SF-Register-Adresse **0x30**). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** dürfen **nicht verwendet** werden, da es sich um eines der oberen 32 Register handelt.

SFIOR = Special Function IO Register

Bit	7	6	5	4	3	2	1	0
SFIOR 0x30	**ADTS2**	**ADTS1**	**ADTS0**	**-**	**ACME**	**PUD**	**PSR2**	**PSR10**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W

ADTSn ***ADC Auto Trigger Source*** ***ADTS2, ADTS1, ADTS0***

Falls der Auto Trigger Modus eingeschaltet wurde (**ADATE** in **ADCSRA**) werden diese drei Bit im **SFIOR**-Register benötigt um die Trigger-Quelle auszuwählen. Eine A/D-Wandlung wird durch die steigende Flanke der ausgewählten Interrupt-Quelle ausgelöst.

ADTS2 ADTS1 ADTS0 2^2 2^1 2^0	Trigger-Quelle
000	Free Running Modus
001	Analoger Komparator
010	Externer Interrupt INT0
011	Timer 0 Compare
100	Timer 0 Overflow
101	Timer 1 Compare B
110	Timer 1 Overflow
111	Timer 1 Capture Event

Das 16-Bit ADC Datenregister

Die beiden 8 Bit-Register **ADCH** und **ADCL** befindet sich auf den SRAM-Adressen **0x0025** und **0x0024** (SF-Register-Adressen **0x05** und **0x04**). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** dürfen verwendet werden.

ADCH = ADC Data Register High

Bit	7	6	5	4	3	2	1	0
ADCH 0x05	- ADC9	- ADC8	- ADC7	- ADC6	- ADC5	- ADC4	ADC9 ADC3	ADC8 ADC2
Startwert	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R

ADCL = ADC Data Register Low

Bit	7	6	5	4	3	2	1	0
ADCL 0x04	ADC7 ADC1	ADC6 ADC0	ADC5 -	ADC4 -	ADC3 -	ADC2 -	ADC1 -	ADC0 -
Startwert	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R

ADCn ***ADC Data Register Bit n***

Das **Resultat einer A/D-Wandlung** befindet sich in diesem 16-Bit Doppelregister. Werden differentielle (symmetrische) Eingänge benutzt, so liegt das Resultat in Zweierkomplementform vor!

ADLAR = 0: Ist das **ADLAR**-Bit im Register **ADMUX** gelöscht, so sind die 10 Bits rechtsbündig angeordnet (obere Zeile). Diese Einstellung ist günstig, wenn alle 10 Bit benötigt werden. Die Wertigkeit der Bits entspricht dann einer 16-Bit Dualzahl. **Das niederwertige Register muss zuerst gelesen werden und dann das hochwertige Register!** Erst nach dem Lesen des hochwertigen Register werden neue Daten in **ADC** abgelegt.

ADLAR = 1: Ist das **ADLAR**-Bit im Register **ADMUX** gesetzt, so sind die 10 Bits linksbündig angeordnet (untere Zeile). Diese Einstellung ist günstig, wenn die zwei niederwertigsten Bit vernachlässigt werden können, man also nur mit 8 Bit weiterarbeitet. Es reicht dann nur **ADCH** auszulesen.

ADLAR = 0

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

ADLAR = 1

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

A/D-Wandlung mittels Polling

Single Conversion Mode

Im ***Single Conversion Mode*** (Einzelwandlungsmodus) muss jede Wandlung manuell neu gestartet werden, da das Bit **ADSC** (starte Wandlung, **ADCSRA**) nach der Wandlung automatisch von der Steuerung wieder zurückgesetzt wird. Dieses Bit kann auch für das Polling, zur Kontrolle ob die Wandlung angeschlossen ist, verwendet werden.

Bemerkung: Eine Variante ist die Überprüfung des **ADIF**-Flag. Dieses Flag muss allerdings manuell, durch Schreiben einer Eins, zurückgesetzt werden.

Programmbeispiel:

Der ATmega32A-Controller soll eine Spannung zwischen 0 V und 5 V an **PA0** (**ADC0**) in eine 8-Bit-Zahl wandeln. **PA0** soll als unsymmetrischer Eingang geschaltet sein. Der Spannungswert soll binär über LEDs an Port C angezeigt werden.
Das Ende der Wandlungszeit wird mittels Polling ermittelt.

```
;***************************************************************************
;*
;*      Titel:  AD-Wandler Polling (single conversion) mit ADSC
;*              (C101_adc_single_conversion_1.asm)
;*      Datum:  15/11/10        Version:        0.2
;*      Autor:  WEIGU
;*
;*      Informationen zur Beschaltung:
;*
;*      Prozessor:      ATmega32A               Quarzfrequenz: 16MHz
;*      Eingaenge:      Analoge Spannung an PA0 (ADC0)
;*      Ausgaenge:      8 LEDs an PORTC
;*
;*      Informationen zur Funktionsweise:
;*      Die Spannung an PA0 (ADC0) wird eingelesen und als 8 Bit-Wert binaer am
;*      PORTC an 8 LEDs ausgegeben sobald die Wandlung abgeschlossen ist.
;*
;***************************************************************************

;---------------------------------------------------------------------------
;       Einbinden der controllerspezifischen Definitionsdatei
;---------------------------------------------------------------------------
.NOLIST                         ;List-Output ausschalten
.INCLUDE "m32def.inc"           ;AVR-Definitionsdatei einbinden
.LIST                           ;List-Output wieder einschalten

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;       Programmspeicher (FLASH)     Programmstart nach RESET ab Adr. 0x0000
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.CSEG                           ;was ab hier folgt kommt in den FLASH-Speicher
.ORG    0x0000                  ;Programm beginnt an der FLASH-Adresse 0x0000
RESET1: rjmp    INIT            ;springe nach INIT (ueberspringe ISR Vektoren)

;---------------------------------------------------------------------------
;       Initialisierungen und eigene Definitionen
;---------------------------------------------------------------------------
.ORG    INT_VECTORS_SIZE        ;Platz fuer ISR Vektoren lassen
INIT:
.DEF    Zero = r15              ;Register 1 wird zum Rechnen benoetigt
        clr     r15             ;und mit Null belegt
.DEF    Tmp1 = r16              ;Register 16 dient als erster Zwischenspeicher
.DEF    Tmp2 = r17              ;Register 17 dient als zweiter Zwischenspeicher
```

```
.DEF    Cnt1 = r18              ;Register 18 dient als Zaehler
.DEF    WL = r24                ;Register 24 und 25 dienen als universelles
.DEF    WH = r25                ;Doppelregister W und zur Parameteruebergabe

        ;ADC initialisieren
        ldi     Tmp1,0b01100000 ;REFS = 01 AVCC als Referenzspannung
        out     ADMUX,Tmp1      ;ADLAR = 1 linksbuendig (obere 8 Bit in ADCH)
                                ;MUX = 00000 unsymetrisch PA0 (ADC0)
        ldi     Tmp1,0b10000111 ;ADEN = 1 (ADC einschalten)
        out     ADCSRA,Tmp1     ;ADPS = 111 (16MHz/128 = 125kHz)
                                ;andere Bits per default = 0

        ;PORTC (8 Bit) als Ausgang
        ser     Tmp1            ;DDRC = 0b11111111
        out     DDRC,Tmp1

;-----------------------------------------------------------------------------
;       Hauptprogramm
;-----------------------------------------------------------------------------
MAIN:   sbi     ADCSRA,ADSC     ;eine AD-Wandlung ausloesen
LOOP:   sbic    ADCSRA,ADSC     ;Polling bis Wandlung fertig
        rjmp    LOOP
        in      Tmp1,ADCH       ;oberste 8 Bit einlesen und ausgeben
        out     PORTC,Tmp1
        rjmp    MAIN            ;Endlosschleife

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.EXIT                           ;Ende des Quelltextes
```

✎ **C101** a) Zeichne das dem Programmbeispiel entsprechende Flussdiagramm.
b) Teste das Programm mit einer variablen Spannungsquelle (0-5 V).
Speichere das Programm als "**C101_adc_single_conversion_1.asm**".
c) Teste eine Variante mit der Abfrage des **ADIF**-Flag.
Speichere das Programm als "**C101_adc_single_conversion_2.asm**".

Free Running Mode

Der ***Free Running Mode*** unterscheidet sich zum einfachen Polling mit **ADIF** dadurch, dass die Wandlung nur einmalig eingeleitet wird. Das Ende einer Wandlung startet automatisch die nächste Wandlung. Dadurch wird um ein Takt schneller gewandelt, da der Vorteiler nicht zurückgesetzt werden muss (siehe Zeitdiagramme im Datenblatt).

Bei der Initialisierung muss das **ADATE**-Bit (*ADc Auto Trigger Enable*) gesetzt werden und der ***Free Running Mode*** im **SFIOR** Register gewählt werden.

Programmbeispiel (Auszug):

```
        ;ADC initialisieren
        ldi     Tmp1,0b01100000 ;REFS = 01 AVCC als Referenzspannung
        out     ADMUX,Tmp1      ;ADLAR = 1 linksb. (obere 8 Bit in ADCH)
                                ;MUX = 00000 unsymetrisch PA0 (ADC0)
        ldi     Tmp1,0b10100111 ;ADEN = 1 (ADC einschalten)
        out     ADCSRA,Tmp1     ;ADATE = 1 (Auto Trigger Modus ein)
                                ;ADPS = 111 (16MHz/128 = 125kHz)
                                ;andere Bits per default = 0
        in      Tmp1,SFIOR      ;ADTS = 000 free running mode
        andi    Tmp1,0b00011111
        out     SFIOR,Tmp1

        ;PORTC (8 Bit) als Ausgang
        ser     Tmp1            ;DDRC = 0b11111111
        out     DDRC,Tmp1
```

```
;-------------------------------------------------------------------------------
;       Hauptprogramm
;-------------------------------------------------------------------------------
MAIN:   sbi     ADCSRA,ADSC     ;AD-Wandlung ausloesen fuer free running mode
                                ;einmaliger Vorgang
LOOP:   sbis    ADCSRA,ADIF     ;Polling ob Wandlung fertig
        rjmp    LOOP
        in      Tmp1,ADCH       ;oberste 8 Bit einlesen und ausgeben
        sbi     ADCSRA,ADIF     ;ADC Interrupt Flag manuell loeschen (mit 1!)
        out     PORTC,Tmp1
        rjmp    LOOP            ;Endlosschleife
```

✎ **C102** a) Teste das obige Programm im ***Free Running Mode***.
Speichere das Programm als "**C102_adc_free_running_1.asm**".

b) Es soll mit Hilfe des Oszilloskop die Wandlungszeit bei allen drei Programmen ermittelt werden. Dazu wird die analoge Eingangsspannung auf Minimum (0 V) eingestellt und das Oszilloskop an Pin **PC0** angeschlossen. Hinter der Ausgabe in den Programmen wird folgende Zeile eingefügt: **sbi PORTC,0**. Bestimme jetzt aus dem Abstand der Impulse die Wandlungszeiten. Notiere und kommentiere die Ergebnisse.

A/D-Wandlung mittels Interrupt (Auto Trigger)

Bei eingeschaltetem ***Auto Trigger***, kann eine von acht Interrupt-Quellen (Trigger-Quellen) eine Wandlung auslösen. Der A/D-Wandler startet eine Wandlung, wenn eine positive Flanke der Interrupt-Quelle erkannt wird. Die Interrupt-Quelle wird mit drei Bit (**ADTS0** - **ADTS2**) im **SFIOR**-Register ausgewählt.

Zusätzlich zu den Initialisierungen beim Polling muss der Interruptvektor initialisiert werden (**ADCCaddr**), der Auto Trigger Modus muss eingeschaltet werden (**ADATE** im SF-Register **ADCSRA**) und die Interrupt-Quelle muss im **SFIOR** ausgewählt werden (**ADTS0** - **ADTS2**).
Der A/D-Wandler-Interrupt muss erlaubt sein (**ADIE** im SF-Register **ADCSRA**) und Interrupts müssen global zugelassen sein (**I** im Statusregister **SREG**).

Free Running Mode

Ein Spezialfall ist der **Free Running Mode**. Hierbei wird kontinuierlich gewandelt. Wurde das Resultat der Wandlung im Datenregister abgelegt, so wird gleich die nächste Wandlung gestartet. Dies geschieht durch die positive Flanke des Interrupt des A/D-Wandlers selbst. Die Wandlung muss nur einmalig mit dem Bit **ADSC** gestartet werden.

Der **Free Running Mode** wird im **SFIOR** aktiviert mit **ADTS = 0b000**.

Die gleiche Aufgabe wie oben soll nun im Free Running-Mode mit Hilfe eines Interrupts programmiert werden.

Programmbeispiel (Auszug):

```
;-------------------------------------------------------------------------------
;       Sprungadressen fuer die Interrupts organisieren (ISR VECTORS)
```

```
;-------------------------------------------------------------------------------
        ;Vektortabelle (im Flash-Speicher)
.ORG    ADCCaddr                ;interner Vektor fuer ADCC (alt.: .ORG 0x0020)
                                ;A/D-Wandlung vollstaendig ADC complete
        rjmp    ISR_AD          ;Springe zur ISR von ADCC

;-------------------------------------------------------------------------------
;       Initialisierungen und eigene Definitionen
;-------------------------------------------------------------------------------
.ORG    INT_VECTORS_SIZE        ;Platz fuer ISR Vektoren lassen
INIT:
.DEF    Zero = r15              ;Register 1 wird zum Rechnen benoetigt
        clr     r15             ;und mit Null belegt
.DEF    Tmp1 = r16              ;Register 16 dient als erster Zwischenspeicher
.DEF    Tmp2 = r17              ;Register 17 dient als zweiter Zwischenspeicher
.DEF    Cnt1 = r18              ;Register 18 dient als Zaehler
.DEF    WL = r24                ;Register 24 und 25 dienen als universelles
.DEF    WH = r25                ;Doppelregister W und zur Parameteruebergabe
...
        ;Stapel initialisieren (fuer Unterprogramme bzw. Interrupts)
        ldi     Tmp1,HIGH(RAMEND)       ;RAMEND (SRAM) ist in der Definitions-
        out     SPH,Tmp1                ;datei festgelegt
        ldi     Tmp1,LOW(RAMEND)
        out     SPL,Tmp1

        ;ADC initialisieren
        ldi     Tmp1,0b01100000 ;REFS = 01 AVCC als Referenzspannung
        out     ADMUX,Tmp1      ;ADLAR = 1 linksb. (obere 8 Bit in ADCH)
                                ;MUX = 00000 unsymetrisch PA0 (ADC0)
        ldi     Tmp1,0b10101111 ;ADEN = 1 (ADC einschalten)
        out     ADCSRA,Tmp1     ;ADATE = 1 (Auto Trigger Modus ein)
                                ;ADIE = 1 (ADC interrupt enable)
                                ;ADPS = 111 (16MHz/128 = 125kHz)
                                ;andere Bits per default = 0
        in      Tmp1,SFIOR      ;ADTS = 000 free running mode
        andi    Tmp1,0b00011111
        out     SFIOR,Tmp1

        ;PORTC (8 Bit) als Ausgang
        ser     Tmp1            ;DDRC = 0b11111111
        out     DDRC,Tmp1

        ;A/D-Wandlung ausloesen (einmalig) Interrupts erlauben
        sbi     ADCSRA,ADSC     ;A/D-Wandlung ausloesen fuer free running mode
        sei                     ;globales Interrupt-Flag setzen (Int. erlauben)

;-------------------------------------------------------------------------------
;       Hauptprogramm
;-------------------------------------------------------------------------------
MAIN:   rjmp      MAIN

;-------------------------------------------------------------------------------
;       Unterprogramme und Interrupt-Behandlungsroutinen
;-------------------------------------------------------------------------------
; Interrupt-Behandlungsroutine ADC
ISR_AD: push    r16             ;benutzte Reg. retten (r16 = Zwischenspeicher)
        in      r16,SREG        ;Statusregister einlesen
        push    r16             ;Statusregister retten

        in      r16,ADCH        ;oberste 8 Bit einlesen und ausgeben
        out     PORTC,r16
        rcall   W1s             ;Anzeige 1* pro Sekunde

        pop     r16             ;Werte der geretteten Register wieder-
        out     SREG,r16        ;herstellen
        pop     r16
        reti                    ;Ruecksprung ins Hauptprogramm aus einer
                                ;Interrupt-Behandlungsroutine
```

✎ **C103** a) Zeichne das dem Programmbeispiel entsprechende Flussdiagramm.
b) Teste das Programm mit einer variablen Spannungsquelle (0-5 V).
Speichere das Programm als "**C103_adc_int_free_running_1.asm**".

Wandlung auslösen mit dem externen Interrupt 0

Bei diesem Fall handelt es sich auch um einen Spezialfall, da gegenüber den anderen sechs Fällen hier kein echter zweiter Interrupt generiert werden muss. Bereits die Initialisierung des SF-Register **MCUCR** bewirkt, dass das Interrupt-Flag (**INTF0**, **INTF1** oder **INTF2**) beim Auftreten einer Flanke am entsprechenden Pin gesetzt wird. Das externe Interrupt-Flag löst dann eine ADC-Wandlung aus.

Wurde der ADC-Interrupt erlaubt, wird dann nach der Wandlung ein ADC-Interrupt ausgelöst.

!! Zum Nutzen des externen Interrupt 0 als Triggerquelle muss nur im SF-Register **MCUCR** festgelegt werden mit welcher Flanke gearbeitet wird!

Die Triggerquelle **INT0** wird im **SFIOR** aktiviert mit **ADTS = 0b010**.

Da hier kein realer externer Interrupt auftritt, wird auch das Flag **INTF0** nicht automatisch gelöscht. Dies muss manuell durch Schreiben einer Eins in der ADC-ISR geschehen.

Eine steigende Flanke am Pin **INT0** (**PD2**) soll eine Wandlung auslösen. Die Spannung am unsymmetrischen Eingang **PA0** wird als 8 Bit-Wert über Port C ausgegeben.

Programmbeispiel (Auszug):

```
;-----------------------------------------------------------------------------
;       Sprungadressen fuer die Interrupts organisieren (ISR VECTORS)
;-----------------------------------------------------------------------------
        ;Vektortabelle (im Flash-Speicher)
.ORG    ADCCaddr                ;interner Vektor fuer ADCC (alt.: .ORG 0x0020)
                                ;AD-Wandlung vollstaendig ADC complete
        rjmp    ISR_AD          ;Springe zur ISR von ADCC

;-----------------------------------------------------------------------------
;       Initialisierungen und eigene Definitionen
;-----------------------------------------------------------------------------
.ORG    INT_VECTORS_SIZE        ;Platz fuer ISR Vektoren lassen
INIT:
.DEF    Zero = r15              ;Register 1 wird zum Rechnen benoetigt
        clr     r15             ;und mit Null belegt
.DEF    Tmp1 = r16              ;Register 16 dient als erster Zwischenspeicher
.DEF    Tmp2 = r17              ;Register 17 dient als zweiter Zwischenspeicher
.DEF    Cnt1 = r18              ;Register 18 dient als Zaehler
.DEF    WL = r24                ;Register 24 und 25 dienen als universelles
.DEF    WH = r25                ;Doppelregister W und zur Parameteruebergabe

        ;Stapel initialisieren (fuer Unterprogramme bzw. Interrupts)
        ldi     Tmp1,LOW(RAMEND)        ;RAMEND (SRAM) ist in der Definitions-
        out     SPL,Tmp1                ;datei festgelegt
        ldi     Tmp1,HIGH(RAMEND)
        out     SPH,Tmp1

        ;ADC initialisieren
        ldi     Tmp1,0b01100000 ;REFS = 01 AVCC als Referenzspannung
        out     ADMUX,Tmp1      ;ADLAR = 1 linksbuendig (obere 8 Bit in ADCH)
```

```
                                ;MUX = 00000 unsymetrisch PA0 (ADC0)
        ldi     Tmp1,0b10101111 ;ADEN = 1 (ADC einschalten)
        out     ADCSRA,Tmp1     ;ADATE = 1 (Auto Trigger Modus aktivieren)
                                ;ADIE = 1       (ADC interrupt enable)
                                ;ADPS = 111 (16MHz/128 = 125kHz)
                                ;andere Bits per default = 0
        in      Tmp1,SFIOR      ;ADTS = 010 external interrupt request 0
        andi    Tmp1,0b01011111
        ori     Tmp1,0b01000000
        out     SFIOR,Tmp1

        ;INT0 steigende Flanke initialisieren
        ;(INT0 muss nicht gesondert ueber GICR aktiviert werden!!)
        in      Tmp1,MCUCR      ;steigende Flanke von INT0 initialisieren
        ori     Tmp1,0b00000011
        out     MCUCR,Tmp1

        ;PORTC (8 Bit) als Ausgang
        ser     Tmp1            ;DDRC = 0b11111111
        out     DDRC,Tmp1

         ;Interrupts global erlauben
         sei                     ;globales Interrupt-Flag setzen (Int. erlauben)

;-----------------------------------------------------------------------------
;       Hauptprogramm
;-----------------------------------------------------------------------------
MAIN:   rjmp    MAIN            ;Endlosschleife

;-----------------------------------------------------------------------------
;       Unterprogramme und Interrupt-Behandlungsroutinen
;-----------------------------------------------------------------------------
; Interrupt-Behandlungsroutine ADC
ISR_AD: push    r16             ;benutzte Reg. retten (r16 = Zwischenspeicher)
        in      r16,SREG        ;Statusregister einlesen
        push    r16             ;Statusregister retten

        in      r16,ADCH        ;oberste 8 Bit einlesen und ausgeben
        out     PORTC,r16

        in      r16,GIFR        ;INTF0-Bit im GIFR manuell loeschen, (mit 1!)
        ori     r16,0b01000000 ;damit erneutes Interrupt erkannt wird!
        out     GIFR,r16

        pop     r16             ;Werte der geretteten Register wieder-
        out     SREG,r16        ;herstellen
        pop     r16
        reti                    ;Ruecksprung ins Hauptprogramm aus einer
                                ;Interrupt-Behandlungsroutine
```

✎ **C104** a) Zeichne das dem Programmbeispiel entsprechende Flussdiagramm.
b) Teste das Programm mit einer variablen Spannungsquelle (0-5 V) .
Speichere das Programm als "**C104_adc_int_int0.asm**".

Wandlung auslösen mit dem Timer 0

Hier soll als Beispiel noch das Auslösen eines Interrupt im Millisekundenrythmus mit dem Timer 0 demonstriert werden. Auf dieses Beispiel kann man zurückkommen, wenn man das Kapitel zum Timer durchgearbeitet hat.

Es werden dazu Timer 0 und Timer 2 im CTC-Modus verwendet. Mit einem Quarz von 16 MHz und einem Teiler von 64 benötigt man 250 Zählschritte um eine Interruptfrequenz von 1000 Hz (t = 1 ms) zu erhalten.

Weitere Aufgaben

✎ **C105** Erweitere das erste Programm um eine Dual-BCD Wandlung. Dazu kann das unten stehende Unterprogramm zum Teilen durch 100 und durch 10 verwendet werden. Der Spannungswert soll als Vielfaches von 10 mV an den drei untersten Stellen des Siebensegment-Displays ausgegeben werden.
Der Einfachheit halber wird der gewandelte 8-Bit-Wert (0-255) mit Zwei multipliziert (Befehl **lsl**). Der dabei gemachte Fehler (0-510 statt 0-500) liegt nur bei zwei Prozent.
Teste das Programm mit einer variablen Spannungsquelle (0-5 V). Speichere das Programm als "**C105_adc_voltmeter.asm**".

```
;-----------------------------------------------------------------------------
;       Ganzzahldivision 2Byte / 1Byte
;-----------------------------------------------------------------------------

;Dividend:       r1,r0 (LSB)
;Divisor         r2
;Schleifenzähler r16
;Resultat Rest in r1 + Quotient in r0
;Achtung Ueberlauf wenn obere n Bit des Dividenden groesser gleich n Bit Divisor

DIV21:  push    r16
        in      r16,SREG
        push    r16

        ldi     r16,8           ;Schleifenzaehler init. (Bitzahl Quotient)

DIV21A: lsl     r0              ;Rotiere Dividend 1 Stelle nach links
        rol     r1              ;rechts wird Null reingeschoben
        brcs    DIV21B          ;Falls Carry muss Subtraktion erfolgen!
        cp      r1,r2           ;Subtraktion testen
        brlo    DIV21C          ;Ueberspringe Subtraktion, wenn kleiner

DIV21B: sub     r1,r2           ;Subtraktion durchfuehren
        inc     r0              ;Bit 0 im Dividenden setzen

DIV21C: dec     r16             ;Schleifenzaehler dekrementieren
        brne    DIV21A

        pop     r16
        out     SREG,r16
        pop     r16
        ret
```

Bemerkung: Zum Erhöhen der Genauigkeit kann noch die oberste Stelle (0-5) mit 2 multipliziert werden (Befehl **lsl**) und dann vom gesamten Wert subtrahiert werden.

✎ **C106** Konfiguriere und programmiere den ATmega32A so, dass sein A/D-Wandler im *Free Running Mode* mit Interrupt arbeitet. Das Assemblerprogramm soll so aufgebaut sein, dass die Spannung an **PA0** (ADC0) gemessen und ihre Höhe über eine Bargraph-Anzeige (Balkenanzeige) ausgegeben wird. Die Bargraph-Anzeige soll mit Hilfe von den 8 LEDs an Port C gebildet werden. Dazu ist es notwendig die Spannung in einen 3 Bit-Wert umzuwandeln (einfach die hochwertigsten drei Bits verwenden!). Benutze zur Programmierung der Bargraph-Anzeige eine Tabelle.
Schließe an **PA0** ein Potentiometer an und überprüfe die Funktionsweise

deines Programms. Bei minimaler Spannung (linker Anschlag) darf nur eine LED leuchten, bei maximaler Spannung (rechter Anschlag) sollen alle LEDs leuchten. Speichere das Programm als "**C106_adc_bargraph.asm**".

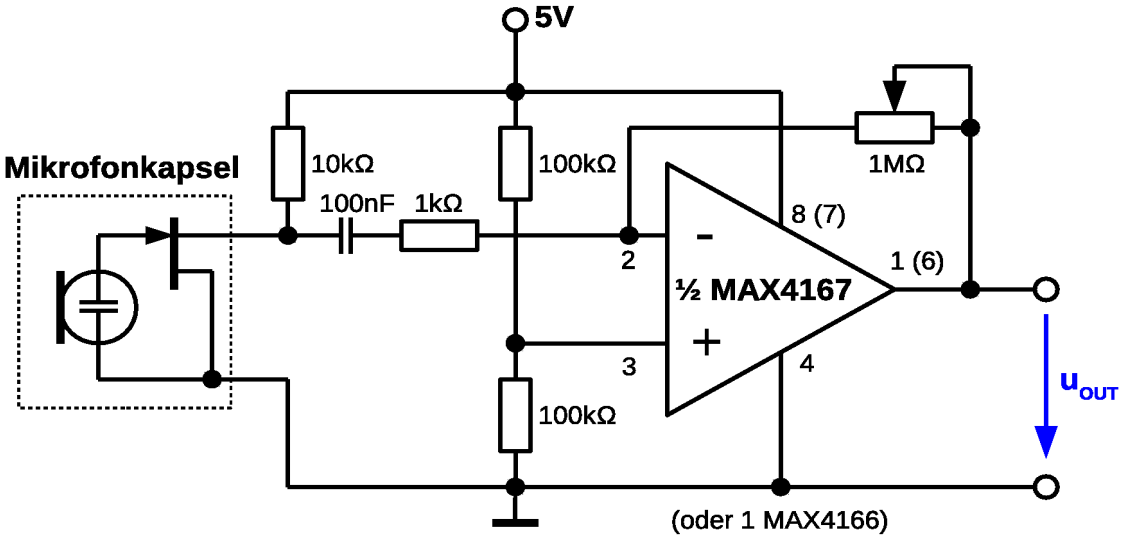
✎ **C107** Entwickle für den ATmega32A ein Assemblerprogramm, das an **PA2** (ADC2) die Spannung misst, wenn an **INT0** eine fallende Signalflanke erscheint. Der umgewandelte Spannungswert (10 Bit) soll mit Hilfe eines Unterprogramms in ein String (mit ASCII-Zeichen) umgewandelt werden und dann über die serielle Schnittstelle mit 19200 Baud (8N1) zu einem PC gesendet werden. Das Unterprogramms mit den symbolischen Namen **B16_STR** wandelt eine 16 Bit-Zahl in einen String mit einer festen Länge von 5 ASCII-Zeichen (00000-65536). Speichere das Programm als "**C107_adc_to_EIA232.asm**".

✎ **C108** Ähnliche Aufgabe wie die vorige Aufgabe, nun soll der String allerdings in mittels Interrupt über die serielle Schnittstelle versendet werden. Speichere das Programm als "**C108_adc_to_EIA232_2.asm**".

✎ **C109** Im Single Conversion Mode soll abwechselnd die Spannung an **PA0** (ADC0) und an **PA1** (ADC1) in einen 8 Bit-Wert umgewandelt und über Port C (für ADC0) bzw. Port B (für ADC1) ausgegeben werden. Entwickle das hierzu notwendige Assemblerprogramm für den ATmega32A. Speichere das Programm als "**C109_adc_dual_conversion.asm**".

✎ **C10A**
a) Baue das folgende Modul um Sprache zu digitalisieren. Das Signal eines Elektret-Kondensatormikrofons[10] wird mittels eines OPV als Differenzverstärker in eine Spannung zwischen 0 und 5 V umgesetzt. Es wird ein sogenannter *rail-to-rail* OPV verwendet, der den gesamten Bereich seiner Versorgungsspannung nutzen kann.
b) Beschreibe ausführlich die Funktionsweise der Schaltung (nutze die Formeln des OPV als Differenzverstärker!)
c) Teste die Schaltung ausgiebig mit den obigen Programmen.

10 Das Elektretmikrofon ist ein Kondensatormikrofon mit einer Elektretfolie. Meist befindet sich in der Mikrofonkapsel noch ein Mikrofonverstärker (FET). Zum Betrieb reicht eine Spannung von ungefähr 1,5 Volt bei 1 mA. Das Elektretmikrofon hat Kugelcharakteristik und deckt den Frequenzbereich von 20 Hz bis 20 kHz ab. Wegen seiner kompakten Bauweise wird er praktisch in allen mobilen Geräten eingesetzt.

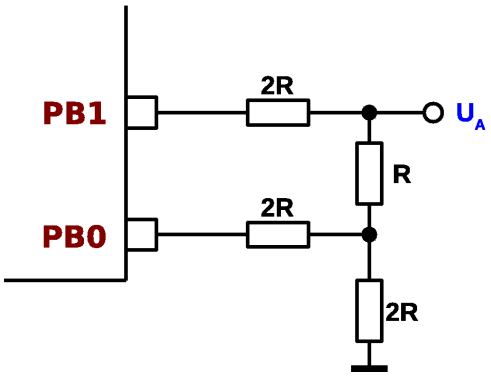


D/A-Wandler

Damit digitale Informationen mittels D/A-Wandlern umgesetzt werden können müssen sie in einem gewichteten Code vorliegen. Der Dualcode (binäres Zahlensystem) ist ein solch gewichteter Code ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$ ...). Umso mehr Bits verwendet werden umso höher ist die Auflösung des D/A-Wandlers. Mit 8 Bits kann ein analoges Signal mit 256 unterschiedlichen Spannungswerten erzeugt werden. Umso höher die Auflösung, umso kleiner wird die Treppenstufung des analogen Signals.

Das R-2R-Netzwerk

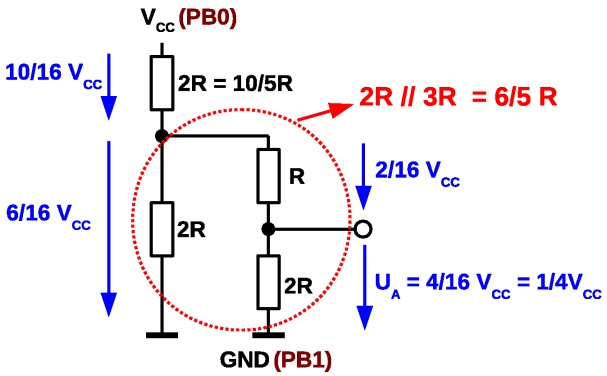
Ein D/A-Wandler lässt sich sehr einfach mit einem sogenannten R-2R-Netzwerk realisieren. Ein solches Netzwerk besitzt nur Widerstände mit den Werten R und 2R. Im folgenden soll ein solches Netzwerk für zwei Stellen des Dualcodes untersucht werden:



1 Fall:
PB1 = 0 und **PB0 = 0**
Die Ausgangsspannung wird in diesem ersten Fall 0 V sein, da keine Spannung anliegt.

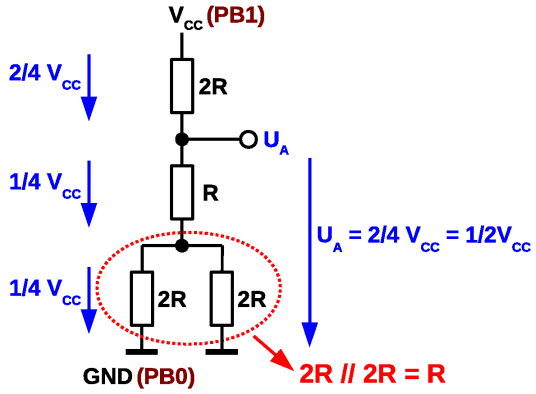
2 Fall:
PB1 = 0 und **PB0 = 1**

An **PB0** = 1 liegt die Betriebsspannung des Controllers an. Es ergibt sich folgende Schaltung:



Die Parallelschaltung von 3R (R+2R) mit 2R ergibt einen Gesamtwiderstand von 6/5 R. Auf diesen fallen in der Reihenschaltung mit 2R = 10/5 R also 6 Anteile der Spannung von 16. Davon tragen dann 2/3 zur Ausgangsspannung bei. Die Ausgangsspannung beträgt ¼ der Gesamtspannung.

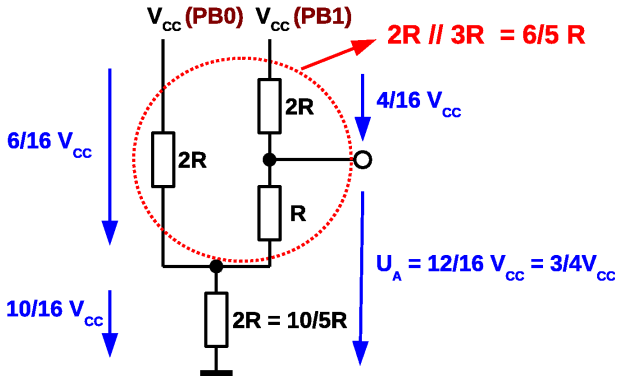
3 Fall:
PB1 = 1 und **PB0 = 0**



Die Spannung wird geteilt und die Hälfte der Betriebsspannung liegt am Ausgang.

4 Fall:
PB1 = 1 und **PB0 = 1**

Die Spannung teilt sich ähnlich wie im 2. Fall auf. Es liegen dann ¾ der Betriebsspannung am Ausgang an.



Damit ergibt sich folgende Tabelle:

Fall	Dezimalwert	PB1 (2^1)	PB0 (2^0)	U_A
1	0	0	0	0
2	1	0	1	¼ V_{CC}
3	2	1	0	½ V_{CC}
4	3	1	1	¾ V_{CC}

Jedes Bit trägt seinen Teil zur resultierenden Ausgangsspannung bei! Die höchste Stelle PB1 (2^1) steuert die Hälfte der Betriebsspannung bei. Die niedrigere Stelle ein Viertel.

Erweitert man das Netzwerk zum Beispiel auf drei Bit, so ergibt sich folgende Tabelle:

Fall	Dezimalwert	PB2 (2^2)	PB1 (2^1)	PB0 (2^0)	U_A
1	0	0	0	0	0
2	1	0	0	1	1/8 V_{CC}
3	2	0	1	0	1/4 V_{CC}
4	3	0	1	1	3/8 V_{CC}
5	4	1	0	0	1/2 V_{CC}
6	5	1	0	1	5/8 V_{CC}
7	6	1	1	0	3/4 V_{CC}
8	7	1	1	1	7/8 V_{CC}

Eine dritte Stelle wird also ein Achtel der Betriebsspannung beisteuern. Ein solches Netzwerk lässt sich beliebig ausbauen.

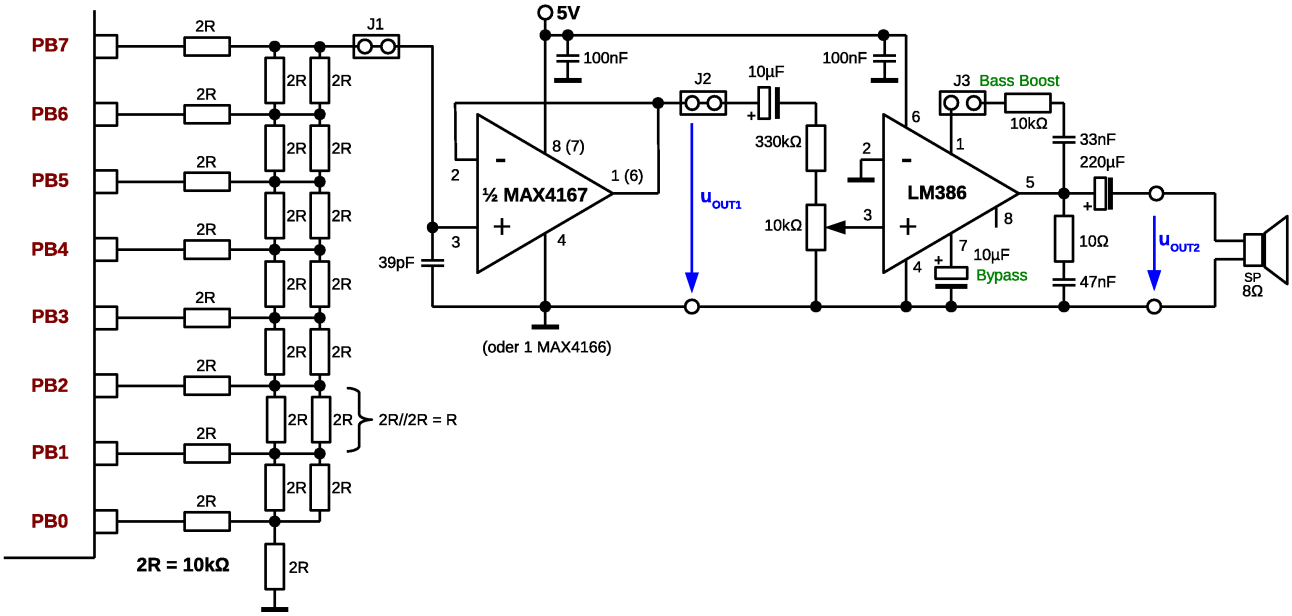
Die **kleinste Änderung** der Ausgangsspannung lässt sich dann mit Hilfe der folgenden Gleichung berechnen :

$$\Delta U_{Amin} = \frac{1}{2^n} \cdot V_{CC}$$

Ein solches R-2R-Netzwerkes hat einen konstanten Innenwiderstand $R_i = R$, egal für wie viele Stellen es ausgelegt wurde. Eine Belastung des Ausgangs bewirkt also keine Verfälschung des Resultats. Nur die maximal abgreifbare Spannung sinkt.

Um in der Praxis ein R-2R-Netzwerk zu bauen braucht man präzise Widerstandswerte (E96-Reihe, 1 %). Mit einer Parallelschaltung kann der Widerstandswert halbiert werden. Um die Ausgänge des Controllers nicht zu sehr zu belasten, sollen die Widerstandswerte nicht kleiner als 10 kΩ sein.

Ein nachgeschalteter Operationsverstärker minimiert die Belastung des Netzwerks und liefert den nötigen Strom. Um den ganzen Spannungsbereich zu nutzen (GND – VCC) wird ein „*rail to rail*“ OPV einzusetzen, der bis zu den Grenzen seiner Betriebsspannung arbeiten kann. Der nachgeschaltete Verstärker (LM386) erlaubt es einen Lautsprecher anzuschließen.



Bemerkungen: Natürlich gibt es Digital-Analog-Wandler mit R-2R-Netzwerke auch in integrierter Form.

Eine andere und einfachere Möglichkeit ein analoges Signal zu erzeugen ist die Pulsweitenmodulation (siehe Timer).

Weitere Aufgaben

✎ **C10B** Berechne die kleinste Spannungsänderung für den obigen 8-Bit-D/A-Wandler. In welcher Schaltung wird der Operationsverstärker betrieben? Ermittle die Verstärkung und die Funktion Bass Boost (J3) aus dem Datenblatt des LM386.

✎ **C10C** Programmiere einen kleinen Funktionsgenerator. Über zwei Schalter S0 und S1 soll die Wellenform ausgewählt werden können (Sinus, Dreieck, Sägezahn und Rechteck). Der Sinus wird mit der folgenden Tabelle erzeugt (Datei auf weigu.lu/a/asm). Sinus, Dreieck und Sägezahn werden mit der höchstmöglichen Frequenz ausgegeben. Beim Rechteck kann eine beliebige Zeitschleife eingefügt werden.

a) Zeichne das dem Programmbeispiel entsprechende Flussdiagramm.

b) Schreibe das Assemblerprogramm und speichere es unter dem Namen **"C10C_DAC_wavegen.asm"**.

c) Ermittle mit Hilfe eines Oszilloskops oder eines Frequenzzählers die maximalen Frequenzen für Sinus, Dreieck und Sägezahn.

d) Die Ausgabefrequenz aller Wellenformen soll nun auf 440 Hz verringert werden. Bestimme die nötige Zeitverzögerungen zwischen zwei Ausgaben und erweitere entsprechend dein Programm.

Sinustabelle:

```
.DB   128, 131, 134, 137, 140, 143, 146, 149
.DB   152, 155, 158, 162, 165, 167, 170, 173
.DB   176, 179, 182, 185, 188, 190, 193, 196
.DB   198, 201, 203, 206, 208, 211, 213, 215
.DB   218, 220, 222, 224, 226, 228, 230, 232
.DB   234, 235, 237, 238, 240, 241, 243, 244
.DB   245, 246, 248, 249, 250, 250, 251, 252
.DB   253, 253, 254, 254, 254, 255, 255, 255
.DB   255, 255, 255, 255, 254, 254, 254, 253
.DB   253, 252, 251, 250, 250, 249, 248, 246
.DB   245, 244, 243, 241, 240, 238, 237, 235
.DB   234, 232, 230, 228, 226, 224, 222, 220
.DB   218, 215, 213, 211, 208, 206, 203, 201
.DB   198, 196, 193, 190, 188, 185, 182, 179
.DB   176, 173, 170, 167, 165, 162, 158, 155
.DB   152, 149, 146, 143, 140, 137, 134, 131
.DB   128, 124, 121, 118, 115, 112, 109, 106
.DB   103, 100, 97, 93, 90, 88, 85, 82
.DB   79, 76, 73, 70, 67, 65, 62, 59
.DB   57, 54, 52, 49, 47, 44, 42, 40
.DB   37, 35, 33, 31, 29, 27, 25, 23
.DB   21, 20, 18, 17, 15, 14, 12, 11
.DB   10, 9, 7, 6, 5, 5, 4, 3
.DB   2, 2, 1, 1, 1, 0, 0, 0
.DB   0, 0, 0, 0, 1, 1, 1, 2
.DB   2, 3, 4, 5, 5, 6, 7, 9
.DB   10, 11, 12, 14, 15, 17, 18, 20
.DB   21, 23, 25, 27, 29, 31, 33, 35
.DB   37, 40, 42, 44, 47, 49, 52, 54
.DB   57, 59, 62, 65, 67, 70, 73, 76
.DB   79, 82, 85, 88, 90, 93, 97, 100
.DB   103, 106, 109, 112, 115, 118, 121, 124
```

✎ **C10D** Für Fortgeschrittene:
Die zwölf Tasten einer Matrixtastatur sollen dazu dienen Musik zu erzeugen. (siehe Tabelle). Erweitere das obige Programm, so dass die erwünschten Töne in einer beliebigen Wellenform anhand der Tasten erzeugt werden können. Zur Abfrage der Tastatur kann das Unterprogramm "**SR_3x4_KEYPAD.asm**"

verwendet werden (siehe Aufgabe B50D).
Schreibe das Assemblerprogramm und speichere es unter dem Namen **"B10D_DAC_tonegen.asm"**.

	f (Hz)	f (Hz)
Do (C)	131	262
Re (D)	147	294
Mi (E)	165	330
Fa (F)	175	349
Sol (G)	196	392
La (A)	220	
Si (H, B)	247	

Tipps: Die Ausgabe aller Wellenform aus 256 Byte großen Flash-Tabellen erleichtert die Aufgabe.

Es muss ein Lautsprecher verwendet werden, der diese tiefen Töne (Sinus hat keine Oberwellen) auch ausgeben kann!

✎ **C10E**

a) Mit Hilfe des Electret-Mikrofons und des A/D-Wandlers sollen jetzt Signale digitalisiert werden, und dann gleich wieder mit dem D/A/Wandler über den Lautsprecher ausgegeben werden.
Schreibe das Assemblerprogramm und speichere es unter dem Namen **"B10E_ADC_DAC.asm"**.

b) Vervollständige das folgende Programm mit den entsprechenden Initialisierungen und beschreibe ausführlich seine Funktionsweise. Speichere es unter dem Namen **"B10E_ADC_DAC_delay.asm"**.

```
;---------------------------------------------------------------------------
;       Organisation des Datenspeichers (SRAM)
;---------------------------------------------------------------------------
.DSEG                           ;was ab hier folgt kommt in den SRAM-Speicher
TAB:  .BYTE  1800               ;1800 Byte grosse Tabelle im Datensegment
```

```
MAIN:   sbi     ADCSRA,ADSC     ;AD-Wandlung ausloesen fuer free running mode
                                ;einmaliger Vorgang
LOOP1:  ldi     XL,LOW(TAB)
        ldi     XH,HIGH(TAB)
        ldi     WL,LOW(1800)
        ldi     WH,HIGH(1800)

LOOP2:  sbis    ADCSRA,ADIF     ;Polling bis Wandlung fertig
        rjmp    LOOP2
        in      Tmp1,ADCH       ;oberste 8 Bit einlesen und ausgeben
        sbi     ADCSRA,ADIF     ;ADC Interrupt Flag manuell loeschen (mit 1!)
        st      X+,Tmp1

        sbiw    WL,1
        breq    LOOP1

        ld      Tmp1,X
        out     PORTB,Tmp1
        rjmp    LOOP2           ;Endlosschleife
```