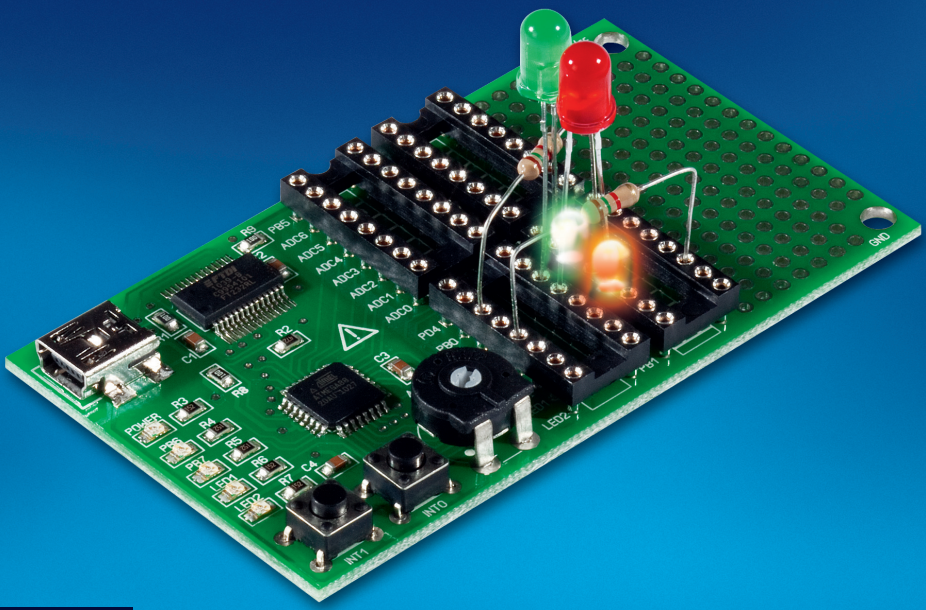
Das Franzis Lernpaket

# Microcontroller in C programmieren




FRANZIS

# Inhaltsverzeichnis

# 1 CD-ROM zum Lernpaket

Diesem Lernpaket liegt eine CD-ROM bei, die verschiedene Programme, Tools, Datenblätter und Beispiele enthält. Sie wird Ihnen das Arbeiten mit diesem Lehrgang erleichtern. Alle hier abgedruckten Beispiele sind auf der CD-ROM enthalten.

## 1.1 GPL (General Public License)

Sie können Ihre eigenen Programme selbstverständlich mit anderen Anwendern über das Internet austauschen. Die Beispielprogramme stehen unter der Open-Source-Lizenz *GPL* (General Public License). Daher sind Sie berechtigt, die Programme unter den Bedingungen der GPL zu modifizieren, zu veröffentlichen und anderen Anwendern zur Verfügung zu stellen, sofern Sie Ihre Programme dann ebenfalls unter die GPL-Lizenz stellen.

## 1.2 Systemvoraussetzung

Betriebssystem:

- Windows XP (x86) Service Pack 3
- Windows Vista (x86) Service Pack 1, Service Pack 2
- Windows XP (x64) Service Pack 2
- Windows Vista (x64) Service Pack 1, Service Pack 2
- Windows 7 (x86 und x64)
- Windows Server 2003 R2 (x86 und x64)

Hardware:

- Computer mit mindestens 1,6 GHz oder schneller
- 1 GB RAM für x86 CPU
- 2 GB RAM für x64 CPU
- Mindestens 512 MB RAM
- 3 GB freier Festplattenspeicher
- 5.400-RPM-Festplatte
- DirectX-9-kompatible Grafikkarte und 1.024 x 768 oder höhere Bildschirmauflösung
- Internetanschluss

## 1.3 Updates und Support

Das Atmel Atmel Studio (derzeit Version 6) wird ständig weiterentwickelt. Updates können Sie kostenlos von der Website des Herstellers laden (*www.atmel.com* – es fallen nur Ihre üblichen Online-Kosten an). Das ist besonders bequem mit der in das Atmel Studio integrierten Autoupdate-Funktion möglich.

## 9.12 Digitalvoltmeter mit Average Digitalfilter

Da wir jetzt in der Lage sind, Daten an den PC zu senden und zu empfangen, verwenden wir unser Mikrocontrollerboard gleich als Messgerät für elektrische Spannungen. Unser Controller enthält dazu einen Analog-digital-Wandler (Analog Digital Converter, ADC) mit einer Auflösung von 10 Bit. D. h., die Spannung wird 2 ^ 10 digitalisiert, was 1.024 Stufen (Steps) entspricht. Ohne Spannungsteiler an einem ADC-Eingang kann man Spannungen von 0–5 V bzw. 0 bis $V_{CC}$ direkt messen. Das entspricht einem digitalen Messwert von 0–1023 und einer Auflösung von 5 V / 1.023 = 0,00488 V oder 4,88 mV. Unser Controller besitzt acht analoge Eingänge (ADC0–ADC7), um elektrische Spannungen zu messen. Im Controller selbst ist nur ein Analog-digital-Wandler verbaut, der mit einem Multiplexer (umschaltbarer Verteiler) verbunden ist. Dazu müssen wir den Multiplexer in der Software immer erst auf den gewünschten Pin (ADC-Eingang) am Controller umschalten, bevor wir eine Messung durchführen können.

**Die Auflösung kann man mit dieser einfachen Formel leicht selbst ausrechnen:**

$U_{step} = U_{ref}$ / Auflösung

$U_{step} = 5$ V / 1.023

$U_{step} = 0{,}00488$V

Den digitalen Wert kann man folgendermaßen berechnen:

Anzeigewert (Raw) = 1.023 x anliegende Spannung (ADC) / $U_{ref}$

Die Genauigkeit schwankt zwischen +/-2 Messschritten (Steps). Bei 5 V Referenzspannung liegt die Genauigkeit also bei +/-0,0097 V. Man kann somit sagen, dass der ADC bei 5 V Referenzspannung eine angelegte Spannung auf zwei Kommastellen genau misst. Voraussetzung ist natürlich eine stabile Referenzspannung.

Als Referenzspannung kann man jede Spannung zischen 2 V und 5 V am AVCC-Eingang verwenden. Bei unserem Experimentier-Board verwenden wir durch die bereits auf dem Board vorhandene Schaltung die Betriebsspannung des USB-Anschlusses, die für experimentelle Zwecke ausreichend genau ist.

**Technische Daten des ATmega88 ADC:**

- 10 Bit Auflösung
- 0,5 LSB Linearitätsfehler
- ± 2 LSB Genauigkeit
- 13 µs bis 260 µs Wandlungszeit
- 0 V–$V_{CC}$ ADC-Eingangsspannung
- Auswählbare 1,1 V ADC interne Spannungsreferenz
- Referenzspannung mindestens: 2 V
- Eingangswiderstand des Referenzspannungs-Pins (AREF): 32 kΩ
- Eingangswiderstand am zu messenden Pin (ADC): min. 55 MΩ, typisch 100 MΩ

| Symbol | Parameter | Condition | Minimum | Typical | Maximum | Units |
|---|---|---|---|---|---|---|
| | Resolution | | | 10 | | Bits |
| | Absolute accuracy (Including INL, DNL, quantization error, gain and offset error) | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | | 2 | | LSB |
| | | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 1MHz | | 4.5 | | |
| | | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz Noise reduction mode | | 2 | | |
| | | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 1MHz Noise reduction mode | | 4.5 | | |
| | Integral non-linearity (INL) | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | | 0.5 | | |
| | Differential non-linearity (DNL) | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | | 0.25 | | |
| | Gain error | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | | 2 | | |
| | Offset error | $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200kHz | | 2 | | |
| | Conversion time | Free running conversion | 13 | | 260 | µs |
| | Clock frequency | | 50 | | 1000 | kHz |
| $AV_{CC}$(1) | Analog supply voltage | | $V_{CC}$ - 0.3 | | $V_{CC}$ + 0.3 | V |
| $V_{REF}$ | Reference voltage | | 1.0 | | $AV_{CC}$ | |
| $V_{IN}$ | Input voltage | | GND | | $V_{REF}$ | |
| | Input bandwidth | | | 38.5 | | kHz |
| $V_{INT}$ | Internal voltage reference | | 1.0 | 1.1 | 1.2 | V |
| $R_{REF}$ | Reference input resistance | | | 32 | | kΩ |
| $R_{AIN}$ | Analog input resistance | | | 100 | | MΩ |

Note: 1. $AV_{CC}$ absolute min./max.: 1.8V/5.5V

**Abb. 9.43:** Auszug aus dem Datenblatt zu den Electrical characteristics des ADC. Deutlich sind die Fehler und die Genauigkeit des ADC zu sehen. Man muss aber immer im Hinterkopf behalten, dass der Hersteller diese Angaben gern etwas beschönigt und die realen Daten geringfügig schlechter ausfallen können.

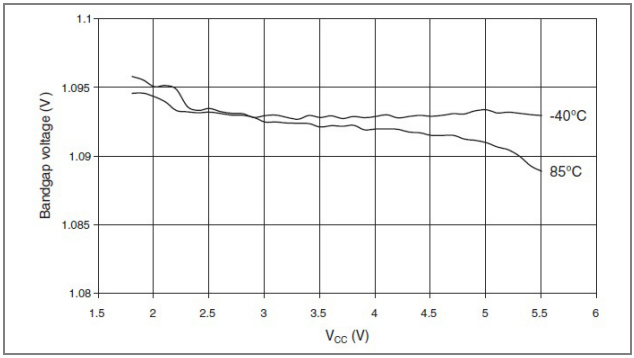



**Abb. 9.44:** Verwendet man nicht $V_{CC}$ (Versorgungsspannung des µC), steht eine interne Referenzdiode zur Verfügung. Man sieht deutlich das Verhalten der Referenzspannung im Zusammenhang von $V_{CC}$ und Umgebungstemperatur. Die Referenzdiode ist hier nicht unbedingt eine Präzisionsdiode, aber für die meisten Anwendungen völlig ausreichend. Da es sich um eine 1,1-V-Referenz handelt, können am ADC ohne zusätzliche Beschaltung auch nur maximal 1,1 V gemessen werden. Spannungen zwischen 0 V und 1,1 V werden mit 0–1023 dargestellt.

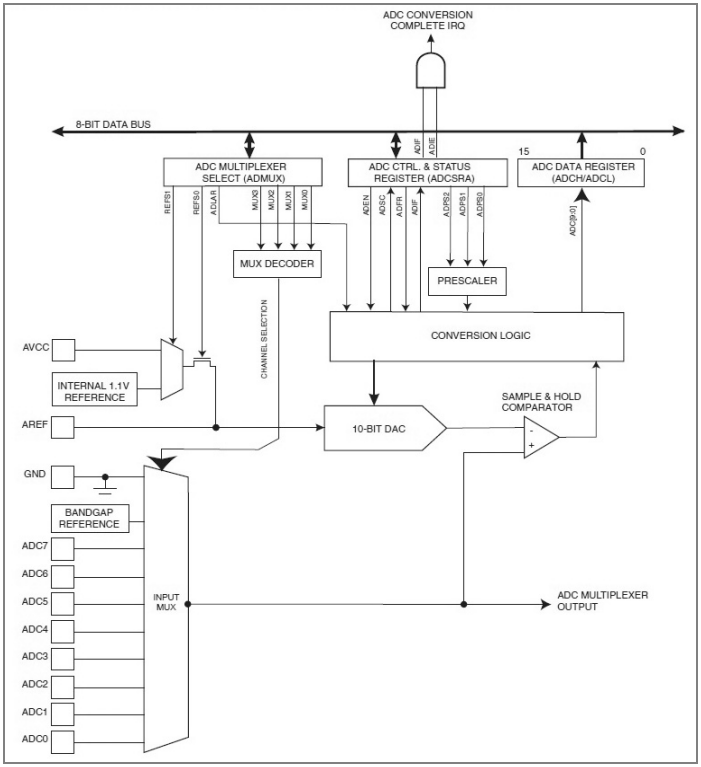



**Abb. 9.45:** Hier sehen wir den internen Aufbau des ADC. Oben links befindet sich der Multiplexer, rechts daneben das Control- und Statusregister zur Konfiguration des ADC. Oben rechts ist das Datenregister. In ihm wird der Messwert abgelegt. Links in der Mitte ist die auswählbare Spannungsreferenz zu sehen. Man kann zwischen AVCC, interner Referenz mit 1,1 V und externer Referenz auswählen. Unten links sehen wir die physikalischen Eingänge des ADC, die direkt auf den Multiplexer gehen.

**Hinweis**
Im Atmel-Datenblatt zum Controller finden Sie noch mehr Informationen zum ADC. Ein Blick lohnt sich immer.

Das folgende Beispiel misst den Analogwert am Analogeingang 7 (ADC7) und gibt am Terminal den digitalen Wert (RAW-Wert 0–1023) und die Spannung aus. Die Spannung wird berechnet und mit den bereits bekannten Formatierungsfunktionen am Terminal ausgegeben. Das Terminal müssen Sie wie folgt einstellen, um eine übersichtliche Ausgabe zu erhalten:


**Abb. 9.46:** *ASCII / Newline at CR-LF /* Häkchen von *Show newline charakter* entfernen; diese Einstellung empfiehlt sich auch für die anderen Beispiele, da so die Ausgabe deutlich übersichtlicher ist.

**Beispiel:** Digitalvoltmeter

```
#define F_CPU 8000000

// Baudrate
#define BAUD 19200UL

// Baudratenberechnung
#define UBRR_VAL ((F_CPU+BAUD*8) / (BAUD*16)-1)

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

// UART-Init ATmega88
void uart_init(void)
{
   // Baudrate einstellen
   UBRR0H = (unsigned char)(UBRR_VAL>>8);
   UBRR0L = (unsigned char)UBRR_VAL;

   // Receiver und Transmitter einschalten
   UCSR0B = (1<<RXEN0)|(1<<TXEN0);

   // Frame Format: Asynchron 8N1
   UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
}

// Charakter schreiben
void uart_putChar(unsigned char c)
{
   // Warten, bis Sendepuffer leer ist
   while (!(UCSR0A & (1<<UDRE0)));
```

```
   // Daten in den Puffer schreiben und senden
   UDR0 = c;
}

// String senden
void uart_putStr(char *s)
{
   while (*s)
   {   // solange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)"
       uart_putChar(*s);
       s++;
   }
}

// ADC initialisieren
void ADC_Init(void)
{
   unsigned int result;

   //ADMUX = (0<<REFS1) | (0<<REFS0);     // AREF, internal Vref off
   ADMUX = (0<<REFS1) | (1<<REFS0);       // AVCC als Referenz benutzen
   //ADMUX = (1<<REFS1) | (1<<REFS0);     // interne Referenzspannung nutzen

   // Bit ADFR ("free running") in ADCSRA steht beim Einschalten
   // schon auf 0, also single conversion


   //ADCSRA = (0<<ADPS2) | (0<<ADPS1) | (1<<ADPS0);  // Frequenzvorteiler 2
   //ADCSRA = (0<<ADPS2) | (1<<ADPS1) | (0<<ADPS0);  // Frequenzvorteiler 4
   ADCSRA = (0<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);    // Frequenzvorteiler 8
   //ADCSRA = (1<<ADPS2) | (0<<ADPS1) | (0<<ADPS0);  // Frequenzvorteiler 16
   //ADCSRA = (1<<ADPS2) | (0<<ADPS1) | (1<<ADPS0);  // Frequenzvorteiler 32
   //ADCSRA = (1<<ADPS2) | (1<<ADPS1) | (0<<ADPS0);  // Frequenzvorteiler 64
   //ADCSRA = (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);  // Frequenzvorteiler 128

   ADCSRA |= (1<<ADEN);    // ADC aktivieren

   // nach Aktivieren des ADC wird ein "Dummy-Readout" empfohlen, man liest
   // also einen Wert und verwirft diesen, um den ADC "warmlaufen" zu lassen

   ADCSRA |= (1<<ADSC);           // Single ADC-Wandlung Start
   while (ADCSRA & (1<<ADSC));    // auf Abschluss der Konvertierung warten

   // ADCW muss einmal gelesen werden,
   // sonst wird Ergebnis der nächsten Wandlung nicht übernommen.
   result = ADCW;
}

// ADC-Einzelmessung
unsigned int ADC_Read(unsigned char channel )
{
   // Kanal wählen, ohne andere Bits zu beeinflussen
```

```
   ADMUX = (ADMUX & ~(0x1F)) | (channel & 0x1F);
   ADCSRA |= (1<<ADSC);        // eine Wandlung "single conversion"
   while (ADCSRA & (1<<ADSC))  // auf Abschluss der Konvertierung warten
   return ADCW;                // ADC auslesen und zurückgeben
}

// ADC-Mehrfachmessung mit Mittelwertbbildung
unsigned int ADC_Read_Avg(unsigned char channel, unsigned char average)
{
   unsigned long result = 0;

   for (unsigned char i = 0; i < average; ++i)
   {
      result += ADC_Read(channel);
      _delay_us(250);
   }

   return (unsigned int)(result / average);
}

void Wait(unsigned int dly_ms)
{
   while(dly_ms)
   {
      dly_ms--;
      _delay_ms(1);
   }
}

// Hauptprogramm
int main(void)
{
   unsigned char str[10];
   unsigned int raw;
   float volt;
   const float volt_convert = 5.0/1023.0;

   // ADC initialisieren
   ADC_Init();

   // Endlosschleife
   while(1)
   {
      // Mittelwertmessung
      raw = ADC_Read_Avg(7,64);

      // Spannungsumrechnung
      volt = raw * volt_convert;

      // ADC-Wert ausgeben
      uart_putStr("ADC7 RAW: ");
```

```
      utoa(raw,str,10);
      uart_putStr(str);
      uart_putStr("\r\n");

      // Spannungswert ausgeben
      uart_putStr(dtostrf(volt, 1, 3, str));
      uart_putStr(" V");
      uart_putStr("\r\n");

      Wait(1000); // Sekunde warten

   }
}
```

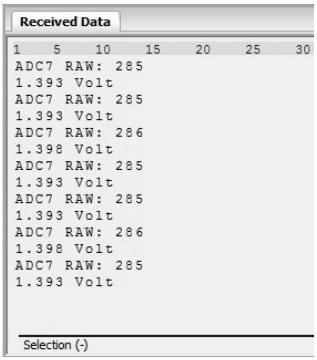



**Abb. 9.47:** So sollte die Ausgabe des Beispiels bei Ihnen aussehen.

Zuerst wird der RAW-Wert angezeigt, danach folgt die Spannungsausgabe.

In der Initialisierungsfunktion wählen wir zuerst die Referenzspannung aus, die wir verwenden möchten, in unserem Beispiel *AVCC*, also die Versorgungsspannung (5 V). Dazu setzen wir im *ADMUX*-Register die beiden Bits REFS1 und REFS0 auf 1. Beim Einschalten des Controllers steht im Register *ADCSRA* (ADC-Statusregister) das *ADFR*(Free Running Mode = Messung erfolgt dauerhaft)-Bit bereits auf 0. Wir müssen es nicht extra auf Null setzen, wenn wir eine Einzelmessung (single conversion) machen möchten. Dabei wird immer eine Messung gestartet, wenn wir das möchten.

```
ADMUX = (0<<REFS1) | (1<<REFS0);    // AVCC als Referenz benutzen
```

**ADMUX – ADC multiplexer selection register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x7C) | REFS1 | REFS0 | ADLAR | – | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| Read/write | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W | |
| Initial value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Abb. 9.48:** Das ADMUX-Register – es ist für die Auswahl des Kanals und der Referenzspannung zuständig und dafür, ob das Ergebnis rechts- oder linksbündig dargestellt werden soll.

- Bit 7:6 – REFS1:0: Reference selection bits
- Bit 5 – ADLAR: ADC left adjust result
- Bits 3:0 – MUX3:0: Analog channel selection bits

| REFS1 | REFS0 | Voltage reference selection |
|---|---|---|
| 0 | 0 | AREF, internal $V_{ref}$ turned off |
| 0 | 1 | $AV_{CC}$ with external capacitor at AREF pin |
| 1 | 0 | Reserved |
| 1 | 1 | Internal 1.1V voltage reference with external capacitor at AREF pin |

**Abb. 9.49:** Tabelle zur Auswahl der Referenzspannung

Nun müssen wir den Vorteiler konfigurieren. Er gibt an, wie schnell der Multiplexer umgeschaltet wird. Hier gilt: je langsamer, desto genauer die Messung. Wir verwenden Prescaler 8, also schon recht flott. Sie können gern mit den restlich aufgezeigten Möglichkeiten experimentieren und beobachten, wie sich der Messwert verhält.

```
ADCSRA = (0<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);   // Frequenzvorteiler 8
```

| ADPS2 | ADPS1 | ADPS0 | Division factor |
|---|---|---|---|
| 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 |

**Abb. 9.50:** Tabelle für den Vorteiler (Prescaler)

| MUX3..0 | Single ended input |
|---|---|
| 0000 | ADC0 |
| 0001 | ADC1 |
| 0010 | ADC2 |
| 0011 | ADC3 |
| 0100 | ADC4 |
| 0101 | ADC5 |
| 0110 | ADC6 |
| 0111 | ADC7 |
| 1000 | (reserved) |
| 1001 | (reserved) |
| 1010 | (reserved) |
| 1011 | (reserved) |
| 1100 | (reserved) |
| 1101 | (reserved) |
| 1110 | 1.1V ($V_{BG}$) |
| 1111 | 0V (GND) |

**Abb. 9.51:** Multiplexer-Tabelle zur Auswahl des Messkanals und der internen Referenz

Nun müssen wir den ADC noch einschalten. Das geschieht im ADC-Statusregister mit dem Bit *ADEN* (Analog Enalbe).

```
ADCSRA |= (1<<ADEN);        // ADC aktivieren
```

**ADCSRA – ADC control and status register A**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x7A) | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| 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 | |

**Abb. 9.52:** ADC Control und Status-Register ADCSRA; hier müssen wir das ADEN-Bit auf 1 setzen.

- Bit 7 – ADEN: ADC enable
- Bit 6 – ADSC: ADC start conversion
- Bit 5 – ADATE: ADC auto trigger enable
- Bit 4 – ADIF: ADC interrupt flag
- Bit 3 – ADIE: ADC interrupt enable
- Bits 2:0 – ADPS2:0: ADC prescaler select bits

Nun starten wir die erste Messung, indem wir im Register *ADCSRA* das Bit *ADCSC* (ADC Single Conversion) auf 1 setzen. Danach warten wir, bis die Messung abgeschlossen ist und das richtige Ergebnis

abgeholt werden kann. Dazu kontrollieren wir im ADC-Statusregister *ADCSRA* das Bit *ADCSC*. Es ist so lange 1, bis die Messung beendet ist.

```
ADCSRA |= (1<<ADSC);                // Single ADC-Wandlung Start
while (ADCSRA & (1<<ADSC));         // auf Abschluss der Konvertierung warten
```

Der Messwert ADCW muss einmal gelesen werden, sonst wird das Ergebnis der nächsten Wandlung nicht übernommen. Dazu holen wir das Ergebnis einmal ab, denn wir können es mit *ADCW* komplett lesen. Das Ergebnis steht eigentlich in zwei getrennten Registern mit den Namen *ADCH* und *ADCL*. Sie können versuchsweise auch die Daten getrennt abholen und zusammensetzen. Dazu müssen Sie beachten, dass zuerst das *ADCL*-Register und dann erst das *ADCH*-Register gelesen wird. Das Ergebnis besteht als High- und Low-Byte. Das High-Byte multiplizieren wir mit 256 (ADCH << 8 ist eine Multiplikation mit 256) und addieren das Low-Byte hinzu.

```
// Auf einmal abholen
return ADCW;

// Einzeln abholen und zusammensetzen
return ADCL + (ADCH<<8);
```

**24.8.3 ADCL and ADCH – The ADC data register**

*24.8.3.1 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 | |
| Read/write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

*24.8.3.2 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 | |
| Read/write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Abb. 9.53:** Die beiden Register, die den Messwert enthalten

**Hinweis**
Nach der Aktivierung des ADC wird ein *Dummy-Readout* empfohlen. Man liest also einen Wert und verwirft ihn, um den ADC »warmlaufen« zu lassen. Die erste Messung des ADC ist in der Regel nicht zu gebrauchen und sollte nicht ins Messergebnis mit einfließen. In der Praxis haben sich als Sicherheit drei Messungen zum Verwerfen bewährt, bevor die eigentliche Messung beginnt.

Wir haben zum komfortablen Lesen des Werts im Beispiel zwei Funktionen: *ADC_Read* und *ADC_Read_Avg.* Die erste Funktion liest den eigentlichen Wert aus dem Register. Dabei setzt sie zuerst den Multiplexer mit *ADMUX = (ADMUX & ~(0x1F)) | (channel & 0x1F)*, ohne dabei die anderen Bits im Register zu beeinflussen, und startet danach die Messung. Er wartet dann, wie bereits bekannt, auf die abgeschlossene Messung, liest den Wert aus und gibt ihn mit *return* zurück.

```
unsigned int ADC_Read(unsigned char channel )
{
   // Kanal wählen, ohne andere Bits zu beeinflussen
   ADMUX = (ADMUX & ~(0x1F)) | (channel & 0x1F);
   ADCSRA |= (1<<ADSC);           // eine Wandlung "single conversion"
   while (ADCSRA & (1<<ADSC))     // auf Abschluss der Konvertierung warten
   return ADCW;                   // ADC auslesen und zurückgeben
}
```

Die Funktion *ADC_Read_Avg* ist ein digitaler Filter zur Mittelwertberechnung. Da der ADC nicht sonderlich hochwertig und genau ist, kann man mittels Mehrfachmessung (arithmetischer Mittelwert) die Genauigkeit steigern. Wir müssen den gewünschten Pin (Channel) angeben und festlegen, wie oft die Messung stattfinden soll. Je mehr Messungen wir machen, desto stabiler wird unser Messwert. Jedoch benötigt dies auch wesentlich mehr Zeit und Speicher (RAM). Hier ist also je nach Aufgabe abzuwägen, was wichtiger ist. Eine weitere Neuerung in diesem Beispiel ist, dass wir in der For-Schleife die Zählervariable deklarieren. *for (unsigned char i = 0; i < average; ++i)* ist eine gebräuchliche Kurzschreibweise und macht den Programm-Code übersichtlicher. Die Messungen werden zunächst mit *result += ADC_Read (channel)* zusammengezählt (gleichzustellen mit result = result + ADC_Read(channel)). Auch hier kann man sich mit dieser Schreibweise Zeit und Tipparbeit sparen. Die Pause nach jeder Messung kann auch entfernt werden, da wir ohnehin warten, bis eine Messung beendet wurde. Sie hat nur bei sehr verrauschten Signalen einen Vorteil, weil so in größeren Intervallen der Mittelwert gebildet wird. Hier können Sie also noch experimentieren. Zuletzt wird das Ergebnis von *result* durch die Anzahl der Messungen dividiert und mit *unsigned int* »gecastet«, da wir der Funktion ein *unsigned int* zurückgeben müssen.

```
unsigned int ADC_Read_Avg(unsigned char channel, unsigned char average)
{
   unsigned long result = 0;

   for (unsigned char i = 0; i < average; ++i)
   {
      result += ADC_Read(channel);
      _delay_us(250);
   }

   return (unsigned int)(result / average);
}
```

Der Aufruf dieser Funktion wäre für 64 Messungen an Kanal 7 *raw = ADC_Read_Avg(7,64);*.

**Tipp**
Weitere allgemeine Informationen über Analog-digital-Konverter und Spannungsteiler zur Messwerterweiterung finden Sie unter *http://de.wikipedia.org/wiki/Analog-Digital-Umsetzer* und *http://de.wikipedia.org/wiki/Spannungsteiler*

## 9.13 Blinker mit einstellbarer Intervallzeit

Da wir nun in der Lage sind, analoge Werte einzulesen, können wir einen Blinker mit variabler Intervallzeit programmieren. Das kleine Programm liest den Wert von ADC7 aus. Ihn verwenden wir als Pausezeit zwischen den Zustandswechseln der LED (ein/aus).

**Beispiel:** Intervallblinker

```
#define F_CPU 8000000

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

// ADC initialisieren
void ADC_Init(void)
{
   unsigned int result;

   ADMUX = (0<<REFS1) | (1<<REFS0);                    // AVCC als Referenz benutzen
   ADCSRA = (0<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);      // Frequenzvorteiler 8
   ADCSRA |= (1<<ADEN);                                // ADC aktivieren

   ADCSRA |= (1<<ADSC);            // Single ADC-Wandlung start
   while (ADCSRA & (1<<ADSC));     // auf Abschluss der Konvertierung warten

   // ADCW muss einmal gelesen werden,
   // sonst wird Ergebnis der nächsten Wandlung nicht übernommen.
   result = ADCW;
}

// ADC-Einzelmessung
unsigned int ADC_Read(unsigned char channel )
{
   // Kanal wählen, ohne andere Bits zu beeinflussen
   ADMUX = (ADMUX & ~(0x1F)) | (channel & 0x1F);
   ADCSRA |= (1<<ADSC);            // eine Wandlung "single conversion"
   while (ADCSRA & (1<<ADSC))      // auf Abschluss der Konvertierung warten
   //return ADCW;                  // ADC auslesen und zurückgeben
   return ADCL + (ADCH<<8);
}

void Wait(unsigned int dly_ms)
{
```

**Weitere Informationen zur Sprache *C* finden Sie im Internet unter folgenden Links:**

*http://www.galileocomputing.de/openbook/c_von_a_bis_z/*

*http://de.wikibooks.org/wiki/C-Programmierung*

*http://suparum.rz.uni-mannheim.de/manuals/c/cde.htm*

*http://www.roboternetz.de/wissen/index.php/C-Tutorial*

*http://www.its.strath.ac.uk/courses/c/*

# 12 Bezugsquellen

Conrad Electronic SE
Klaus-Conrad-Straße 1
92240 Hirschau
*www.conrad.de*

Electronic Assembly GmbH
Zeppelinstraße 19
82205 Gilching bei München
*http://www.lcd-module.de*

Sommer-Robotics
Ulli Sommer
Bahnhofstraße 8
92726 Waidhaus
*www.sommer-robotics.de*

Das Franzis Lernpaket

# Microcontroller in C programmieren

**Die ATMEL-AVR-Mikrocontroller-Familie erfreut sich seit Jahren größter Beliebtheit – nicht nur bei Entwicklungsprofis, sondern zunehmend auch bei Hobbyprogrammierern. Dieses Lernpaket macht Sie auf einfache und praktische Weise damit vertraut: Sie entdecken die spannende Welt der ATmega-Programmierung und setzen auf dieser Basis zahlreiche verblüffende Projekte um.**

## Experimentieren und anwenden

Das Lernpaket eröffnet Ihnen Schritt für Schritt die Grundlagen der Mikrocontroller-Technik, des ATMEL-AVR-ATmega88-Controllers und seiner Programmierung. Für den Aufbau der Versuche erhalten Sie genaue Anweisungen, Zeichnungen, Datenblätter und Fotos. Zu jedem Versuch gibt es außerdem einen Programmcode auf CD, den Sie selbst auf der mitgelieferten Hardware testen und nachvollziehen können. Danach sind Sie in der Lage, eigene Programme in „C" zu schreiben – sei es für die heimische Hausautomatisierung oder für den beruflichen Einsatz – sowie einfache Hardwareschaltungen zu realisieren und diese Controller selbstständig zu programmieren.

## Ein leicht verständliches Handbuch

Das beiliegende Handbuch führt Sie Schritt für Schritt ohne theoretischen Ballast direkt von den technischen Grundlagen zum fertigen Projekt. Der Spaß liegt im Machen und im Erleben! Jedes Experiment wird leicht verständlich und detailliert erklärt. Dazu gehören auch der eigenständige Aufbau einer Mikrocontroller-Schaltung und die Anbindung von Hardware an den Controller sowie die Programmierung mittels eines externen Programmiergeräts für eigene Anwendungen.

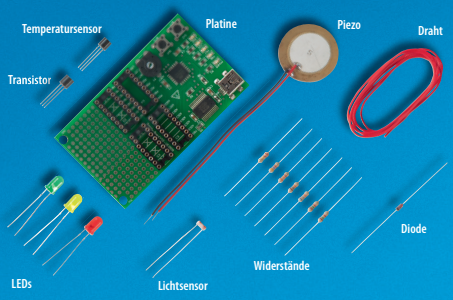
## Projekte, die wirklich funktionieren

Dieses Lernpaket zeichnet sich durch hohe Qualität und Praxistauglichkeit aus. Die Experimente wurden tausendfach durchgespielt. Sie können also sicher sein, dass auch bei Ihnen zu Hause alles klappt. Hand drauf: Dieses Franzis Lernpaket hält, was es verspricht.

## Diese Projekte führen Sie durch:

- Metronom
- Fahrradrücklicht-Flasher
- Licht ein/aus bei Tastendruck
- Code-Schloss
- State machine (Automat)
- Kondensatormessgerät
- Steuern mit dem PC
- Digitalvoltmeter mit Average-Digitalfilter
- Blinker mit einstellbarer Intervallzeit
- Diodentester
- Thermoschalter
- Thermometer LM335
- Thermograph mit StampPlot
- ATmega88-Speicheroszilloskop
- Alarmanlage mit Bewegungsmelder
- Uhr mit Weckfunktion
- Flackerlicht – virtuelle Kerze
- Entscheidungshilfe (Kopf oder Zahl)

## Die Bauteile im Überblick




Das Experimentierboard, das mit einem ATmega88, einem FTDI FT232RL und diverser Peripherie ausgestattet ist, können Sie später auch in Ihrer eigenen Applikation verbauen.

ISBN 978-3-645-65068-7

FRANZIS